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. 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 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 [9], 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 [9,20,10,36] 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 [29], 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. [30] 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 [31], 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 [29] are proved to degenerate to constants. In [12,13], 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 [37], 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 [6], Grain-128AEAD [15], Kreyvium [7] and the authenticated encryption algorithm Acorn [32].

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.
- *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

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 hours
I_1	843	Verified(✓)		4 hours
I_2	843	Verified(✓)		1 hour
I_3	843	Verified(✓)		1.5 hours
I_4	843	Verified(✓)		1 day and 17 hours
I_2	844	Verified(✓)	17 hours	5 hours
I_3	844	Verified(✓)	6 hours	2.5 hours
I_2	845	Verified(✓)	about 16 days	19.5 hours
I_3	845	Verified(✓)	4 days and 9 hours	8.5 hours

†: 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.

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 [6,3]), Grain-128AEAD (a member of the ten finalist candidates of the NIST LWC standardization process [15]), Kreyvium (designed for Fully Homomorphic Encryption [7]) and ACORN (a member of the final portfolio of the CAESAR competition for Lightweight applications [32]). 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 anonymous 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	[9,20,10,38,23]
	808	37	39-41	Practical	key recovery	[23]
	≤ 844	-	-	$2^{75} \sim 2^{79.6}$	key recovery	[23,16,17,12,13,19] [26,39,30,38,36,10]
	845	2	54-55	2^{78}	key recovery	[16]
	846	6	51-54	2^{79}	key recovery	Section 6.1
	847	2	52-53	2^{79}	key recovery	Section 6.1
	848	1	52	2^{79}	key recovery	Section 6.1
Grain‡	169	-	-	Practical	Condit. Diff.	[18]
	-	-	-	Practical	State recovery	[8]
	≤ 190	-	-	$2^{123} \sim 2^{129}$	key recovery	[25,26,29,12,13]
	191	2	95-96	2^{127}	key recovery	[16]
	192	1	94	2^{127}	key recovery	Section 6.2
Kreyvium	≤ 893	-	-	$2^{119} \sim 2^{127}$	key recovery	[26,29,12,13,11,23]
	894	1	119	2^{127}	key recovery	[16]
	895	1	120	2^{127}	key recovery	Section 6.3
Acorn	≤ 774	-	-	2^{127}	key recovery	[26,12,11,29]
	775	6	127	2^{127}	distinguisher	[35]
	775	5	126	2^{126}	distinguisher	[34]
	775	1	126	2^{127}	key recovery	[34]
	776	2	126	2^{127}	key recovery	Section 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 $0 \leq i < n$. 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 $i_j \in \{0 \leq i \leq n-1 : u_i = 1\}$ 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 $u_i \geq u'_i$ for all i. Similarly, we define $\boldsymbol{u} \preceq \boldsymbol{u}'$ if $u_i \leq u'_i$ 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 [33], 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 [33]). *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 Sup.Mat. B.

In our work, we use $\boldsymbol{k}^{(0)} \overset{\mathbb{K}_{\boldsymbol{f}}}{\rightsquigarrow} \boldsymbol{k}^{(r)}$ 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 $\boldsymbol{k}^{(0)} \overset{\mathbb{K}_{\boldsymbol{f}}}{\bowtie} \boldsymbol{k}^{(r)}$, whose size is denoted by $|\boldsymbol{k}^{(0)} \overset{\mathbb{K}_{\boldsymbol{f}}}{\bowtie} \boldsymbol{k}^{(r)}|$. 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 [12,13,14,17]). *Let* $\boldsymbol{f} = \boldsymbol{f}^{(r-1)} \circ \boldsymbol{f}^{(r-2)} \circ \cdots \circ \boldsymbol{f}^{(0)}$ *defined as above.* $\pi_{\boldsymbol{u}^{(0)}}(\boldsymbol{x}^{(0)}) \to \pi_{\boldsymbol{u}^{(r)}}(\boldsymbol{x}^{(r)})$ *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 [22,21,5], which are suitable for MILP modeling. The concrete propagation rules and models of the monomial prediction are provided in 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 [12,13,17], 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 [12,13].

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 $f : \mathbb{F}_2^{n+m} \to \mathbb{F}_2$ 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 $I = \{0 \le i \le n-1 : u_i = 1\} \subseteq \{0, 1, \ldots, n-1\}$, 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 [17,12,13] 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

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

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**
  - $r_n=$ `ChooseRiX`$(\mathbb{S}_u^{(r_l)}, r_l, ...)$
  - $\mathbb{S}^{(r_n)}=$ `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**
    - $\tau=$ `ChooseTiX`(r_n)
    - $(status, p^{(r_n)})=$ `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 $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$, 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 `TermExpanderX`$(\mathbb{S}_u^{(r_l)}, r_l, r_n, ...)$ and `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 $\mathsf{Coe}\left(\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ 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 $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \to f$, 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 $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, e.g.,

$$|\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \bowtie \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})| \equiv \sum_{\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})} |\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \bowtie \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})| \pmod 2. \tag{2}$$

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

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

Terms satisfying (A) $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, (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 $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ (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. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \to \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$. 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 $\boldsymbol{f} : \mathbb{F}_2^{n_0} \to \mathbb{F}_2^{n_1} \to \cdots \to \mathbb{F}_2^{n_{r_m}} (n_0 = n + m)$ 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 [29], 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 at least one monomial associated with cube bits; 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 [29] can be handled in the same way, although the authors of [29] 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 $\mathcal{D}^{1^{nr_m}}_{\mathbb{K}^{(r_m)}}$ 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. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}'} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$. 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 $\sum_{\boldsymbol{k}||\boldsymbol{x}\in\mathbb{X}^{(0)}} \boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}'}$ is unknown for any $\boldsymbol{w}||\boldsymbol{u}' \succeq \boldsymbol{0}^m||\boldsymbol{u}$ and $\sum_{\boldsymbol{k}||\boldsymbol{x}\in\mathbb{X}^{(0)}} \boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}'} = 0$ 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. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}'} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)})$, 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 ([25,29]). In our work, to deal with constant 0 bits, we follow three rules that are described in [29] for Copy, And and Xor, but with our flag technique. These three rules are slightly adjusted and listed in 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:

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

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 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 $\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, 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 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 Eqn. (5). The number of solutions of `NBDP-MPModelX` can be expressed as

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

where $\boldsymbol{k}^{(r_m)}$ and $\boldsymbol{t}^{(r_m)}$ take all allowed values. Note that for a specific $\boldsymbol{t}^{(r_m)}$ satisfying $|\pi_{\boldsymbol{t}^{(r_m)}}(\boldsymbol{s}^{(r_m)}) \bowtie \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})| > 0$ with a high hamming weight, the sum of all $|\boldsymbol{0}^m||\boldsymbol{u} \overset{\mathbb{K}}{\boxtimes} \boldsymbol{k}^{(r_m)}|$ 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 Eqn. (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 $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\boldsymbol{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ and $\pi_{\boldsymbol{\Lambda}^{1_c}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\boldsymbol{w}}||\boldsymbol{x}^{\boldsymbol{0}}$. Let $\boldsymbol{\Lambda}^{1_c}(\boldsymbol{1}^{n_0})$ indicate the 1_c bits of the initial state (round 0), that is, $\Lambda_i^{1_c}(\boldsymbol{1}^{n_0}) = 1$ if $s_i^{(0)}.F = 1_c$, otherwise $\Lambda_i^{1_c}(\boldsymbol{1}^{n_0}) = 0$. Obviously $\boldsymbol{\Lambda}^{1_c}(\boldsymbol{1}^{n_0}) = \boldsymbol{1}^m||\boldsymbol{0}^n$. Considering in Condition B we only require the existence of $\boldsymbol{w}$, $\boldsymbol{w}$ can be any vector satisfying $\boldsymbol{w} \preceq \boldsymbol{\Lambda}^{1_c}(\boldsymbol{1}^{n_0})$. Therefore, we can give a generalization of Condition B by

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

Naturally, we study how to describe

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

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 $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) = \boldsymbol{k}^{\boldsymbol{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ 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*

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

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 $\pi_{\boldsymbol{t}^{(0)}}(\boldsymbol{s}^{(0)}) \to \pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$ 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 $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})}(\boldsymbol{s}^{(0)}) \overset{\text{Core}}{\to} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$, i.e., whether there exists a $\boldsymbol{w} \preceq \boldsymbol{\Lambda}^{1_c}(\boldsymbol{1}^{n_0})$ such that $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(0)})\oplus\boldsymbol{w}}(\boldsymbol{s}^{(0)}) \to \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r)})}(\boldsymbol{s}^{(r)})$. 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, $\exists \boldsymbol{w} \preceq \boldsymbol{\Lambda}^{1_c}(\boldsymbol{1}^{n_0})$ 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.,

$$(y_0, y_1)^{(0,0)} = 1, (y_0, y_1)^{(1,0)} = y_0 = k_0x_0 + k_0 + k_1.$$

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

Considering $(y_0, y_1)^{\boldsymbol{\Lambda}^\delta((1,1))} = y_0y_1$, then $x_0x_1 \overset{\text{Core}}{\to} y_0y_1$ 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 $\boldsymbol{z}$:

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

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* $\boldsymbol{\Lambda}^\delta(\boldsymbol{v}) = (v'_0, \ldots, v'_n)$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{u})}(\boldsymbol{z}) \overset{Core}{\to} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ *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 $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y}) = z_0^{v'_0 \vee v'_1} z_1^{v'_2} \cdots z_{n-1}^{v'_n}$. $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{u})}(\boldsymbol{z}) \overset{\text{Core}}{\to} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ 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 $2 \le i \le n$, therefore $v'_i = u'_{i-1}$ for $2 \le i \le n$. 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 $2 \le i \le n$. □

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*

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

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*

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

The MILP models corresponding to propagation rules can be easily derived, as shown in 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 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:

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

A MILP model based on the structure in Eqn. (7) is described as `CMP-MPModelX` in Algorithm 5.

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

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

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

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

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 Eqn. (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 Eqn. (7) can be solved faster than the model based on Eqn. (4). Indeed, taking the MILP model based on Eqn. (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 Eqn. (4).

Proposition 2 (A property of CMP). *Let $g = \pi_{\boldsymbol{t}_0^{(i)}}(\boldsymbol{s}^{(i)}) \oplus \pi_{\boldsymbol{t}_1^{(i)}}(\boldsymbol{s}^{(i)})$ with $\pi_{\boldsymbol{t}_0^{(i)}}(\boldsymbol{s}^{(i)})$ and $\pi_{\boldsymbol{t}_1^{(i)}}(\boldsymbol{s}^{(i)})$ being two non-zero monomials. If $\boldsymbol{\Lambda}^\delta(\boldsymbol{t}_0^{(i)}) \succeq \boldsymbol{\Lambda}^\delta(\boldsymbol{t}_1^{(i)})$ and $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{Core}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}_1^{(i)})}(\boldsymbol{s}^i) \overset{Core}{\to} g$, then $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{Core}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}_0^{(i)})}(\boldsymbol{s}^i) \overset{Core}{\to} g$. In other words, to evaluate whether $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{Core}{\rightsquigarrow} g$, it is sufficient to evaluate whether $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{Core}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}_0^{(i)})}(\boldsymbol{s}^i) \overset{Core}{\to} g$ and the role of $\pi_{\boldsymbol{t}_1^{(i)}}(\boldsymbol{s}^{(i)})$ can be ignored.*

Let $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{w})}(\boldsymbol{z})$ and $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ be two intermediate monomials involved in a core monomial trail between $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}}$ and $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$, with $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$ and $\boldsymbol{y} = (z_0 \oplus z_1, z_2, \ldots, z_{n-1})$ being the input and output of a XOR function. $\boldsymbol{y}^{\boldsymbol{v}}$ contains two monomials of $\boldsymbol{z}$, namely $z_0^{v_0}z_2^{v_1}\cdots z_{n-1}^{v_{n-2}}$ and $z_1^{v_0}z_2^{v_1}\cdots z_{n-1}^{v_{n-2}}$. We denote these two monomials by $\boldsymbol{z}^{\boldsymbol{v}_0}$ and $\boldsymbol{z}^{\boldsymbol{v}_1}$. Consider $\{z_0.F, z_1.F\} = \{1_c, \delta\}$. Suppose $z_0.F = \delta$ and $z_1.F = 1_c$, then $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v}_0)}(\boldsymbol{z}) \succeq \pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v}_1)}(\boldsymbol{z})$. According to Proposition 2, ignoring $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v}_1)}(\boldsymbol{z})$ won't affect whether $\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}}$ can propagate to $\pi_{\boldsymbol{\Lambda}^\delta(\boldsymbol{v})}(\boldsymbol{y})$ by CMP. In the MILP model based on Eqn. (7), we are only concerned

about whether $\boldsymbol{k}^{\mathbf{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$, therefore we can ignore $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{v}_1)}(\boldsymbol{z})$. Using the notion of Rule 3, the propagation rule of XOR can be reduced to

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

Considering the existence of a division trail of NBDP can also be used to describe Condition B, whose equivalent description is $\boldsymbol{k}^{\mathbf{0}}\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\rightsquigarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_m)})}(\boldsymbol{s}^{(r_m)})$, therefore Proposition 2 is also applicable during the propagation of NBDP. We will show how to simplify the models of NBDP and CMP using Proposition 2 when discussing the application to Acorn in Sect. 6.4.

Towards new coefficient solver and term expander. Finally, we choose the MILP model based on Eqn. (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 Eqn. (7) and Eqn. (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 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 [12,13,16], 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 Sup.Mat. G are directly borrowed from [12,13,16] 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 Sup.Mat. F. As pointed out before, `NBDP-MPModelX` in Algorithm 4 and `CMP-MPModelX` in 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 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.

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

6.2 Superpoly Recovery for 192-Round Grain-128AEAD

In 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 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.

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 hours
I_5	846	New	80	0.50	16 hours
I_6	846	New	80	0.50	5 hours
I_7	846	New	80	0.50	11 hours
I_8	846	New	80	0.50	5 hours
I_9	846	New	80	0.50	1 day and 17.5 hours
I_6	847	New	80	0.50	9 days and 20.5 hours
I_7	847	New	80	0.50	2 days
I_6	848	New	80	0.50	11 days

6.3 Superpoly Recovery for 895-Round Kreyvium

As can be seen in 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 Sup.Mat. K, we verified the results given in [34] 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 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 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 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 $g_t(e)(0 \le t \le n)$ 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 Eqn. (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 Eqn. (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 Eqn. (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 Sup.Mat. M.

Key recovery during Möbius transformation. Let f be as defined in Eqn. (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.

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

```
1  Procedure RecoverKey(The ANF of f):
2      for t = 1 to n do
3          Precompute the ANF of p_(1,0,...,0)(x_0, ..., x_{t-2}) from the ANF of f
4      if g_0(0, ..., 0) = a then
5          Check if (0, ..., 0) is the correct key by calling the encryption oracle once
6          if The check passes then
7              return (0, ..., 0)
8      for t = 1 to n do
9          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})   */
10         for k = 0 to 2^{t-1} - 1 do
11             g_t(2^{t-1} + k) = g_{t-1}(2^{t-1} + k) + g_{t-1}(k)
12             if g_t(2^{t-1} + k) = a then
13                 Let (c_0, ..., c_{n-1}) be the binary representation of 2^{t-1} + k
14                 Check if (c_0, ..., c_{n-1}) is the correct key by calling the encryption oracle once
15                 if The check passes then
16                     return (c_0, ..., c_{n-1})
17             g_t(k) = g_{t-1}(k)
18     return no solution found
```

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:

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. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *Int. J. Wirel. Mob. Comput.*, 5(1):48–59, 2011.
5. Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.
6. Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 244–266. Springer, 2008.
7. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *J. Cryptol.*, 31(3):885–916, 2018.
8. Donghoon Chang and Meltem Sönmez Turan. Recovering the key from the internal state of Grain-128AEAD. *IACR Cryptol. ePrint Arch.*, 2021:439, 2021.
9. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.
10. Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 502–517. Springer, 2013.
11. Yonglin Hao, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. Links between division property and other cube attack variants. *IACR Trans. Symmetric Cryptol.*, 2020(1):363–395, 2020.
12. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 466–495. Springer, 2020.
13. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset. *J. Cryptol.*, 34(3):22, 2021.
14. Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Lower bounds on the degree of block ciphers. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 537–566. Springer, 2020.
15. Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEAD - A lightweight AEAD stream cipher. *NIST Lightweight Cryptography, Round*, 3, 2019.
16. Kai Hu, Siwei Sun, Yosuke Todo, Meiqin Wang, and Qingju Wang. Massive superpoly recovery with nested monomial predictions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 392–421, Cham, 2021. Springer International Publishing.

17. Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020.
18. Michael Lehmann and Willi Meier. Conditional differential cryptanalysis of Grain-128a. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *CANS 2012*, volume 7712, pages 1–11. Springer, 2012.
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. Piotr Mroczkowski and Janusz Szmidt. The cube attack on stream cipher Trivium and quadraticity tests. *Fundam. Informaticae*, 114(3-4):309–318, 2012.
21. Yu Sasaki and Yosuke Todo. New algorithm for modeling s-box in MILP based differential and division trail search. In Pooya Farshim and Emil Simion, editors, *SecITC 2017*, volume 10543 of *LNCS*, pages 150–165. Springer, 2017.
22. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014.
23. Yao Sun. Automatic search of cubes for attacking stream ciphers. *IACR Transactions on Symmetric Cryptology*, 2021(4):100–123, Dec. 2021.
24. Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.
25. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10403 of *LNCS*, pages 250–279. Springer, 2017.
26. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. *IACR Cryptol. ePrint Arch.*, 2017:306, 2017.
27. Yosuke Todo and Masakatu Morii. Bit-based division property and application to Simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.
28. Jianhua Wang, Baofeng Wu, and Zhuojun Liu. Improved degree evaluation and superpoly recovery methods with application to trivium. *CoRR*, abs/2201.06394, 2022.
29. Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10991 of *LNCS*, pages 275–305. Springer, 2018.
30. SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. MILP-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019*, volume 11923 of *LNCS*, pages 398–427. Springer, 2019.
31. SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. A practical method to recover exact superpoly in cube attack. *IACR Cryptology ePrint Archive*, 2019:259, 2019.
32. Hongjun Wu. Acorn v3. *Submission to CAESAR competition*, 2016.

33. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 648–678. Springer, 2016.
34. Jingchun Yang and Dongdai Lin. 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.
35. Jingchun Yang, Meicheng Liu, and Dongdai Lin. Cube cryptanalysis of round-reduced acorn. Cryptology ePrint Archive, Report 2019/1226, 2019. https://ia.cr/2019/1226.
36. Chen-Dong Ye and Tian Tian. A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In Willy Susilo and Guomin Yang, editors, *ACISP 2018*, volume 10946 of *LNCS*, pages 172–187. Springer, 2018.
37. Chen-Dong Ye and Tian Tian. Algebraic method to recover superpolies in cube attacks. *IET Inf. Secur.*, 14(4):430–441, 2020.
38. Chen-Dong Ye and Tian Tian. A practical key-recovery attack on 805-round trivium. *IACR Cryptol. ePrint Arch.*, 2020:1404, 2020.
39. Chendong Ye and Tian Tian. Revisit division property based cube attacks: Key-recovery or distinguishing attacks? *IACR Trans. Symmetric Cryptol.*, 2019(3):81–102, 2019.

Supplementary Material

A

A.1 MILP Models Based on the structure of Eqns. (5) and (7)

Algorithm 4: The MILP model based on the structure of Eqn. (5)

Procedure `NBDP-MPModelX`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m**):**

- Declare an empty MILP model $\mathcal{M}$
- Declare $\mathtt{k}^0$ as $n+m$ MILP variables of $\mathcal{M}$ corresponding to public and secret variables
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 1 \ \forall \ u_i = 1$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 0 \ \forall \ u_i = 0$
- $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i+n}}^0 = 0 \ \forall \ i \in \{0, 1, \ldots, m-1\}$
- Update $\mathtt{k}^0$ according to the propagation rules of NBDP through r_m rounds to $\mathtt{k}^{\mathtt{r_m}}$
- Declare $\mathtt{s}^{\mathtt{r_m}}$ as n_{r_m} MILP variables of $\mathcal{M}$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = 0 \ \forall \ s_i^{(r_m)}.F = 0_c$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \ \forall \ s_i^{(r_m)}.F \neq 0_c$
- Update $\mathtt{s}^{\mathtt{r_m}}$ according to the propagation rules of the monomial prediction through $r - r_m$ rounds to $\mathtt{s}^{\mathtt{r}}$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r}} = 0 \ \forall \ t_i^{(r)} = 0$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r}} = 1 \ \forall \ t_i^{(r)} = 1$
- **return** $\mathcal{M}$

Algorithm 5: The MILP model based on the structure of Eqn. (7)

Procedure `CMP-MPModelX`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m**):**

- Declare an empty MILP model $\mathcal{M}$
- Declare $\mathtt{k}^0$ as $n+m$ MILP variables of $\mathcal{M}$ corresponding to public and secret variables
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 1 \ \forall \ u_i = 1$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 0 \ \forall \ u_i = 0$
- $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i+n}}^0 = 0 \ \forall \ i \in \{0, 1, \ldots, m-1\}$
- Update $\mathtt{k}^0$ according to the propagation rules of CMP through r_m rounds to $\mathtt{k}^{\mathtt{r_m}}$
- Declare $\mathtt{s}^{\mathtt{r_m}}$ as n_{r_m} MILP variables of $\mathcal{M}$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = 0 \ \forall \ s_i^{(r_m)}.F = 0_c$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \ \forall \ s_i^{(r_m)}.F = 1_c$
- $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \ \forall \ s_i^{(r_m)}.F = \delta$
- Update $\mathtt{s}^{\mathtt{r_m}}$ according to the propagation rules of the monomial prediction through $r - r_m$ rounds to $\mathtt{s}^{\mathtt{r}}$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r}} = 0 \ \forall \ t_i^{(r)} = 0$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r}} = 1 \ \forall \ t_i^{(r)} = 1$
- **return** $\mathcal{M}$

A.2 Cubes in [16]

Table 5: Cube indices used in [16]

I	$\lvert I\rvert$	Indices
I_0	56	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, 30, 31, 32, 33, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_1	57	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, 30, 31, 32, 33, 34, 35, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79
I_2	55	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, 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
I_3	54	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, 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_4	76	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, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 73, 75, 77, 79

B Propagation Rules and MILP Models for XOR, AND and COPY

B.1 CBDP

MILP models of XOR, AND and COPY. The division trails can be traced using the following MILP models:

Model 1 (XOR [33]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{XOR}} b$ *be a propagation trail of* XOR*. The following inequalities suffice to describe all the valid trails for* XOR*:*

$$\begin{cases} \mathcal{M}.var \leftarrow a_0, a_1, \ldots, a_{n-1}, b \text{ as binary}; \\ \mathcal{M}.con \leftarrow b = a_0 + a_1 + \cdots + a_{n-1}; \end{cases}$$

Model 2 (AND [33]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{AND}} b$ *be a propagation trail of* AND*. The following inequalities suffice to describe all the valid trails for* AND*:*

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow a_0 + a_1 + \cdots + a_{n-1} \geq b; \\ \mathcal{M}.con \leftarrow b \geq a_i, \ \forall i \in \{0, 1, \ldots, n-1\}. \end{cases}$$

If the MILP solver supports the OR ($\vee$) *operation, then the model can also be represented by*

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b = a_0 \vee a_1 \vee \cdots \vee a_{n-1}; \end{cases}$$

Model 3 (COPY [33]) *Let a* $\xrightarrow{\text{COPY}} (b_0, b_1, \ldots, b_{n-1})$ *be a propagation trail of* AND*. The following inequalities suffice to describe all the valid trails for* COPY*:*

$$\begin{cases}\mathcal{M}.var \leftarrow a, b_0, b_1, \ldots, b_{n-1} \ \textit{as binary};\\ \mathcal{M}.con \leftarrow a = b_0 + b_1 + \cdots + b_{n-1};\end{cases}$$

