Stretching Cube Attacks: Improved Methods to Recover Massive Superpolies

Jiahui He[1,4], Kai Hu[2], Bart Preneel[3], and Meiqin Wang[1,4,5(✉)]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China
hejiahui2020@mail.sdu.edu.cn, mqwang@sdu.edu.cn
[2] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
kai.hu@ntu.edu.sg
[3] imec-COSIC, KU Leuven, Leuven, Belgium
bart.preneel@esat.kuleuven.be
[4] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[5] Quan Cheng Shandong Laboratory, Jinan, China

Abstract. Cube attacks exploit the algebraic properties of symmetric ciphers by recovering a special polynomial, the superpoly, and subsequently the secret key. When the algebraic normal forms of the corresponding Boolean functions are not available, the division property based approach allows to recover the exact superpoly in a clever way. However, the computational cost to recover the superpoly becomes prohibitive as the number of rounds of the cipher increases. For example, the nested monomial predictions (NMP) proposed at ASIACRYPT 2021 stuck at round 845 for Trivium. To alleviate the bottleneck of the NMP technique, i.e., the unsolvable model due to the excessive number of monomial trails, we shift our focus to the so-called valuable terms of a specific middle round that contribute to the superpoly. Two new techniques are introduced, namely, Non-zero Bit-based Division Property (NBDP) and Core Monomial Prediction (CMP), both of which result in a simpler MILP model compared to the MILP model of MP. It can be shown that the CMP technique offers a substantial improvement over the monomial prediction technique in terms of computational complexity of recovering valuable terms. Combining the divide-and-conquer strategy with these two new techniques, we catch the valuable terms more effectively and thus avoid wasting computational resources on intermediate terms contributing nothing to the superpoly. As an illustration of the power of our techniques, we apply our framework to Trivium, Grain-128AEAD, Kreyvium and Acorn. As a result, the computational cost of earlier attacks can be significantly reduced and the exact ANFs of the superpolies for 846-, 847- and 848-round Trivium, 192-round Grain-128AEAD, 895-round Kreyvium and 776-round Acorn can be recovered

Due to page limits, all appendixes and some tables of this paper are provided in our full version [13].

S. Agrawal and D. Lin (Eds.): ASIACRYPT 2022, LNCS 13794, pp. 537–566, 2022.
https://doi.org/10.1007/978-3-031-22972-5_19

in practical time, even though the superpoly of 848-round Trivium contains over 500 million terms; this corresponds to respectively 3, 1, 1 and 1 rounds more than the previous best results. Moreover, by investigating the internal properties of Möbius transformation, we show how to perform key recovery using superpolies involving full key bits, which leads to the best key recovery attacks on the targeted ciphers.

Keywords: Cube attack · Superpoly · Trivium · Grain-128AEAD · Acorn · Kreyvium · Division property · Monomial prediction

1 Introduction

The cube attack, proposed by Dinur and Shamir at EUROCRYPT 2009 [8], is one of the most powerful cryptanalytic techniques against symmetric ciphers. Typically, any output bit of a cipher can be regarded as a polynomial of the public input $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ and the secret input $\boldsymbol{k} = (k_0, k_1, \ldots, k_{m-1})$, denoted by $f(\boldsymbol{x}, \boldsymbol{k})$. For a chosen term $\boldsymbol{x^u} = \prod_{u_i=1} x_i, \boldsymbol{u}, \boldsymbol{x} \in \mathbb{F}_2^n$, $f(\boldsymbol{x}, \boldsymbol{k})$ can be uniquely expressed as

$$f(\boldsymbol{x}, \boldsymbol{k}) = p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k}) \cdot \boldsymbol{x^u} + q(\boldsymbol{x}, \boldsymbol{k}) \ ,$$

where $p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k})$ is a Boolean function of $\boldsymbol{k}$, $\boldsymbol{x}[\bar{\boldsymbol{u}}] = \{x_i : u_i = 0\}$ and each term in $q(\boldsymbol{x}, \boldsymbol{k})$ misses at least one variable from $\{x_i : u_i = 1\}$. The polynomial $p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k})$ is called the superpoly of the cube term $\boldsymbol{x^u}$. After assigning a static value to $\boldsymbol{k}$ and $\boldsymbol{x}[\bar{\boldsymbol{u}}]$, the value of $p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k})$ can be computed by summing $f(\boldsymbol{x}, \boldsymbol{k})$ over a structure called *cube*, denoted as $\mathbb{C}_{\boldsymbol{u}}$, composed of all possible 0/1 combinations of $\{x_i : u_i = 1\}$.

To mount a cube attack, one first recovers the superpoly in an offline phase. Then, the value of the superpoly is obtained by querying the encryption oracle and computing the summation. From the equation between the superpoly and its value, information of the secret key can be revealed. Therefore, the superpoly recovery is a central step in the cube attack.

Traditional cube attacks [8,9,20,35] regard ciphers as black boxes so the superpolies are recovered experimentally. Only linear or quadratic superpolies are applicable. In [25], Todo et al. introduced cube attacks based on the Conventional Bit-based Division Property (CBDP). New methods based on CBDP [27] were proposed to efficiently identify secret variables that are not involved in the superpoly. After removing these uninvolved key bits and collecting the remaining key bits into a set J, the truth table of the superpoly can be recovered with time complexity $2^{|I|+|J|}$, where the set $I = \{i : u_i = 1\}$ is called *cube indices*. In [28], Wang et al. improved the precision of CBDP by considering cancellation characteristics of constant 1 bits, thus further lowering the complexity.

Exact Superpoly Recovery. Although the CBDP never produces a false positive error [17], it cannot accurately predict the existence of a monomial in the superpoly. A substantial amount of works have been carried out to get around this point. At Asiacrypt 2019, Wang et al. [29] managed to recover the exact superpoly for the first time with the pruning technique combined with the three-subset

bit-based division property. However, the value of this technique is limited as it requires the assumption that almost all elements in the so-called 1-subset can be pruned. In [30], Ye and Tian introduced the recursively-expressing method, which recursively splits the output bits into intermediate terms of smaller rounds and filters out these useless terms that contribute nothing to the superpoly. As a result, several superpolies recovered in [28] are proved to degenerate to constants. In [11,12], Hao et al. proposed the three-subset division property without unknown subsets (3SDPwoU) to recover the exact superpolies from the perspective of counting the number of three-subset trails. In [17], Hu et al. established the equivalence between monomial prediction and 3SDPwoU from the viewpoint of monomial propagations. In [36], Ye and Tian also developed a pure algebraic method to recover the exact superpoly. However, as the number of rounds of the cipher increases, such useful cubes are hard to find. Last year, Hu el al. embedded the monomial prediction technique into a nested framework, which allows them to recover massive superpolies [16] that contain almost 20 million terms.

Nested Monomial Predictions. In terms of structure, the nested monomial prediction [16] consists of two components, namely the *coefficient solver* and the *term expander*. Given a cube term $\boldsymbol{x}^{\boldsymbol{u}}$, the coefficient solver is designed to compute the superpoly of $\boldsymbol{x}^{\boldsymbol{u}}$ for a term of the current round, and the term expander is responsible for expressing unsolved terms as terms of a deeper round. At first, from top to bottom, the target output bit is expressed as a polynomial of the state bits of an intermediate round, then by iteratively calling the coefficient solver and expanding unsolved terms into terms of deeper rounds, the final superpoly can be recovered.

As mentioned, the cube attack is one of the powerful tools to evaluate the security of stream ciphers. It is important to explore its limits by recovering superpolies for as many rounds as possible. While the nested monomial predictions is efficient for massive superpolies (e.g., it can recover a superpoly for 845-round Trivium that contains 19,967,968 terms), it has been stuck at 845 rounds of Trivium. In order to recover superpolies for more rounds, novel techniques are required.

Contributions. This paper provides new efficient methods to recover superpolies for more initialization rounds of stream ciphers such as Trivium [5], Grain-128AEAD [15], Kreyvium [6] and the authenticated encryption algorithm Acorn [31].

Recall that the framework of nested monomial predictions consists of two components, i.e., the coefficient solver and the term expander; we design two algorithms to greatly improve the efficiency of both of them.

- *Two-step strategy for the coefficient solver.* Unlike the monomial prediction, our coefficient solver takes two steps to compute the superpoly. During the first stage, the intermediate monomials related to the superpoly are determined utilizing a new technique called core monomial prediction. Next, by applying the monomial prediction to these intermediate monomials and collecting the results, the final superpoly can be recovered quickly.

Table 1. Verification and comparison of superpolies for 843-, 844- and 845-round Trivium† from [16].

I	Round	Status	TimeCost ([16])	TimeCost (ours)
I_0	843	Verified(✓)	Less than 2 weeks	2 h
I_1	843	Verified(✓)		4 h
I_2	843	Verified(✓)		1 h
I_3	843	Verified(✓)		1.5 h
I_4	843	Verified(✓)		1 day and 17 h
I_2	844	Verified(✓)	17 h	5 h
I_3	844	Verified(✓)	6 h	2.5 h
I_2	845	Verified(✓)	about 16 days	19.5 h
I_3	845	Verified(✓)	4 days and 9 h	8.5 h

†: The time consumption of the superpoly recovery of 843-, 844-round Trivium is stated as 'less than two weeks' in [16]. The concrete time cost for 844- and 845-round Trivium was obtained by rerunning the code provided by [16] on our platform.

– *Fast-descent algorithm for the term expander.* Instead of expressing the current terms as a polynomial of indistinguishable terms of a deeper round and then testing them one by one, our term expander uses Gurobi's callback function to automatically filter out the useless terms internally during each expansion, which makes the number of rounds drop faster and reduces the time spent on useless terms.

Our new framework offers substantial efficiency improvements in recovering superpolies compared to the nested monomial prediction. We verified superpolies for Trivium recovered in [16]. As a result, our framework allows to recover superpolies in a few hours rather than in weeks. The comparison is illustrated in Table 1.

More importantly, our framework is able to recover superpolies for more initialization rounds of high profile symmetric-key ciphers including Trivium (ISO/IEC standard [3,5]), Grain-128AEAD (a member of the ten finalist candidates of the NIST LWC standardization process [15]), Kreyvium (designed for Fully Homomorphic Encryption [6]) and Acorn (a member of the final portfolio of the CAESAR competition for Lightweight applications [31]). For Trivium, we are the first to obtain superpolies for up to 848-round Trivium. We also recovered the superpolies of 192-round Grain-128AEAD, 895-round Kreyvium and 776-round Acorn, all penetrating one more round than the previous best results. By investigating the internal properties of Möbius transformation, we propose a novel method to perform key recovery inside Möbius transformation. The summary of our cube attack results and the previous best results are provided in Table 2.

All source codes for recovering the superpolies in this paper are provided in the public git repository https://github.com/viocently/ekignrb9lc.git.

Table 2. Summary of our cube attack results and the previous best results. #Cube means the number of cubes whose superpolies are recovered.

Cipher	Rounds	#Cube	Cube size	Time complexity	Attack types	Reference
Trivium	≤806	–	–	Practical	key recovery	[8,9,20,23,37]
	808	37	39–41	Practical	Key recovery	[23]
	≤844	–	–	$2^{75} \sim 2^{79.6}$	Key recovery	[11,12,16,17,19,23] [9,26,29,35,37,38]
	845	2	54–55	2^{78}	Key recovery	[16]
	846	6	51–54	2^{79}	Key recovery	Sect. 6.1
	847	2	52–53	2^{79}	Key recovery	Sect. 6.1
	848	1	52	2^{79}	Key recovery	Sect. 6.1
Grain‡	169	–	–	Practical	Condit. Diff.	[18]
	–	–	–	Practical	State recovery	[7]
	≤190	–	–	$2^{123} \sim 2^{129}$	Key recovery	[11,12,25,26,28]
	191	2	95–96	2^{127}	Key recovery	[16]
	192	1	94	2^{127}	Key recovery	Sect. 6.2
Kreyvium	≤893	–	–	$2^{119} \sim 2^{127}$	Key recovery	[10–12,23,26,28]
	894	1	119	2^{127}	Key recovery	[16]
	895	1	120	2^{127}	Key recovery	Sect. 6.3
Acorn	≤774	–	–	2^{127}	Key recovery	[10,11,26,28]
	775	6	127	2^{127}	Distinguisher	[34]
	775	5	126	2^{126}	Distinguisher	[33]
	775	1	126	2^{127}	Key recovery	[33]
	776	2	126	2^{127}	Key recovery	Sect. 6.4

‡: Grain-128a or Grain-128AEAD.

