Massive Superpoly Recovery with a Meet-in-the-Middle Framework

Improved Cube Attacks on Trivium and Kreyvium

Jiahui He[1,3], Kai Hu[1,3,4], Hao Lei[1,3], and Meiqin Wang[1,2,3](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China
{hejiahui2020,leihao}@mail.sdu.edu.cn, {kai.hu,mqwang}@sdu.edu.cn
[2] Quan Cheng Shandong Laboratory, Jinan, China
[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[4] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore

Abstract. The cube attack extracts the information of secret key bits by recovering the coefficient called superpoly in the output bit with respect to a subset of plaintexts/IV, which is called a cube. While the division property provides an efficient way to detect the structure of the superpoly, superpoly recovery could still be prohibitively costly if the number of rounds is sufficiently high. In particular, Core Monomial Prediction (CMP) was proposed at ASIACRYPT 2022 as a scaled-down version of Monomial Prediction (MP), which sacrifices accuracy for efficiency but ultimately gets stuck at 848 rounds of Trivium.

In this paper, we provide new insights into CMP by elucidating the algebraic meaning to the core monomial trails. We prove that it is sufficient to recover the superpoly by extracting all the core monomial trails, an approach based solely on CMP, thus demonstrating that CMP can achieve perfect accuracy as MP does. We further reveal that CMP is still MP in essence, but with variable substitutions on the target function. Inspired by the divide-and-conquer strategy that has been widely used in previous literature, we design a meet-in-the-middle (MITM) framework, in which the CMP-based approach can be embedded to achieve a speedup.

To illustrate the power of these new techniques, we apply the MITM framework to Trivium, Grain-128AEAD and Kreyvium. As a result, not only can the previous computational cost of superpoly recovery be reduced (e.g., 5x faster for superpoly recovery on 192-round Grain-128AEAD), but we also succeed in recovering superpolies for up to 851 rounds of Trivium and up to 899 rounds of Kreyvium. This surpasses the previous best results by respectively 3 and 4 rounds. Using the memory-efficient Möbius transform proposed at EUROCRYPT 2021, we can perform key recovery attacks on target ciphers, even though the superpoly

Due to page limits, all appendixes and some tables of this paper are provided in our full version [21].

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14651, pp. 368–397, 2024.
https://doi.org/10.1007/978-3-031-58716-0_13

may contain over 2^{40} monomials. This leads to the best cube attacks on the target ciphers.

Keywords: Cube Attack · Superpoly · Trivium · Grain-128AEAD · Kreyvium · Division Property · Monomial Prediction · Core Monomial Prediction

1 Introduction

Cube Attack. Cube attack was proposed by Dinar and Shamir [13] at EUROCRYPT 2009 and has become one of the general cryptanalytic techniques against symmetric ciphers. Since its proposal, it has been applied to analyze various symmetric ciphers [4,12,14,17,25,27,30,34,36]. In particular, against the Ascon cipher [15] that is selected by NIST for future standardization of the lightweight cryptography, the cube attack shows outstanding effectiveness [5,28,32,33]. The cube attack exploits the fact that each output bit of a cipher can be expressed as a Boolean function of the key bits and plaintext/IV bits. For a randomly chosen set I of indices of the plaintext/IV bits, we can form a monomial t_I as the product of the bits indexed by I. After fixing the plaintext/IV bits outside of I to constant values, we attempt to recover the polynomial related to key bits, called *superpoly*, that is multiplied by t_I in the output bit. If the superpoly is obtained, the value of the superpoly can be computed by summing over a structure called *cube*, denoted by C_I, which consists of all possible combinations of values that those plaintext/IV bits indexed by I can take. Subsequently, the information of key bits may be deduced by solving the equation built from the value of the superpoly.

Division Property. The original division property was proposed at EUROCRYPT 2015 [42] as a generalization of the integral property. By tracking the integral characteristics more accurately with the division property, the long-standing cipher MISTY1 was broken theoretically for the first time [41]. At FSE 2016, the word-based division property was refined into bit-based division property [44], with which the experimentally discovered integral characteristics for bit-based block cipher SIMON32 and Simeck32 are proved for the first time. The corresponding MILP model for deducing the division property was proposed by Xiang et al. [50] at ASIACRYPT 2016, where the propagation rules of basic operations are encoded as linear equalities. This MILP method has been used to improve the integral attack against many other ciphers [18,37,38,47].

Exact Superpoly Recovery. Initially, the cube attack treats the target cipher as a black box [13,17,31], and the structure of the superpoly can only be detected by experimental tests, thus limiting the superpoly to simple forms (e.g., linear or quadratic). Later, the Conventional Bit-based Division Property [44] was introduced into the cube attack [43], so that those secret variables that are not involved in the superpoly could be efficiently identified. In the same year, Liu et al. [29] discovered constant superpolies with the numeric mapping technique.

When determining whether a monomial exists in the superpoly, however, the bit-based division property may produce false positives, so a further series of work was carried out to improve its accuracy. In [48], Wang et al. took the cancellation of constant 1 bits into account and proposed the flag technique to improve the precision of the bit-based division property. At ASIACRYPT 2019, Wang et al. [49] recovered the exact superpoly for the first time with the pruning technique combined with the three-subset division property. However, this technique is limited by its assumption that almost all elements in the 1-subset can be pruned. The inaccuracy problem was finally resolved by Hao et al. in [20], where the unknown subset was discarded and the cancellation of the 1-subset vectors was transformed into the parity of the number of division trails. This new variant of division property is called three-subset division property without unknown subset (3SDPwoU), which is interpreted as the so-called monomial prediction (MP) from a purely algebraic viewpoint [24]. The MILP model of MP can be further optimized if we represent the propagation of MP as a directed graph [10]. Both the MP and 3SDPwoU will encounter a bottleneck if the number of division trails exceeds the upper limit of the number of solutions that can be enumerated by an MILP solver. To this end, Hu et al. [23] proposed an improved framework called *nested monomial prediction* to recover massive superpolies, which can be viewed as a recursive application of the divide-and-conquer strategy. Recently, He et al. [22] proposed the core monomial prediction (CMP), which is claimed to sacrifice accuracy for efficiency compared to MP, thus significantly reducing the computational cost of superpoly recovery.

Motivation. MP can achieve perfect accuracy because we can determine whether a monomial appears in the output bit by evaluating the parity of the number of monomial trails, but what information beyond the existence can be brought to the table by core monomial trails remains unknown. Even in [22], the authors only exploit the existence of a core monomial trail, but do not show how we can benefit from all core monomial trails. Also, we notice that the definition of CMP naturally lends itself to forward propagations, i.e., derivation from round 0 to higher rounds, while recovering the superpoly with MP does not have such a property, because it does not impose any constraints on the secret variables. Forward propagation often implies the possibility of improving accuracy and efficiency, as in the pruning technique [49] where the division property was propagated and filtered from the bottom up round by round.