B.2 Monomial Prediction

Propagation Rules of XOR, AND and COPY. According to [12], the rules for XOR, AND and COPY are the following:

Rule 1 (XOR [12,13]) *Let* $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ *and* $\boldsymbol{y} = (x_0 \oplus x_1, x_2, \ldots, x_{n-1})$ *be the input and output vector of a* XOR *function. Considering a monomial of* $\boldsymbol{x}$ *as* $\boldsymbol{x}^{\boldsymbol{u}}$*, the monomials* $\boldsymbol{y}^{\boldsymbol{v}}$ *of* $\boldsymbol{y}$ *meet the condition that* $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$ *only when* $\boldsymbol{v}$ *satisfies*

$$\boldsymbol{v} = (u_0 + u_1, u_2, \ldots, u_{n-1}), \quad (u_0, u_1) \in \{(0,0), (0,1), (1,0)\}.$$

Rule 2 (AND [12,13]) *Let* $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ *and* $\boldsymbol{y} = (x_0 \vee x_1, x_2, \ldots, x_{n-1})$ *be the input and output vector of an* AND *function. Considering a monomial of* $\boldsymbol{x}$ *as* $\boldsymbol{x}^{\boldsymbol{u}}$*, the monomials* $\boldsymbol{y}^{\boldsymbol{v}}$ *of* $\boldsymbol{y}$ *meet the condition that* $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$ *only when* $\boldsymbol{v}$ *satisfies*

$$\boldsymbol{v} = (u_0, u_2, \ldots, u_{n-1}), \quad (u_0, u_1) \in \{(0,0), (1,1)\}.$$

Rule 3 (COPY [12,13]) *Let* $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$ *and* $\boldsymbol{y} = (x_0, x_0, x_1, x_2, \ldots, x_{n-1})$ *be the input and output vector of a* COPY *function. Considering a monomial of* $\boldsymbol{x}$ *as* $\boldsymbol{x}^{\boldsymbol{u}}$*, the monomials* $\boldsymbol{y}^{\boldsymbol{v}}$ *of* $\boldsymbol{y}$ *meet the condition that* $\boldsymbol{x}^{\boldsymbol{u}} \rightarrow \boldsymbol{y}^{\boldsymbol{v}}$ *only when* $\boldsymbol{v}$ *satisfies*

$$\boldsymbol{v} = \begin{cases}(0, 0, u_2, \ldots, u_{n-1}), & \textit{if } u_0 = 0\\ (0, 1, u_2, \ldots, u_{n-1}), (1, 0, u_2, \ldots, u_{n-1}), (1, 1, u_2, \ldots, u_{n-1}), & \textit{if } u_0 = 1\end{cases}$$

MILP Models of the Propagation Trails. The propagation trails of the monomial prediction can be traced using the following MILP models:

Model 1 (XOR [12,13]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{XOR}} b$ *be a propagation trail of* XOR*. The following inequalities suffice to describe all the valid trails for* XOR*:*

$$\begin{cases}\mathcal{M}.var \leftarrow a_0, a_1, \ldots, a_{n-1}, b \ \textit{as binary};\\ \mathcal{M}.con \leftarrow b = a_0 + a_1 + \cdots + a_{n-1};\end{cases}$$

Model 2 (AND [12,13]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{AND}} b$ *be a propagation trail of* AND*. The following inequalities suffice to describe all the valid trails for* AND*:*

$$\begin{cases}\mathcal{M}.var \leftarrow a_0, a_1, \ldots, a_{n-1}, b \ \textit{as binary};\\ \mathcal{M}.con \leftarrow b = a_i, \ \forall i \in \{0, 1, \ldots n-1\}.\end{cases}$$

Model 3 (COPY [12,13]) *Let a* $\xrightarrow{\text{COPY}}(b_0, b_1, \ldots, b_{n-1})$ *be a propagation trail of* COPY*. The following inequalities suffice to describe all the valid trails for* COPY*:*

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_0, b_1, \ldots, b_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b_0 + b_1 + \cdots + b_{n-1} \geq a; \\ \mathcal{M}.con \leftarrow a \geq b_i, \ \forall i \in \{0, 1, \ldots, n-1\}. \end{cases}$$

If the MILP solver supports the OR ($\vee$) *operation, then the model can also be represented by:*

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_0, b_1, \ldots, b_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow a = b_0 \vee b_1 \vee \cdots \vee b_{n-1}; \end{cases}$$

C Details of NBDP

Rules of handling constant 0 bits. As mentioned in [29], the constant 0 bits mainly affect the AND operation, so once the input of AND involves contant 0 bits, the propagation of AND should be prevented. As for COPY or XOR, if the input bits are all 0's, we can choose to prevent the propagation, or simply declare that the output bits are also all 0s.

Rule 1 (zero bits in COPY [29]) *Let a* $\xrightarrow{\text{COPY}}(b_0, b_1, \ldots, b_{n-1})$ *be a propagation trail of* COPY*. If* $a.F = 0_c$*, then we do not model this operation or simply add the constraint* $b_i = 0$ *for* $i \in \{0, 1, \ldots, n-1\}$*. Otherwise, we use the MILP model of* COPY *to model this operation.*

Rule 2 (zero bits in AND [29]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{AND}} b$ *be a propagation trail of* AND*. If* $\exists\ 0 \leq i \leq n-1$ *s.t.* $a_i.F = 0_c$*, then we do not model this operation. Otherwise, we use the MILP model of* AND *to model this operation.*

Rule 3 (zero bits in XOR [29]) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{XOR}} b$ *be a propagation trail of* XOR*. If* $\exists\ 0 \leq i \leq n-1$ *s.t.* $a_i.F \neq 0_c$*, then we collect these non-zero input bits, say* $a_{n_0}, a_{n_1}, \ldots, a_{n_k}$*, and use the MILP model of* XOR *to model* $(a_{n_0}, a_{n_1}, \ldots, a_{n_k}) \xrightarrow{\text{XOR}} b$*. Otherwise, we do not model this operation or simply add the constraint* $b = 0$*.*

Additional constraints in NBDP. Consider the composition of the operations COPY and AND : $(a, b) \xrightarrow[a\to(a_0,a_1),b\to(b_0,b_1)]{\text{COPY}} (a_0, a_1, b_0, b_1) \xrightarrow[(a_1,b_1)\to c]{\text{AND}} (a_0, b_0, c)$, denoted by $(a, b) \overset{\mathbb{K}}{\rightsquigarrow} (a_0, b_0, c)$. Then $(1, 1) \overset{\mathbb{K}}{\rightsquigarrow} (0, 1, 1)$, $(1, 1) \overset{\mathbb{K}}{\rightsquigarrow} (1, 0, 1)$ and $(1, 1) \overset{\mathbb{K}}{\rightsquigarrow} (0, 0, 1)$ are all valid. The former two transitions are redundant and thus can be eliminated by adding the following constraints:

$$\mathcal{M}.con \leftarrow a_1 = a \wedge c,$$

$$\mathcal{M}.con \leftarrow b_1 = b \wedge c.$$

These two constraints indicate that if $c = 1, a = 1$ (resp. $c = 1, b = 1$), then $a_1 = 1$ (resp. $b_1 = 1$) must also hold.

D Details of CMP

MILP Models of Core Monomial Trails. Based on the propagation rules of CMP, we can easily derive the MILP models corresponding to COPY, AND and XOR. When modeling the propagation of CMP, the flags are computed according to the cube term ahead of time, hence the introduction of the flag technique doesn't affect the efficiency at all.

Model 1 (XOR) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{XOR}} b$ *be a propagation trail of* XOR. *The following inequalities suffice to describe all the valid trails for* XOR*:*

$$\begin{cases} \mathcal{M}.var \leftarrow a_0, a_1, \ldots, a_{n-1}, b \text{ as binary}; \\ \mathcal{M}.con \leftarrow b \geq a_0 + a_1 + \cdots + a_{n-1}, \text{ if } \{1_c, \delta\} \subseteq \{a_0.F, a_1.F, \ldots, a_{n-1}.F\}; \\ \mathcal{M}.con \leftarrow b = a_0 + a_1 + \cdots + a_{n-1}, \text{ otherwise }. \end{cases}$$

Model 2 (AND) *Let* $(a_0, a_1, \ldots, a_{n-1}) \xrightarrow{\text{AND}} b$ *be a propagation trail of* AND. *The following inequalities suffice to describe all the valid trails for* AND*:*

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow a_0 + a_1 + \cdots + a_{n-1} \geq b; \\ \mathcal{M}.con \leftarrow b \geq a_i, \ \forall i \in \{0, 1, \ldots, n-1\}; \\ \mathcal{M}.con \leftarrow b = a_i, \ \forall i \in \{0, 1, \ldots, n-1\} \text{ and } a_i.F = \delta. \end{cases}$$

If the MILP solver supports the OR $(\vee)$ *operation, then the model can also be represented by:*

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b = a_0 \vee a_1 \vee \cdots \vee a_{n-1}; \\ \mathcal{M}.con \leftarrow b = a_i, \ \forall i \in \{0, 1, \ldots, n-1\} \text{ and } a_i.F = \delta. \end{cases}$$

Model 3 (COPY) *Let* $a \xrightarrow{\text{COPY}} (b_0, b_1, \ldots, b_{n-1})$ *be a propagation trail of* AND. *The following inequalities suffice to describe all the valid trails for* COPY*:*

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_0, b_1, \ldots, b_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b_0 + b_1 + \cdots + b_{n-1} \geq a; \\ \mathcal{M}.con \leftarrow a \geq b_i, \ \forall i \in \{0, 1, \ldots, n-1\}. \end{cases}$$

If the MILP solver supports the OR $(\vee)$ *operation, then the model can also be represented by:*

$$\begin{cases} \mathcal{M}.var \leftarrow a, b_0, b_1, \ldots, b_{n-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow a = b_0 \vee b_1 \vee \cdots \vee b_{n-1}; \end{cases}$$

E Our Term Expander Exploiting Callback Functions

In this section, we propose a new term expander to work with the new designed coefficient solver proposed in Sect. 5.

E.1 Motivation

As mentioned in Sect. 3, after computing $\mathbb{S}^{(r_n)}$ from $\mathbb{S}_u^{(r_l)}$ using the term expander, the coefficient solver is applied to each term in $\mathbb{S}^{(r_n)}$. In our experiment with NMP, we found most terms in $\mathbb{S}^{(r_n)}$ are partitioned into $\mathbb{S}_0^{(r_n)}$, while the term expander can't recognize this fact. These useless terms that are later partitioned into $\mathbb{S}_0^{(r_n)}$ not only lead to a lot of wasted time, but also cause the number of rounds to drop slowly.

Consider the following conditions :

$$\text{Condition C: } \sum_{\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}) \in \mathbb{S}_u^{(r_l)}} |\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \bowtie \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})| \equiv 1 \pmod 2,$$

Condition D: $\exists \boldsymbol{k}^{\boldsymbol{w}}$ s.t. $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$.

While the term expander of NMP only captures those $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$'s satisfying Condition C, our new term expander aims to include $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying both Condition C and D into $\mathbb{S}^{(r_n)}$. If we continue to obey similar rules in NMP for selecting r_n, i.e., the expansion doesn't stop until the size of $\mathbb{S}^{(r_n)}$ is larger than N for the first time, it can be predicted that our term expander will select a r_n that is farther away from r_l than the term expander of NMP. As a result, we save time for the coefficient solver while ensuring a fast drop in the number of rounds, which helps to increase the recovery speed of superpoly.

E.2 The theory

Specifically, we can calculate $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying Condition C and Condition D by the following steps:

1. For each $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}) \in \mathbb{S}_u^{(r_l)}$, recover $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \rightsquigarrow \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ and Condition D, and add $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s to a multiset $\mathbb{T}^{(r_n)}$.
2. Count the number of occurrences for each element in $\mathbb{T}^{(r_n)}$ and add elements occurring an odd number of times to $\mathbb{S}^{(r_n)}$, then $\mathbb{S}^{(r_n)}$ is exactly what we want.

How to recover $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$s satisfying $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \rightsquigarrow \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ and Condition D has been discussed in Sect. 5 and two MILP models have been proposed to solve it provided that enumerating all solutions of the MILP model is feasible. Since $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ is partitioned into $\mathbb{S}_u^{(r_l)}$, enumerating all the solutions

turns out to be infeasible here. To circumvent this requirement, we attempt to list all possible values of $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ one by one for the MILP model based on

$$\overbrace{\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \xrightarrow{\textbf{NBDP}} \pi_{\boldsymbol{k}^{(r_c)}}(\boldsymbol{s}^{(r_c)})}^{r_c \text{ rounds}} \hat{\preceq} \overbrace{\pi_{\boldsymbol{t}^{(r_c)}}(\boldsymbol{s}^{(r_c)}) \xrightarrow{\textbf{MP}} \pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \xrightarrow{\textbf{MP}} \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})}^{r_l - r_c \text{ rounds}}, \quad (10)$$

or

$$\overbrace{\boldsymbol{k}^{\boldsymbol{0}}\boldsymbol{x}^{\boldsymbol{u}} \xrightarrow{\textbf{CMP}} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(r_c)})}(\boldsymbol{s}^{(r_c)})}^{r_c \text{ rounds}} \tilde{\preceq} \overbrace{\pi_{\boldsymbol{t}^{(r_c)}}(\boldsymbol{s}^{(r_c)}) \xrightarrow{\textbf{MP}} \pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \xrightarrow{\textbf{MP}} \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})}^{r_l - r_c \text{ rounds}}, \quad (11)$$

where r_c is an arbitrary round close to r_n satisfying $0 < r_c \leq r_n$. Every time we obtain a solution of $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$, we remove it from the solution space and check if $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)}) \rightarrow \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ using the monomial prediction manually, which can be implemented by callback functions in Gurobi. Such a strategy, also used in [28], entails repeatedly finding one solution for a MILP model. If even finding one solution for the MILP model is impossible, which usually happens for a large r_l, say $r_l = 800$ for TRIVIUM, then we have to resort to the expansion strategy in NMP[2]. A threshold value R_s of the number of rounds is set to determine when the expansion strategy of NMP should be used. When our expansion strategy fails, i.e, the time to find a solution exceeds our preset time limit, we also fall back to the expansion strategy of NMP. In our experiment, we found that the MILP model of Eqn. (10) performed better than the one of Eqn. (11) in callback mode, so we decide to use NBDP in the first r_c rounds.

The choice of r_n. When implementing our term expander in practice, we select a moderate value of ε for the cipher and attempt to recover $\pi_{\boldsymbol{t}^{(r_l - t\varepsilon)}}(\boldsymbol{s}^{(r_l - t\varepsilon)})$s satisfying Condition C and D for a t that starts from 1 and is incremented. r_n is chosen as the first value of $r_l - t\varepsilon$ that makes the size of $\mathbb{S}^{(r_l - t\varepsilon)}$ larger than N, where N is equal to 10 000 or 15 000. Note that once the selection of r_n is over, $\mathbb{S}^{(r_n)}$ has also been calculated.

To sum up, an intuitive description of our term expander and how to select r_n is present in Algorithm 6, where we use `ChooceRCiX`(r_n) to represent the procedure of selecting r_c and τ' to represent the preset time limit for our term expander.

[2] The expansion strategy of NMP is independent of the number of rounds, but it will generate thousands of useless terms, which leads to more time consumption.

Algorithm 6: New term expander and how to choose r_n

Procedure `ChooseRiX`($\mathbb{S}_u^{(r_l)}, r_l, \boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, ε)**:**
  if $r_l > R_s$ **then**
    Call the term expander in NMP to expand $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ into $\mathbb{S}^{(r_n)}$
    return r_n
  $\mathbb{S}^{(r_l)} \leftarrow \mathbb{S}_u^{(r_l)}$
  do
    $r_n \leftarrow r_l - \varepsilon$
    $r_c =$ `ChooseRCiX`(r_n)
    Initialize an empty set $\mathbb{S}^{(r_n)}$
    Prepare a hash table $J^{(r_n)}$ whose key is a term of r_n-th round and the value is an integer
    for $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)}) \in \mathbb{S}^{(r_l)}$ **do**
      Prepare an empty set $\mathbb{P}^{(r_n)}$
      $\mathcal{M} =$ `NBDP-MPModelX`$(r_l, \boldsymbol{t}^{(r_l)}, \boldsymbol{u}, r_c)$ `/* See Algorithm 4 */`
      Solve the MILP model $\mathcal{M}$ within the time limit τ'
      Loop
        if *Find a solution for* $\mathcal{M}$ **then**
          Extract the solution corresponding to $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ as $\pi_{\boldsymbol{p}}(\boldsymbol{s}^{(r_n)})$
          Remove $\pi_{\boldsymbol{p}}(\boldsymbol{s}^{(r_n)})$ from the solution space
          Solve a MILP model of the monomial prediction to determine if $\pi_{\boldsymbol{p}}(\boldsymbol{s}^{(r_n)}) \to \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$
          if $\pi_{\boldsymbol{p}}(\boldsymbol{s}^{(r_n)}) \to \pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ **then**
            Add $\pi_{\boldsymbol{p}}(\boldsymbol{s}^{(r_n)})$ to $\mathbb{P}^{(r_n)}$
          Continue to solve $\mathcal{M}$
        else if $\mathcal{M}$ *has no solution* **then**
          Break the loop
        else if $\mathcal{M}$ *reaches the time limit* **then**
          Clear $\mathbb{P}^{(r_n)}$ and call the term expander of NMP to expand $\pi_{\boldsymbol{t}^{(r_l)}}(\boldsymbol{s}^{(r_l)})$ into $\mathbb{P}^{(r_n)}$
          Break the loop
      for *each* $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ *in* $\mathbb{P}^{(r_n)}$ **do**
        Increase $J^{(r_n)}[\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})]$ by 1
      for *each* $\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$ *whose* $J^{(r_n)}[\pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})]$ *is an odd-number* **do**
        $\mathbb{S}^{(r_n)} = \mathbb{S}^{(r_n)} \cup \pi_{\boldsymbol{t}^{(r_n)}}(\boldsymbol{s}^{(r_n)})$
      $\mathbb{S}^{(r_l)} \leftarrow \mathbb{S}^{(r_n)}$
      $r_l \leftarrow r_n$
  while $|\mathbb{S}^{(r_l)}| \leq N$
  return r_n

Procedure `TermExpanderX`($\mathbb{S}_u^{(r_l)}, r_l, r_n, \boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, ε)**:**
  Notice that once r_n is selected, $\mathbb{S}^{(r_n)}$ is already determined, so we extract the set $\mathbb{S}^{(r_n)}$ directly from `ChooseRiX`$(\mathbb{S}_u^{(r_l)}, r_l, \boldsymbol{u}, \varepsilon)$.
  return $\mathbb{S}^{(r_n)}$

F Basic Models and Functions Used for NBDP and CMP

Algorithm 8: Basic models for CMP

Procedure `CMPCopy`($\mathcal{M}, \mathtt{x}$):
  $\mathcal{M}.var \leftarrow \mathtt{y}, \mathtt{z}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{x} = \mathtt{y} \vee \mathtt{z}$
  return $(\mathcal{M}, \mathtt{y}, \mathtt{z})$

Procedure `CMPAnd1`($\mathcal{M}, \mathtt{x}$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}, \mathtt{z}) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x})$
    Add $\mathtt{z}$ to $\mathbb{V}$
  else
    $\mathtt{y} = \mathtt{x}$
  return $(\mathcal{M}, \mathtt{y}, \mathbb{V})$

Procedure `CMPAnd2`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$ *and* $x_1.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1$
    for $\mathtt{x}_i \in \{\mathtt{x}_0, \mathtt{x}_1\}$ *for which* $\mathtt{x}_i.F = \delta$ **do**
      $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_i$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathbb{V})$

Procedure `CMPAnd3`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$, $x_1.F \neq 0_c$ *and* $x_2.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $(\mathcal{M}, \mathtt{y}_2, \mathtt{z}_2) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_2)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1 \vee \mathtt{z}_2$
    for $\mathtt{x}_i \in \{\mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2\}$ *for which* $\mathtt{x}_i.F = \delta$ **do**
      $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_i$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1, \mathtt{y}_2 = \mathtt{x}_2$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathtt{y}_2, \mathbb{V})$

Procedure `CMPAnd4`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$, $x_1.F \neq 0_c$, $x_2.F \neq 0_c$ *and* $x_3.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $(\mathcal{M}, \mathtt{y}_2, \mathtt{z}_2) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_2), (\mathcal{M}, \mathtt{y}_3, \mathtt{z}_3) =$ `CMPCopy`$(\mathcal{M}, \mathtt{x}_3)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1 \vee \mathtt{z}_2 \vee \mathtt{z}_3$
    for $\mathtt{x}_i \in \{\mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3\}$ *for which* $\mathtt{x}_i.F = \delta$ **do**
      $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_i$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1, \mathtt{y}_2 = \mathtt{x}_2, \mathtt{y}_3 = \mathtt{x}_3$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathtt{y}_2, \mathtt{y}_3, \mathbb{V})$

Procedure `CMPXor`($\mathcal{M}$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  $\mathcal{M}.var \leftarrow \mathtt{o}$ as binary
  if $|\mathbb{V}| = 0$ **then**
    $\mathcal{M}.con \leftarrow \mathtt{o} = 0$
  else
    Initialize a linear expression $\mathtt{nv} = 0$
    for $\mathtt{v} \in \mathbb{V}$ **do**
      $\mathtt{nv} \mathrel{+}= \mathtt{v}$
    $\mathcal{M}.con \leftarrow \mathtt{o} = \mathtt{nv}$
  return $(\mathcal{M}, \mathtt{o})$

Algorithm 7: Basic models for NBDP

Procedure `NBDPCopy`($\mathcal{M}, \mathtt{x}$):
  $\mathcal{M}.var \leftarrow \mathtt{y}, \mathtt{z}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{x} = \mathtt{y} + \mathtt{z}$
  return $(\mathcal{M}, \mathtt{y}, \mathtt{z})$

Procedure `NBDPAnd1`($\mathcal{M}, \mathtt{x}$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}, \mathtt{z}) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x})$
    Add $\mathtt{z}$ to $\mathbb{V}$
  else
    $\mathtt{y} = \mathtt{x}$
  return $(\mathcal{M}, \mathtt{y}, \mathbb{V})$

Procedure `NBDPAnd2`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$ *and* $x_1.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1$
    $\mathcal{M}.con \leftarrow \mathtt{z}_0 = \mathtt{a} \wedge \mathtt{x}_0, \mathtt{z}_1 = \mathtt{a} \wedge \mathtt{x}_1$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathbb{V})$

Procedure `NBDPAnd3`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2, \mathtt{x}_3$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$, $x_1.F \neq 0_c$ *and* $x_2.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $(\mathcal{M}, \mathtt{y}_2, \mathtt{z}_2) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_2)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1 \vee \mathtt{z}_2$
    $\mathcal{M}.con \leftarrow \mathtt{z}_0 = \mathtt{a} \wedge \mathtt{x}_0, \mathtt{z}_1 = \mathtt{a} \wedge \mathtt{x}_1, \mathtt{z}_2 = \mathtt{a} \wedge \mathtt{x}_2$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1, \mathtt{y}_2 = \mathtt{x}_2$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathtt{y}_2, \mathbb{V})$

Procedure `NBDPAnd4`($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  if $x_0.F \neq 0_c$, $x_1.F \neq 0_c$, $x_2.F \neq 0_c$ *and* $x_3.F \neq 0_c$ **then**
    $(\mathcal{M}, \mathtt{y}_0, \mathtt{z}_0) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_0), (\mathcal{M}, \mathtt{y}_1, \mathtt{z}_1) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_1)$
    $(\mathcal{M}, \mathtt{y}_2, \mathtt{z}_2) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_2), (\mathcal{M}, \mathtt{y}_3, \mathtt{z}_3) =$ `NBDPCopy`$(\mathcal{M}, \mathtt{x}_3)$
    $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_0 \vee \mathtt{z}_1 \vee \mathtt{z}_2 \vee \mathtt{z}_3$
    $\mathcal{M}.con \leftarrow \mathtt{z}_0 = \mathtt{a} \wedge \mathtt{x}_0, \mathtt{z}_1 = \mathtt{a} \wedge \mathtt{x}_1, \mathtt{z}_2 = \mathtt{a} \wedge \mathtt{x}_2, \mathtt{z}_3 = \mathtt{a} \wedge \mathtt{x}_3$
    Add $\mathtt{a}$ to $\mathbb{V}$
  else
    $\mathtt{y}_0 = \mathtt{x}_0, \mathtt{y}_1 = \mathtt{x}_1, \mathtt{y}_2 = \mathtt{x}_2, \mathtt{y}_3 = \mathtt{x}_3$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \mathtt{y}_2, \mathtt{y}_3, \mathbb{V})$