2 Division Property and Monomial Prediction

2.1 Notations and Definitions

We use bold italic lowercase letters to represent bit vectors. For an n-bit vector $\boldsymbol{u} = (u_0, \cdots, u_{n-1}) \in \mathbb{F}_2^n$, its complementary vector is denoted by $\bar{\boldsymbol{u}}$, where $u_i \oplus \bar{u}_i = 1$ for $0 \le i < n$. The Hamming weight of $\boldsymbol{u}$ is $wt(\boldsymbol{u}) = |\{i : u_i = 1\}|$. The concatenation of $\boldsymbol{u}_0$ and $\boldsymbol{u}_1$ is denoted by $\boldsymbol{u}_0||\boldsymbol{u}_1$. For $\boldsymbol{u}, \boldsymbol{x} \in \mathbb{F}_2^n$, $\boldsymbol{x}[\boldsymbol{u}]$ denotes a sub-vector of $\boldsymbol{x}$ with respect to $\boldsymbol{u}$ as $\boldsymbol{x}[\boldsymbol{u}] = (x_{i_0}, x_{i_1}, \ldots, x_{i_{wt(\boldsymbol{u})-1}}) \in \mathbb{F}_2^{wt(\boldsymbol{u})}$, where $i_j \in \{0 \le i \le n-1 : u_i = 1\}$ and $(i_0, \ldots, i_{wt(\boldsymbol{u})-1})$ is arranged from the least to the greatest. For any n-bit vectors $\boldsymbol{u}$ and $\boldsymbol{u}'$, we define $\boldsymbol{u} \succeq \boldsymbol{u}'$ if $u_i \ge u'_i$ for all i. Similarly, we define $\boldsymbol{u} \preceq \boldsymbol{u}'$ if $u_i \le u'_i$ for all i. Bold italic lowercase letters with superscript are used to represent the bitvector in a certain round. Particularly, $\boldsymbol{u}^{(i)}$ represents a bitvector in round i. We use $\mathbf{0}^n$ or $\mathbf{1}^n$ to represent an all-zeros or all-ones vector of length n.

Blackboard bold uppercase letters (e.g. $\mathbb{S}, \mathbb{K}, \mathbb{U}, \ldots$) are used to represent sets of bit vectors. In the propagation of some algebraic properties such as CBDP, the set generated in the i-th round is denoted as $\mathbb{S}^{(i)}$.

Boolean Function. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function whose *algebraic normal form* (ANF) is

$$f(\boldsymbol{x}) = f(x_0, x_1, \ldots, x_{n-1}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \prod_{i=0}^{n-1} x_i^{u_i} \ ,$$

where $a_{\boldsymbol{u}} \in \mathbb{F}_2$, and

$$\boldsymbol{x}^{\boldsymbol{u}} = \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \prod_{i=0}^{n-1} x_i^{u_i} \text{ with } x_i^{u_i} = \begin{cases} x_i, \text{ if } u_i = 1 \ , \\ 1, \text{ if } u_i = 0 \ , \end{cases}$$

is called a monomial. If the coefficient of $\boldsymbol{x}^{\boldsymbol{u}}$ in f is 1, i.e., $\boldsymbol{x}^{\boldsymbol{u}}$ is contained by f, then we denote it by $\boldsymbol{x}^{\boldsymbol{u}} \to f$. Otherwise, we denote the absence of $\boldsymbol{x}^{\boldsymbol{u}}$ in f by $\boldsymbol{x}^{\boldsymbol{u}} \nrightarrow f$. In this work, we will use $\boldsymbol{x}^{\boldsymbol{u}}$ and $\pi_{\boldsymbol{u}}(\boldsymbol{x})$ interchangeably to avoid using the awkward notation $\boldsymbol{x}^{(i)^{\boldsymbol{u}^{(j)}}}$ when both $\boldsymbol{x}$ and $\boldsymbol{u}$ have superscripts.

Vectorial Boolean Function. Let $\boldsymbol{f} : \mathbb{F}_2^m \to \mathbb{F}_2^n$ be a vectorial Boolean function with $\boldsymbol{y} = (y_0, y_1, \ldots, y_{m-1}) = \boldsymbol{f}(\boldsymbol{x}) = (f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_{n-1}(\boldsymbol{x}))$. For $\boldsymbol{v} \in \mathbb{F}_2^n$, we use $\boldsymbol{y}^{\boldsymbol{v}}$ to denote the product of some coordinates of $\boldsymbol{y}$:

$$\boldsymbol{y}^{\boldsymbol{v}} = \prod_{i=0}^{m-1} y_i^{v_i} = \prod_{i=0}^{m-1} (f_i(\boldsymbol{x}))^{v_i},$$

which is a Boolean function in $\boldsymbol{x}$.

2.2 Conventional Bit-Based Division Property

The word-based division property [24] was proposed by Todo originally as a generalization of integral attack. Subsequently, by shifting the propagation of the division property to the bit level, Todo and Morii [27] introduced the bit-based division property (CBDP).

Definition 1 (Conventional bit-based division property (CBDP) [27]). *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^m$ and $\boldsymbol{k} \in \mathbb{F}_2^m$. When the multiset has the division property $\mathcal{D}_{\mathbb{K}}^{1^m}$, the following conditions are fulfilled:*

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

In [32], Xiang et al. introduced the mixed integer linear programming (MILP) method to search for integral distinguishers of block ciphers based on CBDP. They first introduced the division trail as follows.

Definition 2 (Division Trail of CBDP [32]). *Let $\mathbb{D}_{\mathbb{K}^{(i)}}$ be the division property of the input for the ith round function. Consider the propagation of the division property $\{\boldsymbol{k}\} = \mathbb{K}^{(0)} \to \mathbb{K}^{(1)} \to \mathbb{K}^{(2)} \to \cdots \to \mathbb{K}^{(r)}$. For any bitvector $\boldsymbol{k}^{(i+1)} \in \mathbb{K}^{(i+1)}$, there must exist a bitvector $\boldsymbol{k}^{(i)} \in \mathbb{K}^{(i+1)}$ such that $\boldsymbol{k}^{(i)}$ can propagate to $\boldsymbol{k}^{(i+1)}$ by the propagation rules of CBDP. Furthermore, for $(\boldsymbol{k}^{(0)}, \boldsymbol{k}^{(1)}, ..., \boldsymbol{k}^{(r)}) \in (\mathbb{K}^{(0)} \times \mathbb{K}^{(1)} \times \cdots \times \mathbb{K}^{(r)})$, we call $(\boldsymbol{k}^{(0)} \to \boldsymbol{k}^{(1)} \to \cdots \to \boldsymbol{k}^{(r)})$ an r-round division trial if $\boldsymbol{k}^{(i)}$ can propagate to $\boldsymbol{k}^{(i+1)}$ for all $i \in \{0, 1, \cdots, r-1\}$.*

For a stream cipher, three fundamental operations, i.e., COPY, AND, and XOR are sufficient to cover all division trails. Xiang et al. showed how to model these three operations by inequalities. We present their MILP models in [13, Sup.Mat. B].

In our work, we use $\boldsymbol{k}^{(0)} \overset{\mathbb{K}_f}{\rightsquigarrow} \boldsymbol{k}^{(r)}$ to denote the existence of at least one division trail from $\boldsymbol{k}^{(0)}$ to $\boldsymbol{k}^{(r)}$ through the function $\boldsymbol{f}$. The set of all division trails from $\boldsymbol{k}^{(0)}$ to $\boldsymbol{k}^{(r)}$ is denoted as $\boldsymbol{k}^{(0)} \overset{\mathbb{K}_f}{\bowtie} \boldsymbol{k}^{(r)}$, whose size is denoted by $|\boldsymbol{k}^{(0)} \overset{\mathbb{K}_f}{\bowtie} \boldsymbol{k}^{(r)}|$. When $\boldsymbol{f}$ is not explicitly given or can be inferred from the context, we use $\boldsymbol{k}^{(0)} \overset{\mathbb{K}}{\rightsquigarrow} \boldsymbol{k}^{(r)}$ and $\boldsymbol{k}^{(0)} \overset{\mathbb{K}}{\bowtie} \boldsymbol{k}^{(r)}$ for simplicity.

2.3 Monomial Prediction

Let $\boldsymbol{f} : \mathbb{F}_2^{n_0} \to \mathbb{F}_2^{n_r}$ be a composite vectorial Boolean function built by composition from a sequence of vectorial Boolean functions $\boldsymbol{f}^{(i)} : \mathbb{F}_2^{n_i} \to \mathbb{F}_2^{n_{i+1}}, 0 \le i \le r-1$ whose ANFs are known, i.e.,

$$\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}. \tag{1}$$

Let $\boldsymbol{x}^{(i)} \in \mathbb{F}_2^{n_i}$ and $\boldsymbol{x}^{(i+1)} \in \mathbb{F}_2^{n_{i+1}}$ be the input and output variables of $\boldsymbol{f}^{(i)}$ respectively. We call an i-round monomial $\pi_{\boldsymbol{u}^{(i)}}(\boldsymbol{x}^{(i)})$ $(1 \le i \le r-1)$ an *intermediate monomial* or an *intermediate term*[1]. Starting from a monomial of $\boldsymbol{x}^{(0)}$, say $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$, all monomials of $\boldsymbol{x}^{(1)}$ satisfying $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)})$ can be derived; for every such $\pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)})$, we then find all $\pi_{\boldsymbol{u}^{(2)}}(\boldsymbol{x}^{(2)})$ satisfying $\pi_{\boldsymbol{u}^{(1)}}(\boldsymbol{x}^{(1)}) \to \pi_{\boldsymbol{u}^{(2)}}(\boldsymbol{x}^{(2)})$; such forward expansions continue until we arrive at the monomials of $\boldsymbol{x}^{(r)}$. Each transition from $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ denoted by

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

is called a monomial trail [17], denoted by $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, which is also used to indicate the existence of at least one monomial trail from $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$. All the trails from $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ are denoted by $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, which is the set of all trails. Whether $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ is determined by the size of $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, represented as $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})|$. If there is no trail from $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$, we say $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ and accordingly $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})| = 0$.

[1] In this paper, 'monomial' and 'term' have the same meaning.

Theorem 1 (Integrated from [11,12,14,17]). *Let* $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ *defined as above.* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ *if and only if*

$$|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})| \equiv 1 \pmod 2.$$

Propagation Rules of the Monomial Prediction. Each symmetric cipher can be decomposed into a sequence of the basic operations XOR, AND and COPY, hence it is sufficient to give propagation rules of the monomial prediction for these basic operations. To model the propagation of the monomial prediction for a vectorial Boolean function, a common method is to list all the possible (input, output) tuples according to the definition of the monomial prediction [17]. These tuples can be transformed into a set of linear inequalities [4,21,22], which are suitable for MILP modeling. The concrete propagation rules and models of the monomial prediction are provided in [13, Sup.Mat. B].

Gurobi's PoolSearchMode and Callback Functions. In our work, we choose the Gurobi solver [1] as our MILP tool. Since our coefficient solver follows the idea of counting propagation trails similar to [11,12,17], we turn on Gurobi's PoolSearchMode with $\mathcal{M}$.PoolSearchMode $\leftarrow 1$ to extract all possible solutions of a model. By adding a lazy constraint to the MILP model from within a callback function, Gurobi allows users to cut off a feasible solution during the search. We use $\mathcal{M}$.LazyConstraints $\leftarrow 1$ to turn on lazy constraints. For more on Gurobi's callback functions and PoolSearchMode, readers are requested to refer to the Gurobi manual [2]. We would like to mention that the callback function is also used in the code provided by [11,12].