Our Contributions. This paper aims to recover superpolies for more initialization rounds of stream ciphers, for which we propose purely CMP-based approach and framework to improve the efficiency of superpoly recovery.

– *Refinement of CMP theory: new CMP-based approach.* It was believed in previous works that CMP is a scaled-down version of MP that sacrifices accuracy for efficiency. However, in this paper, we prove that CMP is also perfectly accurate, as the three-subset division property without unknown subset and monomial prediction, which refines the division property family. After investigating how each core monomial trail contributes to the composition of the superpoly, we demonstrate that it is sufficient to recover the exact superpoly by extracting all core monomial trails.

– *Meet-in-the-middle (MITM) framework.* Inspired by the divide-and-conquer strategy, we show that it is possible to split a complex problem of superpoly recovery into multiple simpler problems of superpoly recovery, by performing forward or backward propagation of CMP. By using these two types of propagation interchangeably and recursively, we can embed our CMP-based approach into an MITM framework to further achieve a speedup.

Since it has been shown in [22] that the MILP model for CMP is simpler than that for MP, we claim that our purely CMP-based approach and framework perform better than the method in [22] that combines CMP and MP. The most intuitive evidence for this is that we can reproduce previous superpolies at a much smaller computational cost. For Trivium, we halve the time it took to recover the superpolies (see [21, Table 6]); for 192-round Grain-128AEAD, we reduce the time of superpoly recovery to about $\frac{1}{5}$ of the original (see Sect. 5.2).

Notably, our MITM framework enables us to extend the number of initialization rounds of superpoly recovery for several prominent ciphers, including Trivium (ISO/IEC standard [1,3]) and Kreyvium (designed for Fully Homomorphic Encryption [8]). Ultimately, we succeed in recovering the superpolies for up to 851-round Trivium and up to 899-round Kreyvium, extending the previous best results by 3 and 4 rounds, respectively. With the help of the memory-efficient Möbius transform proposed at EUROCRYPT 2021 [11], we can utilize the recovered superpolies to perform key recovery at a complexity lower than exhaustive search, leading to the best results of cube attacks against target ciphers.

The summary of our cube attack results are provided in Table 1. The source codes for superpoly recovery, as well as some recovered superpolies, can be found in our anonymous git repository

https://github.com/viocently/sdfkjxu192lc78-s0.

2 Cube Attack and Monomial Prediction

2.1 Notations and Definitions

In this paper, we use bold italic or Greek letters to represent binary vectors. For a binary vector $\boldsymbol{x} \in \mathbb{F}_2^m$, its i^{th} bit is represented by $x[i]$; the Hamming weight of $\boldsymbol{x}$ is calculated as $wt(\boldsymbol{x}) = \sum_{i=0}^{m-1} x[i]$; the indices of ones in $\boldsymbol{x}$ are represented by the set $\mathsf{Ind}[\boldsymbol{x}] = \{i \mid x[i] = 1\}$. Given two binary vectors $\boldsymbol{x} \in \mathbb{F}_2^m$ and $\boldsymbol{u} \in \mathbb{F}_2^m$, we use $\boldsymbol{x}^{\boldsymbol{u}}$ to represent $\prod_{i=0}^{m-1} x[i]^{u[i]}$; $\boldsymbol{x}[\boldsymbol{u}] = \big(x[i_0], \ldots, x[i_{wt(\boldsymbol{u})-1}]\big) \in \mathbb{F}_2^{wt(\boldsymbol{u})}$ denotes a sub-vector of $\boldsymbol{x}$ with respect to $\boldsymbol{u}$, where $i_0, \ldots, i_{wt(\boldsymbol{u})-1}$ are elements of $\mathsf{Ind}[\boldsymbol{u}]$ in ascending order. We define $\boldsymbol{x} \succeq \boldsymbol{u}$ (resp. $\boldsymbol{x} \succ \boldsymbol{u}$) if $x[i] \geq u[i]$ (resp. $x[i] > u[i]$) for all i and $\boldsymbol{x} \preceq \boldsymbol{u}$ (resp. $\boldsymbol{x} \prec \boldsymbol{u}$) if $x[i] \leq u[i]$ (resp. $x[i] < u[i]$) for all i. The concatenation of $\boldsymbol{x}$ and $\boldsymbol{u}$ is denoted by $\boldsymbol{x} \| \boldsymbol{u}$. The bitwise operations AND,OR,XOR,NOT are denoted by $\wedge, \vee, \oplus, \neg$ respectively and can be applied