Procedure `NBDPXor`($\mathcal{M}$, the set of $\mathcal{M}.var$'s $\mathbb{V}$):
  $\mathcal{M}.var \leftarrow \mathtt{o}$ as binary
  if $|\mathbb{V}| = 0$ **then**
    $\mathcal{M}.con \leftarrow \mathtt{o} = 0$
  else
    Initialize a linear expression $\mathtt{nv} = 0$
    for $\mathtt{v} \in \mathbb{V}$ **do**
      $\mathtt{nv} \mathrel{+}= \mathtt{v}$
    $\mathcal{M}.con \leftarrow \mathtt{o} = \mathtt{nv}$
  return $(\mathcal{M}, \mathtt{o})$

G Borrowed Models and Functions from [12,13]

G.1 Trivium

Algorithm 9: MILP Model for monomial trails of the update function in Trivium

Procedure MPTriviumCore($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_{287}, i_1, i_2, i_3, i_4, i_5$):
  $\mathcal{M}.var \leftarrow \mathtt{y}_{\mathtt{i}_1}, \mathtt{y}_{\mathtt{i}_2}, \mathtt{y}_{\mathtt{i}_3}, \mathtt{y}_{\mathtt{i}_4}, \mathtt{y}_{\mathtt{i}_5}, \mathtt{z}_1, \mathtt{z}_2, \mathtt{z}_3, \mathtt{z}_4, \mathtt{a}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{x}_{\mathtt{i}_\mathtt{j}} = \mathtt{y}_{\mathtt{i}_\mathtt{j}} \vee \mathtt{z}_\mathtt{j}$ for all $\mathtt{j} \in \{1, 2, 3, 4\}$
  $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_3$
  $\mathcal{M}.con \leftarrow \mathtt{a} = \mathtt{z}_4$
  $\mathcal{M}.con \leftarrow \mathtt{y}_{\mathtt{i}_5} = \mathtt{x}_{\mathtt{i}_5} + \mathtt{a} + \mathtt{z}_1 + \mathtt{z}_2$
  for $\mathtt{i} \in \{0, 1, \ldots, 287\}$ *w/o* $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4, \mathtt{i}_5$ **do** $\mathtt{y}_\mathtt{i} = \mathtt{x}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{287})$
Procedure MPTriviumUpdate($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}$):
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}, 65, 170, 90, 91, 92$)
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176$)
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287$)
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{z}_{287}, \mathtt{z}_0, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1})$

G.2 Kreyvium

Algorithm 10: MILP Model for monomial trails of the update function in Kreyvium

Procedure MPLFSR($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{127}$):
  $\mathcal{M}.var \leftarrow \mathtt{a}, \mathtt{b}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{x}_0 = \mathtt{a} \vee \mathtt{b}$
  $(\mathtt{y}_0, \ldots, \mathtt{y}_{127}) = (\mathtt{x}_1, \ldots, \mathtt{x}_{126}, \mathtt{a})$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{127}, \mathtt{b})$
Procedure MPKreyviumUpdate($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}}$):
  $(\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{a}^\mathtt{i}) \leftarrow$ MPLFSR($\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}$)
  $(\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1}, \mathtt{b}^\mathtt{i}) \leftarrow$ MPLFSR($\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}}$)
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}, 65, 170, 90, 91, 92$)
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176$)
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287})$ = MPTriviumCore($\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287$)
  $\mathcal{M}.var \leftarrow \mathtt{t}_1^\mathtt{i}, \mathtt{t}_3^\mathtt{i}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{t}_1^\mathtt{i} = \mathtt{z}_{92} + \mathtt{b}^\mathtt{i}$
  $\mathcal{M}.con \leftarrow \mathtt{t}_3^\mathtt{i} = \mathtt{z}_{287} + \mathtt{a}^\mathtt{i}$
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{t}_3^\mathtt{i}, \mathtt{z}_0, \ldots, \mathtt{z}_{91}, \mathtt{t}_1^\mathtt{i}, \mathtt{z}_{93}, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1})$

G.3 ACORN

Algorithm 11: MILP Models for monomial trails of *Maj* and *Ch* in ACORN

Procedure $\texttt{MPMaj}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, \texttt{i}, \texttt{j}, \texttt{k})$:
  $\mathcal{M}.var \leftarrow \texttt{a}, \texttt{b}, \texttt{c}, \texttt{y}_\texttt{i}, \texttt{y}_\texttt{j}, \texttt{y}_\texttt{k}, \texttt{o}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{i} = \texttt{a} \vee \texttt{b} \vee \texttt{y}_\texttt{i}$
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{j} = \texttt{a} \vee \texttt{c} \vee \texttt{y}_\texttt{j}$
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{k} = \texttt{b} \vee \texttt{c} \vee \texttt{y}_\texttt{k}$
  $\mathcal{M}.con \leftarrow \texttt{o} = \texttt{a} + \texttt{b} + \texttt{c}$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $\texttt{i}, \texttt{j}, \texttt{k}$ **do** $\texttt{y}_\texttt{i} = \texttt{x}_\texttt{i}$
  return $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{o})$

Procedure $\texttt{MPCh}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, \texttt{i}, \texttt{j}, \texttt{k})$:
  $\mathcal{M}.var \leftarrow \texttt{a}, \texttt{b}, \texttt{c}, \texttt{y}_\texttt{i}, \texttt{y}_\texttt{j}, \texttt{y}_\texttt{k}, \texttt{o}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{i} = \texttt{a} \vee \texttt{b} \vee \texttt{y}_\texttt{i}$
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{j} = \texttt{a} \vee \texttt{y}_\texttt{j}$
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{k} = \texttt{b} \vee \texttt{c} \vee \texttt{y}_\texttt{k}$
  $\mathcal{M}.con \leftarrow \texttt{o} = \texttt{a} + \texttt{b} + \texttt{c}$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $\texttt{i}, \texttt{j}, \texttt{k}$ **do** $\texttt{y}_\texttt{i} = \texttt{x}_\texttt{i}$
  return $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{o})$

Algorithm 12: MILP Models for monomial trails of the LFSR in ACORN

Procedure $\texttt{MPxorFB}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, \texttt{i}, \texttt{j}, \texttt{k})$:
  $\mathcal{M}.var \leftarrow \texttt{b}, \texttt{c}, \texttt{y}_\texttt{j}, \texttt{y}_\texttt{k}, \texttt{y}_\texttt{i}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{j} = \texttt{b} \vee \texttt{y}_\texttt{j}$
  $\mathcal{M}.con \leftarrow \texttt{x}_\texttt{k} = \texttt{c} \vee \texttt{y}_\texttt{k}$
  $\mathcal{M}.con \leftarrow \texttt{y}_\texttt{i} = \texttt{x}_\texttt{i} + \texttt{b} + \texttt{c}$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $\texttt{i}, \texttt{j}, \texttt{k}$ **do** $\texttt{y}_\texttt{i} = \texttt{x}_\texttt{i}$
  return $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292})$

Algorithm 13: MILP Models for monomial trails of *ksg128* and *fbk128* in ACORN

Procedure $\texttt{MPksg128}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292})$:
  $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{c}) = \texttt{MPMaj}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 235, 61, 193)$
  $(\mathcal{M}, \texttt{z}_0, \texttt{z}_1, \ldots, \texttt{z}_{292}, \texttt{d}) = \texttt{MPCh}(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292}, 230, 111, 66)$
  $\mathcal{M}.var \leftarrow \texttt{a}, \texttt{b}, \texttt{t}_{12}, \texttt{t}_{154}, \texttt{o}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{z}_{12} = \texttt{a} \vee \texttt{t}_{12}$
  $\mathcal{M}.con \leftarrow \texttt{z}_{154} = \texttt{b} \vee \texttt{t}_{154}$
  $\mathcal{M}.con \leftarrow \texttt{o} = \texttt{a} + \texttt{b} + \texttt{c} + \texttt{d}$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $12, 154$ **do** $\texttt{t}_\texttt{i} = \texttt{z}_\texttt{i}$
  return $(\mathcal{M}, \texttt{t}_0, \texttt{t}_1, \ldots, \texttt{t}_{292}, \texttt{o})$

Procedure $\texttt{MPfbk128}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292})$:
  $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{d}) = \texttt{MPMaj}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 244, 23, 160)$
  $\mathcal{M}.var \leftarrow \texttt{a}, \texttt{b}, \texttt{c}, \texttt{t}_0, \texttt{t}_{107}, \texttt{t}_{196}, \texttt{o}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{y}_0 = \texttt{a} \vee \texttt{t}_0$
  $\mathcal{M}.con \leftarrow \texttt{y}_{107} = \texttt{b} \vee \texttt{t}_{107}$
  $\mathcal{M}.con \leftarrow \texttt{y}_{196} = \texttt{c} \vee \texttt{t}_{196}$
  $\mathcal{M}.con \leftarrow \texttt{o} \geq \texttt{a} + \texttt{b} + \texttt{c} + \texttt{d}$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $0, 107, 196$ **do** $\texttt{t}_\texttt{i} = \texttt{y}_\texttt{i}$
  return $(\mathcal{M}, \texttt{t}_0, \texttt{t}_1, \ldots, \texttt{t}_{292}, \texttt{o})$

Algorithm 14: MILP Models for monomial trails of the update function in ACORN

Procedure $\texttt{MPgenM}(\mathcal{M}, \texttt{k}_0, \ldots, \texttt{k}_{127}, \texttt{v}_0, \ldots, \texttt{v}_{127}, r)$**:**
- $\mathcal{M}.var \leftarrow \texttt{a}, \texttt{x}$ as binary
- **if** $r < 128$ **then**
  - $\mathcal{M}.con \leftarrow \texttt{k}_\texttt{r} = \texttt{a} \vee \texttt{x}$
  - $\texttt{k}'_\texttt{r} = \texttt{x}$
  - **for** $\texttt{i} \in \{0, 1, \ldots, 127\}$ *w/o* r **do** $\texttt{k}'_\texttt{i} = \texttt{k}_\texttt{i}$
  - **return** $(\mathcal{M}, \texttt{k}'_0, \ldots, \texttt{k}'_{127}, \texttt{v}_0, \ldots, \texttt{v}_{127}, \texttt{a})$
- **else if** $r < 256$ **then**
  - $\mathcal{M}.con \leftarrow \texttt{v}_{\texttt{r}-128} = \texttt{a} \vee \texttt{x}$
  - $\texttt{v}'_{\texttt{r}-128} = \texttt{x}$
  - **for** $\texttt{i} \in \{0, 1, \ldots, 127\}$ *w/o* $r - 128$ **do** $\texttt{v}'_\texttt{i} = \texttt{v}_\texttt{i}$
  - **return** $(\mathcal{M}, \texttt{k}_0, \ldots, \texttt{k}_{127}, \texttt{v}'_0, \ldots, \texttt{v}'_{127}, \texttt{a})$
- **else**
  - $\mathcal{M}.con \leftarrow \texttt{k}_{\texttt{r} \bmod 128} = \texttt{a} \vee \texttt{x}$
  - $\texttt{k}'_{\texttt{r} \bmod 128} = \texttt{x}$
  - **for** $\texttt{i} \in \{0, 1, \ldots, 127\}$ *w/o* $r \bmod 128$ **do** $\texttt{k}'_\texttt{i} = \texttt{k}_\texttt{i}$
  - **if** $r \neq 256$ **then**
    - **return** $(\mathcal{M}, \texttt{k}'_0, \ldots, \texttt{k}'_{127}, \texttt{v}_0, \ldots, \texttt{v}_{127}, \texttt{a})$
  - **else**
    - $\mathcal{M}.var \leftarrow \texttt{o}$ as binary
    - $\mathcal{M}.con \leftarrow \texttt{o} \geq \texttt{a}$
    - **return** $(\mathcal{M}, \texttt{k}'_0, \ldots, \texttt{k}'_{127}, \texttt{v}_0, \ldots, \texttt{v}_{127}, \texttt{o})$

Procedure $\texttt{MPAcornUpdate}(\mathcal{M}, \texttt{s}^\texttt{i}_0, \ldots, \texttt{s}^\texttt{i}_{292}, \texttt{k}^\texttt{i}_0, \ldots, \texttt{k}^\texttt{i}_{127}, \texttt{v}^\texttt{i}_0, \ldots, \texttt{v}^\texttt{i}_{127})$**:**
- $(\mathcal{M}, \texttt{t}_0, \ldots, \texttt{t}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{s}^\texttt{i}_0, \ldots, \texttt{s}^\texttt{i}_{292}, 289, 235, 230)$
- $(\mathcal{M}, \texttt{u}_0, \ldots, \texttt{u}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{t}_0, \ldots, \texttt{t}_{292}, 230, 196, 193)$
- $(\mathcal{M}, \texttt{w}_0, \ldots, \texttt{w}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{u}_0, \ldots, \texttt{u}_{292}, 193, 160, 154)$
- $(\mathcal{M}, \texttt{l}_0, \ldots, \texttt{l}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{w}_0, \ldots, \texttt{w}_{292}, 154, 111, 107)$
- $(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{l}_0, \ldots, \texttt{l}_{292}, 107, 66, 61)$
- $(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292}) = \texttt{MPxorFB}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 61, 23, 0)$
- $(\mathcal{M}, \texttt{a}_0, \ldots, \texttt{a}_{292}, \texttt{e}) = \texttt{MPksg128}(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292})$
- $(\mathcal{M}, \texttt{z}_0, \ldots, \texttt{z}_{292}, \texttt{b}) = \texttt{MPfbk128}(\mathcal{M}, \texttt{a}_0, \ldots, \texttt{a}_{292})$
- $(\mathcal{M}, \texttt{k}^{\texttt{i}+1}_0, \ldots, \texttt{k}^{\texttt{i}+1}_{127}, \texttt{v}^{\texttt{i}+1}_0, \ldots, \texttt{v}^{\texttt{i}+1}_{127}, \texttt{m}) = \texttt{MPgenM}(\mathcal{M}, \texttt{k}^\texttt{i}_0, \ldots, \texttt{k}^\texttt{i}_{127}, \texttt{v}^\texttt{i}_0, \ldots, \texttt{v}^\texttt{i}_{127}, \texttt{i})$
- $\mathcal{M}.var \leftarrow \texttt{o}$ as binary
- $\mathcal{M}.con \leftarrow \texttt{o} = \texttt{e} + \texttt{m} + \texttt{b}$
- $\mathcal{M}.con \leftarrow \texttt{z}_0 = 0$
- $(\texttt{s}^{\texttt{i}+1}_0, \ldots, \texttt{s}^{\texttt{i}+1}_{292}) = (\texttt{z}_1, \texttt{z}_2, \ldots, \texttt{z}_{292}, \texttt{o})$
- **return** $(\mathcal{M}, \texttt{s}^{\texttt{i}+1}_0, \ldots, \texttt{s}^{\texttt{i}+1}_{292}, \texttt{k}^{\texttt{i}+1}_0, \ldots, \texttt{k}^{\texttt{i}+1}_{127}, \texttt{v}^{\texttt{i}+1}_0, \ldots, \texttt{v}^{\texttt{i}+1}_{127})$

G.4 Grain-128AEAD

Algorithm 15: MILP Models for monomial trails of XOR and AND in Grain-128AEAD

Procedure $\texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, I, J)$**:**
- $\mathcal{M}.var \leftarrow \texttt{b}'_\texttt{i}, \texttt{x}_\texttt{i}\ \forall\ i \in I$ as binary
- $\mathcal{M}.var \leftarrow \texttt{s}'_\texttt{j}, \texttt{y}_\texttt{j}\ \forall\ j \in J$ as binary
- $\mathcal{M}.var \leftarrow \texttt{z}$ as binary
- $\mathcal{M}.con \leftarrow \texttt{b}_\texttt{i} = \texttt{b}'_\texttt{i} \vee \texttt{x}_\texttt{i}\ \forall\ i \in I$
- $\mathcal{M}.con \leftarrow \texttt{s}_\texttt{i} = \texttt{s}'_\texttt{j} \vee \texttt{y}_\texttt{j}\ \forall\ j \in J$
- $\mathcal{M}.con \leftarrow \texttt{z} = \texttt{x}_\texttt{i}\ \forall\ i \in I$
- $\mathcal{M}.con \leftarrow \texttt{z} = \texttt{y}_\texttt{j}\ \forall\ j \in J$
- **for** $i \in \{0, 1, \ldots, 127\} \backslash I$ **do** $\texttt{s}'_\texttt{i} = \texttt{s}_\texttt{i}$
- **for** $j \in \{0, 1, \ldots, 127\} \backslash J$ **do** $\texttt{b}'_\texttt{j} = \texttt{b}_\texttt{j}$
- **return** $(\mathcal{M}, \texttt{b}'_0, \ldots, \texttt{b}'_{127}, \texttt{s}'_0, \ldots, \texttt{s}'_{127}, \texttt{z})$

Procedure $\texttt{MPXOR}((\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, I, J)$**:**
- $\mathcal{M}.var \leftarrow \texttt{b}'_\texttt{i}, \texttt{x}_\texttt{i}\ \forall\ i \in I$ as binary
- $\mathcal{M}.var \leftarrow \texttt{s}'_\texttt{j}, \texttt{y}_\texttt{j}\ \forall\ j \in J$ as binary
- $\mathcal{M}.con \leftarrow \texttt{z}$ as binary
- $\mathcal{M}.con \leftarrow \texttt{b}_\texttt{i} = \texttt{b}'_\texttt{i} \vee \texttt{x}_\texttt{i}\ \forall i \in I$
- $\mathcal{M}.con \leftarrow \texttt{s}_\texttt{j} = \texttt{s}'_\texttt{j} \vee \texttt{y}_\texttt{j}\ \forall j \in J$
- $\mathcal{M}.con \leftarrow \texttt{z} = \sum_{\texttt{i} \in \texttt{I}} \texttt{x}_\texttt{i} + \sum_{\texttt{j} \in \texttt{J}} \texttt{y}_\texttt{j}$
- **for** $i \in \{0, 1, \ldots, 127\} \backslash I$ **do** $b'_i = b_i$
- **for** $j \in \{0, 1, \ldots, 127\} \backslash J$ **do** $s'_j = s_j$
- **return** $(\mathcal{M}, \texttt{b}'_0, \ldots, \texttt{b}'_{127}, \texttt{s}'_0, \ldots, \texttt{s}'_{127}, \texttt{z})$

Algorithm 16: MILP Models for monomial trails of the NFSR and LFSR in Grain-128AEAD

Procedure $\texttt{MPfuncZ}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127})$**:**
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{a}_1) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \{12\}, \{8\})$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{a}_2) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \varnothing, \{13, 20\})$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{a}_3) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \{95\}, \{42\})$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{a}_4) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \varnothing, \{60, 79\})$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{a}_5) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \{12, 95\}, \{94\})$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{x}_1) = \texttt{MPXOR}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{2, 15, 36, 45, 64, 73, 89\}, \varnothing)$
- $(\mathcal{M}, \varnothing, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{x}_2) = \texttt{MPXOR}(\mathcal{M}, \varnothing, \texttt{s}_0, \ldots, \texttt{s}_{127}, \varnothing, \{93\})$
- $\mathcal{M}.var \leftarrow \texttt{z}$ as binary
- $\mathcal{M}.con \leftarrow \texttt{z} = \texttt{x}_1 + \texttt{x}_2 + \sum_{\texttt{i}=1}^{5} \texttt{a}_\texttt{i}$
- **return** $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{z})$

Procedure $\texttt{MPfuncF}(\mathcal{M}, \texttt{s}_0, \ldots, \texttt{s}_{127})$**:**
- $(\mathcal{M}, \varnothing, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{f}) = \texttt{MPXOR}(\mathcal{M}, \varnothing, \texttt{s}_0, \ldots, \texttt{s}_{127}, \varnothing, \{0, 7, 38, 70, 81, 96\})$
- **return** $(\mathcal{M}, \texttt{s}_0, \ldots, \texttt{s}_{127}, \texttt{f})$

Procedure $\texttt{MPfuncG}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127})$**:**
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_1) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{3, 67\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_2) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{11, 13\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_3) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{17, 18\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_4) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{27, 59\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_5) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{40, 48\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_6) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{61, 65\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_7) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{68, 84\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_8) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{88, 92, 93, 95\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_9) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{22, 24, 25\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{a}_{10}) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{70, 78, 82\}, \varnothing)$
- $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \texttt{x}) = \texttt{MPAND}(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \varnothing, \{0, 26, 56, 91, 96\}, \varnothing)$
- $\mathcal{M}.var \leftarrow \texttt{g}$ as binary
- $\mathcal{M}.con \leftarrow \texttt{g} = \texttt{x} + \sum_{\texttt{i}=1}^{10} \texttt{a}_\texttt{i}$
- **return** $(\mathcal{M}, \texttt{b}_0, \ldots, \texttt{b}_{127}, \texttt{g})$

Algorithm 17: MILP Models for monomial trails of the update function in Grain-128AEAD

Procedure MPGrainUpdate($\mathcal{M}, \mathtt{b}_0^{r-1}, \ldots, \mathtt{b}_{127}^{r-1}, \mathtt{s}_0^{r-1}, \ldots, \mathtt{s}_{127}^{r-1}$):
  $(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127}, \mathtt{z}^{r-1}) = \mathtt{MPfuncZ}(\mathcal{M}, \mathtt{b}_0^{r-1}, \ldots, \mathtt{b}_{127}^{r-1}, \mathtt{s}_0^{r-1}, \ldots, \mathtt{s}_{127}^{r-1})$
  $\mathcal{M}.var \leftarrow \mathtt{zg}, \mathtt{zf}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{z}^{r-1} = \mathtt{zg} \vee \mathtt{zf}$
  $(\mathcal{M}, \mathtt{b}''_0, \ldots, \mathtt{b}''_{127}, \mathtt{g}) = \mathtt{MPfuncG}(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127})$
  $(\mathcal{M}, \mathtt{s}''_0, \ldots, \mathtt{s}''_{127}, \mathtt{f}) = \mathtt{MPfuncF}(\mathcal{M}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127})$
  for $i = 0$ *to* 126 **do**
    $b_i^r = b''_{i+1}$
    $s_i^r = s''_{i+1}$
  $\mathcal{M}.var \leftarrow \mathtt{b}_{127}^r, \mathtt{s}_{127}^r$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{b}''_0 = 0$
  $\mathcal{M}.con \leftarrow \mathtt{b}_{127}^r = \mathtt{g} + \mathtt{s}''_0 + \mathtt{zg}$
  $\mathcal{M}.con \leftarrow \mathtt{s}_{127}^r = \mathtt{f} + \mathtt{zf}$
  /* Additional constraint in [12] */
  $\mathcal{M}.con \leftarrow \mathtt{s}_0^{r-1} + \mathtt{z}^{r-1} \leq 1$
  return $(\mathcal{M}, \mathtt{b}_0^r, \ldots, \mathtt{b}_{127}^r, \mathtt{s}_0^r, \ldots, \mathtt{s}_{127}^r)$

H Application to Trivium

Trivium is an NLFSR-based stream cipher designed by De Cannière and Preneel [6]. Its initial state is represented by a 288-bit state $(s_0, s_1, \ldots, s_{287})$. In the initialization phase, the 80-bit key is loaded to the first register and the 80-bit IV is loaded to the second register. The other state bits are set to 0 except the last three bits in the third register. Namely, the initial state bits are given by

$$\begin{aligned}
&(s_0, s_1, \ldots, s_{92}) \leftarrow (K_0, K_1, \ldots, K_{79}, 0, \ldots, 0)\\
&(s_{93}, s_{95}, \ldots, s_{176}) \leftarrow (IV_0, IV_1, \ldots, IV_{79}, 0, \ldots, 0)\\
&(s_{177}, s_{179}, \ldots, s_{287}) \leftarrow (0, \ldots, 0, 1, 1, 1) \ .
\end{aligned}$$

Then from the initial state, the state is updated 1152 times without producing an output. This process can be represented by the following pseudo-code:

for $i = 0$ to 1151 **do**
  $t_1 \leftarrow s_{65} \oplus s_{90} \cdot s_{91} \oplus s_{92} \oplus s_{170}$
  $t_2 \leftarrow s_{161} \oplus s_{174} \cdot s_{175} \oplus s_{176} \oplus s_{263}$
  $t_3 \leftarrow s_{242} \oplus s_{285} \cdot s_{286} \oplus s_{287} \oplus s_{68}$
  $(s_0, s_1, \ldots, s_{92}) \leftarrow (t_3, s_0, s_1, \ldots, s_{91})$
  $(s_{93}, s_{95}, \ldots, s_{176}) \leftarrow (t_1, s_{93}, s_{94}, \ldots, s_{175})$
  $(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (t_2, s_{177}, s_{178} \ldots, s_{286})$
end for

After the initialization, one key stream bit is produced by $z = s_{65} \oplus s_{92} \oplus s_{161} \oplus s_{176} \oplus s_{242} \oplus s_{287}$ during each iteration. This paper studies reduced-round variants of Trivium in which the number of state updates in the initialization is reduced to r.