2.4 Cube Attack

In the context of the cube attack, the output bit of a symmetric cipher is typically regarded as a parameterized Boolean function $f : \mathbb{F}_2^{n+m} \rightarrow \mathbb{F}_2$ whose inputs are the public variables $\boldsymbol{x} \in \mathbb{F}_2^n$ and the secret ones $\boldsymbol{k} \in \mathbb{F}_2^m$. For a constant bitvector $\boldsymbol{u} \in \mathbb{F}_2^n$ indexed by $I = \{0 \leq i \leq n-1 : u_i = 1\} \subseteq \{0, 1, \ldots, n-1\}$, the ANF of $f(\boldsymbol{x}, \boldsymbol{k})$ can be uniquely represented as

$$f(\boldsymbol{x}, \boldsymbol{k}) = p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k}) \cdot \boldsymbol{x}^{\boldsymbol{u}} + q(\boldsymbol{x}, \boldsymbol{k}),$$

where each term of $q(\boldsymbol{x}, \boldsymbol{k})$ misses at least one variable from $\{x_i : u_i = 1\}$. $\boldsymbol{x}^{\boldsymbol{u}}$ is called a *cube term*, and $\mathbb{C}_{\boldsymbol{u}}$ (or $\mathbb{C}_I$) is called a *cube*, which is the set $\{\boldsymbol{x} \in \mathbb{F}_2^n : \boldsymbol{x} \preceq \boldsymbol{u}\}$. The sum of f over all values of the cube $\mathbb{C}_{\boldsymbol{u}}$ is

$$\bigoplus_{\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{u}}} f(\boldsymbol{x}, \boldsymbol{k}) = \bigoplus_{\boldsymbol{x} \in \mathbb{C}_{\boldsymbol{u}}} (p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k}) \cdot \boldsymbol{x}^{\boldsymbol{u}} \oplus q(\boldsymbol{x}, \boldsymbol{k})) = p(\boldsymbol{x}[\bar{\boldsymbol{u}}], \boldsymbol{k}),$$

which is exactly the coefficient of $\boldsymbol{x}^{\boldsymbol{u}}$ in $f(\boldsymbol{x}, \boldsymbol{k})$, denoted by $\mathsf{Coe}(f(\boldsymbol{x}, \boldsymbol{k}), \boldsymbol{x}^{\boldsymbol{u}})$ in our work. If we assign a fixed value to $\boldsymbol{x}[\bar{\boldsymbol{u}}]$, then $\mathsf{Coe}(f(\boldsymbol{x}, \boldsymbol{k}), \boldsymbol{x}^{\boldsymbol{u}})$ becomes a Boolean function of $\boldsymbol{k}$.

As mentioned, the superpoly recovery is of significant importance in the cube attack. If the recovered superpoly is constant 0 or 1, we actually find a distinguisher for the cipher. If the superpoly is a Boolean function of $\boldsymbol{k}$, then key bits can be extracted. In particular, a balanced superpoly always contains one bit of information on average. The remaining key bits can be recovered through exhaustive search.

2.5 Superpoly Recovery with the Monomial Prediction/3SDPwoU

To the best of our knowledge, monomial prediction/3DSPwoU [11,12,17] can reach the perfect accuracy in determining the existence of a certain monomial in f. To recover the superpoly of a cube term $\boldsymbol{x}^{\boldsymbol{u}}$ with the monomial prediction/3DSPwoU, the initial state variables of the MILP model are divided into three parts: the public input (plaintext, IV or tweak), the secret input (the key bits) and the constant input.

The public input variables are constrained to be equal to $\boldsymbol{u}$. The secret input variables are left as free variables without any constraints. For the constant 0 bits, we constrain the corresponding MILP variables to 0, while for the constant 1 bits, we let their variables be free. We then model the propagation of monomial trails to f. Each solution of the model is a valid monomial trail of the form $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow f$. By collecting monomials $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}}$ occurring an odd number of times in all solutions and adding them, we can obtain the superpoly of $\boldsymbol{x}^{\boldsymbol{u}}$ as

$$\mathsf{Coe}(f, \boldsymbol{x}^{\boldsymbol{u}}) = \bigoplus_{|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \bowtie f| \equiv 1 \pmod 2} \boldsymbol{k}^{\boldsymbol{w}}.$$

In [17], Hu et al. observed that for the composite function f, where

$$f = f^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)},$$

if $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow f$, then for $0 < i < r$,

$$|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f| \equiv \sum_{\pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)}) \to f} \left|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})\right| \pmod 2.$$

Instead of computing $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie f|$ for a large r, we can compute $|\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \bowtie \pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})|$ for all $\pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)})$ satisfying $\pi_{\boldsymbol{u}^{(r-i)}}(\boldsymbol{x}^{(r-i)}) \to f$ with a lower computational difficulty. In practice, such a divide-and-conquer strategy resulted in a significant speed-up of the search.

3 Nested Monomial Predictions (NMP)

At Asiacrypt 2021, Hu et al. proposed a nested framework, called Nested Monomial Predictions, to recover the superpoly of Trivium up to 845 rounds. In this section, we briefly introduce the workflow of this framework and divide the structure of this framework into two parts, namely the *coefficient solver* and the *term expander*.

3.1 The Workflow

Given a parameterized Boolean function which consists of a sequence of simple vectorial Boolean functions as

$$f(\boldsymbol{x}, \boldsymbol{k}) = f^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}(\boldsymbol{x}, \boldsymbol{k}),$$

let the output of $\boldsymbol{f}^{(i)}$ be $\boldsymbol{s}^{(i+1)}$. Assume we want to compute $\mathsf{Coe}\,(f, \boldsymbol{x}^{\boldsymbol{u}})$. The nested monomial predictions works as follows:

1. Initialize a variable $r_l = r$ and a set $\mathbb{S}_u^{(r_l)} = \{f\}$.
2. Choose r_n such that $0 < r_n < r_l$ according to some criterion.
3. Express each term in $\mathbb{S}_u^{(r_l)}$ as a polynomial of $\boldsymbol{s}^{(r_n)}$ using the monomial prediction technique and save the terms of this polynomial in a multiset $\mathbb{T}^{(r_n)}$.
4. Count the number of occurrences for each element in $\mathbb{T}^{(r_n)}$ and add the elements occurring an odd number of times to a set $\mathbb{S}^{(r_n)}$.
5. For each term $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \in \mathbb{S}^{(r_n)}$, construct a MILP model of the monomial prediction and invoke Gurobi to solve it. If the model has solutions and is successfully solved, then this term is partitioned into $\mathbb{S}_p^{(r_n)}$ and we can compute $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$, which is collected as a part of $\mathsf{Coe}\,(f, \boldsymbol{x}^{\boldsymbol{u}})$; if the model has no solutions, then this term is partitioned into $\mathbb{S}_0^{(r_n)}$ and discarded; if the model isn't solved in limited time, we partition this term into the set $\mathbb{S}_u^{(r_n)}$.
6. If the set $\mathbb{S}_u^{(r_n)}$ is not empty, we update the variable $r_l = r_n$ and regard the set $\mathbb{S}_u^{(r_n)}$ as $\mathbb{S}_u^{(r_l)}$, then jump to step 2. Otherwise we have successfully compute $\mathsf{Coe}\,(f, \boldsymbol{x}^{\boldsymbol{u}})$.

In step 2 of NMP, r_n is chosen as the round that makes the size of $\mathbb{T}^{(r_n)}$ larger than N for the first time, where N can take the value 10 000 or 100 000. Interested readers can refer to [16] for more details.

3.2 The Structure of the Nested Monomial Prediction

In terms of the structure, the nested monomial prediction consists of two components. In Sect. 3.1, step 3 and 4 are responsible for expanding terms in $\mathbb{S}_u^{(r_l)}$ into terms of a deeper round r_n represented by $\mathbb{S}^{(r_n)}$, while the step 5 attempts to compute $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ for each term $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ in $\mathbb{S}^{(r_n)}$. This leads to the following two concepts.

Term Expander. For an algorithm $\mathcal{H}$ of a specific cryptographic algorithm $\mathtt{X}$, if given the last round r_l, the set $\mathbb{S}_u^{(r_l)}$ containing terms of round r_l, the next round r_n and other auxiliary parameters as input, the algorithm $\mathcal{H}$ can always output all $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying $\sum_{\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}) \in \mathbb{S}_u^{(r_l)}} |\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \bowtie \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})| \equiv 1 \pmod 2$, then we say $\mathcal{H}$ is a term expander of $\mathtt{X}$.

Coefficient Solver. For an algorithm $\mathcal{H}$ of a specific cryptographic algorithm $\mathtt{X}$, if given the last round r_l, a term $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ of round r_l, $\boldsymbol{u}$ indicating the

Algorithm 1: Generic structure of the nested monomial predictions [16]

Procedure `SuperpolyRecFramework`(*the target output bit* $f(\boldsymbol{x},\boldsymbol{k})$*, the target round* r*,* $\boldsymbol{u}$ *indicating the cube term*)**:**
- Prepare a polynomial $p = 0$
- Initialize $r_l = r, \mathbb{S}_u^{(r_l)} = \{f\}$
- **while** $\mathbb{S}_u^{(r_l)} \neq \emptyset$ **do**
  - $r_n = \texttt{ChooseRiX}(\mathbb{S}_u^{(r_l)}, r_l, ...)$
  - $\mathbb{S}^{(r_n)} = \texttt{TermExpanderX}(\mathbb{S}_u^{(r_l)}, r_l, r_n, ...)$
  - **for** $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \in \mathbb{S}^{(r_n)}$ **do**
    - $\tau = \texttt{ChooseTiX}(r_n)$
    - $(status, p^{(r_n)}) = \texttt{CofSolverX}(\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}), r_n, \boldsymbol{u}, \tau, ...)$
    - **if** *status* = SOLVED **then**
      - $p = p \oplus p^{(r_n)}$
    - **else if** *status* = TIMEOUT **then**
      - Insert $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ into $\mathbb{S}_u^{(r_n)}$
  - $r_l \leftarrow r_n$
  - $\mathbb{S}_u^{(r_l)} \leftarrow \mathbb{S}_u^{(r_n)}$
- **return** p

cube term $\boldsymbol{x}^{\boldsymbol{u}}$ (or other parameters that can identify the cube), the time limit τ and other auxiliary parameters as input, the algorithm $\mathcal{H}$ can always output either $\texttt{Coe}\left(\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$, no solution or timeout, then we say $\mathcal{H}$ is a coefficient solver of `X`.

In this paper, we denote the term expander and the coefficient solver by $\texttt{TermExpanderX}(\mathbb{S}_u^{(r_l)}, r_l, r_n, ...)$ and $\texttt{CofSolverX}(\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}), r_l, \boldsymbol{u}, \tau, ...)$ respectively, where we use ... to represent arbitrary auxiliary parameters. `TermExpanderX` returns a set containing all $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying $\sum_{\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}) \in \mathbb{S}_u^{(r_l)}} |\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \bowtie \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})| \equiv 1 \pmod 2$. `CofSolverX` returns a 2-tuple $(status, result)$, where *status* takes SOLVED, NOSOLUTION or TIMEOUT and *result* represents $\texttt{Coe}\left(\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ only when *status* = SOLVED. Using these notions, the generic structure of the nested monomial predictions can be described in Algorithm 1, where `ChooseRiX` and `ChooseTiX` represent the process of selecting r_n and τ.

Following the generic structure, the nested monomial predictions utilizes the monomial prediction technique to build the term expander and the coefficient solver. As a result, the superpoly recovery of the target output bit is divided into superpoly recoveries of thousands of terms of fewer rounds, thereby reducing the computational difficulty. Our work in this paper also follows the generic structure, but with a more efficient term expander and coefficient solver.

4 New Coefficient Solver

4.1 Motivation

Although the monomial prediction technique can reach perfect accuracy in detecting if $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightarrow f$, it requires counting the number of monomial trails.

Such a task is impractical for a high number of rounds of a well-designed cryptographic algorithms, as the number of monomial trails grows almost exponentially with the number of rounds.

As mentioned in Sect. 2.5, the divide-and-conquer strategy can speed up the search compared with counting the number of monomial trails directly. Inspired by this, we construct a new coefficient solver that first divides the output bit of the current round into terms of a quite deep round, then solve these terms using the monomial prediction technique.

4.2 The Theory

For simplicity, we assume the term of the current round is $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ and we want to compute $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ with the coefficient solver. We divide $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ into terms of a reduced number $r_m < r$ of rounds. Naturally, we introduce the concept of valuable terms to capture those terms in round r_m that contribute to $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$.