Table 1. Summary of the key recovery attacks on Trivium and Kreyvium

Cipher	Rounds	#Cube*	Cube size	Attack type	Data	Time	Reference
Trivium	672	63	12	Cube	$2^{18.6}$	2^{17}	[13]
	709	80	22–23	Cube	2^{23}	$2^{29.14}$	[31]
	767	35	28–31	Cube	2^{31}	2^{45}	[13]
	784	42	30–33	Cube	2^{33}	2^{39}	[17]
	799	18	32–37	Cube	2^{38}	2^{62}	[17]
	802	8	34–37	Cube	2^{37}	2^{72}	[51]
	805	42	32–38	Cube	2^{38}	$2^{41.4}$	[52]
	805	28	28	Correlation Cube	2^{28}	2^{73}	[30]
	806	16	34–37	Cube	$2^{38.64}$	2^{64}	[52]
	806	29	34–37	Cube	2^{39}	2^{39}	[40]
	808	37	39–41	Cube	2^{44}	$2^{44.58}$	[40]
	810	39	40–42	Cube	2^{44}	$2^{44.17}$	[26]
	815	35	44–46	Cube	2^{47}	$2^{47.32}$	[9]
	820	30	48–51	Cube	2^{53}	$2^{53.17}$	[9]
	820†	2^{13}	38	Correlation Cube	2^{51}	2^{60}	[45]
	825	31	49–52	Cube	2^{53}	$2^{53.09}$	[26]
	825†	2^{12}	41	Correlation Cube	2^{53}	2^{60}	[45]
	830†	2^{13}	41	Correlation Cube	2^{54}	2^{60}	[45]
	832	1	72	Cube	2^{72}	2^{79}	[43, 49]
	835	41	35	Correlation Cube	2^{35}	2^{75}	[30]
	840	1	78	Cube	2^{78}	$2^{79.6}$	[20]
	840	3	75	Cube	$2^{76.6}$	$2^{77.8}$	[24]
	840	6	47–62	Cube	2^{62}	$2^{76.32}$	[23]
	841	1	78	Cube	2^{78}	$2^{79.6}$	[20]
	841	2	76	Cube	2^{77}	$2^{78.6}$	[24]
	841	3	56–76	Cube	2^{76}	2^{78}	[23]
	842	1	78	Cube	2^{78}	$2^{79.6}$	[20]
	842	2	76	Cube	2^{77}	$2^{78.6}$	[24]
	842	3	56–76	Cube	2^{76}	2^{78}	[23]
	843	2	78	Cube	2^{78}	$2^{79.6}$	[40]
	843	5	56–76	Cube	2^{56}	2^{77}	[23]
	844	2	54–55	Cube	2^{56}	2^{78}	[23]
	845	2	54–55	Cube	2^{56}	2^{78}	[23]
	846	6	51–54	Cube	2^{51}	2^{79}	[22]
	847	2	52–53	Cube	2^{52}	2^{79}	[22]
	848	1	52	Cube	2^{52}	2^{79}	[22]
	849	**2**	**44**	**Cube**	$\mathbf{2^{44}}$	$\mathbf{2^{79}}$	**Sect.** 5.1
	850	**1**	**44**	**Cube**	$\mathbf{2^{44}}$	$\mathbf{2^{79}}$	**Sect.** 5.1
	851	**1**	**44**	**Cube**	$\mathbf{2^{44}}$	$\mathbf{2^{79}}$	**Sect.** 5.1
Kreyvium	≤ 893	-	≤ 119	Cube	$\leq 2^{119}$	$\leq 2^{127}$	[19, 20, 40, 43, 48]
	894	1	119	Cube	2^{119}	2^{127}	[23]
	895	1	120	Cube	2^{120}	2^{127}	[22]
	896	**2**	**123–124**	**Cube**	$\mathbf{2^{123}}$	$\mathbf{2^{127}}$	**Sect.** 5.3
	897	**1**	**124**	**Cube**	$\mathbf{2^{124}}$	$\mathbf{2^{127}}$	**Sect.** 5.3
	898*	2	126	Cube	2^{127}	$2^{127.58}$	[16]
	898	**1**	**124**	**Cube**	$\mathbf{2^{124}}$	$\mathbf{2^{127}}$	**Sect.** 5.3
	899*	1	126	Cube	2^{126}	$2^{127.58}$	[16]
	899	**1**	**124**	**Cube**	$\mathbf{2^{124}}$	$\mathbf{2^{127}}$	**Sect.** 5.3
	900*	1	126	Cube	2^{126}	$2^{127.58}$	[16]

* #Cube represents the number of cubes whose superpolies are recovered, but this may not be equal to the number of cubes eventually used in the key recovery attack.
† The 820-, 825- and 830-round attacks in [45] work for only $2^{79.8}$, $2^{79.7}$ and $2^{79.3}$ of the keys in the key space, respectively.
* We notice that after our submission, the superpolies of up to 900 rounds of Kreyvium have been recovered in [16], where the complexity analysis is based on the concept of implementation dependency. This leads to the cube attacks against up to 900 rounds of Kreyvium.

to bits or binary vectors. As a special case, we use $\mathbf{0}$ and $\mathbf{1}$ to refer to the all-zero vector and the all-one vector, respectively.

We add subscripts to distinguish n binary vectors that use the same letter (e.g., $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}$) and superscripts to represent the binary vectors associated with a specific number of rounds (e.g., $\boldsymbol{x}^i$ is a binary vector at round i). For clarity, we will use $\pi(\boldsymbol{x}, \boldsymbol{u})$ instead of $\boldsymbol{x}^{\boldsymbol{u}}$ when both $\boldsymbol{x}$ and $\boldsymbol{u}$ have superscripts or subscripts.

When introducing a concrete MILP model, we use regular italic letters to represent MILP variables, and similarly we add superscripts to denote the number of rounds if they correspond to a certain round and add subscripts to distinguish them if they use a same letter.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function whose *algebraic normal form* (ANF) is represented as $f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{u} \in \mathbb{F}_2^n} a_{\boldsymbol{u}} \boldsymbol{x}^{\boldsymbol{u}}$, where $a_{\boldsymbol{u}} \in \mathbb{F}_2$ and $\boldsymbol{x} \in \mathbb{F}_2^n$. $\boldsymbol{x}^{\boldsymbol{u}}$ is called a monomial. We say a monomial $\boldsymbol{x}^{\boldsymbol{u}}$ appears in f, if the coefficient of $\boldsymbol{x}^{\boldsymbol{u}}$ in f is 1, i.e., $a_{\boldsymbol{u}} = 1$, and we denote this case by $\boldsymbol{x}^{\boldsymbol{u}} \to f$; otherwise, we denote the absence of $\boldsymbol{x}^{\boldsymbol{u}}$ in f by $\boldsymbol{x}^{\boldsymbol{u}} \not\to f$.

Let $\boldsymbol{f} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a vectorial Boolean function with $\boldsymbol{x}$ and $\boldsymbol{y}$ being the input and output, respectively. Given a monomial $\boldsymbol{y}^{\boldsymbol{v}}$ of $\boldsymbol{y}$, we can derive a Boolean function g of $\boldsymbol{x}$ by taking $\boldsymbol{y}^{\boldsymbol{v}}$ as the output of g. In the remainder of the paper, notations of the form $\boldsymbol{y}^{\boldsymbol{v}}$ may represent either a monomial of $\boldsymbol{y}$ or the Boolean function g derived from it, depending on the context. We then write $\boldsymbol{x}^{\boldsymbol{u}} \to \boldsymbol{y}^{\boldsymbol{v}}$ if $\boldsymbol{x}^{\boldsymbol{u}} \to g$; otherwise we write $\boldsymbol{x}^{\boldsymbol{u}} \not\to \boldsymbol{y}^{\boldsymbol{v}}$. The ANF of g is denoted by $\mathsf{Expr}\langle \boldsymbol{y}^{\boldsymbol{v}}, \boldsymbol{x} \rangle$, which represents a Boolean polynomial of $\boldsymbol{x}$ determined by $\boldsymbol{y}^{\boldsymbol{v}}$. For a polynomial p of $\boldsymbol{y}$, $\mathsf{Expr}\langle p, \boldsymbol{x} \rangle$ is defined as the summation of $\mathsf{Expr}\langle \boldsymbol{y}^{\boldsymbol{v}}, \boldsymbol{x} \rangle$ over all monomials $\boldsymbol{y}^{\boldsymbol{v}}$ appearing in p. We would like to point out that when we use above notations, we may not give $\boldsymbol{f}$ explicitly, so the readers should be able to derive $\boldsymbol{f}$ from the context on their own.

2.2 Cube Attack

The cube attack was proposed by Dinur and Shamir at EUROCRYPT 2009 [13] as an extension of the higher-order differential attack. Given a cipher with secret variables $\boldsymbol{k} \in \mathbb{F}_2^n$ and public variables $\boldsymbol{v} \in \mathbb{F}_2^m$ being the input, any output bit can be represented as a Boolean function of $\boldsymbol{k}$ and $\boldsymbol{v}$, denoted by $f(\boldsymbol{k}, \boldsymbol{v})$.