MILP Model. The MILP model of monomial trails of the update function in Trivium is illustrated as `MPTriviumUpdate` in Algorithm 9. For the term expander, the MILP model `NBDP-MPModelTrivium` is illustrated in Algorithm 18. For the coefficient solver, the MILP model `CMP-MPModelTrivium` is presented in Algorithm 20.

Parameters. For the term expander, we fix $\varepsilon, R_s, N, \tau'$ to $20, 600, 15000, 3000$ respectively. r_c is chosen to be the same value as r_n according to the procedure `ChooseRCiTrivium`, as shown in Algorithm 19. For the coefficient solver, τ is selected in the same way as in [16], as shown in Algorithm 21. r_m is fixed to 90 in `CMP-MPModelTrivium` to ensure all $VT^{(90)}$s can be solved by the monomial prediction.

Superpoly Verification for 843-, 844- and 845-Round Trivium. In [16], five cube indices are chosen heuristically to construct the new cubes, as shown in Table 5 in Sup.Mat. A.2. We verified their results by our new nested framework, but with a much lower time complexity. The verification results are present in Table 1.

Algorithm 18: MILP model for the term expander when applying our nested framework to TRIVIUM

Procedure NBDPTriviumCore($\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_{287}, i_1, i_2, i_3, i_4, i_5$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_1}, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{x}_{\mathtt{i}_1}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_2}, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{x}_{\mathtt{i}_2}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_3}, \mathtt{y}_{\mathtt{i}_4}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{x}_{\mathtt{i}_3}, \mathtt{x}_{\mathtt{i}_4}, \mathbb{V}$)
  if $\mathtt{x}_{\mathtt{i}_5}.F \neq 0_c$ **then**
    Add $\mathtt{x}_{\mathtt{i}_5}$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_5})$ = NBDPXor($\mathcal{M}, \mathbb{V}$)
  for $\mathtt{i} \in \{0, 1, \ldots, 287\}$ *w/o* $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4, \mathtt{i}_5$ **do** $\mathtt{y}_\mathtt{i} = \mathtt{x}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{287})$

Procedure NBDPTriviumUpdate($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}$):
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287})$ = NBDPTriviumCore($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}, 65, 170, 90, 91, 92$)
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287})$ = NBDPTriviumCore($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176$)
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287})$ = NBDPTriviumCore($\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287$)
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{z}_{287}, \mathtt{z}_0, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1})$

Procedure NBDP-MPModelTrivium(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m):
  Declare an empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow \mathtt{s}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 287\}$
  for $\mathtt{i} = 0$ *to* 92 *and* $\mathtt{i} = 93 + 80$ *to* 287 **do** $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 0$
  for $\mathtt{i} = 93$ *to* 172 **do**
    $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 1 \ \forall \ u_{i-93} = 1$
    $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0 \ \forall \ u_{i-93} = 0$
  for $i = 0$ *to* $r_m - 1$ **do**
    $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1})$ = NBDPTriviumUpdate($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}$)
  $\mathcal{M}.var \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$ for $\mathtt{i} \in \{0, 1, \ldots, 287\}$
  for $i = 0$ *to* 287 **do**
    if $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
  for $i = r_m$ *to* $r - 1$ **do**
    $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}+1})$ = MPTriviumUpdate($\mathcal{M}, \mathtt{ss}_0^\mathtt{i}, \ldots, \mathtt{ss}_{287}^\mathtt{i}$)
  for $i = 0$ *to* 287 **do**
    if $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^\mathtt{r} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^\mathtt{r} = 1$
  return $\mathcal{M}$

Algorithm 19: ChooseRCiTrivium

Procedure ChooseRCiTrivium(r_n):
  $r_c = r_n$
  return r_c

Algorithm 20: MILP model for the coefficient solver when applying our nested framework to TRIVIUM

Procedure `CMPTriviumCore`$(\mathcal{M}, \mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_{287}, i_1, i_2, i_3, i_4, i_5)$**:**
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_1}, \mathbb{V}) =$ `CMPAnd1`$(\mathcal{M}, \mathtt{x}_{\mathtt{i}_1}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_2}, \mathbb{V}) =$ `CMPAnd1`$(\mathcal{M}, \mathtt{x}_{\mathtt{i}_2}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_3}, \mathtt{y}_{\mathtt{i}_4}, \mathbb{V}) =$ `CMPAnd2`$(\mathcal{M}, \mathtt{x}_{\mathtt{i}_3}, \mathtt{x}_{\mathtt{i}_4}, \mathbb{V})$
  if $\mathtt{x}_{\mathtt{i}_5}.F \neq 0_c$ **then**
    Add $\mathtt{x}_{\mathtt{i}_5}$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{y}_{\mathtt{i}_5}) =$ `CMPXor`$(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 287\}$ *w/o* $\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3, \mathtt{i}_4, \mathtt{i}_5$ **do** $\mathtt{y}_\mathtt{i} = \mathtt{x}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{287})$

Procedure `CMPTriviumUpdate`$(\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i})$**:**
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}) =$ `CMPTriviumCore`$(\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i}, 65, 170, 90, 91, 92)$
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}) =$ `CMPTriviumCore`$(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176)$
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287}) =$ `CMPTriviumCore`$(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287)$
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{z}_{287}, \mathtt{z}_0, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1})$

Procedure `CMP-MPModelTrivium`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m**):**
  Declare an empty MILP Model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow \mathtt{s}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 287\}$
  for $\mathtt{i} = 0$ *to* 92 *and* $\mathtt{i} = 93 + 80$ *to* 287 **do** $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 0$
  for $\mathtt{i} = 93$ *to* 172 **do**
    $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 1 \ \forall\ u_{i-93} = 1$
    $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0 \ \forall\ u_{i-93} = 0$
  for $i = 0$ *to* $r_m - 1$ **do**
    $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) =$ `CMPTriviumUpdate`$(\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{287}^\mathtt{i})$
  $\mathcal{M}.var \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$ for $\mathtt{i} \in \{0, 1, \ldots, 287\}$
  for $i = 0$ *to* 287 **do**
    if $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} = 0$
    else if $s_i^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
    else $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
  for $i = r_m$ *to* $r - 1$ **do**
    $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}+1}) =$ `MPTriviumUpdate`$(\mathcal{M}, \mathtt{ss}_0^\mathtt{i}, \ldots, \mathtt{ss}_{287}^\mathtt{i})$
  for $i = 0$ *to* 287 **do**
    if $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^\mathtt{r} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^\mathtt{r} = 1$
  return $\mathcal{M}$

Algorithm 21: `ChooseTiTrivium`

Procedure `ChooseTiTrivium`(r_n)**:**
  if $r_n \geq 600$ **then** $\tau = 40$ seconds
  else if $r_n \geq 500$ **then** $\tau = 80$ seconds
  else if $r_n \geq 400$ **then** $\tau = 160$ seconds
  else if $r_n \geq 300$ **then** $\tau = 320$ seconds
  else if $r_n \geq 200$ **then** $\tau = 640$ seconds
  else if $r_n \geq 100$ **then** $\tau = 1200$ seconds
  else if $r_n \geq 20$ **then** $\tau = 3600$ seconds
  else if $r_n \geq 0$ **then** $\tau = \infty$
  return τ

I Application to Grain-128AEAD

Grain-128AEAD [15] is an authenticated encryption algorithm and also one of the ten finalist candidates of the NIST LWC standardization process. The design of Grain-128AEAD inherits from Grain-128a, which was proposed in 2011 [4]. Hereinafter, we follow the assumption in [12,13] that the first bit of the pre-output key stream can be observed, so there is no difference between Grain-128a and Grain-128AEAD under this assumption.

The initial state of Grain-128AEAD can be represented by two 128-bit states, $(b_0, b_1, ..., b_{127})$ and $(s_0, s_1, \ldots, s_{127})$. The 128-bit key is loaded to the first register $\boldsymbol{b}$ and the 96-bit initialization vector is loaded to the second register $\boldsymbol{s}$. The other state bits of $\boldsymbol{s}$ are set to 1 except the least one bit. Namely, the initial state bits are represented as

$$\begin{aligned}(b_0, b_1, \ldots, b_{127}) &= (K_0, K_1, \ldots, K_{127}),\\ (s_0, s_1, \ldots, s_{127}) &= (N_0, N_1, \ldots, N_{95}, 1, \ldots, 1, 0).\end{aligned}$$

The pseudo code of the update function in the initialization is given as follows.

$$\begin{aligned}
g &\leftarrow b_0 \oplus b_{26} \oplus b_{56} \oplus b_{91} \oplus b_{96} \oplus b_3 b_{67} \oplus b_{11} b_{13} \oplus b_{17} b_{18} \oplus b_{27} b_{59} \oplus b_{40} b_{48}\\
&\quad \oplus b_{61} b_{65} \oplus b_{68} b_{84} \oplus b_{88} b_{92} b_{93} b_{95} \oplus b_{22} b_{24} b_{25} \oplus b_{70} b_{78} b_{82},\\
f &\leftarrow s_0 \oplus s_7 \oplus s_{38} \oplus s_{70} \oplus s_{81} \oplus s_{96},\\
h &\leftarrow b_{12} s_8 \oplus s_{13} s_{20} \oplus b_{95} s_{42} \oplus s_{60} s_{79} \oplus b_{12} b_{95} s_{94},\\
z &\leftarrow h \oplus s_{93} \oplus b_2 \oplus b_{15} \oplus b_{36} \oplus b_{45} \oplus b_{64} \oplus b_{73} \oplus b_{89},\\
(b_0, b_1, \ldots, b_{127}) &\leftarrow (b_1, \ldots, b_{127}, g \oplus s_0 \oplus z),\\
(s_0, s_1, \ldots, s_{127}) &\leftarrow (s_1, \ldots, s_{127}, f \oplus z).
\end{aligned}$$

In the initialization phase, the 256-bit state is updated 256 times without producing an output. After the initialziation, z is used as a pre-output key instead of being fed to the state. We study the variant of Grain-128AEAD whose initialization phase is reduced to r rounds.

MILP Model. The MILP model for monomial trails of the update function is illustrated as `MPGrainUpdate` in Algorithm 17, whose supporting functions such as `MPfuncZ`, `MPfuncG` and `MPfuncF` are directly borrowed from [12,13] and shown in Algorithm 16. For the term expander, the MILP model `NBDP-MPModelGrain` presented in Algorithm 24 corresponds to Eqn. (5), which is built based on `MPGrainUpdate` and `NBDPGrainUpdate` in Algorithm 23. The MILP models of supporting functions are provided in Algorithm 22. For the coefficient solver, the MILP model `CMP-MPGrainUpdate` in Algorithm 28 corresponds to Eqn. (7). The MILP model of core monomial trails of the update function is shown in Algorithm 27.

Parameters. For the term expander, we fix $\varepsilon, R_s, N, \tau'$ to $1, 131, 15000, 3600$ respectively. r_c is chosen to be the same value as r_n according to the procedure `ChooseRCiGrain`, as shown in Algorithm 25. For the coefficient solver, the choice

of τ is shown in Algorithm 29. r_m is fixed to 40 in `CMP-MPModelGrain` to ensure the second stage of the coefficient solver can be completed quickly.

Superpoly verification for 191-Round Grain-128AEAD. In [16], the superpolies of two cubes indexed by $\{0, 1, 2, \ldots, 95\}$ and $\{0, 1, 2, \ldots, 95\}\backslash\{30\}$ are recovered for Grain-128AEAD reduced to 191 rounds. We verified their results with our new framework and it took us about 3 days.

Algorithm 22: MILP Models for division trails of NBDP of the NFSR and LFSR in Grain-128AEAD

Procedure NBDPfuncZ($\mathcal{M}$, $\mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{b}_{12}, \mathtt{s}_8, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{12}, \mathtt{s}_8, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{s}_{13}, \mathtt{s}_{20}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{s}_{13}, \mathtt{s}_{20}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{95}, \mathtt{s}_{42}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{95}, \mathtt{s}_{42}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{s}_{60}, \mathtt{s}_{79}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{s}_{60}, \mathtt{s}_{79}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{12}, \mathtt{b}_{95}, \mathtt{s}_{94}, \mathbb{V})$ = NBDPAnd3($\mathcal{M}, \mathtt{b}_{12}, \mathtt{b}_{95}, \mathtt{s}_{94}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{s}_{93}, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{s}_{93}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_i, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{b}_i, \mathbb{V}$) $\forall \mathtt{i} \in \{2, 15, 36, 45, 64, 73, 89\}$
  $(\mathcal{M}, \mathtt{z})$ = NBDPXor($\mathcal{M}, \mathbb{V}$)
  return $(\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}, \mathtt{z})$
Procedure NBDPfuncF($\mathcal{M}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{s}_i, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{s}_i, \mathbb{V}$) $\forall \mathtt{i} \in \{0, 7, 38, 70, 81, 96\}$
  $(\mathcal{M}, \mathtt{f})$ = NBDPXor($\mathcal{M}, \mathbb{V}$)
  return $(\mathcal{M}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}, \mathtt{f})$
Procedure NBDPfuncG($\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{b}_3, \mathtt{b}_{67}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_3, \mathtt{b}_{67}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{11}, \mathtt{b}_{13}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{11}, \mathtt{b}_{13}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{17}, \mathtt{b}_{18}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{17}, \mathtt{b}_{18}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{27}, \mathtt{b}_{59}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{27}, \mathtt{b}_{59}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{40}, \mathtt{b}_{48}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{40}, \mathtt{b}_{48}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{61}, \mathtt{b}_{65}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{61}, \mathtt{b}_{65}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{68}, \mathtt{b}_{84}, \mathbb{V})$ = NBDPAnd2($\mathcal{M}, \mathtt{b}_{68}, \mathtt{b}_{84}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{88}, \mathtt{b}_{92}, \mathtt{b}_{93}, \mathtt{b}_{95}, \mathbb{V})$ = NBDPAnd4($\mathcal{M}, \mathtt{b}_{88}, \mathtt{b}_{92}, \mathtt{b}_{93}, \mathtt{b}_{95}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{22}, \mathtt{b}_{24}, \mathtt{b}_{25}, \mathbb{V})$ = NBDPAnd3($\mathcal{M}, \mathtt{b}_{22}, \mathtt{b}_{24}, \mathtt{b}_{25}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_{70}, \mathtt{b}_{78}, \mathtt{b}_{82}, \mathbb{V})$ = NBDPAnd3($\mathcal{M}, \mathtt{b}_{70}, \mathtt{b}_{78}, \mathtt{b}_{82}, \mathbb{V}$)
  $(\mathcal{M}, \mathtt{b}_i, \mathbb{V})$ = NBDPAnd1($\mathcal{M}, \mathtt{b}_i, \mathbb{V}$) $\forall \mathtt{i} \in \{0, 26, 56, 91, 96\}$
  $(\mathcal{M}, \mathtt{g})$ = NBDPXor($\mathcal{M}, \mathbb{V}$)
  return $(\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{g})$

Algorithm 23: MILP Models for division trails of NBDP of the update function in Grain-128AEAD

Procedure NBDPGrainUpdate($\mathcal{M}, \mathtt{b}_0^{\mathtt{r}-1}, \ldots, \mathtt{b}_{127}^{\mathtt{r}-1}, \mathtt{s}_0^{\mathtt{r}-1}, \ldots, \mathtt{s}_{127}^{\mathtt{r}-1}$):

- $(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127}, \mathtt{z}^{\mathtt{r}-1}) = \mathtt{NBDPfuncZ}(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}-1}, \ldots, \mathtt{b}_{127}^{\mathtt{r}-1}, \mathtt{s}_0^{\mathtt{r}-1}, \ldots, \mathtt{s}_{127}^{\mathtt{r}-1})$
- $\mathcal{M}.var \leftarrow \mathtt{zg}, \mathtt{zf}$ as binary
- $\mathcal{M}.con \leftarrow \mathtt{z}^{\mathtt{r}-1} = \mathtt{zg} + \mathtt{zf}$
- $(\mathcal{M}, \mathtt{b}''_0, \ldots, \mathtt{b}''_{127}, \mathtt{g}) = \mathtt{NBDPfuncG}(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127})$
- $(\mathcal{M}, \mathtt{s}''_0, \ldots, \mathtt{s}''_{127}, \mathtt{f}) = \mathtt{NBDPfuncF}(\mathcal{M}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127})$
- **for** $i = 0$ *to* 126 **do**
  - $b_i^r = b''_{i+1}$
  - $s_i^r = s''_{i+1}$
- $\mathcal{M}.var \leftarrow \mathtt{b}_{127}^{\mathtt{r}}, \mathtt{s}_{127}^{\mathtt{r}}$ as binary
- $\mathcal{M}.con \leftarrow \mathtt{b}''_0 = 0$
- $\mathcal{M}.con \leftarrow \mathtt{b}_{127}^{\mathtt{r}} = \mathtt{g} + \mathtt{s}''_0 + \mathtt{zg}$
- $\mathcal{M}.con \leftarrow \mathtt{s}_{127}^{\mathtt{r}} = \mathtt{f} + \mathtt{zf}$
- /* Additional constraint in [12] */
- $\mathcal{M}.con \leftarrow \mathtt{s}_0^{\mathtt{r}-1} + \mathtt{z}^{\mathtt{r}-1} \leq 1$
- **return** $(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}}, \ldots, \mathtt{b}_{127}^{\mathtt{r}}, \mathtt{s}_0^{\mathtt{r}}, \ldots, \mathtt{s}_{127}^{\mathtt{r}})$

Algorithm 24: MILP model for the term expander when applying our nested framework to Grain-128AEAD

Procedure NBDP-MPModelGrain(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m):

- Declare a empty MILP Model $\mathcal{M}$
- $\mathcal{M}.var \leftarrow \mathtt{b}_{\mathtt{i}}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.var \leftarrow \mathtt{s}_{\mathtt{i}}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- **for** $\mathtt{i} = 96$ *to* 127 **do** $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^0 = 0$
- **for** $\mathtt{i} = 0$ *to* 127 **do** $\mathcal{M}.con \leftarrow \mathtt{b}_{\mathtt{i}}^0 = 0$
- **for** $\mathtt{i} = 0$ *to* 95 **do**
  - $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 1 \ \forall \ u_i = 1$
  - $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 0 \ \forall \ u_i = 0$
- **for** $r = 0$ *to* $r_m - 1$ **do**
  - $(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}+1}, \ldots, \mathtt{b}_{127}^{\mathtt{r}+1}, \mathtt{s}_0^{\mathtt{r}+1}, \ldots, \mathtt{s}_{127}^{\mathtt{r}+1}) = \mathtt{NBDPGrainUpdate}(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}}, \ldots, \mathtt{b}_{127}^{\mathtt{r}}, \mathtt{s}_0^{\mathtt{r}}, \ldots, \mathtt{s}_{127}^{\mathtt{r}})$
- $\mathcal{M}.var \leftarrow \mathtt{bb}_{\mathtt{i}}^{\mathtt{r_m}}$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.var \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- /* $\boldsymbol{s}^{(r_m)} = (s_0^{(r_m)}, \ldots, s_{256}^{(r_m)})$ denotes the concatenation of $\boldsymbol{b}^{(s_m)}$ and $\boldsymbol{s}^{(r_m)}$ */
- **for** $i = 0$ *to* 127 **do**
  - **if** $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{bb}_{\mathtt{i}}^{\mathtt{r_m}} = \mathtt{b}_{\mathtt{i}}^{\mathtt{r_m}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{bb}_{\mathtt{i}}^{\mathtt{r_m}} \geq \mathtt{b}_{\mathtt{i}}^{\mathtt{r_m}}$
  - **if** $s_{i+128}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}} = \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}} \geq \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}}$
- **for** $r = r_m$ *to* $r - 1$ **do**
  - $(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}+1}, \ldots, \mathtt{b}_{127}^{\mathtt{r}+1}, \mathtt{s}_0^{\mathtt{r}+1}, \ldots, \mathtt{s}_{127}^{\mathtt{r}+1}) = \mathtt{MPGrainUpdate}(\mathcal{M}, \mathtt{b}_0^{\mathtt{r}}, \ldots, \mathtt{b}_{127}^{\mathtt{r}}, \mathtt{s}_0^{\mathtt{r}}, \ldots, \mathtt{s}_{127}^{\mathtt{r}})$
- /* $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{256}^{(r)})$ corresponds to $\mathtt{bb}^{\mathtt{r}}$ and $\mathtt{ss}^{\mathtt{r}}$ */
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{bb}_{\mathtt{i}}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{bb}_{\mathtt{i}}^{\mathtt{r_m}} = 1$
  - **if** $t_{i+128}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}} = 1$
- **return** $\mathcal{M}$

Algorithm 25: `ChooseRCiGrain`

Procedure `ChooseRCiGrain`(r_n):
  $r_c = r_n$
  return r_c

Algorithm 26: MILP Models for core monomial trails of the NFSR and LFSR in Grain-128AEAD