Valuable Terms. According to the divide-and-conquer strategy, the monomial trails of the form $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ can be divided into monomial trails of the form $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ for each $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ satisfying $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, e.g.,

$$|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})| \equiv \sum_{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})} |\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| \pmod 2. \quad (2)$$

Note that if $|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| = 0$, $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ contributes nothing to $|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})|$. Therefore, to make it precise we rewrite the Eq. (2) as

$$|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})| \equiv \sum_{\substack{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}) \\ \boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})}} |\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \ltimes \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| \pmod 2. \quad (3)$$

Terms satisfying (A) $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, (B) $\exists \boldsymbol{k}^{\boldsymbol{w}}$ such that $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ are called *valuable terms* of round r_m, denoted by $VT^{(r_m)}$. Usually r_m is chosen not too large, say 90 for Trivium. Once we have recovered all $VT^{(r_m)}$s, we can compute $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ easily by applying the monomial prediction to compute $\mathsf{Coe}\left(VT^{(r_m)}, \boldsymbol{x}^{\boldsymbol{u}}\right)$ for each $VT^{(r_m)}$. Briefly speaking, the workflow of our coefficient solver is as follows:

1. Develop a method to recover $VT^{(r_m)}$s within the time limit τ. If it times out, return TIMEOUT; else if no $VT^{(r_m)}$ could be recovered, return NOSOLUTION; otherwise, if $VT^{(r_m)}$s are recovered successfully, go to step 2.
2. Apply the monomial prediction to each $VT^{(r_m)}$ to compute $\mathsf{Coe}\left(VT^{(r_m)}, \boldsymbol{x}^{\boldsymbol{u}}\right)$.
3. Sum all $\mathsf{Coe}\left(VT^{(r_m)}, \boldsymbol{x}^{\boldsymbol{u}}\right)$s to compute $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ and return SOLVED.

How to Recover $VT^{(r_m)}$. If a term $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ is a $VT^{(r_m)}$, then two conditions are necessary and sufficient, namely $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ (Condition

A) and $\exists \boldsymbol{k}^{\boldsymbol{w}}$ s.t. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ (Condition B). We need to construct a MILP model to describe these two conditions simultaneously.

At a first glance, both conditions can be described by the monomial prediction with the following structure

$$\overbrace{\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \xrightarrow{\mathbf{MP}} \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})}^{r_m \ rounds} = \overbrace{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \xrightarrow{\mathbf{MP}} \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})}^{r-r_m \ rounds}, \tag{4}$$

where $\xrightarrow{\mathbf{MP}}$ means the propagation is described according to the propagation rules of MP. However, since we need to incorporate these two conditions into one MILP model, such a structure is equivalent to computing $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ by the monomial prediction directly. We do not gain efficiency improvement from valuable terms.

Note that when describing Condition B, the monomial prediction is so accurate that we can even determine whether $\exists \boldsymbol{k}^{\boldsymbol{w}}$ s.t. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$. Therefore, a natural idea is to sacrifice some accuracy in exchange for efficiency. In next section, we provide two variants of bit-based division properties for efficient descriptions of Condition B. The first variant is called non-zero bit-based division property (NBDP): it simply excludes the propagation of CBDP related to constant 0 bits. The second is called the core monomial prediction (recall that the monomial prediction is an explanation of division properties): it ignores the role of constant bits and attempts to establish a set of rules to characterize those non-constant bits. Both variants play important roles in our new algorithms for recovering superpolies.

5 Two Variants of the Division Property for Describing Condition B

In this section, we present two techniques to describe Condition B under the assumption that non-cube public variables are set to 0. For convenience, we always consider an r_m-round cryptographic function $\boldsymbol{f} : \mathbb{F}_2^{n_0} \rightarrow \mathbb{F}_2^{n_1} \rightarrow \cdots \rightarrow \mathbb{F}_2^{n_{r_m}}(n_0 = n + m)$ with $\boldsymbol{x}, \boldsymbol{k}$ as input and $\boldsymbol{s}^{(i+1)}$ as output of the i-th round $(0 < i \leq r_m - 1)$, where $\boldsymbol{x} \in \mathbb{F}_2^n$ and $\boldsymbol{k} \in \mathbb{F}_2^m$ denote the public and secret variables, respectively. Let the cube term be $\boldsymbol{x}^{\boldsymbol{u}}$. The output term of round r_m is represented as $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$. Correspondingly, Condition B should be expressed as $\exists \boldsymbol{k}^{\boldsymbol{w}}$ s.t. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$.

Flag Technique. Similar to [28], we propose a flag technique to classify bits. In our work, we treat $\boldsymbol{k}$ as non-zero constants and set $\boldsymbol{x}[\bar{\boldsymbol{u}}]$ to 0, then each bit involved in the round function of $\boldsymbol{f}$ can be represented as an ANF of $\boldsymbol{k}$ and $\boldsymbol{x}[\boldsymbol{u}]$. For each involved bit b, we assign it an additional flag $b.F \in \{1_c, 0_c, \delta\}$. 0_c means b is constant 0; δ means the ANF of b involves $\boldsymbol{x}[\boldsymbol{u}]$, i.e., it contains a monomial associated with at least one cube variable; 1_c means b is non-zero and its ANF doesn't involve $\boldsymbol{x}[\boldsymbol{u}]$. Since the ANF will become intractable as the number of rounds increases, these flags are precomputed according to COPY, XOR and

AND without considering the effect of cancellation characteristics. Note that the computation of our flags does not require the help of MILP models, and the flags in [28] can be handled in the same way, although the authors of [28] encoded the flags into their MILP models. We define $=$, $\oplus$ and $\times$ operations for the elements of set $\{1_c, 0_c, \delta\}$ corresponding to the basic operations COPY, XOR and AND, respectively. The $=$ operation sets one element equal to another element. The $\oplus$ operation follows the rules:

$$\begin{cases} 1_c \oplus 1_c = 1_c \ , \\ 0_c \oplus x = x \oplus 0_c = x \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \ , \\ \delta \oplus x = x \oplus \delta = \delta \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \ . \end{cases}$$

The $\times$ operation follows the rules:

$$\begin{cases} 1_c \times x = x \times 1_c = x \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \ , \\ 0_c \times x = x \times 0_c = 0_c \text{ for arbitrary } x \in \{1_c, 0_c, \delta\} \ , \\ \delta \times \delta = \delta \ . \end{cases}$$

Bits flagged by x ($x \in \{1_c, 0_c, \delta\}$) are referred to as x bits in this paper. For a bit vector $\boldsymbol{t}^{(j)}$, suppose the state bits in the j-th round are denoted by $\boldsymbol{s}^{(j)}$, we use $\boldsymbol{\Lambda}^{1_c}(\boldsymbol{t}^{(j)})$, $\boldsymbol{\Lambda}^{0_c}(\boldsymbol{t}^{(j)})$, $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(j)})$ to divide $\boldsymbol{t}^{(j)}$ into three bit vectors according to the $1_c, 0_c, \delta$ part of $\boldsymbol{s}^{(j)}$ respectively, i.e.,

$$\Lambda_i^x(\boldsymbol{t}^{(j)}) = t_i^{(j)} \ \forall \ s_i^{(j)}.F = x, \text{ otherwise } \Lambda_i^x(\boldsymbol{t}^{(j)}) = 0$$

for arbitrary $x \in \{1_c, 0_c, \delta\}$. When introducing MILP models, for a MILP variable $v \in \mathcal{M}.var$ assigned to the bit b, we may use $v.F$ to represent $b.F$ implicitly. Once the cube term is given, the flags of all state bits of $\boldsymbol{f}$ are determined. A specific example of our flag computation can be found in Example 1.

5.1 Non-zero Bit-Based Division Property (NBDP)

First, we revisit the roles of CBDP in recovering the superpoly from a perspective of the monomial propagation.

Proposition 1. *Given a term $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ of round r_m. Assuming the initial CBDP $\mathcal{D}^{1^{m+n}}_{\{\boldsymbol{0}^m||\boldsymbol{u}\}}$ propagates to $\mathcal{D}^{1^{n_{r_m}}}_{\mathbb{K}^{(r_m)}}$ after evaluating $\boldsymbol{f}$ through r_m rounds, if $\exists \boldsymbol{k}^{(r_m)} \in \mathbb{K}^{(r_m)}$ such that $\boldsymbol{k}^{(r_m)} \preceq \boldsymbol{t}^{(r_m)}$, then there must $\exists \boldsymbol{w} \succeq \boldsymbol{0}^m, \boldsymbol{u'} \succeq \boldsymbol{u}$ s.t. $\boldsymbol{k^w x^{u'}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$. The converse is also true.*

Proof. Let the input and output multisets of CBDP be $\mathbb{X}^{(0)}$ and $\mathbb{X}^{(r)}$ respectively. According to the definition of CBDP, we assume $\sum_{\boldsymbol{k}||\boldsymbol{x} \in \mathbb{X}^{(0)}} \boldsymbol{k^w x^{u'}}$ is unknown for any $\boldsymbol{w}||\boldsymbol{u'} \succeq \boldsymbol{0}^m||\boldsymbol{u}$ and $\sum_{\boldsymbol{k}||\boldsymbol{x} \in \mathbb{X}^{(0)}} \boldsymbol{k^w x^{u'}} = 0$ for any $\boldsymbol{w}||\boldsymbol{u'} \not\succeq \boldsymbol{0}^m||\boldsymbol{u}$.

If $\exists \boldsymbol{w} \succeq \boldsymbol{0}, \boldsymbol{u'} \succeq \boldsymbol{u}$ s.t. $\boldsymbol{k^w x^{u'}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, then $\sum_{\boldsymbol{s}^{(r_m)} \in \mathbb{X}^{(r_m)}} \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ must be unknown, which means $\exists \boldsymbol{k}^{(r_m)} \in \mathbb{K}^{(r_m)}$ such that

$\boldsymbol{k}^{(r_m)} \preceq \boldsymbol{t}^{(r_m)}$. Conversely, if $\exists \boldsymbol{k}^{(r_m)} \in \mathbb{K}^{(r_m)}$ such that $\boldsymbol{k}^{(r_m)} \preceq \boldsymbol{t}^{(r_m)}$, then $\sum_{\boldsymbol{s}^{(r_m)} \in \mathbb{X}^{(r_m)}} \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ must be unknown. We can deduce there exists $\boldsymbol{w} \succeq \boldsymbol{0}^m, \boldsymbol{u}' \succeq \boldsymbol{u}$ s.t. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}'} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, because otherwise $\sum_{\boldsymbol{s}^{(r_m)} \in \mathbb{X}^{(r_m)}} \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ would be 0 rather than unknown. □

Based on Proposition 1, a natural idea to describe Condition B is to exclude all division trails related to $\boldsymbol{x}[\bar{\boldsymbol{u}}]$. Since $\boldsymbol{x}[\bar{\boldsymbol{u}}]$ is set to constant 0, we only need to handle constant 0 bits during the propagation of CBDP. A lot of work has been conducted on this research line ([25,28]). In our work, to deal with constant 0 bits, we follow three rules that are described in [28] for Copy, And and Xor, but with our flag technique. These three rules are slightly adjusted and listed in [13, Sup.Mat. C], together with some additional constraints that can be added to remove redundancy.

In addition, to describe Condition B, the partial order in CBDP should also consider the effect of constant 0 bits, so we modify the partial order in CBDP.

Definition 3 (The Partial Order). *Let $\boldsymbol{v}'$ and $\boldsymbol{v}$ be two bit vectors. We say $\boldsymbol{v}' \hat{\succeq} \boldsymbol{v}$ on $\boldsymbol{y}$ or simply $\boldsymbol{y}^{\boldsymbol{v}'} \hat{\succeq} \boldsymbol{y}^{\boldsymbol{v}}$, if*

$$\begin{cases} v'_i = v_i = 0 & y_i.F = 0_c \\ v'_i \geq v_i & y_i.F \neq 0_c \end{cases}.$$

We denote this variant of CBDP as *non-zero bit-based division property* (NBDP). Using NBDP, if $\exists \boldsymbol{k}^{\boldsymbol{w}}$ such that $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, then there must exist $\boldsymbol{k}^{(r_m)}$ propagated from $\boldsymbol{0}^m||\boldsymbol{u}$ such that $\boldsymbol{k}^{(r_m)} \hat{\succeq} \boldsymbol{t}^{(r_m)}$ on $\boldsymbol{s}^{(r_m)}$. Hence, we can construct a MILP model to recover $VT^{(r_m)}$s as follows:

$$\overbrace{\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \xrightarrow{\textbf{NBDP}} \pi_{\boldsymbol{k}^{(r_m)}}(\boldsymbol{s}^{(r_m)})}^{r_m \ rounds} \hat{\succeq} \overbrace{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \xrightarrow{\textbf{MP}} \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})}^{r-r_m \ rounds}, \tag{5}$$

where NBDP propagates from $\boldsymbol{0}^m||\boldsymbol{u}$ to $\boldsymbol{k}^{(r_m)}$ in the first r_m rounds. Such a MILP model is described as `NBDP-MPModelX` in [13, Algorithm 4]. After extracting all solutions of this MILP model, for each $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ we can count the number of monomial trails between $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ and $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ to determine if $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, then determine if this term $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ is a $VT^{(r_m)}$. In this way, a new coefficient solver can be developed by first recovering $VT^{(r_m)}$s and then applying the monomial prediction to each $VT^{(r_m)}$, as stated in Sect. 4. We did test such a new coefficient solver for 846-round Trivium by setting $r_m = 90$, combined with the term expander used in NMP. As a result, the superpoly of the cube term I_3 in [13, Table 5] is recovered in about two days on our platform. However, apart from this result, no other superpolies were recovered.