Given $I \subseteq \{0, \ldots, m-1\}$ as a set of indices of the public variables, we can uniquely express $f(\boldsymbol{k}, \boldsymbol{v})$ as

$$f(\boldsymbol{k}, \boldsymbol{v}) = p(\boldsymbol{k}, \boldsymbol{v}) \cdot t_I + q(\boldsymbol{k}, \boldsymbol{v}),$$

where $t_I = \prod_{i \in I} v[i]$, $p(\boldsymbol{k}, \boldsymbol{v})$ only relates to $v[s]$'s ($s \notin I$) and the secret variables $\boldsymbol{k}$, and each monomial appearing in $q(\boldsymbol{k}, \boldsymbol{v})$ misses at least one variable from $\{v[i] \mid i \in I\}$. I is called *cube indices*, whose size is denoted by $|I|$. If we assign all the possible combinations of 0/1 values to $v[j]$'s ($j \in I$) and leave $v[s]$'s ($s \notin I$) undetermined, we can determine a set C_I from I, which is called *cube*. The coefficient $p(\boldsymbol{k}, \boldsymbol{v})$ is called the *superpoly* of the cube C_I or the cube indices

I, which can be computed by summing the output bit $f(\boldsymbol{k}, \boldsymbol{v})$ over the cube, namely

$$p(\boldsymbol{k}, \boldsymbol{v}) = \sum_{v \in C_I} f(\boldsymbol{k}, \boldsymbol{v}).$$

If we set the non-cube variables $v[s]$'s ($s \notin I$) to constants, the coefficient $p(\boldsymbol{k}, \boldsymbol{v})$ reduces to a polynomial that only relates to $\boldsymbol{k}$, which we denote by $\mathsf{Coe}\langle f, t_I \rangle$.

The typical process for carrying out a cube attack can be summarized as follows:

- In the offline phase, the attacker recovers superpolies for selected cubes of the cipher without knowledge of the secret key.
- In the online phase, the attacker exploits the output bits generated under the unknown key to evaluate the recovered superpolies. This allows building a system of equations in the key bits.
- Solving this system of equations recovers part of the key. The remaining key bits can be obtained through an exhaustive search.

The core idea is that the successful recovery of superpolies allows to construct a solvable system of equations that leak key bits, which can break the security of the cipher by facilitating full key recovery.

2.3 Monomial Prediction (MP)

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

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

where $\boldsymbol{x}^i \in \mathbb{F}_2^{n_i}$ and $\boldsymbol{x}^{i+1} \in \mathbb{F}_2^{n_{i+1}}$ are the input and output of $\boldsymbol{f}^i$, respectively.

Given a starting number r_s and an ending number r_e with $0 \le r_s < r_e \le r$, let $r' = r_e - r_s$. Given $r'+1$ monomials $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}), \cdots, \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$, if for each $j, r_s \le j \le r_e - 1$ we have $\pi(\boldsymbol{x}^j, \boldsymbol{u}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{u}^{j+1})$, we write the connection of these transitions as

$$\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{u}^{r_s+1}) \rightarrow \cdots \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e}),$$

which is called an r'-round *monomial trail*. If there exists at least one monomial trail from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$, we write $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \rightsquigarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$. The set containing all the monomial trails from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ is denoted by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \bowtie \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$, whose size is represented as $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \bowtie \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})|$. If there is no trail from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$, we denote it by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \not\rightsquigarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ and accordingly we have $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \bowtie \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})| = 0$.

The monomial prediction focuses on how to determine accurately whether $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ for two given monomials $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ and $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$, and the following theorem relates this problem to the number of monomial trails.

Theorem 1 ([24, Proposition 1]). *Use the notations defined above. We have* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ *if and only if*

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \bowtie \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})| \equiv 1 \pmod 2.$$

Theorem 2 (Superpoly Recovery [24]). *Let f be the output bit of a cipher represented as a monomial of the output state, which is generated from the secret variables $\boldsymbol{k}$ and the public variables $\boldsymbol{x}$ through a series of round functions. Given the cube indices I and $t_I = \prod_{i \in I} v[i]$, set the non-cube variables $v[s]$'s ($s \notin I$) to* 0*, then*

$$\mathsf{Coe}\langle f, t_I \rangle = \sum_{|\boldsymbol{k}^{\boldsymbol{w}} t_I \bowtie f| \equiv 1 \pmod 2} \boldsymbol{k}^{\boldsymbol{w}},$$

where $\boldsymbol{k}^{\boldsymbol{w}} t_I \bowtie f$ is the set of monomial trails that propagate $\boldsymbol{k}^{\boldsymbol{w}} t_I$ to f through the round functions.

Propagation Rules and MILP Models. Since any symmetric primitive can be constructed from basic operations like XOR, AND and COPY, it is sufficient to define propagation rules for these basic functions. By listing all input-output pairs that exhibit monomial prediction, and encoding these pairs as linear inequalities [6,35,39], we can model the propagation of monomial prediction in a way that is amenable to efficient MILP solving. We provide the concrete propagation rules and MILP models in [21, Sup.Mat. A]. In this paper, we choose the state-of-the-art commercial MILP solver, Gurobi [2], to solve our MILP models.

3 Recalling Core Monomial Prediction (CMP)

This paper targets an r-round cipher represented by a parameterized vectorial Boolean function $\boldsymbol{f}(\boldsymbol{k}, \boldsymbol{v})$ with secret variables $\boldsymbol{k}$ and public variables $\boldsymbol{v}$ being the input. $\boldsymbol{f}$ can be written as the composition of a sequence of simple round functions whose ANFs are known, i.e.,

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

where $\boldsymbol{f}^i, 0 \le i \le r-1$, represents the round function at round i, with input variables $\boldsymbol{x}^i \in \mathbb{F}_2^{n_i}$ and output variables $\boldsymbol{x}^{i+1} \in \mathbb{F}_2^{n_{i+1}}$. The initial state $\boldsymbol{x}^0$ is loaded with $\boldsymbol{k}$, $\boldsymbol{v}$, constant 1 bits and constant 0 bits. The output bit z of the cipher is defined as the sum of several monomials of $\boldsymbol{x}^r$. After choosing the cube indices I and setting the non-cube variables to constants (not necessarily constant 0), we aim to efficiently compute $\mathsf{Coe}\langle z, t_I \rangle$, where $t_I = \prod_{i \in I} v[i]$. We first recall some details about the core monomial prediction proposed in [22].