Procedure `CMPfuncZ`($\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{b}_{12}, \mathtt{s}_8, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{12}, \mathtt{s}_8, \mathbb{V})$
  $(\mathcal{M}, \mathtt{s}_{13}, \mathtt{s}_{20}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{s}_{13}, \mathtt{s}_{20}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{95}, \mathtt{s}_{42}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{95}, \mathtt{s}_{42}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{s}_{60}, \mathtt{s}_{79}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{s}_{60}, \mathtt{s}_{79}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{12}, \mathtt{b}_{95}, \mathtt{s}_{94}, \mathbb{V}) = \mathtt{CMPAnd3}(\mathcal{M}, \mathtt{b}_{12}, \mathtt{b}_{95}, \mathtt{s}_{94}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{s}_{93}, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{s}_{93}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_\mathtt{i}, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{b}_\mathtt{i}, \mathbb{V})\ \forall \mathtt{i} \in \{2, 15, 36, 45, 64, 73, 89\}$
  $(\mathcal{M}, \mathtt{z}) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  return $(\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}, \mathtt{z})$
Procedure `CMPfuncF`($\mathcal{M}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{s}_\mathtt{i}, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{s}_\mathtt{i}, \mathbb{V})\ \forall \mathtt{i} \in \{0, 7, 38, 70, 81, 96\}$
  $(\mathcal{M}, \mathtt{f}) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  return $(\mathcal{M}, \mathtt{s}_0, \ldots, \mathtt{s}_{127}, \mathtt{f})$
Procedure `CMPfuncG`($\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}$):
  Initialize a empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{b}_3, \mathtt{b}_{67}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_3, \mathtt{b}_{67}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{11}, \mathtt{b}_{13}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{11}, \mathtt{b}_{13}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{17}, \mathtt{b}_{18}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{17}, \mathtt{b}_{18}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{27}, \mathtt{b}_{59}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{27}, \mathtt{b}_{59}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{40}, \mathtt{b}_{48}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{40}, \mathtt{b}_{48}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{61}, \mathtt{b}_{65}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{61}, \mathtt{b}_{65}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{68}, \mathtt{b}_{84}, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{b}_{68}, \mathtt{b}_{84}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{88}, \mathtt{b}_{92}, \mathtt{b}_{93}, \mathtt{b}_{95}, \mathbb{V}) = \mathtt{CMPAnd4}(\mathcal{M}, \mathtt{b}_{88}, \mathtt{b}_{92}, \mathtt{b}_{93}, \mathtt{b}_{95}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{22}, \mathtt{b}_{24}, \mathtt{b}_{25}, \mathbb{V}) = \mathtt{CMPAnd3}(\mathcal{M}, \mathtt{b}_{22}, \mathtt{b}_{24}, \mathtt{b}_{25}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_{70}, \mathtt{b}_{78}, \mathtt{b}_{82}, \mathbb{V}) = \mathtt{CMPAnd3}(\mathcal{M}, \mathtt{b}_{70}, \mathtt{b}_{78}, \mathtt{b}_{82}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{b}_\mathtt{i}, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{b}_\mathtt{i}, \mathbb{V})\ \forall \mathtt{i} \in \{0, 26, 56, 91, 96\}$
  $(\mathcal{M}, \mathtt{g}) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  return $(\mathcal{M}, \mathtt{b}_0, \ldots, \mathtt{b}_{127}, \mathtt{g})$

Algorithm 27: MILP Models for core monomial trails of the update function in Grain-128AEAD

Procedure `CMPGrainUpdate`($\mathcal{M}, \mathtt{b}_0^{r-1}, \ldots, \mathtt{b}_{127}^{r-1}, \mathtt{s}_0^{r-1}, \ldots, \mathtt{s}_{127}^{r-1}$):
  $(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127}, \mathtt{z}^{r-1}) = \mathtt{CMPfuncZ}(\mathcal{M}, \mathtt{b}_0^{r-1}, \ldots, \mathtt{b}_{127}^{r-1}, \mathtt{s}_0^{r-1}, \ldots, \mathtt{s}_{127}^{r-1})$
  $\mathcal{M}.var \leftarrow \mathtt{zg}, \mathtt{zf}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{z}^{r-1} = \mathtt{zg} \vee \mathtt{zf}$
  $(\mathcal{M}, \mathtt{b}''_0, \ldots, \mathtt{b}''_{127}, \mathtt{g}) = \mathtt{CMPfuncG}(\mathcal{M}, \mathtt{b}'_0, \ldots, \mathtt{b}'_{127})$
  $(\mathcal{M}, \mathtt{s}''_0, \ldots, \mathtt{s}''_{127}, \mathtt{f}) = \mathtt{CMPfuncF}(\mathcal{M}, \mathtt{s}'_0, \ldots, \mathtt{s}'_{127})$
  for $i = 0$ *to* 126 **do**
    $b_i^r = b''_{i+1}$
    $s_i^r = s''_{i+1}$
  $\mathcal{M}.var \leftarrow \mathtt{b}_{127}^r, \mathtt{s}_{127}^r$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{b}''_0 = 0$
  $\mathcal{M}.con \leftarrow \mathtt{b}_{127}^r = \mathtt{g} + \mathtt{s}''_0 + \mathtt{zg}$
  $\mathcal{M}.con \leftarrow \mathtt{s}_{127}^r = \mathtt{f} + \mathtt{zf}$
  /* Additional constraint in [12] */
  $\mathcal{M}.con \leftarrow \mathtt{s}_0^{r-1} + \mathtt{z}^{r-1} \leq 1$
  return $(\mathcal{M}, \mathtt{b}_0^r, \ldots, \mathtt{b}_{127}^r, \mathtt{s}_0^r, \ldots, \mathtt{s}_{127}^r)$

Algorithm 28: MILP model for the coefficient solver when applying our nested framework to Grain-128AEAD

Procedure `CMP-MPModelGrain`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m`)`:

- Declare a empty MILP Model $\mathcal{M}$
- $\mathcal{M}.var \leftarrow \mathtt{b}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.var \leftarrow \mathtt{s}_\mathtt{i}^0$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- **for** $\mathtt{i} = 96$ *to* 127 **do** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0$
- **for** $\mathtt{i} = 0$ *to* 127 **do** $\mathcal{M}.con \leftarrow \mathtt{b}_\mathtt{i}^0 = 0$
- **for** $\mathtt{i} = 0$ *to* 95 **do**
  - $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 1 \ \forall \ u_i = 1$
  - $\mathcal{M}.con \leftarrow \mathtt{s}_i^0 = 0 \ \forall \ u_i = 0$
- **for** $r = 0$ *to* $r_m - 1$ **do**
  - $(\mathcal{M}, \mathtt{b}_0^{r+1}, \ldots, \mathtt{b}_{127}^{r+1}, \mathtt{s}_0^{r+1}, \ldots, \mathtt{s}_{127}^{r+1}) = \mathtt{CMPGrainUpdate}(\mathcal{M}, \mathtt{b}_0^r, \ldots, \mathtt{b}_{127}^r, \mathtt{s}_0^r, \ldots, \mathtt{s}_{127}^r)$
- $\mathcal{M}.var \leftarrow \mathtt{bb}_\mathtt{i}^{r_m}$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.var \leftarrow \mathtt{ss}_\mathtt{i}^{r_m}$ for $\mathtt{i} \in \{0, 1, \ldots, 127\}$ as binary

/* $\boldsymbol{s}^{(r_m)} = (s_0^{(r_m)}, \ldots, s_{256}^{(r_m)})$ denotes the concatenation of $\boldsymbol{b}^{(s_m)}$ and $\boldsymbol{s}^{(r_m)}$ */

- **for** $i = 0$ *to* 127 **do**
  - **if** $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{bb}_\mathtt{i}^{r_m} = \mathtt{b}_\mathtt{i}^{r_m} = 0$
  - **else if** $s_i^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{bb}_\mathtt{i}^{r_m} \geq \mathtt{b}_\mathtt{i}^{r_m}$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{bb}_\mathtt{i}^{r_m} = \mathtt{b}_\mathtt{i}^{r_m}$
  - **if** $s_{i+128}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{r_m} = \mathtt{s}_\mathtt{i}^{r_m} = 0$
  - **else if** $s_{i+128}^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{r_m} \geq \mathtt{s}_\mathtt{i}^{r_m}$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{r_m} = \mathtt{s}_\mathtt{i}^{r_m}$
- **for** $r = r_m$ *to* $r - 1$ **do**
  - $(\mathcal{M}, \mathtt{b}_0^{r+1}, \ldots, \mathtt{b}_{127}^{r+1}, \mathtt{s}_0^{r+1}, \ldots, \mathtt{s}_{127}^{r+1}) = \mathtt{MPGrainUpdate}(\mathcal{M}, \mathtt{b}_0^r, \ldots, \mathtt{b}_{127}^r, \mathtt{s}_0^r, \ldots, \mathtt{s}_{127}^r)$

/* $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{256}^{(r)})$ corresponds to $\mathtt{bb}^r$ and $\mathtt{ss}^r$ */

- **for** $i = 0$ *to* 127 **do**
  - **if** $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{bb}_\mathtt{i}^r = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{bb}_\mathtt{i}^{r_m} = 1$
  - **if** $t_{i+128}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^r = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{r_m} = 1$
- **return** $\mathcal{M}$

Algorithm 29: `ChooseTiGrain`

Procedure `ChooseTiGrain`(r_n)**:**

- **if** $r_n \geq 120$ **then** $\tau = 60$ seconds
- **else if** $r_n \geq 110$ **then** $\tau = 120$ seconds
- **else if** $r_n \geq 100$ **then** $\tau = 180$ seconds
- **else if** $r_n \geq 10$ **then** $\tau = 360$ seconds
- **else if** $r_n \geq 0$ **then** $\tau = \infty$
- **return** τ

J Application to Kreyvium

Specification of Kreyvium. Kreyvium is a stream cipher designed for the use of Fully Homomorphic Encryption [7]. Kreyvium accepts a 128-bit IV and supports 128-bit security with an internal structure similar to Trivium. Kreyvium is composed of 5 registers. Two of them are LFSRs, denoted by K^* and IV^*, respectively. The remaining three ones are NFSRs that are identical to those of Trivium. The initial state of Kreyvium is set as

$$\begin{aligned}
&(s_0, s_1, \ldots, s_{92}) \leftarrow (K_0, K_1, \ldots, K_{92}) \\
&(s_{93}, s_{95}, \ldots, s_{176} \leftarrow (IV_0, IV_1, \ldots, IV_{83}) \\
&(s_{177}, s_{179}, \ldots, s_{287}) \leftarrow (IV_{85}, \ldots, IV_{127}, 1, \ldots, 1, 0) \\
&(IV^*_{127}, \ldots, IV^*_0) \leftarrow (IV_0, \ldots, IV_{127}) \\
&(K^*_{127}, \ldots, K^*_0) \leftarrow (K_0, \ldots, K_{127}) \ .
\end{aligned}$$

Then, the state is updated over 1152 rounds using the following update function:

for $i = 0$ to 1151 **do**

$t_1 \leftarrow s_{65} \oplus s_{92}, \quad t_2 \leftarrow s_{161} \oplus s_{176}, \quad t_3 \leftarrow s_{242} \oplus s_{287} \oplus K^*_0$

$z_i \leftarrow t_1 \oplus t_2 \oplus t_3$

$t_1 \leftarrow t_1 \oplus s_{90}s_{91} \oplus s_{170} \oplus IV^*_0$

$t_2 \leftarrow t_2 \oplus s_{174}s_{175} \oplus s_{263}$

$t_3 \leftarrow t_3 \oplus s_{285}s_{286} \oplus s_{68}$

$t_4 \leftarrow K^*_0, \ t_5 \leftarrow IV^*_0$

$(s_0, s_1, \ldots, s_{92}) \leftarrow (t_3, s_0, s_1, \ldots, s_{91})$

$(s_{92}, s_{93}, \ldots, s_{176}) \leftarrow (t_1, s_{93}, s_{94}, \ldots, s_{175})$

$(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (t_2, s_{177}, s_{178}, \ldots, s_{286})$

$(K^*_{127}, K^*_{126}, \ldots, K^*_0) \leftarrow (t_4, K^*_{127}, K^*_{126}, \ldots, K^*_1)$

$(IV^*_{127}, IV^*_{126}, \ldots, IV^*_0) \leftarrow (t_5, IV^*_{127}, IV^*_{126}, \ldots, IV^*_1)$

end for

After the initialization phase, the key stream bit z_i ($i \geq 1152$) is output. We analyze the variant of Kreyvium whose initialization phase is reduced to r rounds, where the key stream bit is denoted by z_r.

MILP Model. The MILP model of monomial trails of the update function in Kreyvium is illustrated as `MPKreyviumUpdate` in Algorithm 10. `NBDPKreyviumUpdate` in Algorithm 30 and `CMPKreyviumUpdate` in Algorithm 31 model the propagation of NBDP and CMP for the update function of Kreyvium respectively. They are embedded as subroutines when introducing the MILP models for the term expander and coefficient solver in Algorithm 32 and Algorithm 33.

Parameters. For the term expander, we fix $\varepsilon, R_s, N, \tau'$ to $20, 420, 10000, 14400$, respectively. r_c is chosen to be the same value as r_n according to the procedure `ChooseRCiKreyvium` in Algorithm 34. For the coefficient solver, the procedure `ChooseTiKreyvium` for selecting the time limit is presented in Algorithm 35.

In `CMP-MPModelKreyvium`, r_m is fixed to 120 to ensure the second stage of the coefficient solver can be completed quicly.

Superpoly verification for 894-Round Kreyvium. In [16], the superpoly of a 119-dimensional cube indexed by

$$I = \{0, 1, \ldots, 127\} \backslash \{6, 66, 72, 73, 78, 101, 106, 109, 110\}$$

is recovered for 894-round Kreyvium. We verified this result by our nested framework, which took about two weeks.

Algorithm 30: MILP model for division trails of NBDP of the update function in Kreyvium

Procedure NBDPLSFR($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{127}$):
  $\mathcal{M}.var \leftarrow \mathtt{b}$ as binary
  if $x_0.F \neq 0_c$ **then**
    $\mathcal{M}.var \leftarrow \mathtt{a}$ as binary
    $\mathcal{M}.con \leftarrow \mathtt{x}_0 = \mathtt{a} + \mathtt{b}$
    $(\mathtt{y}_0, \ldots, \mathtt{y}_{127}) = (\mathtt{x}_1, \ldots, \mathtt{x}_{127}, \mathtt{a})$
  else
    $(\mathtt{y}_0, \ldots, \mathtt{y}_{127}) = (\mathtt{x}_1, \ldots, \mathtt{x}_{127}, \mathtt{x}_0)$
    $\mathcal{M}.con \leftarrow \mathtt{b} = 0$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{127}, \mathtt{b})$
Procedure NBDPKreyviumUpdate($\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}}$):
  $(\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{a}^{\mathtt{i}}) \leftarrow$ NBDPLSFR$(\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}})$
  $(\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1}, \mathtt{b}^{\mathtt{i}}) \leftarrow$ NBDPLSFR$(\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}})$
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287})$ = NBDPTriviumCore$(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, 65, 170, 90, 91, 92)$
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287})$ = NBDPTriviumCore$(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176)$
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287})$ = NBDPTriviumCore$(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287)$
  $\mathcal{M}.var \leftarrow \mathtt{t}_1^{\mathtt{i}}, \mathtt{t}_3^{\mathtt{i}}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{t}_1^{\mathtt{i}} = \mathtt{z}_{92} + \mathtt{b}^{\mathtt{i}}$
  $\mathcal{M}.con \leftarrow \mathtt{t}_3^{\mathtt{i}} = \mathtt{z}_{287} + \mathtt{a}^{\mathtt{i}}$
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{t}_3^{\mathtt{i}}, \mathtt{z}_0, \ldots, \mathtt{z}_{91}, \mathtt{t}_1^{\mathtt{i}}, \mathtt{z}_{93}, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1})$

Algorithm 31: MILP model for core monomial trails of the update function in Kreyvium

Procedure CMPLSFR($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{127}$):
  $\mathcal{M}.var \leftarrow \mathtt{b}$ as binary
  if $x_0.F \neq 0_c$ **then**
    $\mathcal{M}.var \leftarrow \mathtt{a}$ as binary
    $\mathcal{M}.con \leftarrow \mathtt{x}_0 = \mathtt{a} \vee \mathtt{b}$
    $(\mathtt{y}_0, \ldots, \mathtt{y}_{127}) = (\mathtt{x}_1, \ldots, \mathtt{x}_{127}, \mathtt{a})$
  else
    $(\mathtt{y}_0, \ldots, \mathtt{y}_{127}) = (\mathtt{x}_1, \ldots, \mathtt{x}_{127}, \mathtt{x}_0)$
    $\mathcal{M}.con \leftarrow \mathtt{b} = 0$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{127}, \mathtt{b})$

Procedure CMPKreyviumUpdate($\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}}$):
  $(\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{a}^{\mathtt{i}}) \leftarrow$ CMPLSFR$(\mathcal{M}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}})$
  $(\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1}, \mathtt{b}^{\mathtt{i}}) \leftarrow$ CMPLSFR$(\mathcal{M}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}})$
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}) =$ CMPTriviumCore$(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, 65, 170, 90, 91, 92)$
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287})$ = CMPTriviumCore$(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{287}, 161, 263, 174, 175, 176)$
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{287})$ = CMPTriviumCore$(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{287}, 242, 68, 285, 286, 287)$
  $\mathcal{M}.var \leftarrow \mathtt{t}_1^{\mathtt{i}}, \mathtt{t}_3^{\mathtt{i}}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{t}_1^{\mathtt{i}} = \mathtt{z}_{92} + \mathtt{b}^{\mathtt{i}}$
  $\mathcal{M}.con \leftarrow \mathtt{t}_3^{\mathtt{i}} = \mathtt{z}_{287} + \mathtt{a}^{\mathtt{i}}$
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = (\mathtt{t}_3^{\mathtt{i}}, \mathtt{z}_0, \ldots, \mathtt{z}_{91}, \mathtt{t}_1^{\mathtt{i}}, \mathtt{z}_{93}, \ldots, \mathtt{z}_{286})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1})$

Algorithm 32: MILP model for the term expander when applying our nested framework to Kreyvium

Procedure NBDP-MPModelKreyvium(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m**):**

- Declare an empty MILP Model $\mathcal{M}$
- $\mathcal{M} \leftarrow \mathtt{s}_\mathtt{i}^0 \ \forall\ i \in \{0, 1, \ldots, 287\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{x}_\mathtt{i} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{k}_\mathtt{i} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- **for** $\mathtt{i} = 0$ *to* 127 **do**
  - $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i}^0 = 1 \ \forall\ u_i = 1$
  - $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i}^0 = 0 \ \forall\ u_i = 0$
  - $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 0$
- $\mathcal{M} \leftarrow \mathtt{K}_\mathtt{i}^{*,0} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{IV}_\mathtt{i}^{*,0} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.con \leftarrow \mathtt{K}_\mathtt{i}^{*,0} = 0$ for $i = 0, \ldots, 127$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0$ for $i = 0, \ldots, 92$ and $i = 93 + 128, \ldots, 287$
- $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i} = \mathtt{IV}_{128-\mathtt{i}}^{*,0} + \mathtt{s}_{93+\mathtt{i}}^0$ for $i = 0, \ldots, 127$
- **for** $i = 0$ *to* $r_m - 1$ **do**
  - $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1}) =$ NBDPKreyviumUpdate$(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}})$
- $\mathcal{M} \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 287\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- /* $\boldsymbol{s}^{(r_m)} = (s_0^{(r_m)}, \ldots, s_{543}^{(r_m)})$ denotes the concatenation of r_m-round state, IV^{*,r_m} and K^{*,r_m} */
- **for** $i = 0$ *to* 127 **do**
  - **if** $s_{i+128}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{IV}_\mathtt{i}^{*,\mathtt{r_m}} = \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{IV}_\mathtt{i}^{*,\mathtt{r_m}} = \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}}$
- **for** $i = 0$ *to* 287 **do**
  - **if** $s_{i+256}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
- **for** $i = r_m$ *to* $r - 1$ **do**
  - $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}+1}, \mathtt{KK}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{KK}_{127}^{*,\mathtt{i}+1}, \mathtt{VV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{VV}_{127}^{*,\mathtt{i}+1}) =$ MPKreyviumUpdate$(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}}, \mathtt{KK}_0^{*,\mathtt{i}}, \ldots, \mathtt{KK}_{127}^{*,\mathtt{i}}, \mathtt{VV}_0^{*,\mathtt{i}}, \ldots, \mathtt{VV}_{127}^{*,\mathtt{i}})$
- /* $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{543}^{(r)})$ corresponds to $\mathtt{KK}^{*,\mathtt{r}}, \mathtt{VV}^{*,\mathtt{r}}, \mathtt{ss}^{\mathtt{r}}$ */
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r}} = 1$
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_{i+128}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r}} = 1$
- **for** $i = 0$ *to* 287 **do**
  - **if** $t_{i+256}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r}} = 1$
- **return** $\mathcal{M}$

Algorithm 33: MILP model for the coefficient solver when applying our nested framework to Kreyvium

Procedure `CMP-MPModelKreyvium`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{u}$ *indicating the cube term* $\boldsymbol{x}^{\boldsymbol{u}}$, *the middle round* r_m**):**

- Declare an empty MILP Model $\mathcal{M}$
- $\mathcal{M} \leftarrow \mathtt{s}_\mathtt{i}^0 \ \forall\ i \in \{0, 1, \ldots, 287\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{x}_\mathtt{i} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{k}_\mathtt{i} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- **for** $\mathtt{i} = 0$ *to* 127 **do**
  - $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i}^0 = 1 \ \forall\ u_i = 1$
  - $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i}^0 = 0 \ \forall\ u_i = 0$
  - $\mathcal{M}.con \leftarrow \mathtt{k}_\mathtt{i}^0 = 0$
- $\mathcal{M} \leftarrow \mathtt{K}_\mathtt{i}^{*,0} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{IV}_\mathtt{i}^{*,0} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M}.con \leftarrow \mathtt{K}_\mathtt{i}^{*,0} = 0$ for $i = 0, \ldots, 127$
- $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^0 = 0$ for $i = 0, \ldots, 92$ and $i = 93 + 128, \ldots, 287$
- $\mathcal{M}.con \leftarrow \mathtt{x}_\mathtt{i} = \mathtt{IV}_{128-\mathtt{i}}^{*,0} \vee \mathtt{s}_{93+\mathtt{i}}^0$ for $i = 0, \ldots, 127$
- **for** $i = 0$ *to* $r_m - 1$ **do**
  - $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}, \mathtt{K}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}+1}, \mathtt{IV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}+1}) =$ `CMPKreyviumUpdate`$(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}}, \mathtt{K}_0^{*,\mathtt{i}}, \ldots, \mathtt{K}_{127}^{*,\mathtt{i}}, \mathtt{IV}_0^{*,\mathtt{i}}, \ldots, \mathtt{IV}_{127}^{*,\mathtt{i}})$
- $\mathcal{M} \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 287\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}} \ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- /* $\boldsymbol{s}^{(r_m)} = (s_0^{(r_m)}, \ldots, s_{543}^{(r_m)})$ `denotes the concatenation of` r_m`-round state,` IV^{*,r_m} `and` K^{*,r_m} */
- **for** $i = 0$ *to* 127 **do**
  - **if** $s_{i+128}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{IV}_\mathtt{i}^{*,\mathtt{r_m}} = \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}} = 0$
  - **else if** $s_{i+128}^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{IV}_\mathtt{i}^{*,\mathtt{r_m}} \leq \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}}$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{IV}_\mathtt{i}^{*,\mathtt{r_m}} = \mathtt{VV}_\mathtt{i}^{*,\mathtt{r_m}}$
- **for** $i = 0$ *to* 287 **do**
  - **if** $s_{i+256}^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}} = 0$
  - **else if** $s_{i+256}^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} \leq \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{s}_\mathtt{i}^{\mathtt{r_m}} = \mathtt{ss}_\mathtt{i}^{\mathtt{r_m}}$
- **for** $i = r_m$ *to* $r - 1$ **do**
  - $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}+1}, \mathtt{KK}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{KK}_{127}^{*,\mathtt{i}+1}, \mathtt{VV}_0^{*,\mathtt{i}+1}, \ldots, \mathtt{VV}_{127}^{*,\mathtt{i}+1}) =$ `MPKreyviumUpdate`$(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}}, \ldots, \mathtt{ss}_{287}^{\mathtt{i}}, \mathtt{KK}_0^{*,\mathtt{i}}, \ldots, \mathtt{KK}_{127}^{*,\mathtt{i}}, \mathtt{VV}_0^{*,\mathtt{i}}, \ldots, \mathtt{VV}_{127}^{*,\mathtt{i}})$
- /* $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{543}^{(r)})$ `corresponds to` $\mathtt{KK}^{*,\mathtt{r}}, \mathtt{VV}^{*,\mathtt{r}}, \mathtt{ss}^{\mathtt{r}}$ */
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{KK}_\mathtt{i}^{*,\mathtt{r}} = 1$
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_{i+128}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{VV}_\mathtt{i}^{*,\mathtt{r}} = 1$
- **for** $i = 0$ *to* 287 **do**
  - **if** $t_{i+256}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_\mathtt{i}^{\mathtt{r}} = 1$
- **return** $\mathcal{M}$