The Bottleneck of Recovering $VT^{(r_m)}$s Based on Eq. (5). The number of solutions of `NBDP-MPModelX` can be expressed as

$$\sum_{\substack{\boldsymbol{k}^{(r_m)} \in \mathbb{F}_2^{nr_m}, \boldsymbol{t}^{(r_m)} \in \mathbb{F}_2^{nr_m} \\ \boldsymbol{k}^{(r_m)} \hat{\succeq} \boldsymbol{t}^{(r_m)} \text{ on } \boldsymbol{s}^{(r_m)}}} |\boldsymbol{0}^m||\boldsymbol{u} \overset{\mathbb{K}}{\bowtie} \boldsymbol{k}^{(r_m)}| \cdot |\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \bowtie \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})|,$$

where $\boldsymbol{k}^{(r_m)}$ and $\boldsymbol{t}^{(r_m)}$ take all allowed values. Note that for a specific $\boldsymbol{t}^{(r_m)}$ satisfying $|\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \ltimes \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})| > 0$ with a high hamming weight, the sum of all $|\mathbf{0}^m||\boldsymbol{u} \overset{\mathbb{K}}{\ltimes} \boldsymbol{k}^{(r_m)}|$ satisfying $\boldsymbol{k}^{(r_m)} \hat{\preceq} \boldsymbol{t}^{(r_m)}$ on $\boldsymbol{s}^{(r_m)}$ may be extraordinarily large, which makes it hard to extract all solutions of `NBDP-MPModelX` within limited time.

5.2 Core Monomial Prediction (CMP)

To overcome the bottleneck of recovering $VT^{(r_m)}$s based on Eq. (5), we next propose an alternative approach to characterize Condition B from the perspective of monomial propagation. This new technique is called *Core Monomial Prediction (CMP)*, which can be regarded as a relaxed version of monomial prediction.

Generalization of Condition B. Notice that given a non-zero term $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, what determines whether Condition B holds is $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})$. Moreover, denoting the initial term $\boldsymbol{k}^{\boldsymbol{w}}||\boldsymbol{x}^{\boldsymbol{u}}$ in Condition B by $\pi_{\boldsymbol{t}^{(0)}}(\boldsymbol{s}^{(0)})$, notice that $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\mathbf{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ and $\pi_{\boldsymbol{\Lambda}^{1_c}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\boldsymbol{w}}||\boldsymbol{x}^{\mathbf{0}}$. Let $\boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0})$ indicate the 1_c bits of the initial state (round 0), that is, $\Lambda_i^{1_c}(\mathbf{1}^{n_0}) = 1$ if $s_i^{(0)}.F = 1_c$, otherwise $\Lambda_i^{1_c}(\mathbf{1}^{n_0}) = 0$. Obviously $\boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0}) = \mathbf{1}^m||\mathbf{0}^n$. Considering in Condition B we only require the existence of $\boldsymbol{x}^{\boldsymbol{u}}$, $\boldsymbol{x}^{\boldsymbol{u}}$ can be any vector satisfying $\boldsymbol{x}^{\boldsymbol{u}} \preceq \boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0})$. Therefore, we can give a generalization of Condition B by

$$\exists \boldsymbol{x}^{\boldsymbol{u}} \preceq \boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0}), \text{ such that } \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})\oplus \boldsymbol{x}^{\boldsymbol{u}}}(\boldsymbol{s}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)}). \tag{6}$$

Naturally, we study how to describe

$$\exists \boldsymbol{x}^{\boldsymbol{u}} \preceq \boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_i}), \text{ such that } \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(i)})\oplus \boldsymbol{x}^{\boldsymbol{u}}}(\boldsymbol{s}^{(i)}) \rightsquigarrow \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(j)})}(\boldsymbol{s}^{(j)})$$

for arbitrary $i < j$. Note that in the process of generalizing Condition B, we aggressively assume that $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \neq 0$, whereas in practice, it is entirely possible that $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ equals to 0. Recalling that NBDP is derived by considering the effect of constant 0 bits in the propagation and partial order of CBDP, whose propagation rules are established first, it is reasonable that we first study the case where constant 0 bits are not taken into account.

The Definition and Propagation of CMP. Let $\boldsymbol{g} : \mathbb{F}_2^{n_{\text{in}}} \to \mathbb{F}_2^{n_{\text{out}}}$ be a vectorial Boolean function mapping $\boldsymbol{z} = (z_0, \cdots, z_{n_{\text{in}}-1})$ to $\boldsymbol{y} = (y_0, \cdots, y_{n_{\text{out}}-1})$ with $y_i = g_i(\boldsymbol{z})$. In [17], the monomial prediction is defined as the problem of determining the presence or absence of a particular monomial $\boldsymbol{z}^{\boldsymbol{u}}$ in $\boldsymbol{y}^{\boldsymbol{v}}$, that is, whether $\boldsymbol{z}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$. Similarly, the *core monomial prediction* is defined as the problem of determining whether $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v})}(\boldsymbol{y})$ contains at least one monomial, say $\boldsymbol{z}^{\boldsymbol{u}}$, whose δ part $(\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{u})}(\boldsymbol{z}))$ is a particular monomial. We denote this problem by whether $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{u})}(\boldsymbol{z}) \overset{\text{Core}}{\to} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v})}(\boldsymbol{y})$. Similar to the monomial trail, the definition of CMP gives rise to the concept of the *core monomial trail*.

Definition 4 (Core Monomial Trail). *Given the cube term $\boldsymbol{x}^{\boldsymbol{u}}$, let $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\mathbf{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ and $\boldsymbol{s}^{(i+1)} = \boldsymbol{f}^{(i)}(\boldsymbol{s}^{(i)})$ for $0 \leq i < r$. We call a sequence*

of monomials $(\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}), \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(1)})}(\boldsymbol{s}^{(1)}), \ldots, \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)}))$ *an* r*-round core monomial trail connecting* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)})$ *and* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$ *with respect to the composite function* $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(i-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ *if*

$$\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{Core}{\rightarrow} \cdots \overset{Core}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(i)})}(\boldsymbol{s}^{(i)}) \overset{Core}{\rightarrow} \cdots \overset{Core}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)}),$$

If there is at least one core monomial trail connecting $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)})$ *to* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$*, we write* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{Core}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$*. All core monomial trails between* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)})$ *and* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$ *are denoted by the set* $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{Core}{\bowtie} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$.

The monomial prediction determines whether $\pi_{\boldsymbol{t}^{(0)}}(\boldsymbol{s}^{(0)}) \rightarrow \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ by counting the number of monomial trails between $\pi_{\boldsymbol{t}^{(0)}}(\boldsymbol{s}^{(0)})$ and $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$. However, the number of core monomial trails between $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)})$ and $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$ can not reflect precisely whether $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{\text{Core}}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$, i.e., whether there exists a $\boldsymbol{w} \preceq \boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0})$ such that $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})\oplus\boldsymbol{w}}(\boldsymbol{s}^{(0)}) \rightarrow \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$. Since the information of 1_c bits is ignored by the core monomial trail, we can only draw a weaker conclusion from the existence of a core monomial trail, that is, $\exists \boldsymbol{w} \preceq \boldsymbol{\Lambda}^{1_c}(\mathbf{1}^{n_0})$ such that $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})\oplus\boldsymbol{w}}(\boldsymbol{s}^{(0)}) \rightsquigarrow \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$. Notice that this is exactly the generalization of Condition B, which means the existence of a core monomial trail between $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)})$ and $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$ is another equivalent description of Condition B.

To better understand how core monomial trails are generated, we give a concrete example.

Example 1. Let $\boldsymbol{z} = (z_0, z_1) = \boldsymbol{f}^{(1)}(y_0, y_1) = (y_0y_1, y_0 + y_1 + 1)$, $\boldsymbol{y} = (y_0, y_1) = \boldsymbol{f}^{(0)}(x_0, x_1, x_2, k_0, k_1) = (k_0x_0 + k_1x_0x_2 + k_0 + k_1, k_0k_1x_1 + k_0k_1x_0 + k_0)$. Consider the cube term $(x_0, x_1, x_2)^{(1,1,0)} = x_0x_1$. First, we can compute the flags of $\boldsymbol{x}, \boldsymbol{k}, \boldsymbol{y}, \boldsymbol{z}$, i.e.,

$$\begin{aligned}
&x_0.F = x_1.F = \delta, x_2.F = 0_c.\ k_0.F = k_1.F = 1_c.\\
&y_0.F = 1_c \times \delta \oplus 1_c \times \delta \times 0_c \oplus 1_c \oplus 1_c = \delta.\\
&y_1.F = 1_c \times 1_c \times \delta \oplus 1_c \times 1_c \times \delta \oplus 1_c = \delta.\\
&z_0.F = \delta \times \delta = \delta.\ z_1.F = \delta \oplus \delta \oplus 1_c = \delta.
\end{aligned}$$

Since the ANF of $\boldsymbol{f}^{(0)}$ is available, we can compute all monomials of $\boldsymbol{y}$ (x_2 is set to 0), i.e.,

$$\begin{aligned}
&(y_0, y_1)^{(0,0)} = 1, (y_0, y_1)^{(1,0)} = y_0 = k_0x_0 + k_0 + k_1.\\
&(y_0, y_1)^{(0,1)} = y_1 = k_0k_1x_1 + k_0k_1x_0 + k_0.\\
&(y_0, y_1)^{(1,1)} = y_0y_1 = k_0k_1\underline{x_0x_1} + k_0k_1x_0 + k_0x_0 + k_0 + k_0k_1.
\end{aligned}$$

Considering $(y_0, y_1)^{\boldsymbol{\Lambda}^\delta((1,1))} = y_0y_1$, then $x_0x_1 \overset{\text{Core}}{\rightarrow} y_0y_1$ is the only core monomial trail of $\boldsymbol{f}^{(0)}$ connecting x_0x_1 and the δ part of monomials of $\boldsymbol{y}$. Similarly, we can compute all monomials of $\boldsymbol{z}$ as follows,

$$(z_0, z_1)^{(0,0)} = 1, (z_0, z_1)^{(1,0)} = z_0 = \underline{y_0y_1}, (z_0, z_1)^{(0,1)} = z_1 = y_0 + y_1 + 1,$$
$$(z_0, z_1)^{(1,1)} = z_0z_1 = \underline{y_0y_1}.$$

Since $z_0.F = z_1.F = \delta$, we have $(z_0, z_1)^{\boldsymbol{\Lambda}^\delta((1,0))} = z_0$ and $(z_0, z_1)^{\boldsymbol{\Lambda}^\delta((1,1))} = z_0z_1$. Finally, we obtain two core monomial trails of $\boldsymbol{f}$ connecting x_0x_1 and the δ part of monomials of $\boldsymbol{z}$:

$$x_0x_1 \overset{\text{Core}}{\rightarrow} y_0y_1 \overset{\text{Core}}{\rightarrow} z_0, \quad x_0x_1 \overset{\text{Core}}{\rightarrow} y_0y_1 \overset{\text{Core}}{\rightarrow} z_0z_1.$$

Recalling in [17], the propagation rules of 3SDPwoU are revisited from the algebraic perspective according to the definition of the monomial prediction. In a similar way, the propagation rules of the core monomial prediction can be derived from its definition. And we only give the rule of COPY an algebraic proof, as the others can be interpreted in a same way. As mentioned, we do not take constant 0 bits into account, so we assume the bits of $\boldsymbol{z}$ and $\boldsymbol{y}$ below are all non-zero, i.e., their flags are not 0_c.