Superpoly Recovery Method in [22]. Since z is the sum of several monomials of $\boldsymbol{x}^r$, we consider computing $\mathsf{Coe}\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r), t_I \rangle$ for each $\pi(\boldsymbol{x}^r, \boldsymbol{u}^r)$ satisfying $\pi(\boldsymbol{x}^r, \boldsymbol{u}^r) \rightarrow z$. There are two steps for computing $\mathsf{Coe}\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r), t_I \rangle$ in [22]. In the first step, the authors choose a fixed middle round r_m and recover all $\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m})$'s that satisfy: (A) $\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m}) \rightarrow \pi(\boldsymbol{x}^r, \boldsymbol{u}^r)$, (B) $\exists \boldsymbol{w}$ such that

$\boldsymbol{k}^{\boldsymbol{w}} t_I \rightsquigarrow \pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m})$. In the second step, compute $\mathsf{Coe}\langle\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m}), t_I\rangle$ by MP. The sum of all $\mathsf{Coe}\langle\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m}), t_I\rangle$'s is exactly $\mathsf{Coe}\langle\pi(\boldsymbol{x}^{r}, \boldsymbol{u}^{r}), t_I\rangle$. In [22], Condition B was characterized by a focus on those bits in $\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m})$ that relate to cube variables, thus leading to the flag technique.

Flag Technique for CMP [22]**.** Let b be one bit of an intermediate state $\boldsymbol{x}^i$, b can have three types of flags:

1. If b is 0, denote its flag by $b.F = 0_c$;
2. Otherwise, express b as the polynomial of $\boldsymbol{k}$ and cube variables, if none of cube bits appear in b, denote its flag by $b.F = 1_c$;
3. Otherwise, denote its flag by $b.F = \delta$.

The flags of all bits in $\boldsymbol{x}^i$ are denoted by a vector $\boldsymbol{x}^i.F = (x^i[0].F, \ldots, x^i[n_i - 1].F)$, which can be calculated from $\boldsymbol{x}^0.F$ by the following operation rules:

$$\begin{array}{lll} 1_c \times x = x \times 1_c = x & 1_c \oplus 1_c = 1_c & 0_c \oplus x = x \oplus 0_c = x \\ 0_c \times x = x \times 0_c = 0_c & \delta \oplus x = x \oplus \delta = \delta & \delta \times \delta = \delta \end{array}$$

where x can be any of $\{0_c, 1_c, \delta\}$.

Remark 1. Note that the flag technique for CMP is essentially different from the one for the two-subset division property used in [48]. The most significant difference lies in how to process the secret key bits. In the flag technique for CMP, the secret key bits are regarded as 1_c bits and it is an unalienable part of the CMP technique, whereas in [48], the secret keys are treated as free variables and the flag technique is only a skill to improve the precision and efficiency of the division property.

Definition of Core Monomial Trail [22]**.** Let $\mathsf{Ind}[\boldsymbol{M}^{i,\delta}] = \{j \mid x^i[j].F = \delta\}$; $\mathsf{Ind}[\boldsymbol{M}^{i,1_c}] = \{j \mid x^i[j].F = 1_c\}$; $\mathsf{Ind}[\boldsymbol{M}^{i,0_c}] = \{j \mid x^i[j].F = 0_c\}$. The vectors $\boldsymbol{M}^{i,\delta}, \boldsymbol{M}^{i,1_c}, \boldsymbol{M}^{i,0_c}$ are called *flag masks.* Given two monomials $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ and $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ for $0 \le r_s < r_e \le r$, with $\boldsymbol{t}^{r_s} \preceq \boldsymbol{M}^{r_s,\delta}$ and $\boldsymbol{t}^{r_e} \preceq \boldsymbol{M}^{r_e,\delta}$, if there exists a monomial $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ such that $\boldsymbol{u}^{r_s} \wedge \boldsymbol{M}^{r_s,\delta} = \boldsymbol{t}^{r_s}$, $\boldsymbol{u}^{r_s} \wedge \boldsymbol{M}^{r_s,0_c} = \boldsymbol{0}$ and $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, then we say $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ can propagate to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ under the *core monomial prediction*, denoted by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$; otherwise we denote it by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

Let $r' = r_e - r_s$. We call the connection of r' transitions $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}) \xrightarrow{\mathcal{C}} \cdots \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ an r'-round *core monomial trail.* If there is at least one r'-round core monomial trail from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, we denote it by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$; otherwise we write $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. The set containing all the trails from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ is denoted by $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. The propagation rules and MILP models of CMP are provided in [21, Sup.Mat. B]. In the propagation of CMP, the 0_c bits are excluded, the 1_c bits are treated as constants that can be ignored, thus only the δ bits are tracked.

Limitations of the CMP Theory in [22]**.** In [22], only the existence property of a core monomial trail was used in theory, and the CMP technique was considered as a compromised version of MP which sacrificed accuracy for efficiency. However, we observe that more information has been associated with a core monomial trail besides the existence property, which was ignored by [22]. When considering these information, CMP can be as precise as MP. To intuitively show this, let us consider a simple example.

Example 1. Consider a simple cipher $\boldsymbol{f} = \boldsymbol{f}^2 \circ \boldsymbol{f}^1 \circ \boldsymbol{f}^0$ where

$$\begin{aligned}
\boldsymbol{x}^0 &= (v[0], v[1], v[2], k[0], k[1], k[2], 0, 1),\\
\boldsymbol{x}^1 &= \boldsymbol{f}^0(\boldsymbol{x}^0) = (x^0[0]x^0[1] + x^0[1]x^0[2] + x^0[1]x^0[4],\\
&\qquad\qquad\qquad\qquad x^0[0]x^0[3] + x^0[3], x^0[4] + x^0[5], x^0[7] + x^0[6]),\\
\boldsymbol{x}^2 &= \boldsymbol{f}^1(\boldsymbol{x}^1) = (x^1[0]x^1[2] + x^1[0]x^1[1], x^1[2], x^1[3]),\\
\boldsymbol{x}^3 &= \boldsymbol{f}^2(\boldsymbol{x}^2) = (x^2[0]x^2[1] + x^2[0]x^2[2]).
\end{aligned}$$

Assume $t_I = v[0]v[1], v[2] = 1$ and we want to compute $\mathsf{Coe}\langle x^3[0], t_I\rangle$.