Algorithm 34: `ChooseRCiKreyvium`

Procedure `ChooseRCiKreyvium`(r_n)**:**
    $r_c = r_n$
    return r_c

Algorithm 35: `ChooseTiKreyvium`

Procedure `ChooseTiKreyvium`(r_n)**:**
    if $r_n \geq 600$ **then** $\tau = 60$ seconds
    else if $r_n \geq 500$ **then** $\tau = 120$ seconds
    else if $r_n \geq 400$ **then** $\tau = 180$ seconds
    else if $r_n \geq 300$ **then** $\tau = 360$ seconds
    else if $r_n \geq 200$ **then** $\tau = 720$ second
    else if $r_n \geq 0$ **then** $\tau = \infty$
    return τ

K Application to ACORN

Specification of ACORN. ACORN [32] is a stream cipher for authenticated encryption, which has been selected as one of the finalists in CAESAR competition. ACORN has a 128-bit key and a 128-bit IV, denoted by $\boldsymbol{K}$ and $\boldsymbol{IV}$ respectively. Its internal state is represented by a 293-bit state $\boldsymbol{s} = (s_0, s_1, \ldots, s_{292})$. The update function is composed of two component functions $ks = ksg128(\boldsymbol{s})$ and $f = fbk128(\boldsymbol{s}, ca, cb)$ that are defined as

$$\begin{aligned} ks &= s_{12} \oplus s_{154} \oplus maj(s_{235}, s_{61}, s_{193}) \oplus ch(s_{230}, s_{111}, s_{66}), \\ f &= s_0 \oplus (s_{107} \oplus 1) \oplus maj(s_{244}, s_{23}, s_{160}) \oplus (ca \wedge s_{196}) \oplus (cb \wedge ks). \end{aligned}$$

ks is the key stream bit, and maj and ch are defined as

$$maj(x, y, z) = xy + xz + yz, ch(x, y, z) = xy + xz + z.$$

Initialized as $\boldsymbol{s}^{(0)} = (0, 0, \ldots, 0)$, the state of r-th round is generated by the following update function:

$$s_{289}^{(r-1)} \leftarrow s_{289}^{(r-1)} \oplus s_{235}^{(r-1)} \oplus s_{230}^{(r-1)}, s_{230}^{(r-1)} \leftarrow s_{230}^{(r-1)} \oplus s_{196}^{(r-1)} \oplus s_{193}^{(r-1)} \tag{12}$$

$$s_{193}^{(r-1)} \leftarrow s_{193}^{(r-1)} \oplus s_{160}^{(r-1)} \oplus s_{154}^{(r-1)}, s_{154}^{(r-1)} \leftarrow s_{154}^{(r-1)} \oplus s_{111}^{(r-1)} \oplus s_{107}^{(r-1)} \tag{13}$$

$$s_{107}^{(r-1)} \leftarrow s_{107}^{(r-1)} \oplus s_{66}^{(r-1)} \oplus s_{61}^{(r-1)}, s_{61}^{(r-1)} \leftarrow s_{61}^{(r-1)} \oplus s_{23}^{(r-1)} \oplus s_{0}^{(r-1)} \tag{14}$$

$$ks^{(r-1)} = ksg128(\boldsymbol{s}^{(r-1)}), f^{(r-1)} = fbk128(\boldsymbol{s}^{(r-1)}, 1, 1) \tag{15}$$

$$\boldsymbol{s}^{(r)} = (s_0^r, \ldots, s_{292}^r) \leftarrow (s_1^{(r-1)}, \ldots, s_{292}^{(r-1)}, f^{(r-1)} \oplus m^{(r-1)}) \tag{16}$$

The bit m_t is generated according to following rules:

$$m^{(t)} = \begin{cases} K_t & \text{for } t \in \{0, \ldots, 127\} \\ IV_{t-128} & \text{for } t \in \{128, \ldots, 255\} \\ K_0 \oplus 1 & \text{for } t = 256 \\ K_{t \bmod 128} & \text{for } t \in \{257, \ldots, 1791\} \end{cases}.$$

Two tricks. When recovering the superpoly of ACORN, we use two tricks based on the structure of ACORN to speed things up. We take an example to illustrate the first trick. For the function $maj = xy+xz+yz$, if $x.F = \delta, y.F = \delta, z.F = 1_c$, then according to Proposition 2, when modeling maj using NBDP or CMP based on the structure of Eqn. (5) or (7), we only need to model $maj = xy$ and the role of xz, yz can be ignored. For convenience, we call xy a *nice term*. Such an optimization is also applicable to ch.

Second, assuming the cube term is denoted by $\boldsymbol{x}^{\boldsymbol{u}}$ and the state of 256th round ACORN is denoted by $\boldsymbol{s}^{(256)}$, we find that for some cubes, there exists only one core monomial trail from $\boldsymbol{k}^{\boldsymbol{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ to the δ part of monomials of $\boldsymbol{s}^{(256)}$. Let this core monomial trail be

$$\boldsymbol{k}^{\boldsymbol{0}}||\boldsymbol{x}^{\boldsymbol{u}} \overset{\text{Core}}{\rightarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(1)})}(\boldsymbol{s}^{(1)}) \overset{\text{Core}}{\rightarrow} \cdots \overset{\text{Core}}{\rightarrow} \pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(255)})}(\boldsymbol{s}^{(255)}) \overset{\text{Core}}{\rightarrow} \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)}),$$

where $\boldsymbol{l}_{\boldsymbol{u}}^{(256)}$ is a specific vector with respect to the cube. This means we can regard the key stream bit $ks^{(r)}$ as a polynomial of δ bits of $\boldsymbol{s}^{(256)}$ with 1_c bits of $\boldsymbol{s}^{(256)}$ as the coefficients. We then attempt to recover the coefficient of $\pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})$, denoted by $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$, which should be a polynomial of 1_c bits of $\boldsymbol{s}^{(256)}$ and can be further expanded to be a polynomial with respect to $\boldsymbol{k}$. Considering each $\boldsymbol{t}^{(256)}$ satisfying $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(256)}}(\boldsymbol{s}^{(256)})$ must satisfy $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)}) = \boldsymbol{l}_{\boldsymbol{u}}^{(256)}$, the superpoly of $\boldsymbol{x}^{\boldsymbol{u}}$ can be computed as the product of $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$ and $\mathsf{Coe}\left(\pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$. The latter can be computed quickly by the monomial prediction technique, and $\boldsymbol{l}_{\boldsymbol{u}}^{(256)}$ can be determined by following the propagation of CMP from $\boldsymbol{k}^{\boldsymbol{0}}||\boldsymbol{x}^{\boldsymbol{u}}$ through 256 rounds.

We have confirmed using CMP that the second trick is applicable to all cube terms of Acorn we mentioned, then the problem of computing $\mathsf{Coe}\left(ks^{(r)}, \boldsymbol{x}^{\boldsymbol{u}}\right)$ is reduced to the problem of recovering the coefficient of $\pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})$ in $ks^{(r)}$, hence all our MILP models for Acorn are built from the 256th round.

Fast Evaluation of a Superpoly. For the superpoly of a cube that satisfies the condition of the second trick, i.e., each $\boldsymbol{t}^{(256)}$ satisfying $\boldsymbol{k}^{\boldsymbol{w}}\boldsymbol{x}^{\boldsymbol{u}} \rightsquigarrow \pi_{\boldsymbol{t}^{(256)}}(\boldsymbol{s}^{(256)})$ must satisfy $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)}) = \boldsymbol{l}_{\boldsymbol{u}}^{(256)}$, we can evaluate the superpoly by evaluating $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$ and $\mathsf{Coe}\left(\pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ respectively.

For all cube terms of Acorn we mentioned, we experimentally found that $\mathsf{Coe}\left(\pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)}), \boldsymbol{x}^{\boldsymbol{u}}\right)$ is always equal to 1. This is reasonable because from the 128th round to the 256th round, each IV bit is inserted into the last bit (s_{292}) of the state as a linear term (IV_{t-128}). $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$ is a polynomial of 1_c bits of $\boldsymbol{s}^{(256)}$ and its expression can be figured out during the superpoly recovery, therefore each time we assign values to $\boldsymbol{K}$, we can first compute the values of 1_c bits of $\boldsymbol{s}^{(256)}$ and then compute the value of $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$ quickly.

Consequently, the cost of evaluating a superpoly once is equal to one 256-round Acorn call, together with one evaluation of $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$. For our cubes I_1 and I_2 mentioned in Sect. 6.4, their $\mathsf{Coe}\left(ks^{(r)}, \pi_{\boldsymbol{l}_{\boldsymbol{u}}^{(256)}}(\boldsymbol{s}^{(256)})\right)$ only contain 484 and 280 terms respectively, so in this paper we assume one evaluation of the superpoly of Acorn is equivalent to one 256-round Acorn call.

MILP Model. The MILP model describing the monomial propagation of the update function in Acorn is shown in Algorithm 14. The subroutines are described in Algorithm 11, 12, 13, where `xorFB` denotes the procedure of LFSR update (12) and `genM` denotes the procedure of generating $m^{(t)}$. These models are directly adapted from [12,13,16].

Algorithm 36, 37, 38 and Algorithm 42, 43, 44 provide the propagation model of NBDP and CMP for the components of Acorn. Using these components,

the propagation of NBDP and CMP in the update function of ACORN can be traced, as shown in Algorithm 39 and 45. According to the second trick above, they are constructed from round 256. Finally, the MILP models used for the term expander and coefficient solver are presented in Algorithm 41 and 47 respectively.

Parameters. For the term expander, we fix $\varepsilon, R_s, N, \tau'$ to $5, 540, 15000, 7200$, respectively. r_c is chosen to be $min(r_n - 25, 350)$ according to the procedure `ChooseRCiAcorn` in Algorithm 40. For the coefficient solver, the procedure `ChooseTiAcorn` for selecting the time limit is present in Algorithm 46. In `CMP-MPModelAcorn`, r_m is fixed to 350 to ensure the second stage of the coefficient solver can be completed quickly.

Superpoly verification for 775-Round ACORN. In [34], the superpoly of a 126-dimensional cube indexed by

$$I = \{0, 1, \ldots, 127\} \backslash \{1, 26\}$$

is recovered. Besides, superpolies of five 126-dimensional cubes indexed by

$$I = \{0, 1, \ldots, 127\} \backslash \{i, j\}, \{i, j\} \in \{\{1, 2\}, \{1, 11\}, \{1, 18\}, \{2, 18\}, \{11, 18\}\},$$

are found to be constants. It took us about 6 days to verify the correctness of these results.

Algorithm 36: MILP Models for division trails of NBDP of *Maj* and *Ch* in ACORN

```
Procedure NBDPMaj(M, x_0, ..., x_292, i, j, k):
    Initialize an empty set V of M.var
    /* Suppose i, j, k correspond to the bit x_i, x_j, x_k                    */
    if x_i x_j is a nice term then
        (M, y_i, y_j, V) = NBDPAnd2(M, x_i, x_j, V)
    if x_i x_k is a nice term then
        (M, z_i, y_k, V) = NBDPAnd2(M, y_i, x_k, V)
    if x_j x_k is a nice term then
        (M, z_j, z_k, V) = NBDPAnd2(M, y_j, y_k, V)
    (M, o) = NBDPXor(M, V)
    for i ∈ {0, 1, ..., 287} w/o i, j, k do z_i = x_i
    return (M, z_0, z_1, ..., z_292, o)
Procedure NBDPCh(M, x_0, ..., x_292, i, j, k):
    Initialize an empty set V of M.var
    /* Suppose i, j, k correspond to the bit x_i, x_j, x_k                    */
    if x_i x_j is a nice term then
        (M, y_i, z_j, V) = NBDPAnd2(M, x_i, x_j, V)
    if x_i x_k is a nice term then
        (M, z_i, z_k, V) = NBDPAnd2(M, y_i, x_k, V)
    (M, o) = NBDPXor(M, V)
    for i ∈ {0, 1, ..., 287} w/o i, j, k do z_i = x_i
    return (M, z_0, z_1, ..., z_292, o)
```

Algorithm 37: MILP Models for division trails of NBDP of the LFSR in ACORN

Procedure NBDPxorFB($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, \mathtt{i}, \mathtt{j}, \mathtt{k}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  Add $\mathtt{x}_\mathtt{i}$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{y}_\mathtt{j}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{x}_\mathtt{j}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_\mathtt{k}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{x}_\mathtt{k}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_\mathtt{i}) = \mathtt{NBDPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $\mathtt{i}, \mathtt{j}, \mathtt{k}$ **do** $\mathtt{y}_\mathtt{i} = \mathtt{x}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{292})$

Algorithm 38: MILP Models for division trails of NBDP of *ksg128* and *fbk128* in ACORN

Procedure NBDPksg128($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{292}, \mathtt{c}) = \mathtt{NBDPMaj}(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, 235, 61, 193)$
  $(\mathcal{M}, \mathtt{z}_0, \mathtt{z}_1, \ldots, \mathtt{z}_{292}, \mathtt{d}) = \mathtt{NBDPCh}(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{292}, 230, 111, 66)$
  $(\mathcal{M}, \mathtt{t}_{12}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{z}_{12}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{t}_{154}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{z}_{154}, \mathbb{V})$
  Add $\mathtt{c}, \mathtt{d}$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{o}) = \mathtt{NBDPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $12, 154$ **do** $\mathtt{t}_\mathtt{i} = \mathtt{z}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{t}_0, \mathtt{t}_1, \ldots, \mathtt{t}_{292}, \mathtt{o})$
Procedure NBDPfbk128($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{292}, \mathtt{d}) = \mathtt{NBDPMaj}(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, 244, 23, 160)$
  $(\mathcal{M}, \mathtt{t}_0, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{y}_0, \mathbb{V})$
  $(\mathcal{M}, \mathtt{t}_{107}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{y}_{107}, \mathbb{V})$
  $(\mathcal{M}, \mathtt{t}_{196}, \mathbb{V}) = \mathtt{NBDPAnd1}(\mathcal{M}, \mathtt{y}_{196}, \mathbb{V})$
  Add $\mathtt{d}$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{o}) = \mathtt{NBDPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $0, 107, 196$ **do** $\mathtt{t}_\mathtt{i} = \mathtt{y}_\mathtt{i}$
  return $(\mathcal{M}, \mathtt{t}_0, \mathtt{t}_1, \ldots, \mathtt{t}_{292}, \mathtt{o})$

Algorithm 39: MILP Models for division trails of NBDP of the update function in ACORN

Procedure NBDPAcornUpdateFrom256($\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{292}^\mathtt{i}$):
  $(\mathcal{M}, \mathtt{t}_0, \ldots, \mathtt{t}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{s}_0^\mathtt{i}, \ldots, \mathtt{s}_{292}^\mathtt{i}, 289, 235, 230)$
  $(\mathcal{M}, \mathtt{u}_0, \ldots, \mathtt{u}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{t}_0, \ldots, \mathtt{t}_{292}, 230, 196, 193)$
  $(\mathcal{M}, \mathtt{w}_0, \ldots, \mathtt{w}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{u}_0, \ldots, \mathtt{u}_{292}, 193, 160, 154)$
  $(\mathcal{M}, \mathtt{l}_0, \ldots, \mathtt{l}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{w}_0, \ldots, \mathtt{w}_{292}, 154, 111, 107)$
  $(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{l}_0, \ldots, \mathtt{l}_{292}, 107, 66, 61)$
  $(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{292}) = \mathtt{NBDPxorFB}(\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, 61, 23, 0)$
  $(\mathcal{M}, \mathtt{a}_0, \ldots, \mathtt{a}_{292}, \mathtt{e}) = \mathtt{NBDPksg128}(\mathcal{M}, \mathtt{y}_0, \ldots, \mathtt{y}_{292})$
  $(\mathcal{M}, \mathtt{z}_0, \ldots, \mathtt{z}_{292}, \mathtt{b}) = \mathtt{NBDPfbk128}(\mathcal{M}, \mathtt{a}_0, \ldots, \mathtt{a}_{292})$
  $\mathcal{M}.var \leftarrow \mathtt{o}$ as binary
  $\mathcal{M}.con \leftarrow \mathtt{o} = \mathtt{e} + \mathtt{b}$
  $\mathcal{M}.con \leftarrow \mathtt{z}_0 = 0$
  $(\mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{292}^{\mathtt{i}+1}) = (\mathtt{z}_1, \mathtt{z}_2, \ldots, \mathtt{z}_{292}, \mathtt{o})$
  return $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{292}^{\mathtt{i}+1})$

Algorithm 40: `ChooseRCiAcorn`

Procedure `ChooseRCiAcorn`(r_n)**:**
  $r_c = r_n - 25$
  return r_c

Algorithm 41: MILP model for the term expander when applying our nested framework to ACORN

Procedure `NBDP-MPModelAcorn`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)})$ *indicating the cube term* $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)})}(\boldsymbol{s}^{(256)})$, *the middle round* $r_m (r_m > 256)$)**:**
  Declare an empty MILP Model $\mathcal{M}$
  $\mathcal{M} \leftarrow \mathtt{s}_{\mathtt{i}}^{256}\ \forall\ i \in \{0, 1, \ldots, 292\}$ as binary
  for $\mathtt{i} = 0$ *to* 292 **do**
    $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{256} = 1\ \forall\ \Lambda_i^{\delta}(\boldsymbol{t}^{(256)}) = 1$
  for $i = 256$ *to* $r_m - 1$ **do**
    $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) =$ `NBDPAcornUpdateFrom256`$(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}})$
  $\mathcal{M} \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 292\}$ as binary
  $\mathcal{M} \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
  $\mathcal{M} \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
  for $i = 0$ *to* 292 **do**
    if $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} = \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} \leq \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}$
  for $i = r_m$ *to* $r - 1$ **do**
    $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{292}^{\mathtt{i}+1}, \mathtt{k}_0^{\mathtt{i}+1}, \ldots, \mathtt{k}_{127}^{\mathtt{i}+1}, \mathtt{v}_0^{\mathtt{i}+1}, \ldots, \mathtt{v}_{127}^{\mathtt{i}+1}) =$
     `MPAcornUpdate`$(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}}, \ldots, \mathtt{ss}_{292}^{\mathtt{i}}, \mathtt{k}_0^{\mathtt{i}}, \ldots, \mathtt{k}_{127}^{\mathtt{i}}, \mathtt{v}_0^{\mathtt{i}}, \ldots, \mathtt{v}_{127}^{\mathtt{i}})$
  /* $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{549}^{(r)})$ corresponds to $\mathtt{ss}^{\mathtt{r}}, \mathtt{v}^{\mathtt{r}}, \mathtt{k}^{\mathtt{r}}$ */
  for $i = 0$ *to* 127 **do**
    if $t_{421+i}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r}} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r}} = 1$
  for $i = 0$ *to* 127 **do**
    if $t_{293+i}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r}} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r}} = 1$
  for $i = 0$ *to* 292 **do**
    if $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r}} = 0$
    else $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r}} = 1$
  return $\mathcal{M}$

Algorithm 42: MILP Models for core monomial trails of *Maj* and *Ch* in ACORN

Procedure CMPMaj($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, \mathtt{i}, \mathtt{j}, \mathtt{k}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  /* Suppose $\mathtt{i}, \mathtt{j}, \mathtt{k}$ correspond to the bit x_i, x_j, x_k */
  if $x_i x_j$ *is a nice term* **then**
    $(\mathcal{M}, \mathtt{y}_i, \mathtt{y}_j, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{x}_i, \mathtt{x}_j, \mathbb{V})$
  if $x_i x_k$ *is a nice term* **then**
    $(\mathcal{M}, \mathtt{z}_i, \mathtt{y}_k, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{y}_i, \mathtt{x}_k, \mathbb{V})$
  if $x_j x_k$ *is a nice term* **then**
    $(\mathcal{M}, \mathtt{z}_j, \mathtt{z}_k, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{y}_j, \mathtt{y}_k, \mathbb{V})$
  $(\mathcal{M}, \mathtt{o}) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 287\}$ *w/o* $\mathtt{i}, \mathtt{j}, \mathtt{k}$ **do** $\mathtt{z}_i = \mathtt{x}_i$
  return $(\mathcal{M}, \mathtt{z}_0, \mathtt{z}_1, \ldots, \mathtt{z}_{292}, \mathtt{o})$
Procedure CMPCh($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, \mathtt{i}, \mathtt{j}, \mathtt{k}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  /* Suppose $\mathtt{i}, \mathtt{j}, \mathtt{k}$ correspond to the bit x_i, x_j, x_k */
  if $x_i x_j$ *is a nice term* **then**
    $(\mathcal{M}, \mathtt{y}_i, \mathtt{z}_j, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{x}_i, \mathtt{x}_j, \mathbb{V})$
  if $x_i x_k$ *is a nice term* **then**
    $(\mathcal{M}, \mathtt{z}_i, \mathtt{z}_k, \mathbb{V}) = \mathtt{CMPAnd2}(\mathcal{M}, \mathtt{y}_i, \mathtt{x}_k, \mathbb{V})$
  $(\mathcal{M}, \mathtt{o}) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 287\}$ *w/o* $\mathtt{i}, \mathtt{j}, \mathtt{k}$ **do** $\mathtt{z}_i = \mathtt{x}_i$
  return $(\mathcal{M}, \mathtt{z}_0, \mathtt{z}_1, \ldots, \mathtt{z}_{292}, \mathtt{o})$

Algorithm 43: MILP Models for core monomial trails of the LFSR in ACORN

Procedure CMPxorFB($\mathcal{M}, \mathtt{x}_0, \ldots, \mathtt{x}_{292}, \mathtt{i}, \mathtt{j}, \mathtt{k}$):
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  Add $\mathtt{x}_i$ to $\mathbb{V}$
  $(\mathcal{M}, \mathtt{y}_j, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{x}_j, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_k, \mathbb{V}) = \mathtt{CMPAnd1}(\mathcal{M}, \mathtt{x}_k, \mathbb{V})$
  $(\mathcal{M}, \mathtt{y}_i) = \mathtt{CMPXor}(\mathcal{M}, \mathbb{V})$
  for $\mathtt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $\mathtt{i}, \mathtt{j}, \mathtt{k}$ **do** $\mathtt{y}_i = \mathtt{x}_i$
  return $(\mathcal{M}, \mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_{292})$

Algorithm 44: MILP Models for core monomial trails of *ksg128* and *fbk128* in ACORN

Procedure $\texttt{CMPksg128}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292})$**:**
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{c}) = \texttt{CMPMaj}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 235, 61, 193)$
  $(\mathcal{M}, \texttt{z}_0, \texttt{z}_1, \ldots, \texttt{z}_{292}, \texttt{d}) = \texttt{CMPCh}(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292}, 230, 111, 66)$
  $(\mathcal{M}, \texttt{t}_{12}, \mathbb{V}) = \texttt{CMPAnd1}(\mathcal{M}, \texttt{z}_{12}, \mathbb{V})$
  $(\mathcal{M}, \texttt{t}_{154}, \mathbb{V}) = \texttt{CMPAnd1}(\mathcal{M}, \texttt{z}_{154}, \mathbb{V})$
  Add $\texttt{c}, \texttt{d}$ to $\mathbb{V}$
  $(\mathcal{M}, \texttt{o}) = \texttt{CMPXor}(\mathcal{M}, \mathbb{V})$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $12, 154$ **do** $\texttt{t}_\texttt{i} = \texttt{z}_\texttt{i}$
  return $(\mathcal{M}, \texttt{t}_0, \texttt{t}_1, \ldots, \texttt{t}_{292}, \texttt{o})$

Procedure $\texttt{CMPfbk128}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292})$**:**
  Initialize an empty set $\mathbb{V}$ of $\mathcal{M}.var$
  $(\mathcal{M}, \texttt{y}_0, \texttt{y}_1, \ldots, \texttt{y}_{292}, \texttt{d}) = \texttt{CMPMaj}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 244, 23, 160)$
  $(\mathcal{M}, \texttt{t}_0, \mathbb{V}) = \texttt{CMPAnd1}(\mathcal{M}, \texttt{y}_0, \mathbb{V})$
  $(\mathcal{M}, \texttt{t}_{107}, \mathbb{V}) = \texttt{CMPAnd1}(\mathcal{M}, \texttt{y}_{107}, \mathbb{V})$
  $(\mathcal{M}, \texttt{t}_{196}, \mathbb{V}) = \texttt{CMPAnd1}(\mathcal{M}, \texttt{y}_{196}, \mathbb{V})$
  Add $\texttt{d}$ to $\mathbb{V}$
  $(\mathcal{M}, \texttt{o}) = \texttt{CMPXor}(\mathcal{M}, \mathbb{V})$
  for $\texttt{i} \in \{0, 1, \ldots, 292\}$ *w/o* $0, 107, 196$ **do** $\texttt{t}_\texttt{i} = \texttt{y}_\texttt{i}$
  return $(\mathcal{M}, \texttt{t}_0, \texttt{t}_1, \ldots, \texttt{t}_{292}, \texttt{o})$