Rule 1 (COPY). *Let* $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$ *and* $\boldsymbol{y} = (z_0, z_0, z_1, z_2, \ldots, z_{n-1})$ *be the input and output vector of a* COPY *function. Let* $\boldsymbol{\Lambda}^\delta(\boldsymbol{u}) = (u'_0, \ldots, u'_{n-1})$ *and* $\boldsymbol{\Lambda}^\delta(\boldsymbol{v}) = (v'_0, \ldots, v'_n)$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{u})}(\boldsymbol{z}) \overset{Core}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ *only when* $\boldsymbol{\Lambda}^\delta(\boldsymbol{v})$ *satisfies*

$$\boldsymbol{\Lambda}^\delta(\boldsymbol{v}) = \begin{cases} (0, 0, \ldots, u'_{n-1}), & \text{if } u'_0 = 0 \ , \\ (0, 1, \ldots, u'_{n-1}), (1, 0, \ldots, u'_{n-1}), (1, 1, \ldots, u'_{n-1}), & \text{if } u'_0 = 1 \ . \end{cases}$$

Proof. Let $\boldsymbol{z}.F = (z_0.F, \ldots, z_{n-1}.F)$ and $\boldsymbol{y}.F = (y_0.F, \ldots, y_n.F)$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ can be expressed as $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y}) = z_0^{v'_0 \vee v'_1} z_1^{v'_2} \cdots z_{n-1}^{v'_n}$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{u})}(\boldsymbol{z}) \overset{\text{Core}}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ only when $\boldsymbol{\Lambda}^\delta((v'_0 \vee v'_1, v'_2, \ldots, v'_n)) = (u'_0, u'_1, \ldots, u'_{n-1})$, where $\boldsymbol{\Lambda}^\delta((v'_0 \vee v'_1, v'_2, \ldots, v'_n))$ depends on $\boldsymbol{z}.F$.

Notice that $\boldsymbol{\Lambda}^\delta((v'_0, \ldots, v'_n)) = (v'_0, \ldots, v'_n)$ according to $\boldsymbol{y}.F$ and $y_i.F = z_{i-1}.F$ for $2 \leq i \leq n$, therefore $v'_i = u'_{i-1}$ for $2 \leq i \leq n$. Next we consider $z_0.F$. If $z_0.F = y_0.F = y_1.F = 1_c$, then $v'_0 = v'_1 = 0$ and $u'_0 = 0$; otherwise if $z_0.F = y_0.F = y_1.F = \delta$, we can deduce that $v'_0 \vee v'_1 = u'_0$. To sum up, the propagation rule of COPY can be concluded as $v'_0 \vee v'_1 = u'_0$ and $v'_i = u'_{i-1}$ for $2 \leq i \leq n$. □

Rule 2 (AND). *Let* $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$ *and* $\boldsymbol{y} = (z_0 \wedge z_1, z_2, \ldots, z_{n-1})$ *be the input and output vector of an* AND *function. Let* $\boldsymbol{\Lambda}^\delta(\boldsymbol{u}) = (u'_0, \ldots, u'_{n-1})$ *and* $\boldsymbol{\Lambda}^\delta(\boldsymbol{v}) = (v'_0, \ldots, v'_{n-2})$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{u})}(\boldsymbol{z}) \overset{Core}{\rightarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ *only when* $\boldsymbol{\Lambda}^\delta(\boldsymbol{u}), \boldsymbol{\Lambda}^\delta(\boldsymbol{v})$ *satisfies*

$$\begin{aligned} (v'_0, v'_1, \ldots, v'_{n-2}) &= (u'_0 \vee u'_1, u'_2, \ldots, u'_{n-1}) \ , \\ v'_0 &= u'_0, \text{ if } z_0.F = \delta \ , \\ v'_0 &= u'_1, \text{ if } z_1.F = \delta \ . \end{aligned}$$

Rule 3 (XOR). *Let* $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$ *and* $\boldsymbol{y} = (z_0 \oplus z_1, z_2, \ldots, z_{n-1})$ *be the input and output vector of a* XOR *function. Let* $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{u}) = (u'_0, \ldots, u'_{n-1})$ *and* $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v}) = (v'_0, \ldots, v'_{n-2})$. $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{u})}(\boldsymbol{z}) \overset{Core}{\rightarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v})}(\boldsymbol{y})$ *only when* $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v})$ *satisfies*

$$\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v}) = \begin{cases} (v'_0, u'_2, \ldots, u'_{n-1}) \text{ with } v'_0 \geq u'_0 + u'_1, & \text{if } \{z_0.F, z_1.F\} = \{1_c, \delta\} \ , \\ (u'_0 + u'_1, u'_2, \ldots, u'_{n-1}), & \text{otherwise} \ . \end{cases}$$

The MILP models corresponding to propagation rules can be easily derived, as shown in [13, Sup.Mat. D]. Next we consider the effect of constant 0 bits. In the propagation of CMP, we treat constant 0 bits in the same way as NBDP. Namely, we follow the rules listed in [13, Sup.Mat. C]. Recall in the generalization of Condition B, we assume $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ is non-zero. However, $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{\text{Core}}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$ cannot guarantee $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \neq 0$. Therefore, like the proposal of the new partial order in NBDP, we propose a new partial order to impose stricter constraints on $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$.

Definition 5 (New Partial Order of CMP). *Let* $\boldsymbol{v}'$ *and* $\boldsymbol{v}$ *be two bit vectors. We say* $\boldsymbol{v}' \widetilde{\succeq} \boldsymbol{v}$ *on* $\boldsymbol{y}$ *or simply* $\boldsymbol{y}^{\boldsymbol{v}'} \widetilde{\succeq} \boldsymbol{y}^{\boldsymbol{v}}$, *if*

$$\begin{cases} v'_i = v_i = 0 & y_i.F = 0_c \ , \\ v'_i \geq v_i & y_i.F = 1_c \ , \\ v'_i = v_i & y_i.F = \delta \ . \end{cases}$$

Then, the generalization of Condition B (6) holds if and only if $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{\text{Core}}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$ and $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \widetilde{\succeq} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$.

Recovering $VT^{(r_m)}$s with CMP. Based on the discussion above, we can construct a MILP model using CMP to recover $VT^{(r_m)}$s as follows:

$$\overbrace{\boldsymbol{k}^{\mathbf{0}}\boldsymbol{x}^{\boldsymbol{u}} \xrightarrow{\mathbf{CMP}} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})}^{r_m \ rounds} \widetilde{\succeq} \overbrace{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \xrightarrow{\mathbf{MP}} \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})}^{r-r_m \ rounds}. \tag{7}$$

A MILP model based on the structure in Eq. (7) is described as `CMP-MPModelX` in [13, Algorithm 5].

CMP versus MP. We can prove that the MILP model based on Eq. (7) has fewer solutions than the MILP model based on Eq. (4). The number of solutions of the MILP model based on Eq. (7) can be represented by

$$\sum_{\boldsymbol{t}^{(r_m)} \in \mathbb{F}_2^{n r_m}} |\boldsymbol{k}^{\mathbf{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\bowtie} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})| \cdot |\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \bowtie \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})|.$$

The number of solutions of the MILP model based on Eq. (4) can be represented by

$$\sum_{\boldsymbol{t}^{(r_m)} \in \mathbb{F}_2^{n r_m}, \boldsymbol{w} \in \mathbb{F}_2^{m}} |\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \bowtie \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| \cdot |\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \bowtie \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})|.$$

Notice that we can always map a r_m-round monomial trail $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \to \pi_{\boldsymbol{t}^{(1)}}(\boldsymbol{s}^{(1)}) \to \cdots \to \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ to a r_m-round core monomial trail $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\to} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(1)})}(\boldsymbol{s}^{(1)}) \overset{\text{Core}}{\to} \cdots \overset{\text{Core}}{\to} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$. Furthermore, notice that whether Condition B holds can be determined by checking whether $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$, which means this mapping is surjective. As a result, for a specific non-zero monomial $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, we have

$$\sum_{\boldsymbol{w}\in\mathbb{F}_2^m} |\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \bowtie \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| \geq |\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\bowtie} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})|,$$

meaning the model based on Eq. (7) has fewer solutions.

In addition, recall that when modeling Condition B, the propagation model of CMP only describes the δ part of monomials, and the 1_c or 0_c bits are constrained to 0, while the propagation model of MP needs to consider not only the δ part, but it also need to track the propagation of the 1_c bits. The difference between CMP and MP can be seen most intuitively from their algebraic interpretation, which was described earlier in this paper.

Naturally, we believe the MILP model based on Eq. (7) can be solved faster than the model based on Eq. (4). Indeed, taking the MILP model based on Eq. (7) as the core component of our coefficient solver, we successfully recovered the superpoly of 848-round Trivium, a result that can not be achieved by the model based on Eq. (4).

The MILP models based on Eq. (7) can be further optimized using [13, Proposition 2], where the XOR propagation rule (Rule 3) of CMP is reduced to

$$\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v}) = (u'_0 + u'_1, u'_2, \ldots, u'_{n-1}).$$

Moreover, when constructing the MILP models based on Eq. (7) and Eq. (5) for Acorn, this proposition helps to exclude some redundant propagation trails, thus simplifying the models.

Towards New Coefficient Solver and Term Expander. Finally, we choose the MILP model based on Eq. (7) to recover $VT^{(r_m)}$s in a more efficient way than the monomial prediction. Notice that if $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ is a $VT^{(r_m)}$, we can split $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ into 1_c part and δ part, namely

$$\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) = \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)}) \cdot \pi_{\boldsymbol{\Lambda}^{1_c}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)}).$$

$\pi_{\boldsymbol{\Lambda}^{1_c}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$ is the product of some 1_c bits whose ANFs can be computed beforehand, hence the monomial prediction technique is only applied to δ part of $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$ to compute $\mathsf{Coe}\left(\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$. Such a strategy can speed up the coefficient solver for some ciphers.

Combined with the callback function interface provided in Gurobi, MILP models based on Eq. (7) and Eq. (5) can also be extended to construct a new term expander. However, we prefer this to be an implementation improvement rather than a theoretical innovation. In other words, even if we use the term

expander in NMP with our new coefficient solver, we can still get the results listed in this paper, but it might take slightly more time. For this reason, we put the introduction of our term expander in [13, Sup.Mat. E].

6 Applications

Using our designed term expander and coefficient solver, we can assemble a new nested framework according to Algorithm 1. We apply this new nested framework to four NLFSR-based ciphers, namely Trivium, Grain-128AEAD, Kreyvium and Acorn. As a result, the exact ANFs of the superpolies for 846-, 847- and 848-round Trivium, 192-round Grain-128AEAD, 895-round Kreyvium and 776-round Acorn are recovered. All experiments are performed using Gurobi Solver (version 9.1.2) on a work station with high-speed processors, (totally 32 cores and 64 threads). The source code (as well as some superpolies we recovered) is available in our git repository.

In [11,12,16], the MILP models of Trivium, Grain-128AEAD, Kreyvium, and Acorn for tracing the three-subset division/monomial trails are proposed. In this section, the propagation models of monomial trails in [13, Sup.Mat. G] are directly borrowed from [11,12,16] and we adjust them slightly to fit our new framework. The MILP models of basic operations that NBDP and CMP rely on are provided in [13, Sup.Mat. F]. As pointed out before, `NBDP-MPModelX` in [13, Algorithm 4] and `CMP-MPModelX` in [13, Algorithm 5] are the most important parts in the framework, so next we only describe how to construct these two MILP models for a specific cipher, along with the selection of related parameters.

6.1 Superpoly Recovery for Trivium up to 848 Rounds

As shown in [13, Sup.Mat. H], we applied our framework to Trivium and verified the correctness of some previous superpolies with significantly less time cost.

Superpoly Recovery for 846-, 847- and 848-Round Trivium. To the best of our knowledge, currently there is no effective method for choosing a good cube, hence we heuristically choose cubes with similar structure to I_0–I_4. Finally, we find some other cubes applicable to Trivium up to 848 rounds. They are listed in Table 3. Since the sizes of these superpolies are too large, we only provide our codes in the git repository. The details of these superpolies are given in Table 4. The balancedness of each superpoly is estimated by testing 2^{15} random keys.

6.2 Superpoly Recovery for 192-Round Grain-128AEAD

In [13, Sup.Mat. I], we introduce the specification of Grain-128AEAD and apply our new framework to it. We successfully verified the results given in [16]. For 192-round Grain-128AEAD, we heuristically choose a 94-dimensional cube indexed by $\{0, 1, 2, \ldots, 95\} \backslash \{42, 43\}$. The superpoly of this cube for 192-round Grain-128AEAD is recovered in about 45 days using our new framework. The

Table 3. Cube indices for the superpoly recovery of Trivium up to 848 rounds

I	$\|I\|$	Indices
I_5	53	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_6	52	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_7	51	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_8	53	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_9	53	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 30, 31, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79

Table 4. Details related to the superpolies of 846-, 847- and 848-round Trivium.