We first compute $\boldsymbol{x}^0.F = (\delta, \delta, 1_c, 1_c, 1_c, 1_c, 0_c, 1_c), \boldsymbol{x}^1.F = (\delta, \delta, 1_c, 1_c)$, $\boldsymbol{x}^2.F = (\delta, 1_c, 1_c)$ and $\boldsymbol{x}^3.F = (\delta)$. Then, we expand $x^3[0]$ into a polynomial of $\boldsymbol{x}^2$ and combine the monomials according to δ bits.

$$x^3[0] = \underline{(x^2[1] + x^2[2])} \cdot x^2[0]$$

Note that $x^2[1].F = x^2[2].F = 1.c$, we derive

$$x^3[0] = \underline{(x^2[1] + x^2[2])} \cdot x^2[0] = \underline{(1 + k[1] + k[2])} \cdot x^2[0].$$

Similarly, for $x^2[0]$ we have

$$x^2[0] = \underline{(x^1[2])} \cdot x^1[0] + x^1[0]x^1[1] = \underline{(k[1] + k[2])} \cdot x^1[0] + x^1[0]x^1[1],$$

and further for $x^1[0]$ and $x^1[0]x^1[1]$ we have

$$\begin{aligned}
x^1[0] &= \underline{(x^0[2] + x^0[4])} \cdot x^0[1] + x^0[0]x^0[1] = \underline{(1 + k[1])} \cdot x^0[1] + x^0[0]x^0[1],\\
x^1[0]x^1[1] &= \underline{(x^0[2]x^0[3] + x^0[3]x^0[4])} \cdot x^0[0]x^0[1] + \underline{(x^0[2]x^0[3] + x^0[3]x^0[4])} \cdot x^0[1]\\
&= \underline{(k[0] + k[0]k[1])} \cdot x^0[0]x^0[1] + \underline{(k[0] + k[0]k[1])} \cdot x^0[1].
\end{aligned}$$

Thus, there are two core monomial trails

$$\begin{aligned}
&x^0[0]x^0[1] \xrightarrow{\mathcal{C}} x^1[0] \xrightarrow{\mathcal{C}} x^2[0] \xrightarrow{\mathcal{C}} x^3[0],\\
&x^0[0]x^0[1] \xrightarrow{\mathcal{C}} x^1[0]x^1[1] \xrightarrow{\mathcal{C}} x^2[0] \xrightarrow{\mathcal{C}} x^3[0],
\end{aligned}$$

Multiply the coefficients of each core monomial trail. We take the first core monomial trail as an example. From $x^0[0]x^0[1] \xrightarrow{\mathcal{C}} x^1[0]$, the corresponding coefficient of $x^0[0]x^0[1]$ is 1; from $x^1[0] \xrightarrow{\mathcal{C}} x^2[0]$, the corresponding coefficient of $x^1[0]$ is $k[1] + k[2]$; from $x^2[0] \xrightarrow{\mathcal{C}} x^3[0]$, the corresponding coefficient of $x^2[0]$ is $1 + k[1] + k[2]$. Thus, the first core monomial trail leads to $(1 + k[1] + k[2])(k[1] + k[2])$. Similarly, the second core monomial trail leads to $(1 + k[1] + k[2])(k[0] + k[0]k[1])$. It is easy to verify that $\mathsf{Coe}\langle x^3[0], t_I\rangle$ is just $(1 + k[1] + k[2])(k[1] + k[2]) + (1 + k[1] + k[2])(k[0] + k[0]k[1])$.

In [22], all possible concatenations of a core monomial trail of the first r_m rounds and a monomial trail of the subsequent $r - r_m$ rounds are enumerated when solving the MILP model in practice. However, the above example reveals that it is sufficient to compute $\mathsf{Coe}\langle\pi(\boldsymbol{x}^r, \boldsymbol{u}^r), t_I\rangle$ by computing all core monomial trails of the r rounds and then extracting the coefficients from each core monomial trail, where the latter step is a fully offline process independently from the MILP solver. Hence, the new method is surely more efficient than the method in [22].

4 Beyond Existence: Refinement of CMP Theory

In this section, we prove that CMP can reach perfect accuracy as MP does by developing a purely CMP-based approach for the superpoly recovery, thus addressing the limitations of the CMP theory. On this basis, we further design an MITM framework to enhance the CMP-based approach.

4.1 Extending CMP Theory Using SR Problem

In order to study the CMP theory independently from a specific cryptographic context, we start by breaking the association between flags and cube indices.

Indeterminate Flags. While the flag technique presented in Sect. 3 is defined based on the chosen cube indices I, the definition and propagation of CMP are independent of how the flags are defined. Therefore, in the rest of the paper, we drop the previous definition of flags based on cube variables, which is to say, the flag of a bit b is no longer based on representing b as a Boolean polynomial of $\boldsymbol{k}$ and cube variables. Instead, we consider flags as variables that can take values δ, 1_c and 0_c, which are referred to as *indeterminate flags*, but the operation rules remain unchanged. As a result, the flag masks become variables as well, but for $0 \leq i \leq r$, the requirement that $\mathsf{Ind}[\boldsymbol{M}^{i,\delta}]$, $\mathsf{Ind}[\boldsymbol{M}^{i,1_c}]$ and $\mathsf{Ind}[\boldsymbol{M}^{i,0_c}]$ form a partition of $\{0, \ldots, n_i - 1\}$ still holds.

Note that different values assigned to the flag masks can result in different propagation of CMP. Therefore, when we discuss the propagation of CMP from round r_s to round r_e for $0 \leq r_s < r_e \leq r$, if the values of $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}, \boldsymbol{M}^{r_s,0_c}$ are not clear from the context, we will give specific values for $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}, \boldsymbol{M}^{r_s,0_c}$, and implicitly assume that the values of $\boldsymbol{M}^{j,\delta}, \boldsymbol{M}^{j,1_c}, \boldsymbol{M}^{j,0_c}, r_s < j \leq r_e$ are calculated from round r_s according to operation rules.

Extending CMP Theory with SR Problem. In the context of indeterminate flags, we analyze the reasons why the previous CMP theory is considered inaccurate. Assume the flag masks $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}, \boldsymbol{M}^{r_s,0_c}$ take the values $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}, \boldsymbol{\alpha}^{r_s,0_c} = \neg(\boldsymbol{\alpha}^{r_s,\delta} \vee \boldsymbol{\alpha}^{r_s,1_c})$. By the definition of CMP, the transition $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ emphasizes the existence of a $\boldsymbol{w} \preceq \boldsymbol{\alpha}^{r_s,1_c}$ such that $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, but does not provide explicit information about the exact value of $\boldsymbol{w}$. In other words, the definition of CMP does not give any precise information related to the exact expressions of the monomials appearing in $\mathsf{Expr}\langle\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\rangle$. Consequently, it may give the impression that the previous CMP theory is inaccurate.

In order to refine the CMP theory to be precise, it is necessary to capture all $\boldsymbol{w}$'s that satisfy $\boldsymbol{w} \preceq \boldsymbol{\alpha}^{r_s,1_c}$ and $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. This can be easily achieved if we can obtain the concrete expression of $\mathsf{Expr}\langle\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\rangle$ (e.g., when the vectorial Boolean function mapping $\boldsymbol{x}^{r_s}$ to $\boldsymbol{x}^{r_e}$ is simple). However, when $\mathsf{Expr}\langle\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\rangle$ is not available, the situation becomes much more complicated, which deserves further investigation, thus we formalize it as the following SR problem.