Algorithm 45: MILP Models for core monomial trails of the update function in ACORN

Procedure $\texttt{CMPAcornUpdateFrom256}(\mathcal{M}, \texttt{s}_0^\texttt{i}, \ldots, \texttt{s}_{292}^\texttt{i})$**:**
  $(\mathcal{M}, \texttt{t}_0, \ldots, \texttt{t}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{s}_0^\texttt{i}, \ldots, \texttt{s}_{292}^\texttt{i}, 289, 235, 230)$
  $(\mathcal{M}, \texttt{u}_0, \ldots, \texttt{u}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{t}_0, \ldots, \texttt{t}_{292}, 230, 196, 193)$
  $(\mathcal{M}, \texttt{w}_0, \ldots, \texttt{w}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{u}_0, \ldots, \texttt{u}_{292}, 193, 160, 154)$
  $(\mathcal{M}, \texttt{l}_0, \ldots, \texttt{l}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{w}_0, \ldots, \texttt{w}_{292}, 154, 111, 107)$
  $(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{l}_0, \ldots, \texttt{l}_{292}, 107, 66, 61)$
  $(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292}) = \texttt{CMPxorFB}(\mathcal{M}, \texttt{x}_0, \ldots, \texttt{x}_{292}, 61, 23, 0)$
  $(\mathcal{M}, \texttt{a}_0, \ldots, \texttt{a}_{292}, \texttt{e}) = \texttt{CMPksg128}(\mathcal{M}, \texttt{y}_0, \ldots, \texttt{y}_{292})$
  $(\mathcal{M}, \texttt{z}_0, \ldots, \texttt{z}_{292}, \texttt{b}) = \texttt{CMPfbk128}(\mathcal{M}, \texttt{a}_0, \ldots, \texttt{a}_{292})$
  $\mathcal{M}.var \leftarrow \texttt{o}$ as binary
  $\mathcal{M}.con \leftarrow \texttt{o} = \texttt{e} + \texttt{b}$
  $\mathcal{M}.con \leftarrow \texttt{z}_0 = 0$
  $(\texttt{s}_0^{\texttt{i}+1}, \ldots, \texttt{s}_{292}^{\texttt{i}+1}) = (\texttt{z}_1, \texttt{z}_2, \ldots, \texttt{z}_{292}, \texttt{o})$
  return $(\mathcal{M}, \texttt{s}_0^{\texttt{i}+1}, \ldots, \texttt{s}_{292}^{\texttt{i}+1})$

Algorithm 46: `ChooseTiAcorn`

Procedure $\texttt{ChooseTiAcorn}(r_n)$**:**
  if $r_n \geq 600$ **then** $\tau = 600$ seconds
  else if $r_n \geq 500$ **then** $\tau = 1800$ seconds
  else if $r_n \geq 400$ **then** $\tau = 3600$ seconds
  else if $r_n \geq 300$ **then** $\tau = 7200$ seconds
  else if $r_n \geq 200$ **then** $\tau = \infty$
  return τ

Algorithm 47: MILP model for the coefficient solver when applying our nested framework to ACORN

Procedure `CMP-MPModelAcorn`(*the number of rounds* r, $\boldsymbol{t}^{(r)}$ *indicating the output bit* $\pi_{\boldsymbol{t}^{(r)}}(\boldsymbol{s}^{(r)})$, $\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)})$ *indicating the cube term* $\pi_{\boldsymbol{\Lambda}^{\delta}(\boldsymbol{t}^{(256)})}(\boldsymbol{s}^{(256)})$, *the middle round* $r_m (r_m > 256)$)**:**

- Declare an empty MILP Model $\mathcal{M}$
- $\mathcal{M} \leftarrow \mathtt{s}_{\mathtt{i}}^{256}\ \forall\ i \in \{0, 1, \ldots, 292\}$ as binary
- **for** $\mathtt{i} = 0$ *to* 292 **do**
  - $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{256} = 1\ \forall\ \Lambda_i^{\delta}(\boldsymbol{t}^{(256)}) = 1$
- **for** $i = 256$ *to* $r_m - 1$ **do**
  - $(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}+1}, \ldots, \mathtt{s}_{287}^{\mathtt{i}+1}) = \mathtt{CMPAcornUpdateFrom256}(\mathcal{M}, \mathtt{s}_0^{\mathtt{i}}, \ldots, \mathtt{s}_{287}^{\mathtt{i}})$
- $\mathcal{M} \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 292\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- $\mathcal{M} \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r_m}}\ \forall\ i \in \{0, 1, \ldots, 127\}$ as binary
- **for** $i = 0$ *to* 292 **do**
  - **if** $s_i^{(r_m)}.F = 0_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} = \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}} = 0$
  - **else if** $s_i^{(r_m)}.F = 1_c$ **then** $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} \leq \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{s}_{\mathtt{i}}^{\mathtt{r_m}} = \mathtt{ss}_{\mathtt{i}}^{\mathtt{r_m}}$
- **for** $i = r_m$ *to* $r - 1$ **do**
  - $(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}+1}, \ldots, \mathtt{ss}_{292}^{\mathtt{i}+1}, \mathtt{k}_0^{\mathtt{i}+1}, \ldots, \mathtt{k}_{127}^{\mathtt{i}+1}, \mathtt{v}_0^{\mathtt{i}+1}, \ldots, \mathtt{v}_{127}^{\mathtt{i}+1}) =$ `MPAcornUpdate`$(\mathcal{M}, \mathtt{ss}_0^{\mathtt{i}}, \ldots, \mathtt{ss}_{292}^{\mathtt{i}}, \mathtt{k}_0^{\mathtt{i}}, \ldots, \mathtt{k}_{127}^{\mathtt{i}}, \mathtt{v}_0^{\mathtt{i}}, \ldots, \mathtt{v}_{127}^{\mathtt{i}})$

`/*` $\boldsymbol{t}^{(r)} = (t_0^{(r)}, \ldots, t_{549}^{(r)})$ `corresponds to` $\mathtt{ss}^{\mathtt{r}}, \mathtt{v}^{\mathtt{r}}, \mathtt{k}^{\mathtt{r}}$ `*/`

- **for** $i = 0$ *to* 127 **do**
  - **if** $t_{421+i}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{k}_{\mathtt{i}}^{\mathtt{r}} = 1$
- **for** $i = 0$ *to* 127 **do**
  - **if** $t_{293+i}^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{v}_{\mathtt{i}}^{\mathtt{r}} = 1$
- **for** $i = 0$ *to* 292 **do**
  - **if** $t_i^{(r)} = 0$ **then** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r}} = 0$
  - **else** $\mathcal{M}.con \leftarrow \mathtt{ss}_{\mathtt{i}}^{\mathtt{r}} = 1$
- **return** $\mathcal{M}$

L Previous Methods to Recover Key Bits from Multiple Superpolies

We now recall existing methods that can recover key bits from multiple superpolies:

Linear or quadratic superpolies. As mentioned, in the early stage of cube attacks [9,20,10,36], the superpolies recovered have to be extremely simple (typically linear or quadratic functions). Even in [38], Ye and Tian et al. extended the practical key-recovery attack on TRIVIUM to 806 rounds, the superpolies they are concerned about are still linear. The time to solve a linear system of equations is negligible. As for solving quadratic equations, this can be done by hand since the form of superpoly is usually very simple. However, we have to admit that with the increase of the number of rounds, the probability that a superpoly is linear or quadratic will become lower and lower.

Superpolies with balanced secret variables. In [23], Yao et al. proposed a heuristic algorithm to search for superpolies with balanced bits using monomial predictions. After solving the linear equations in previous works to obtain some values of secret variables, they finally ended up with a nonlinear or non-quadratic system of equations. The presence of balanced bits provides an opportunity to iteratively transform the nonlinear equation into a linear equation by guessing bits in order, so such a system can also be solved soon. However, with the number of rounds grows, the balanced secret variables tend to disappear in a superpoly.

Superpolies not involving all secret variables. In many previous cube attacks on such as [17,12,13,29,25], the recovered superpoly tends not to carry all secret variables. Suppose for a balanced superpoly of the cube whose dimensions are not larger than t, the involved key bits are represented by a set J, then one bit information can be recovered with time complexity $2^{|J|}+2^t$. For m such superpolies, we can recover m bit information with time complexity $m\cdot(2^{|J|}+2^t)$. As long as this time complexity is lower than the complexity of exhaustive search, the cube attack is considered successful.

The disjoint set. In [16], Hu el al. recovered more than one superpolies for 843-, 844- and 845-round TRIVIUM, and these superpolies involve all key bits with a balancedness estimated to be almost 0.50. The previous methods no longer work for these superpolies. To extract as much information as possible from superpolies, they split each superpoly $p(\boldsymbol{k})$ into the following form according to a disjoint set $D=\{k_{i_0},k_{i_1},\ldots,k_{i_{m-1}}\}$:

$$p(\boldsymbol{k})=\Big(\bigoplus_{0\le i<m} k_i\cdot p_i(J)\Big)\oplus p_m(J),$$

where $J=\{k_0,k_1,\ldots,k_{n-1}\}/D$. In the offline phase, the true table of each $p_i(J)$ is computed by the Möbius transformation. Then in the online phase, after computing the value of $p(\boldsymbol{k})$ and guessing the values of secret variables in J, we

can obtain a system of linear equations involving secret variables in D, which should be solved quickly. In other words, the disjoint set reduces the problem of solving a system of nonlinear or non-quadratic equations to the problem of solving $2^{|J|}$ systems of linear equations. However, as the number of rounds grows, even a disjoint is hard to find. Since the superpolies of 846- and 847-round TRIVIUM are not linear or quadratic and involve all secret variables, we tried to search a disjoint set for them. Unfortunately, no disjoint set was found.

M Simplified Möbius Transformation

Let the degree of f be d ($d > 0$). If $wt(c_t, \ldots, c_{n-1}) > d$, then $p_{(c_t,\ldots,c_{n-1})}(x_0, \ldots, x_{t-1}) = 0$. According to Proposition 3, we have $g_t(x_0, \ldots, x_{t-1}, c_t, \ldots, c_{n-1}) = 0$ if $wt(c_t, \ldots, c_{n-1}) > d$. Recall in the process of Möbius Transformation shown in Algorithm 2, $g_t(2^t j + 2^{t-1} + k)$ is updated as

$$g_{t-1}(2^t j + 2^{t-1} + k) + g_{t-1}(2^t j + k).$$

We can skip this update step if $g_{t-1}(2^t j + k) = 0$. This inspires us to design a algorithm to simplify the process of Möbius transformation, as shown in Algorithm 48. Usually n is large and the structure of f is sparse, which means we can neglect the cost of Line 4, therefore the time complexity of the simplified Möbius transformation is at most

$$\sum_{t=1}^{n} 2^{t-1} \times (\binom{n-t}{0} + \binom{n-t}{1} + \cdots + \binom{n-t}{d})^{3}$$

bitwise XORs, while the memory complexity remains 2^n bits. In [16], the author also proposed an improved variant of Möbius Transformation using sparse ANFs, as a result, the time complexity of Möbius Transformation was reduced to $n \times 2^{n-2}$ XORs. It is hard to compare our variant of Möbius Transformation with their variant, we therefore hope that this may provide new inspiration for optimizing Möbius Transformation on a sparse f.

[3] $\binom{0}{0} = 1$ and $\binom{n}{r} = 0$ if $r > n$.

Algorithm 48: Simplified Möbius transformation on f of degree d in Eqn. (8)

Procedure `MobiusTransformation`(The ANF of f):
  for $t = 1$ to n **do**
    Initialize g_t to be the same as g_{t-1}
    Express f as

$$f = \sum_{(c_{t-1},\ldots,c_{n-1})\in\mathbb{F}_2^{n-t+1}} p_{(c_{t-1},\ldots,c_{n-1})}(x_0,\ldots,x_{t-2}) \cdot \prod_{i=t-1}^{n-1} x_i^{c_i}.$$

    /* When $t = 1$, we assume $p_{(c_{t-1},\ldots,c_{n-1})}(x_0,\ldots,x_{t-2}) = g_0(c_{t-1},\ldots,c_{n-1})$ */
    for $(0, c_t, \ldots, c_{n-1})$ whose $p_{(0,c_t,\ldots,c_{n-1})}(x_0,\ldots,x_{t-2}) \neq 0$ **do**
      Let $(c_t, \ldots, c_{n-1})$ be the binary representation of j
      /* When $t = n$, we assume $(0, c_t, \ldots, c_{n-1}) = (0)$ and $j = 0$ */
      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

N Superpolies of 776-round Acorn

$$p_{I_1} = 1 + s_{162}^{(256)} + s_{161}^{(256)} s_{193}^{(256)} + s_{161}^{(256)} s_{166}^{(256)} + s_{161}^{(256)} s_{162}^{(256)} + s_{160}^{(256)} s_{166}^{(256)} + s_{160}^{(256)} s_{164}^{(256)} + s_{160}^{(256)} s_{164}^{(256)} s_{193}^{(256)} + s_{160}^{(256)} s_{162}^{(256)} + s_{160}^{(256)} s_{161}^{(256)} + s_{160}^{(256)} s_{161}^{(256)} s_{166}^{(256)} s_{193}^{(256)} + s_{159}^{(256)} + s_{158}^{(256)} + s_{158}^{(256)} s_{160}^{(256)} + s_{158}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{157}^{(256)} s_{164}^{(256)} + s_{157}^{(256)} s_{161}^{(256)} + s_{155}^{(256)} s_{164}^{(256)} + s_{155}^{(256)} s_{161}^{(256)} + s_{155}^{(256)} s_{160}^{(256)} + s_{154}^{(256)} s_{161}^{(256)} + s_{154}^{(256)} s_{160}^{(256)} s_{164}^{(256)} + s_{154}^{(256)} s_{160}^{(256)} s_{161}^{(256)} s_{166}^{(256)} + s_{154}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{153}^{(256)} s_{164}^{(256)} + s_{153}^{(256)} s_{161}^{(256)} + s_{153}^{(256)} s_{160}^{(256)} + s_{152}^{(256)} + s_{150}^{(256)} + s_{150}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{149}^{(256)} s_{160}^{(256)} + s_{149}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{148}^{(256)} + s_{148}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{147}^{(256)} s_{160}^{(256)} + s_{147}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{146}^{(256)} + s_{146}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{145}^{(256)} + s_{145}^{(256)} s_{160}^{(256)} + s_{145}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{145}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{145}^{(256)} s_{157}^{(256)} + s_{145}^{(256)} s_{155}^{(256)} + s_{145}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{145}^{(256)} s_{153}^{(256)} + s_{144}^{(256)} s_{193}^{(256)} + s_{144}^{(256)} s_{160}^{(256)} + s_{144}^{(256)} s_{154}^{(256)} + s_{142}^{(256)} + s_{141}^{(256)} + s_{139}^{(256)} + s_{139}^{(256)} s_{160}^{(256)} + s_{138}^{(256)} + s_{138}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{136}^{(256)} + s_{136}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{135}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{134}^{(256)} + s_{134}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{133}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{132}^{(256)} + s_{131}^{(256)} + s_{130}^{(256)} + s_{130}^{(256)} s_{160}^{(256)} + s_{130}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{128}^{(256)} + s_{127}^{(256)} s_{160}^{(256)} + s_{127}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{127}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{124}^{(256)} s_{160}^{(256)} + s_{123}^{(256)} + s_{119}^{(256)} + s_{117}^{(256)} s_{193}^{(256)} + s_{117}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{117}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{117}^{(256)} s_{154}^{(256)} + s_{117}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{117}^{(256)} s_{144}^{(256)} + s_{116}^{(256)} + s_{115}^{(256)} + s_{115}^{(256)} s_{193}^{(256)} + s_{115}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{115}^{(256)} s_{160}^{(256)} + s_{115}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{115}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{115}^{(256)} s_{154}^{(256)} + s_{115}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{115}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{115}^{(256)} s_{150}^{(256)} s_{160}^{(256)} + s_{115}^{(256)} s_{148}^{(256)} s_{160}^{(256)} + s_{115}^{(256)} s_{146}^{(256)} s_{160}^{(256)} + s_{115}^{(256)} s_{144}^{(256)} + s_{113}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{113}^{(256)} s_{160}^{(256)} + s_{113}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{113}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{113}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{113}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{113}^{(256)} s_{150}^{(256)} s_{160}^{(256)} + s_{113}^{(256)} s_{148}^{(256)} s_{160}^{(256)} + s_{113}^{(256)} s_{146}^{(256)} s_{160}^{(256)} + s_{113}^{(256)} s_{144}^{(256)} + s_{112}^{(256)} s_{193}^{(256)} + s_{112}^{(256)} s_{164}^{(256)} + s_{112}^{(256)} s_{161}^{(256)} + s_{112}^{(256)} s_{160}^{(256)} + s_{112}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{112}^{(256)} s_{154}^{(256)} + s_{112}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{112}^{(256)} s_{145}^{(256)} + s_{112}^{(256)} s_{144}^{(256)} + s_{112}^{(256)} s_{117}^{(256)} s_{193}^{(256)} + s_{112}^{(256)} s_{117}^{(256)}$$

$$s_{160}^{(256)} + s_{112}^{(256)} s_{117}^{(256)} s_{154}^{(256)} + s_{112}^{(256)} s_{115}^{(256)} s_{193}^{(256)} + s_{112}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{112}^{(256)} s_{115}^{(256)} s_{154}^{(256)} + s_{112}^{(256)} s_{113}^{(256)} s_{193}^{(256)} + s_{112}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{112}^{(256)} s_{113}^{(256)} s_{154}^{(256)} + s_{111}^{(256)} s_{193}^{(256)} + s_{111}^{(256)} s_{166}^{(256)} + s_{111}^{(256)} s_{161}^{(256)} + s_{111}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{111}^{(256)} s_{154}^{(256)} + s_{111}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{111}^{(256)} s_{150}^{(256)} s_{160}^{(256)} + s_{111}^{(256)} s_{148}^{(256)} s_{160}^{(256)} + s_{111}^{(256)} s_{146}^{(256)} s_{160}^{(256)} + s_{111}^{(256)} s_{135}^{(256)} + s_{111}^{(256)} s_{118}^{(256)} + s_{111}^{(256)} s_{116}^{(256)} + s_{111}^{(256)} s_{114}^{(256)} + s_{111}^{(256)} s_{112}^{(256)} + s_{110}^{(256)} s_{164}^{(256)} + s_{110}^{(256)} s_{161}^{(256)} + s_{110}^{(256)} s_{145}^{(256)} + s_{109}^{(256)} s_{164}^{(256)} + s_{109}^{(256)} s_{161}^{(256)} + s_{109}^{(256)} s_{160}^{(256)} + s_{109}^{(256)} s_{145}^{(256)} + s_{108}^{(256)} s_{164}^{(256)} + s_{108}^{(256)} s_{161}^{(256)} + s_{108}^{(256)} s_{160}^{(256)} + s_{108}^{(256)} s_{145}^{(256)} + s_{107}^{(256)} + s_{107}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{107}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{107}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{107}^{(256)} s_{111}^{(256)} s_{160}^{(256)} + s_{106}^{(256)} s_{160}^{(256)} + s_{106}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{105}^{(256)} + s_{103}^{(256)} s_{193}^{(256)} + s_{103}^{(256)} s_{154}^{(256)} + s_{102}^{(256)} + s_{102}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{102}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{102}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{102}^{(256)} s_{111}^{(256)} s_{160}^{(256)} + s_{101}^{(256)} + s_{101}^{(256)} s_{160}^{(256)} + s_{101}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{100}^{(256)} s_{160}^{(256)} + s_{100}^{(256)} s_{157}^{(256)} + s_{100}^{(256)} s_{155}^{(256)} + s_{100}^{(256)} s_{153}^{(256)} + s_{100}^{(256)} s_{112}^{(256)} + s_{100}^{(256)} s_{110}^{(256)} + s_{100}^{(256)} s_{109}^{(256)} + s_{100}^{(256)} s_{108}^{(256)} + s_{99}^{(256)} + s_{99}^{(256)} s_{193}^{(256)} + s_{99}^{(256)} s_{160}^{(256)} + s_{99}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{99}^{(256)} s_{154}^{(256)} + s_{99}^{(256)} s_{117}^{(256)} + s_{99}^{(256)} s_{115}^{(256)} + s_{99}^{(256)} s_{113}^{(256)} + s_{99}^{(256)} s_{112}^{(256)} + s_{98}^{(256)} s_{193}^{(256)} + s_{98}^{(256)} s_{154}^{(256)} + s_{97}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{94}^{(256)} + s_{94}^{(256)} s_{160}^{(256)} + s_{94}^{(256)} s_{111}^{(256)} + s_{93}^{(256)} + s_{90}^{(256)} s_{111}^{(256)} + s_{89}^{(256)} + s_{89}^{(256)} s_{160}^{(256)} + s_{88}^{(256)} + s_{86}^{(256)} + s_{86}^{(256)} s_{160}^{(256)} + s_{86}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{86}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{85}^{(256)} + s_{85}^{(256)} s_{160}^{(256)} + s_{83}^{(256)} + s_{82}^{(256)} s_{160}^{(256)} + s_{82}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{82}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{81}^{(256)} + s_{80}^{(256)} + s_{79}^{(256)} + s_{79}^{(256)} s_{160}^{(256)} + s_{78}^{(256)} + s_{76}^{(256)} + s_{75}^{(256)} s_{111}^{(256)} + s_{74}^{(256)} s_{193}^{(256)} + s_{74}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{74}^{(256)} s_{160}^{(256)} + s_{74}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{74}^{(256)} s_{154}^{(256)} + s_{74}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{74}^{(256)} s_{144}^{(256)} + s_{74}^{(256)} s_{112}^{(256)} s_{193}^{(256)} + s_{74}^{(256)} s_{112}^{(256)} s_{160}^{(256)} + s_{74}^{(256)} s_{112}^{(256)} s_{154}^{(256)} + s_{74}^{(256)} s_{99}^{(256)} + s_{73}^{(256)} s_{164}^{(256)} + s_{73}^{(256)} s_{161}^{(256)} + s_{73}^{(256)} s_{160}^{(256)} + s_{73}^{(256)} s_{145}^{(256)} + s_{73}^{(256)} s_{100}^{(256)} + s_{72}^{(256)} + s_{72}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{72}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{72}^{(256)} s_{150}^{(256)} s_{160}^{(256)} + s_{72}^{(256)} s_{148}^{(256)} s_{160}^{(256)} + s_{72}^{(256)} s_{146}^{(256)} s_{160}^{(256)} + s_{72}^{(256)} s_{107}^{(256)} s_{160}^{(256)} + s_{72}^{(256)} s_{102}^{(256)} s_{160}^{(256)} + s_{71}^{(256)} s_{160}^{(256)} + s_{70}^{(256)} + s_{70}^{(256)} s_{160}^{(256)} + s_{70}^{(256)} s_{111}^{(256)} + s_{69}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{69}^{(256)} s_{160}^{(256)} + s_{69}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{69}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{69}^{(256)} s_{144}^{(256)} + s_{69}^{(256)} s_{112}^{(256)} s_{193}^{(256)} + s_{69}^{(256)} s_{112}^{(256)} s_{160}^{(256)} + s_{69}^{(256)} s_{112}^{(256)} s_{154}^{(256)} + s_{69}^{(256)} s_{99}^{(256)} + s_{68}^{(256)} + s_{68}^{(256)} s_{193}^{(256)} + s_{68}^{(256)} s_{160}^{(256)} + s_{68}^{(256)} s_{154}^{(256)} + s_{67}^{(256)} + s_{67}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{150}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{148}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{146}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{144}^{(256)} + s_{67}^{(256)} s_{117}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{117}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{117}^{(256)} s_{154}^{(256)} + s_{67}^{(256)} s_{115}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{115}^{(256)} s_{154}^{(256)} + s_{67}^{(256)} s_{113}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{113}^{(256)} s_{154}^{(256)} + s_{67}^{(256)} s_{111}^{(256)} + s_{67}^{(256)} s_{107}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{102}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{99}^{(256)} + s_{67}^{(256)} s_{74}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{74}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{74}^{(256)} s_{154}^{(256)} + s_{67}^{(256)} s_{69}^{(256)} s_{193}^{(256)} + s_{67}^{(256)} s_{69}^{(256)} s_{160}^{(256)} + s_{67}^{(256)} s_{69}^{(256)} s_{154}^{(256)} + s_{66}^{(256)} s_{193}^{(256)} + s_{66}^{(256)} s_{166}^{(256)} + s_{66}^{(256)} s_{161}^{(256)} + s_{66}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{66}^{(256)} s_{154}^{(256)} + s_{66}^{(256)} s_{135}^{(256)} + s_{66}^{(256)} s_{118}^{(256)} + s_{66}^{(256)} s_{116}^{(256)} + s_{66}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{66}^{(256)} s_{114}^{(256)} + s_{66}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{66}^{(256)} s_{112}^{(256)} + s_{66}^{(256)} s_{111}^{(256)} s_{160}^{(256)} + s_{66}^{(256)} s_{94}^{(256)} + s_{66}^{(256)} s_{90}^{(256)} + s_{66}^{(256)} s_{75}^{(256)} + s_{66}^{(256)} s_{72}^{(256)} s_{160}^{(256)} + s_{66}^{(256)} s_{70}^{(256)} + s_{66}^{(256)} s_{67}^{(256)} + s_{66}^{(256)} s_{67}^{(256)} s_{160}^{(256)} + s_{65}^{(256)} + s_{65}^{(256)} s_{193}^{(256)} + s_{65}^{(256)} s_{160}^{(256)} + s_{65}^{(256)} s_{157}^{(256)} + s_{65}^{(256)} s_{155}^{(256)}$$