I	Round	Status	#Involved key bits	Balancedness	TimeCost
I_3	846	New	80	0.50	1 day and 12 h
I_5	846	New	80	0.50	16 h
I_6	846	New	80	0.50	5 h
I_7	846	New	80	0.50	11 h
I_8	846	New	80	0.50	5 h
I_9	846	New	80	0.50	1 day and 17.5 h
I_6	847	New	80	0.50	9 days and 20.5 h
I_7	847	New	80	0.50	2 days
I_6	848	New	80	0.50	11 days

superpoly is a 34-degree polynomial involving 534 077 971 terms and 128 key bits. The balancedness is estimated to be 0.49 after testing 2^{15} random keys. Since the size of this superpoly is too large, we only provide our codes in the git repository.

6.3 Superpoly Recovery for 895-Round Kreyvium

As can be seen in [13, Sup.Mat. J], we verified the superpoly of the 119-dimensional cube given in [16] with our new framework. For 895-round Kreyvium, we heuristically choose a 120-dimensional cube indexed by

$$I_2 = \{0, 1, \ldots, 127\} \backslash \{66, 72, 73, 78, 101, 106, 109, 110\}.$$

The superpoly of I_2 for 895-Round Kreyvium is recovered in about two weeks using our nested framework. The superpoly is a 7-degree polynomial that involves

19411 terms and 128 key bits. The balancedness is estimated to be 0.50 after testing 2^{15} random keys.

6.4 Superpoly Recovery for ACORN up to 776 Rounds

As can be seen in [13, Sup.Mat. K], we verified the results given in [33] with our new framework. For 776-round ACORN, we heuristically choose two 127-dimensional cubes indexed by $I_1 = \{0, 1, \ldots, 127\} \backslash \{1, 28\}$ and $I_2 = \{0, 1, \ldots, 127\} \backslash \{2, 28\}$. The superpoly recoveries of these two cubes are completed after about 8 days using our nested framework.

The superpoly of I_1 is an 8-degree polynomial involving 123 key bits and 2 464 007 terms, with k_{104}, k_{105} and k_{115} as single balanced bits. Bits not involved are $k_{100}, k_{103}, k_{106}, k_{114}$ and k_{126}. The superpoly of I_2 is an 8-degree polynomial involving 121 key bits and 2 521 399 terms, with k_{104} and k_{126} as single balanced bits. Bits not involved are $k_{99}, k_{100}, k_{101}, k_{103}, k_{106}, k_{110}$ and k_{112}. The concrete expressions of these two superpolies, denoted by p_{I_1} and p_{I_2}, are shown in [13, Sup.Mat. N], where they are represented by 1_c bits of 256th round.

7 Towards Efficient Key-Recovery Attacks

Though we have recovered more than one superpolies for 846- and 847-round TRIVIUM, how to recover the information of key bits from multiple superpolies remains a problem. We briefly discuss several previous approaches to this problem in [13, Sup.Mat. L].

Cube Attacks Against 776-Round Acorn. Since the two superpolies of 776-round ACORN do not involve full key bits, we can mount a key recovery attack against 776-round ACORN as follows:

1. We can obtain the real values of the two superpolies during the online phase, which requires 2^{127} ACORN calls.
2. We guess the values of $\{k_0, ..., k_{127}\} \backslash \{k_{100}, k_{103}, k_{106}\}$ and check if the values of the two superpolies are correct. As mentioned in [13, Sup.Mat. K], one evaluation of the superpoly of ACORN is equivalent to one 256-round ACORN call or approximately $\frac{1}{3}$ 776-round ACORN call, so the complexity of this step is about $2 \times 2^{125} \times \frac{1}{3} \approx 2^{124.4}$ 776-round ACORN calls.
3. For the remaining 2^{126} candidates of key bits, we can find the correct key by an exhaustive search with time complexity of 2^{126} 776-round ACORN calls.

Therefore, the final complexity is slightly more than 2^{127} 776-round ACORN calls to recover all the secret key bits. Next we show how to mount the cube attack using the superpoly involving full key bits.

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

$$f = \bigoplus_{(c_0,\ldots,c_{n-1}) \in \mathbb{F}_2^n} g_0(c_0, \ldots, c_{n-1}) \cdot \prod_{i=0}^{n-1} x_i^{c_i}. \tag{8}$$

The process of Möbius transformation on f from ANF to truth table can be represented by Algorithm 2, where t represents the t-th step and g_n is exactly f. For simplicity, we also use $g_t(e)(0 \leq t \leq n)$ to represent $g_t(c_0, \ldots, c_{n-1})$, where $e = c_0 + c_1 2^1 + \cdots + c_{n-1}2^{n-1}$ and $(c_0, \ldots, c_{n-1})$ is called the binary representation of e. We assume in this paper that Möbius transformation requires $n \times 2^{n-1}$ bitwise XORs and 2^n-bits memory complexity.

Algorithm 2: Möbius transformation on f in Eq. (8)

```
Procedure MobiusTransformation(The ANF of f):
    for t = 1 to n do
        Initialize g_t to be the same as g_{t-1}
        for j = 0 to 2^{n-t} - 1 do
            for k = 0 to 2^{t-1} - 1 do
                g_t(2^t j + 2^{t-1} + k) = g_{t-1}(2^t j + 2^{t-1} + k) + g_{t-1}(2^t j + k)
    return g_n
```

Proposition 3. *Let $f, g_0, g_1, \ldots, g_n$ be defined as above. After the t-th $(1 \leq t \leq n-1)$ step of Möbius transformation on the ANF of f, if we represent f as*

$$f = \sum_{(c_t,\ldots,c_{n-1}) \in \mathbb{F}_{n-t}} p_{(c_t,\ldots,c_{n-1})}(x_0, \ldots, x_{t-1}) \cdot \prod_{i=t}^{n-1} x_i^{c_i}, \tag{9}$$

where $p_{(c_t,\ldots,c_{n-1})}(x_0, \ldots, x_{t-1})$ is a Boolean polynomial of $(x_0, \ldots, x_{t-1})$ determined by $(c_t, \ldots, c_{n-1})$, then for any value of $(c_t, \ldots, c_{n-1})$,

$$g_t(x_0, \ldots, x_{t-1}, c_t, \ldots, c_{n-1}) = p_{(c_t,\ldots,c_{n-1})}(x_0, \ldots, x_{t-1}).$$

An intuitive description of Eq. (9) is, we regard $(x_t, \ldots, x_{n-1})$ as variables and $(x_0, \ldots, x_{t-1})$ as constants, then $p_{(c_t,\ldots,c_{n-1})}(x_0, \ldots, x_{t-1})$ is exactly the coefficient of the monomial $\prod_{i=t}^{n-1} x_i^{c_i}$.

Proof. It can be easily verified that when $t = 1$, the conclusion holds. Assume the conclusion holds for $t = l$ $(1 \leq l \leq n-2)$, next we prove that the conclusion is also true for $t = l + 1$.

According to Eq. (9), $p_{(c_{l+1},\ldots,c_{n-1})}(x_0, \ldots, x_l)$ can be expressed as the sum of $p_{(0,c_{l+1},\ldots,c_{n-1})}(x_0, \ldots, x_{l-1})$ and $p_{(1,c_{l+1},\ldots,c_{n-1})}(x_0, \ldots, x_{l-1}) \cdot x_l$, that is,

$$g_l(x_0, \ldots, x_{l-1}, 0, c_{l+1}, \ldots, c_{n-1}) + g_l(x_0, \ldots, x_{l-1}, 1, c_{l+1}, \ldots, c_{n-1}) \cdot x_l.$$

Considering that x_l takes 0 or 1, $p_{(c_{l+1},\ldots,c_{n-1})}(x_0, \ldots, x_l)$ is equal to

$$\begin{cases} g_l(x_0, \ldots, x_{l-1}, 0, c_{l+1}, \ldots, c_{n-1}), & \text{if } x_l = 0, \\ g_l(x_0, \ldots, x_{l-1}, 0, c_{l+1}, \ldots, c_{n-1}) + g_l(x_0, \ldots, x_{l-1}, 1, c_{l+1}, \ldots, c_{n-1}), & \text{if } x_l = 1. \end{cases}$$

This is the same as how $g_{l+1}(x_0, \ldots, x_l, c_{l+1}, \ldots, c_{n-1})$ is generated during the process of Möbius transformation. Hence, $g_{l+1}(x_0, \ldots, x_l, c_{l+1}, \ldots, c_{n-1})$ is exactly $p_{(c_{l+1},\ldots,c_{n-1})}(x_0, \ldots, x_l)$. By mathematical induction, the conclusion is true for all t $(1 \leq t \leq n-1)$. □

Example 2. Let $f(x_0, x_1, x_2, x_3) = x_0x_1x_2 + x_2x_3 + x_1x_3 + x_2$. The process of Möbius transformation on the ANF of f is shown in the following table, where each row is the truth table of g_t $(0 \leq t \leq 4)$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
g_0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	0	0
g_1	0	0	0	0	1	1	0	1	0	0	1	1	1	1	0	0
g_2	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1
g_3	0	0	0	0	1	1	1	0	0	0	1	1	1	1	0	0
g_4	0	0	0	0	1	1	1	0	0	0	1	1	0	0	1	0

Consider g_2. We regard (x_0, x_1) as constants and (x_2, x_3) as variables, f can be expressed as $f = (x_0x_1+1)\cdot x_2 + x_1 \cdot x_3 + 1 \cdot x_2x_3$. Then $g_2(x_0, x_1, 0, 0) = 0$, which is the coefficient of $x_2^0x_3^0$ in f; $g_2(x_0, x_1, 1, 0) = x_0x_1 + 1$, which is the coefficient of $x_2^1x_3^0$; $g_2(x_0, x_1, 0, 1) = x_1$, which is the coefficient of $x_2^0x_3^1$; $g_2(x_0, x_1, 1, 1) = 1$, which is the coefficient of $x_2^1x_3^1$. This corresponds to truth table of g_2. Note that $g_2(j) = g_2(x_0, x_1, x_2, x_3)$, where $j = x_0 + x_1 2^1 + x_2 2^2 + x_3 2^3$.

We can further simplify Möbius transformation exploiting Proposition 3 and the degree of f. Since this would not affect our final complexity, we discuss it in [13, Sup.Mat. M]

Key Recovery During Möbius Transformation. Let f be as defined in Eq. (8) and $(x_0, \ldots, x_{n-1})$ be n secret key variables. It can be deduced from Proposition 3 that if $(c_t, \ldots, c_{n-1}) = (0, \ldots, 0)$, $f(x_0, \ldots, x_{t-1}, 0, \ldots, 0)$ is equal to $g_t(x_0, \ldots, x_{t-1}, 0, \ldots, 0)$, therefore after t steps of Möbius transformation, we can already obtain the function values of $f(x_0, \ldots, x_{t-1}, 0, \ldots, 0)$. Using this property, we can recover the key during Möbius transformation. We use Example 3 to illustrate our basic idea.

Example 3. Let f and its Möbius transformation be as defined in Example 2. Now we want to recover the key from the equation $f(x_0, x_1, x_2, x_3) = a$.
1. At the beginning, $f(0) = g_0(0)$. If $f(0) = a$, we test whether $(0, 0, 0, 0)$ is the correct key by one encryption call. And if it is incorrect, we go to next step.
2. Compute $g_1(1) = g_0(0) + g_0(1), g_1(0) = g_0(0)$, then $f(1) = g_1(1)$. If $f(1) = a$, we test whether $(1, 0, 0, 0)$ is the correct key by one encryption call. And if it is incorrect, go to next step.
3. First, we compute $g_1(3) = g_0(2) + g_0(3), g_1(2) = g_0(2)$. Compute $g_2(2) = g_1(2) + g_1(0), g_2(3) = g_1(3) + g_1(1), g_2(0) = g_1(0), g_2(1) = g_1(1)$, then $f(2) = g_2(2), f(3) = g_2(3)$. If $f(2) = a$, we test if $(0, 1, 0, 0)$ is the correct key by one

encryption call; if $f(3) = a$, we test if $(1, 1, 0, 0)$ is the correct key by one encryption call. And if none of them is correct, go to next step.
4. First, we compute $g_1(i)$ $(i = 4, 5, 6, 7)$ from $g_0(i)$ $(i = 4, 5, 6, 7)$ and $g_2(i)$ $(i = 4, 5, 6, 7)$ from $g_1(i)$ $(i = 4, 5, 6, 7)$. Compute $g_3(j)$ $(j = 0, \ldots, 7)$ from $g_2(j)$ $(j = 0, \ldots, 7)$, then $f(i) = g_3(i)$ $(i = 4, 5, 6, 7)$. If $f(i) = a$ $(i = 4, 5, 6, 7)$, we test if the binary representation of i is the correct key by one encryption call. And if none of them is correct, go to next step.
5. First, we compute $g_1(i)$ $(i = 8, \ldots, 15)$ from $g_0(i)$ $(i = 8, \ldots, 15)$, $g_2(i)$ $(i = 8, \ldots, 15)$ from $g_1(i)$ $(i = 8, \ldots, 15)$ and $g_3(i)$ $(i = 8, \ldots, 15)$ from $g_2(i)$ $(i = 8, \ldots, 15)$. Compute $g_4(j)$ $(j = 0, \ldots, 15)$ from $g_3(j)$ $(j = 0, \ldots, 15)$, then $f(i) = g_4(i)$ $(i = 8, \ldots, 15)$. If $f(i) = a$ $(i = 8, \ldots, 15)$, we test if the binary representation of i is the correct key by one encryption call. And if none of them is correct, we claim this equation has no solution.