Definition 1 (SR Problem). *Let the target cipher $\boldsymbol{f}$ be as defined in Eq. (1). Given $r_s, r_e, 0 \leq r_s < r_e \leq r$ and the values $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}, \boldsymbol{\alpha}^{r_s,0_c} = \neg(\boldsymbol{\alpha}^{r_s,\delta} \vee \boldsymbol{\alpha}^{r_s,1_c})$ assigned to the flag masks $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}, \boldsymbol{M}^{r_s,0_c}$, for each $j, r_s < j \leq r_e$, let $\boldsymbol{M}^{j,\delta}, \boldsymbol{M}^{j,1_c}, \boldsymbol{M}^{j,0_c}$ take the values $\boldsymbol{\alpha}^{j,\delta}, \boldsymbol{\alpha}^{j,1_c}, \boldsymbol{\alpha}^{j,0_c}$ that are calculated from $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}, \boldsymbol{\alpha}^{r_s,0_c}$ according to the operation rules of flags. Given two monomials $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ and $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ that satisfy $\boldsymbol{t}^{r_s} \preceq \boldsymbol{\alpha}^{r_s,\delta}$ and $\boldsymbol{t}^{r_e} \preceq \boldsymbol{\alpha}^{r_e,\delta}$, we can uniquely and symbolically express $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ as a polynomial of $\boldsymbol{x}^{r_s}$, i.e.,*

$$\mathsf{Expr}\langle\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\rangle = p(\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,1_c} \vee \boldsymbol{\alpha}^{r_s,0_c}]) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) + q(\boldsymbol{x}^{r_s}), \tag{2}$$

where each monomial $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s})$ appearing in $q(\boldsymbol{x}^{r_s})$ satisfies $\boldsymbol{u}^{r_s} \wedge \boldsymbol{\alpha}^{r_s,\delta} \neq \boldsymbol{t}^{r_s}$. If we set $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$, then the coefficient $p(\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,1_c} \vee \boldsymbol{\alpha}^{r_s,0_c}])$ reduces to a polynomial that only relates to $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,1_c}]$. The question is, what is the exact expression of this polynomial?

We use $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ to denote a concrete instance of the SR problem, which is uniquely determined by six parameters, namely the numbers r_s, r_e of rounds, the values $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}$ assigned to $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}$ and the vectors $\boldsymbol{t}^{r_s}, \boldsymbol{t}^{r_e}$ corresponding to the monomials $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}), \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. The solution of this instance, denoted by $\mathsf{Sol}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$, is the coefficient $p(\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,1_c} \vee \boldsymbol{\alpha}^{r_s,0_c}])$ in Eq. (2) after setting $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$. When $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}$ can be inferred from the context without ambiguity, we write $\mathsf{SR}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and $\mathsf{Sol}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ for simplicity. Each instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ uniquely corresponds to a CMP transition $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. In particular, $\mathsf{Sol}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ is exactly the sum of all $\boldsymbol{w} \preceq \boldsymbol{\alpha}^{r_s,1_c}$ that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. When $\mathsf{Sol}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle\boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ is available, the transition $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ is considered accurate.

The SR problem can be considered as an extension of the CMP theory used to specify the precise algebraic information implied by a CMP transition. Hence, the solution of an instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ not only uniquely determines a CMP transition, but also reflects the information of exact monomials in the algebraic composition of $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\right\rangle$, as stated in Lemma 1 and Lemma 2. Since these two lemmas are direct consequences of Definition 1, we omit the proofs of them here.

Lemma 1. *Letting $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}$ be any values assigned to $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}$ and $\boldsymbol{M}^{r_s,0_c} = \neg(\boldsymbol{\alpha}^{r_s,\delta} \vee \boldsymbol{\alpha}^{r_s,1_c})$, for each $j, r_s < j \le r_e$ we calculate the values of $\boldsymbol{M}^{j,\delta}, \boldsymbol{M}^{j,1_c}, \boldsymbol{M}^{j,0_c}$ as $\boldsymbol{\alpha}^{j,\delta}, \boldsymbol{\alpha}^{j,1_c}, \boldsymbol{\alpha}^{j,0_c}$. Then, $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ is not equal to 0 if and only if $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.*

Lemma 2. *Let the values of flag masks be defined as in Lemma 1. Given any monomial $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ satisfying $\boldsymbol{t}^{r_e} \preceq \boldsymbol{\alpha}^{r_e,\delta}$, after setting $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$, if $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\right\rangle \neq 0$, then we can uniquely express $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ as*

$$\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\right\rangle = \sum_{\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})} \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}),$$

where the summation is over all $\boldsymbol{t}^{r_s}$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

We would also like to point out that both the SR problem and MP are concerned with the presence of specific monomials in the polynomial expanded from a higher-round monomial; the only difference is that MP is concerned with whether a single monomial exists in the polynomial, whereas the SR problem is concerned with the existence of several monomials of the same form in the polynomial. For example, the instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ is concerned with whether monomials of the form $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ appear in the polynomial $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\right\rangle$, where $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w})$ is a monomial in $\mathsf{Sol}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$.

Relationships Between Superpoly Recovery and SR Problem. Since the SR problem is accurate as an extension of CMP, we can utilize it to compute the exact superpoly.

Proposition 1 (Reducing Superpoly Recovery to Solving SR Problem). *Recall that the initial state $\boldsymbol{x}^0$ is loaded with $\boldsymbol{k}$, $\boldsymbol{v}$, constant 1 bits and constant 0 bits. Let $\boldsymbol{M}^{0,\delta}, \boldsymbol{M}^{0,1_c}$ and $\boldsymbol{M}^{0,0_c}$ take the particular values $\boldsymbol{\gamma}^{0,\delta}, \boldsymbol{\gamma}^{0,1_c}$ and $\boldsymbol{\gamma}^{0,0_c}$ respectively, where*

$$\begin{aligned}
\mathsf{Ind}[\boldsymbol{\gamma}^{0,\delta}] &= \{i \mid x^{(0)}[i] \textit{ is loaded with a cube variable}\},\\
\mathsf{Ind}[\boldsymbol{\gamma}^{0,0_c}] &= \{i \mid x^{(0)}[i] \textit{ is loaded with a non-cube variable that is set to constant } 0\}\\
&\quad \bigcup \{i \mid x^{(0)}[i] \textit{ is loaded with constant } 0\},\\
\boldsymbol{\gamma}^{0,1_c} &= \neg\left(\boldsymbol{\gamma}^{0,\delta} \vee \boldsymbol{\gamma}^{0,0_c}\right).
\end{aligned} \tag{3}$$