$$+ s_{65}^{(256)} s_{154}^{(256)} + s_{65}^{(256)} s_{153}^{(256)} + s_{65}^{(256)} s_{112}^{(256)} + s_{65}^{(256)} s_{111}^{(256)} + s_{65}^{(256)} s_{110}^{(256)} + s_{65}^{(256)} s_{109}^{(256)} + s_{65}^{(256)} s_{108}^{(256)} + s_{65}^{(256)} s_{73}^{(256)} + s_{65}^{(256)} s_{66}^{(256)} + s_{64}^{(256)} s_{193}^{(256)} + s_{64}^{(256)} s_{154}^{(256)} + s_{63}^{(256)} + s_{63}^{(256)} s_{161}^{(256)} + s_{63}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{63}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{63}^{(256)} s_{160}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{63}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{63}^{(256)} s_{154}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{62}^{(256)} s_{193}^{(256)} + s_{62}^{(256)} s_{160}^{(256)} + s_{62}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{62}^{(256)} s_{154}^{(256)} + s_{62}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{62}^{(256)} s_{111}^{(256)} + s_{62}^{(256)} s_{66}^{(256)} + s_{62}^{(256)} s_{63}^{(256)} s_{193}^{(256)} + s_{62}^{(256)} s_{63}^{(256)} s_{160}^{(256)} + s_{62}^{(256)} s_{63}^{(256)} s_{154}^{(256)} + s_{61}^{(256)} s_{166}^{(256)} + s_{61}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{61}^{(256)} s_{158}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{61}^{(256)} s_{144}^{(256)} + s_{61}^{(256)} s_{115}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{112}^{(256)} + s_{61}^{(256)} s_{111}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{103}^{(256)} + s_{61}^{(256)} s_{99}^{(256)} + s_{61}^{(256)} s_{98}^{(256)} + s_{61}^{(256)} s_{72}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{69}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{69}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{69}^{(256)} s_{154}^{(256)} + s_{61}^{(256)} s_{67}^{(256)} + s_{61}^{(256)} s_{67}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{65}^{(256)} + s_{61}^{(256)} s_{64}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{64}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{64}^{(256)} s_{154}^{(256)} + s_{61}^{(256)} s_{63}^{(256)} s_{161}^{(256)} + s_{61}^{(256)} s_{62}^{(256)} + s_{61}^{(256)} s_{62}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{62}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{62}^{(256)} s_{154}^{(256)} + s_{60}^{(256)} + s_{60}^{(256)} s_{193}^{(256)} + s_{60}^{(256)} s_{160}^{(256)} + s_{60}^{(256)} s_{157}^{(256)} + s_{60}^{(256)} s_{155}^{(256)} + s_{60}^{(256)} s_{154}^{(256)} + s_{60}^{(256)} s_{153}^{(256)} + s_{60}^{(256)} s_{112}^{(256)} + s_{60}^{(256)} s_{111}^{(256)} + s_{60}^{(256)} s_{110}^{(256)} + s_{60}^{(256)} s_{109}^{(256)} + s_{60}^{(256)} s_{108}^{(256)} + s_{60}^{(256)} s_{73}^{(256)} + s_{60}^{(256)} s_{66}^{(256)} + s_{60}^{(256)} s_{61}^{(256)} + s_{59}^{(256)} + s_{59}^{(256)} s_{160}^{(256)} + s_{58}^{(256)} s_{161}^{(256)} + s_{58}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{58}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{58}^{(256)} s_{160}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{58}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{58}^{(256)} s_{154}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{58}^{(256)} s_{62}^{(256)} s_{193}^{(256)} + s_{58}^{(256)} s_{62}^{(256)} s_{160}^{(256)} + s_{58}^{(256)} s_{62}^{(256)} s_{154}^{(256)} + s_{58}^{(256)} s_{61}^{(256)} s_{161}^{(256)} + s_{57}^{(256)} + s_{57}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{160}^{(256)} s_{193}^{(256)} + s_{57}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{111}^{(256)} + s_{57}^{(256)} s_{66}^{(256)} + s_{57}^{(256)} s_{63}^{(256)} s_{193}^{(256)} + s_{57}^{(256)} s_{63}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{63}^{(256)} s_{154}^{(256)} + s_{57}^{(256)} s_{61}^{(256)} + s_{57}^{(256)} s_{61}^{(256)} s_{193}^{(256)} + s_{57}^{(256)} s_{61}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{61}^{(256)} s_{154}^{(256)} + s_{57}^{(256)} s_{58}^{(256)} s_{193}^{(256)} + s_{57}^{(256)} s_{58}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{58}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{166}^{(256)} + s_{56}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{160}^{(256)} s_{164}^{(256)} + s_{56}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{160}^{(256)} s_{161}^{(256)} s_{166}^{(256)} + s_{56}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{145}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{127}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{117}^{(256)} + s_{56}^{(256)} s_{117}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{115}^{(256)} s_{56}^{(256)} s_{115}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{113}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{112}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{112}^{(256)} s_{117}^{(256)} + s_{56}^{(256)} s_{112}^{(256)} s_{115}^{(256)} + s_{56}^{(256)} s_{112}^{(256)} s_{113}^{(256)} + s_{56}^{(256)} s_{86}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{82}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{74}^{(256)} + s_{56}^{(256)} s_{74}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{74}^{(256)} s_{112}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} s_{56}^{(256)} s_{69}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} s_{112}^{(256)} + s_{56}^{(256)} s_{68}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} s_{117}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} s_{115}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} s_{113}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} s_{74}^{(256)} + s_{56}^{(256)} s_{67}^{(256)} s_{69}^{(256)} + s_{56}^{(256)} s_{64}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{64}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{64}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{63}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} s_{63}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} s_{69}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} s_{64}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} s_{62}^{(256)} + s_{56}^{(256)} s_{58}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{58}^{(256)} s_{62}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{63}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{61}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{58}^{(256)} + s_{55}^{(256)} s_{160}^{(256)} + s_{55}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{52}^{(256)} + s_{51}^{(256)} + s_{51}^{(256)} s_{160}^{(256)} + s_{51}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{49}^{(256)} + s_{49}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{48}^{(256)} + s_{47}^{(256)} + s_{47}^{(256)} s_{111}^{(256)} + s_{47}^{(256)} s_{66}^{(256)} + s_{46}^{(256)} + s_{46}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{45}^{(256)} + s_{43}^{(256)} + s_{42}^{(256)} + s_{42}^{(256)}$$

$s_{193}^{(256)} + s_{42}^{(256)}\ s_{160}^{(256)} + s_{42}^{(256)}\ s_{154}^{(256)} + s_{42}^{(256)}\ s_{61}^{(256)} + s_{41}^{(256)} + s_{40}^{(256)} + s_{40}^{(256)}\ s_{160}^{(256)}$
$s_{161}^{(256)} + s_{39}^{(256)} + s_{37}^{(256)} + s_{37}^{(256)}\ s_{193}^{(256)} + s_{37}^{(256)}\ s_{154}^{(256)} + s_{37}^{(256)}\ s_{61}^{(256)}$.

$p_{I_2} = s_{193}^{(256)} + s_{164}^{(256)}\ s_{193}^{(256)} + s_{163}^{(256)} + s_{162}^{(256)} + s_{161}^{(256)}\ s_{193}^{(256)} + s_{161}^{(256)}\ s_{167}^{(256)} +$
$s_{161}^{(256)}\ s_{162}^{(256)}\ s_{193}^{(256)} + s_{160}^{(256)} + s_{160}^{(256)}\ s_{167}^{(256)} + s_{160}^{(256)}\ s_{164}^{(256)} + s_{160}^{(256)}\ s_{161}^{(256)} + s_{160}$
$s_{161}^{(256)}\ s_{193}^{(256)} + s_{160}^{(256)}\ s_{161}^{(256)}\ s_{167}^{(256)}\ s_{193}^{(256)} + s_{160}^{(256)}\ s_{161}^{(256)}\ s_{162}^{(256)} + s_{159}^{(256)} + s_{158}^{(256)} +$
$s_{158}^{(256)}\ s_{161}^{(256)} + s_{157}^{(256)}\ s_{161}^{(256)} + s_{157}^{(256)}\ s_{160}^{(256)} + s_{156}^{(256)}\ s_{160}^{(256)} + s_{155}^{(256)}\ s_{161}^{(256)} + s_{155}$
$s_{160}^{(256)} + s_{154}^{(256)} + s_{154}^{(256)}\ s_{164}^{(256)} + s_{154}^{(256)}\ s_{161}^{(256)} + s_{154}^{(256)}\ s_{161}^{(256)}\ s_{162}^{(256)} + s_{154}^{(256)}\ s_{160}$
$+ s_{154}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{154}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)}\ s_{167}^{(256)} + s_{153}^{(256)}\ s_{161}^{(256)} + s_{153}^{(256)}\ s_{160}^{(256)} +$
$s_{152}^{(256)}\ s_{160}^{(256)} + s_{151}^{(256)} + s_{151}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{150}^{(256)} + s_{145}^{(256)}\ s_{160}^{(256)} + s_{144}^{(256)}\ s_{160}$
$+ s_{142}^{(256)}\ s_{161}^{(256)} + s_{141}^{(256)} + s_{139}^{(256)} + s_{135}^{(256)} + s_{134}^{(256)}\ s_{160}^{(256)} + s_{133}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} +$
$s_{132}^{(256)}\ s_{160}^{(256)} + s_{130}^{(256)}\ s_{160}^{(256)} + s_{126}^{(256)} + s_{125}^{(256)} + s_{123}^{(256)} + s_{122}^{(256)} + s_{120}^{(256)}\ s_{160}^{(256)} +$
$s_{119}^{(256)}\ s_{160}^{(256)} + s_{118}^{(256)} + s_{118}^{(256)}\ s_{160}^{(256)} + s_{118}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{118}^{(256)}\ s_{154}^{(256)}\ s_{160}^{(256)} +$
$s_{117}^{(256)} + s_{117}^{(256)}\ s_{193}^{(256)} + s_{117}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{117}^{(256)}\ s_{154}^{(256)} + s_{117}^{(256)}\ s_{154}^{(256)}\ s_{160}^{(256)} +$
$s_{116}^{(256)}\ s_{160}^{(256)} + s_{115}^{(256)} + s_{115}^{(256)}\ s_{193}^{(256)} + s_{115}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{115}^{(256)}\ s_{154}^{(256)} + s_{115}^{(256)}$
$s_{154}^{(256)}\ s_{160}^{(256)} + s_{114}^{(256)} + s_{114}^{(256)}\ s_{160}^{(256)} + s_{113}^{(256)} + s_{113}^{(256)}\ s_{193}^{(256)} + s_{113}^{(256)}\ s_{160}^{(256)} + s_{113}$
$s_{160}^{(256)}\ s_{193}^{(256)} + s_{113}^{(256)}\ s_{154}^{(256)} + s_{113}^{(256)}\ s_{154}^{(256)}\ s_{160}^{(256)} + s_{112}^{(256)} + s_{112}^{(256)}\ s_{161}^{(256)} + s_{111}^{(256)}$
$+ s_{111}^{(256)}\ s_{167}^{(256)} + s_{111}^{(256)}\ s_{161}^{(256)} + s_{111}^{(256)}\ s_{161}^{(256)}\ s_{193}^{(256)} + s_{111}^{(256)}\ s_{160}^{(256)} + s_{111}^{(256)}\ s_{160}^{(256)}$
$s_{161}^{(256)} + s_{111}^{(256)}\ s_{154}^{(256)}\ s_{161}^{(256)} + s_{111}^{(256)}\ s_{144}^{(256)} + s_{111}^{(256)}\ s_{130}^{(256)} + s_{110}^{(256)}\ s_{161}^{(256)} + s_{110}^{(256)}$
$s_{160}^{(256)} + s_{109}^{(256)} + s_{109}^{(256)}\ s_{161}^{(256)} + s_{108}^{(256)}\ s_{161}^{(256)} + s_{107}^{(256)}\ s_{160}^{(256)} + s_{106}^{(256)} + s_{106}^{(256)}\ s_{160}^{(256)}$
$s_{161}^{(256)} + s_{105}^{(256)} + s_{101}^{(256)} + s_{100}^{(256)} + s_{100}^{(256)}\ s_{160}^{(256)} + s_{99}^{(256)}\ s_{160}^{(256)} + s_{99}^{(256)}\ s_{111}^{(256)} +$
$s_{98}^{(256)} + s_{97}^{(256)} + s_{97}^{(256)}\ s_{161}^{(256)} + s_{92}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{91}^{(256)}\ s_{160}^{(256)} + s_{90}^{(256)} + s_{89}$
$+ s_{88}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{86}^{(256)}\ s_{160}^{(256)} + s_{85}^{(256)} + s_{85}^{(256)}\ s_{111}^{(256)} + s_{82}^{(256)} + s_{81}^{(256)} +$
$s_{81}^{(256)}\ s_{160}^{(256)} + s_{80}^{(256)} + s_{80}^{(256)}\ s_{160}^{(256)} + s_{79}^{(256)} + s_{78}^{(256)} + s_{78}^{(256)}\ s_{160}^{(256)} + s_{76}^{(256)} +$
$s_{75}^{(256)}\ s_{160}^{(256)} + s_{74}^{(256)} + s_{74}^{(256)}\ s_{193}^{(256)} + s_{74}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{74}^{(256)}\ s_{154}^{(256)} + s_{74}^{(256)}$
$s_{154}^{(256)}\ s_{160}^{(256)} + s_{73}^{(256)} + s_{73}^{(256)}\ s_{161}^{(256)} + s_{73}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{73}^{(256)}\ s_{154}^{(256)}\ s_{160}^{(256)} +$
$s_{72}^{(256)}\ s_{161}^{(256)} + s_{72}^{(256)}\ s_{160}^{(256)} + s_{71}^{(256)}\ s_{160}^{(256)} + s_{70}^{(256)}\ s_{161}^{(256)} + s_{70}^{(256)}\ s_{160}^{(256)} + s_{69}^{(256)}$
$+ s_{69}^{(256)}\ s_{193}^{(256)} + s_{69}^{(256)}\ s_{160}^{(256)} + s_{69}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{69}^{(256)}\ s_{154}^{(256)} + s_{69}^{(256)}\ s_{154}$
$s_{160}^{(256)} + s_{68}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{68}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{68}^{(256)}\ s_{154}^{(256)}\ s_{160}^{(256)} + s_{68}^{(256)}$
$s_{150}^{(256)} + s_{68}^{(256)}\ s_{148}^{(256)} + s_{68}^{(256)}\ s_{146}^{(256)} + s_{68}^{(256)}\ s_{117}^{(256)} + s_{68}^{(256)}\ s_{115}^{(256)} + s_{68}^{(256)}\ s_{113}^{(256)}$
$+ s_{68}^{(256)}\ s_{107}^{(256)} + s_{68}^{(256)}\ s_{102}^{(256)} + s_{68}^{(256)}\ s_{74}^{(256)} + s_{68}^{(256)}\ s_{69}^{(256)} + s_{67}^{(256)} + s_{67}^{(256)}\ s_{161}^{(256)}$
$+ s_{67}^{(256)}\ s_{160}^{(256)} + s_{66}^{(256)} + s_{66}^{(256)}\ s_{167}^{(256)} + s_{66}^{(256)}\ s_{161}^{(256)}\ s_{193}^{(256)} + s_{66}^{(256)}\ s_{160}^{(256)} + s_{66}^{(256)}$
$s_{160}^{(256)}\ s_{161}^{(256)} + s_{66}^{(256)}\ s_{154}^{(256)}\ s_{161}^{(256)} + s_{66}^{(256)}\ s_{144}^{(256)} + s_{66}^{(256)}\ s_{130}^{(256)} + s_{66}^{(256)}\ s_{99}^{(256)} +$
$s_{66}^{(256)}\ s_{85}^{(256)} + s_{66}^{(256)}\ s_{68}^{(256)} + s_{65}^{(256)}\ s_{161}^{(256)} + s_{65}^{(256)}\ s_{160}^{(256)} + s_{63}^{(256)} + s_{63}^{(256)}\ s_{193}^{(256)}$
$+ s_{63}^{(256)}\ s_{161}^{(256)} + s_{63}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{63}^{(256)}\ s_{160}^{(256)}\ s_{161}^{(256)} + s_{63}^{(256)}\ s_{154}^{(256)} + s_{63}^{(256)}$
$s_{154}^{(256)}\ s_{160}^{(256)} + s_{63}^{(256)}\ s_{150}^{(256)} + s_{63}^{(256)}\ s_{148}^{(256)} + s_{63}^{(256)}\ s_{146}^{(256)} + s_{63}^{(256)}\ s_{117}^{(256)} + s_{63}^{(256)}$
$s_{115}^{(256)} + s_{63}^{(256)}\ s_{113}^{(256)} + s_{63}^{(256)}\ s_{107}^{(256)} + s_{63}^{(256)}\ s_{102}^{(256)} + s_{63}^{(256)}\ s_{74}^{(256)} + s_{63}^{(256)}\ s_{69}^{(256)}$
$+ s_{63}^{(256)}\ s_{66}^{(256)} + s_{62}^{(256)}\ s_{160}^{(256)} + s_{62}^{(256)}\ s_{160}^{(256)}\ s_{193}^{(256)} + s_{62}^{(256)}\ s_{157}^{(256)} + s_{62}^{(256)}\ s_{155}^{(256)}$

$$+ s_{62}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{62}^{(256)} s_{153}^{(256)} + s_{62}^{(256)} s_{112}^{(256)} + s_{62}^{(256)} s_{111}^{(256)} + s_{62}^{(256)} s_{110}^{(256)}$$
$$+ s_{62}^{(256)} s_{109}^{(256)} + s_{62}^{(256)} s_{108}^{(256)} + s_{62}^{(256)} s_{73}^{(256)} + s_{62}^{(256)} s_{68}^{(256)} s_{160}^{(256)} + s_{62}^{(256)} s_{66}^{(256)}$$
$$+ s_{62}^{(256)} s_{63}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} + s_{61}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{167}^{(256)} + s_{61}^{(256)} s_{164}^{(256)} + s_{61}^{(256)}$$
$$s_{161}^{(256)} + s_{61}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{61}^{(256)} s_{160}^{(256)} + s_{61}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{61}^{(256)} s_{154}^{(256)} +$$
$$s_{61}^{(256)} s_{154}^{(256)} s_{161}^{(256)} + s_{61}^{(256)} s_{68}^{(256)} + s_{61}^{(256)} s_{63}^{(256)} + s_{61}^{(256)} s_{62}^{(256)} + s_{60}^{(256)} + s_{60}^{(256)}$$
$$s_{161}^{(256)} + s_{60}^{(256)} s_{160}^{(256)} + s_{58}^{(256)} s_{193}^{(256)} + s_{58}^{(256)} s_{154}^{(256)} + s_{57}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{160}^{(256)}$$
$$s_{193}^{(256)} + s_{57}^{(256)} s_{157}^{(256)} + s_{57}^{(256)} s_{155}^{(256)} + s_{57}^{(256)} s_{154}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{153}^{(256)} + s_{57}^{(256)}$$
$$s_{112}^{(256)} + s_{57}^{(256)} s_{111}^{(256)} + s_{57}^{(256)} s_{110}^{(256)} + s_{57}^{(256)} s_{109}^{(256)} + s_{57}^{(256)} s_{108}^{(256)} + s_{57}^{(256)} s_{73}^{(256)}$$
$$+ s_{57}^{(256)} s_{68}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{66}^{(256)} + s_{57}^{(256)} s_{63}^{(256)} s_{160}^{(256)} + s_{57}^{(256)} s_{61}^{(256)} + s_{56}^{(256)}$$
$$s_{193}^{(256)} + s_{56}^{(256)} s_{167}^{(256)} + s_{56}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{161}^{(256)} s_{193}^{(256)} + s_{56}^{(256)} s_{161}^{(256)} s_{162}^{(256)}$$
$$+ s_{56}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{160}^{(256)} s_{161}^{(256)} s_{167}^{(256)} + s_{56}^{(256)} s_{154}^{(256)} + s_{56}^{(256)} s_{154}^{(256)} s_{161}^{(256)}$$
$$+ s_{56}^{(256)} s_{118}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{117}^{(256)} + s_{56}^{(256)} s_{117}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{115}^{(256)} + s_{56}^{(256)}$$
$$s_{115}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{113}^{(256)} + s_{56}^{(256)} s_{113}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{111}^{(256)} s_{161}^{(256)} + s_{56}^{(256)}$$
$$s_{74}^{(256)} + s_{56}^{(256)} s_{74}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{73}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{69}^{(256)} + s_{56}^{(256)} s_{69}^{(256)}$$
$$s_{160}^{(256)} + s_{56}^{(256)} s_{68}^{(256)} + s_{56}^{(256)} s_{68}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{66}^{(256)} s_{161}^{(256)} + s_{56}^{(256)} s_{63}^{(256)}$$
$$s_{160}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} + s_{56}^{(256)} s_{62}^{(256)} s_{160}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} + s_{56}^{(256)} s_{61}^{(256)} s_{161}^{(256)} +$$
$$s_{56}^{(256)} s_{58}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} + s_{56}^{(256)} s_{57}^{(256)} s_{160}^{(256)} + s_{54}^{(256)} + s_{54}^{(256)} s_{160}^{(256)} s_{161}^{(256)} +$$
$$s_{52}^{(256)} + s_{48}^{(256)} + s_{47}^{(256)} + s_{45}^{(256)} s_{160}^{(256)} s_{161}^{(256)} + s_{43}^{(256)} + s_{42}^{(256)} s_{160}^{(256)} + s_{39}^{(256)} +$$
$$s_{38}^{(256)} + s_{37}^{(256)} s_{160}^{(256)}.$$