In each step, we use the minimum memory and the least XOR operations to calculate the necessary bits. In step 1, only 1-bit memory $(g_0(0))$ is sufficient. In step 2, 1 XOR and 2-bits memory $(g_1(1), g_1(0))$ are sufficient. In step 3, $2 + 1$ XORs and 4-bits memory $(g_2(i)$ $(i = 1, 2, 3, 4))$ are sufficient. In step 4, $2 + 2 + 4$ XORs and 8-bits memory $(g_3(j)$ $(j = 0, \ldots, 7))$ are sufficient. Finally in step 5, $4 + 4 + 4 + 8$ XORs and 16-bits memory $(g_4(j)$ $(j = 0, \ldots, 15))$ are sufficient. Note that in each step, the first computation can be regarded as performing Möbius transformation on part of the ANF of f. For example, in step 4, we first compute $g_2(i)$ $(i = 4, 5, 6, 7)$ iteratively from $g_0(i)$ $(i = 4, 5, 6, 7)$. This can be regarded as performing Möbius transformation on the ANF of $p_{(1,0)}(x_0, x_1)$, namely $x_0x_1 + 1$.

We summarize our key recovery strategy in Algorithm 3, which can be seen as embedding the key testing procedure into a Möbius transformation with an adjusted computational order. Though a more accurate estimation of the average complexity can be given, here we roughly estimate the time cost (only considering the loop starting at Line 8) of Algorithm 3 as one Möbius transformation together with required encryption calls, that is, $q \cdot 2^n$ encryption calls $+ n \cdot 2^{n-1}$ XORs, where q denotes the probability that $f(x_0, \ldots, x_{n-1}) = a$. The cost of the comparison is ignored. The memory complexity in the worst case is 2^n bits. Comparing with the traditional key recovery method of first constructing a large truth table and then performing queries, our method naturally saves the query cost.

Key Recovery Attacks on 848-Round Trivium. Our further evaluation shows that the superpoly of 848-round Trivium, denoted by $p(k_0, ..., k_{79})$, is a polynomial whose degree is upper bounded by 25. It contains about $2^{30.5}$ terms, but is still very sparse compared with a random polynomial (a random polynomial may contain about 2^{79} terms). A natural idea is to treat $p(k_0, ..., k_{79})$ as f in Algorithm 3. However, Möbius transformation also incurs memory access cost. In the worse case, Algorithm 3 requires 2^{80}-bits memory and the memory access cost of such a big table is unbearable. To address this difficulty, we propose the following strategies:

Algorithm 3: Recover the key from the equation $f(x_0, \ldots, x_{n-1}) = a$

```
Procedure RecoverKey(The ANF of f):
    for t = 1 to n do
        Precompute the ANF of p_{(1,0,...,0)}(x_0, ..., x_{t-2}) from the ANF of f
    if g_0(0, ..., 0) = a then
        Check if (0, ..., 0) is the correct key by calling the encryption oracle once
        if The check passes then
            return (0, ..., 0)
    for t = 1 to n do
        Perform Möbius transformation on the ANF of p_{(1,0,...,0)}(x_0, ..., x_{t-2}) to obtain the
          truth table of g_{t-1}(x_0, ..., x_{t-2}, 1, 0, ..., 0)
        /* When t = 1, we assume p_{(c_{t-1},...,c_{n-1})}(x_0, ..., x_{t-2}) = g_0(c_{t-1}, ..., c_{n-1})  */
        for k = 0 to 2^{t-1} - 1 do
            g_t(2^{t-1} + k) = g_{t-1}(2^{t-1} + k) + g_{t-1}(k)
            if g_t(2^{t-1} + k) = a then
                Let (c_0, ..., c_{n-1}) be the binary representation of 2^{t-1} + k
                Check if (c_0, ..., c_{n-1}) is the correct key by calling the encryption oracle once
                if The check passes then
                    return (c_0, ..., c_{n-1})
            g_t(k) = g_{t-1}(k)
    return no solution found
```

1. We guess the values of $(k_{40}, \ldots, k_{79})$. Let $(v_{40}, \ldots, v_{79})$ denote the values of $(k_{40}, \ldots, k_{79})$, then the equation $p(k_0, \ldots, k_{79}) = a$ can be reduced to $p'(k_0, ..., k_{39}) = p(k_0, \ldots, k_{39}, v_{40}, \ldots, v_{79}) = a$.
2. For each guess, treat $p'(k_0, ..., k_{39})$ as f and apply Algorithm 3 to it. Once Algorithm 3 returns the correct values of $(k_0, ..., k_{39})$, denoted by $(v_0, \ldots, v_{39})$, the correct key is found as $(v_0, \ldots, v_{79})$.

Assuming reducing $p(k_0, \ldots, k_{79})$ to $p'(k_0, ..., k_{39})$ for all guesses requires $2^{40} \times 2^{30.5} = 2^{70.5}$ XORs, the final time complexity is approximately 2^{79} Trivium calls and 40×2^{79} XORs. Assuming one 848-round Trivium call is equivalent to $848 \times 9 = 7632$ XORs, finally our key recovery strategy requires slightly more than 2^{79} 848-round Trivium calls, but only about 2^{40}-bits memory. Similarly, we can recover the key of 192-round Grain-128AEAD and 895-round Kreyvium with time complexity 2^{127} and 2^{127}, respectively.

8 Conclusion

In this paper, we revisit the two core components of nested monomial predictions, namely the coefficient solver and the term expander. The coefficient solver is responsible for performing the superpoly recovery for a given term, while the term expander is used to transform output bits into multiple terms of fewer rounds. We try to improve the coefficient solver by first recovering valuable intermediate terms of a middle round, then applying the monomial prediction to each of them. This idea gives rise to two techniques called NBDP and core

monomial prediction that identify the necessary condition that a valuable intermediate term should satisfy. The core monomial prediction presents a substantial improvement over monomial prediction in terms of efficiency of enumerating solutions, hence we choose it to build our coefficient solver. Besides, we construct an improved term expander using NBDP in order to spend less time on useless terms of fewer rounds. We apply our new framework to Trivium, Grain-128AEAD, Acorn and Kreyvium and recover superpolies for reduced-round versions of the four ciphers with 848, 192, 776 and 895 rounds. This results in attacks that are more efficient and cover more rounds than earlier work.

Acknowledgment.. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. The research leading to these results has received funding from the National Natural Science Foundation of China (Grant No. 62002201, Grant No. 62032014), the National Key Research and Development Program of China (Grant No. 2018YFA0704702), and the Major Basic Research Project of Natural Science Foundation of Shandong Province, China (Grant No. ZR202010220025). Bart Preneel was supported by CyberSecurity Research Flanders with reference number VR20192203. Kai Hu is supported by the "ANR-NRF project SELECT". The scientific calculations in this paper have been done on the HPC Cloud Platform of Shandong University.

References

1. Gorubi Optimization. https://www.gurobi.com
2. Gorubi Optimization Reference Manual. https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.1/refman.pdf
3. ISO/IEC 29192-3:2012: Information technology—Security techniques—Lightweight cryptography—part 3: Stream ciphers. https://www.iso.org/standard/56426.html
4. Boura, C., Coggia, D.: Efficient MILP modelings for sboxes and linear layers of SPN ciphers. IACR Trans. Symmetric Cryptol. **2020**(3), 327–361 (2020)
5. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_18
6. Canteaut, A., et al.: Stream ciphers: a practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018)
7. Chang, D., Turan, M.S.: Recovering the key from the internal state of Grain-128AEAD. IACR Cryptology ePrint Archive 2021:439 (2021)
8. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_16
9. Fouque, P.-A., Vannet, T.: Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 502–517. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_26
10. Hao, Y., Jiao, L., Li, C., Meier, W., Todo, Y., Wang, Q.: Links between division property and other cube attack variants. IACR Trans. Symmetric Cryptol. **2020**(1), 363–395 (2020)

11. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and grain-128AEAD. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 466–495. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_17
12. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. J. Cryptol. **34**(3), 22 (2021)
13. He, J., Hu, K., Preneel, B., Wang, M.: Stretching cube attacks: improved methods to recover massive superpolies. Cryptology ePrint Archive, Paper 2022/1218 (2022). https://eprint.iacr.org/2022/1218
14. Hebborn, P., Lambin, B., Leander, G., Todo, Y.: Lower bounds on the degree of block ciphers. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 537–566. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_18
15. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: Grain-128AEAD - a lightweight AEAD stream cipher. NIST Lightweight Cryptography, Round, 3 (2019)
16. Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 392–421. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_14
17. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: revisiting degree evaluations, cube attacks, and key-independent sums. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 446–476. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_15
18. Lehmann, M., Meier, W.: Conditional Differential Cryptanalysis of Grain-128a. In: Pieprzyk, J., Sadeghi, A.-R., Manulis, M. (eds.) CANS 2012. LNCS, vol. 7712, pp. 1–11. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35404-5_1
19. Meicheng Liu. Degree evaluation of NFSR-based cryptosystems. In Jonathan Katz and Hovav Shacham, editors, CRYPTO 2017, volume 10403 of LNCS, pages 227–249. Springer, 2017
20. Mroczkowski, P., Szmidt, J.: The cube attack on stream cipher Trivium and quadraticity tests. Fundam. Informaticae **114**(3–4), 309–318 (2012)
21. Sasaki, Yu., Todo, Y.: New algorithm for modeling S-box in MILP based differential and division trail search. In: Farshim, P., Simion, E. (eds.) SecITC 2017. LNCS, vol. 10543, pp. 150–165. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69284-5_11
22. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
23. Sun, Y.: Automatic search of cubes for attacking stream ciphers. IACR Trans. Symmetric Cryptol. **2021**(4), 100–123 (2021)
24. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
25. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 250–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_9

26. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. IACR Cryptology ePrint Archive 2017:306 (2017)
27. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
28. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 275–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_10
29. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: MILP-aided method of searching division property using three subsets and applications. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 398–427. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_14
30. Wang, S.P., Bin, H., Guan, J., Zhang, K., Shi, T.: A practical method to recover exact superpoly in cube attack. IACR Cryptology ePrint Archive 2019:259 (2019)
31. Wu, H.: Acorn v3. Submission to CAESAR competition (2016)
32. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
33. Yang, J., Lin, D.: Searching cubes in division property based cube attack: applications to round-reduced acorn. Cryptology ePrint Archive, Report 2020/1128 (2020). https://ia.cr/2020/1128
34. Yang, J., Liu, M., Lin, D.: Cube cryptanalysis of round-reduced acorn. Cryptology ePrint Archive, Report 2019/1226 (2019). https://ia.cr/2019/1226
35. Ye, C., Tian, T.: A new framework for finding nonlinear superpolies in cube attacks against Trivium-like ciphers. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 172–187. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_11
36. Ye, C.-D., Tian, T.: Algebraic method to recover superpolies in cube attacks. IET Inf. Secur. **14**(4), 430–441 (2020)
37. Ye, C.-D., Tian, T.: A practical key-recovery attack on 805-round trivium. IACR Cryptology ePrint Archive 2020:1404 (2020)
38. Ye, C., Tian, T.: Revisit division property based cube attacks: key-recovery or distinguishing attacks? IACR Trans. Symmetric Cryptol. **2019**(3), 81–102 (2019)