For each $j, 0 < j \le r$, let $\boldsymbol{\gamma}^{j,\delta}, \boldsymbol{\gamma}^{j,1_c}, \boldsymbol{\gamma}^{j,0_c}$ be calculated from $\boldsymbol{\gamma}^{0,\delta}, \boldsymbol{\gamma}^{0,1_c}, \boldsymbol{\gamma}^{0,0_c}$ according to the operation rules of flags. For $t_I = \prod_{i\in I} v[i]$ and the output bit z as the sum of monomials of $\boldsymbol{x}^r$, we define two sets $S^0 = \{\boldsymbol{t}^0 \mid \boldsymbol{t}^0 \preceq$

$\boldsymbol{\gamma}^{0,\delta}, \mathsf{Expr}\left\langle\pi(\boldsymbol{x}^0,\boldsymbol{t}^0),\boldsymbol{v}\right\rangle = t_I\}$ *and* $S^r = \{\boldsymbol{u}^r \mid \pi(\boldsymbol{x}^r,\boldsymbol{u}^r) \rightarrow z, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,0_c} = \boldsymbol{0}\}$. *Then, either* $|S^r| = 0$*, which is easy to verify and indicates that* $\mathsf{Coe}\langle z, t_I\rangle = 0$*, or we can compute* $\mathsf{Coe}\langle z, t_I\rangle$ *as*

$$\mathsf{Coe}\langle z,t_I\rangle = \sum_{\boldsymbol{u}^r\in S^r}\left(\sum_{\boldsymbol{t}^0\in S^0}\mathsf{Expr}\left\langle\mathsf{Sol}\langle\boldsymbol{u}^r\wedge\boldsymbol{\gamma}^{r,\delta},\boldsymbol{t}^0\rangle,\boldsymbol{k}\right\rangle\cdot\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^r,\boldsymbol{u}^r\wedge\boldsymbol{\gamma}^{r,1_c}),\boldsymbol{k}\right\rangle\right). \tag{4}$$

Proof. Due to page limits, the proof of this proposition is provided in [21, Sect. 4.1]. □

In Eq. (4), it is assumed that $\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^r,\boldsymbol{u}^r\wedge\boldsymbol{\gamma}^{r,1_c}),\boldsymbol{k}\right\rangle$ can be calculated quickly based on the round functions. In practice, due to the diffusion of round functions, usually the state bits of round r (e.g., $r = 850$ for Trivium) are all δ bits, resulting in $\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^r,\boldsymbol{u}^r\wedge\boldsymbol{\gamma}^{r,1_c}),\boldsymbol{k}\right\rangle$ being 1. Therefore, computing $\mathsf{Coe}\langle z,t_I\rangle$ naturally reduces to the problem of seeking solutions for instances of the form $\mathsf{Sol}_{\boldsymbol{\gamma}^{0,\delta},\boldsymbol{\gamma}^{0,1_c}}\langle\boldsymbol{t}^r,\boldsymbol{t}^0\rangle$, where the values of flag masks are given as in Proposition 1 and $\boldsymbol{t}^r,\boldsymbol{t}^0$ can be any vectors that satisfy $\boldsymbol{t}^0\in S^0, \boldsymbol{t}^r\preceq\boldsymbol{\gamma}^{r,\delta}$.

4.2 Solving SR Problem with CMP

A CMP transition can be refined to be precise if the instance of the SR problem corresponding to this transition can be solved. Unfortunately, this is not always achievable for a complex instance. To address this problem, it would be necessary to study how to split a high-round instance into multiple instances of lower round, so that we can compute the solution of a complex instance from the solutions of simpler instances. With a little derivation, we immediately have the following lemma.

Lemma 3. *Assuming* $r_s + 1 < r_e$*, let the values of flag masks be defined as in Lemma 1. Given a monomial* $\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})$ *satisfying* $\boldsymbol{t}^{r_e}\preceq\boldsymbol{\alpha}^{r_e,\delta}$*, after setting* $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ *to* $\boldsymbol{0}$*, then for any* $j, r_s < j < r_e$*, either* $\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e}),\boldsymbol{x}^j\right\rangle = 0$ *or we can calculate* $\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^{r_s}\rangle$ *as*

$$\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^{r_s}\rangle = \sum_{\pi(\boldsymbol{x}^j,\boldsymbol{t}^j)\xrightarrow{\mathcal{C}}\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})}\mathsf{Expr}\left\langle\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^j\rangle,\boldsymbol{x}^{r_s}\right\rangle\cdot\mathsf{Sol}\langle\boldsymbol{t}^j,\boldsymbol{t}^{r_s}\rangle, \tag{5}$$

where the summation is over all $\boldsymbol{t}^j$*'s that satisfy* $\pi(\boldsymbol{x}^j,\boldsymbol{t}^j)\xrightarrow{\mathcal{C}}\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})$*.*

Proof. After setting $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$, $\boldsymbol{x}^j[\boldsymbol{\alpha}^{j,0_c}]$ is also set to $\boldsymbol{0}$ according to operation rules. According to Lemma 2, either $\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e}),\boldsymbol{x}^j\right\rangle = 0$ or we can express $\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})$ as

$$\mathsf{Expr}\left\langle\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e}),\boldsymbol{x}^j\right\rangle = \sum_{\pi(\boldsymbol{x}^j,\boldsymbol{t}^j)\xrightarrow{\mathcal{C}}\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})}\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^j\rangle\cdot\pi(\boldsymbol{x}^j,\boldsymbol{t}^j).$$

Notice that for each $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ satisfying $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j \rangle$ only relates to $\boldsymbol{x}^j[\boldsymbol{\alpha}^{j,1_c}]$ and can be expressed as polynomial of $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,1_c}]$, so it suffices to calculate $\mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s} \rangle$, thus proving the lemma. $\square$

According to Lemma 3, we can calculate $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle$ as

$$\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle = \sum_{\boldsymbol{t}^{r_e-1}} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-1} \rangle, \boldsymbol{x}^{r_s} \right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{r_e-1}, \boldsymbol{t}^{r_s} \rangle,$$

where the summation of over all $\boldsymbol{t}^{r_e-1}$'s that satisfy $\pi(\boldsymbol{x}^{r_e-1}, \boldsymbol{t}^{r_e-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. By applying Lemma 3 to each $\mathsf{Sol}\langle \boldsymbol{t}^{r_e-1}, \boldsymbol{t}^{r_s} \rangle$ again, we have

$$\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle = \sum_{\boldsymbol{t}^{r_e-2}, \boldsymbol{t}^{r_e-1}} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-1} \rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{r_e-1}, \boldsymbol{t}^{r_e-2} \rangle, \boldsymbol{x}^{r_s} \right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{r_e-2}, \boldsymbol{t}^{r_s} \rangle,$$

where the summation is over all $\boldsymbol{t}^{r_e-2}, \boldsymbol{t}^{r_e-1}$'s that satisfy $\pi(\boldsymbol{x}^{r_e-2}, \boldsymbol{t}^{r_e-2}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e-1}, \boldsymbol{t}^{r_e-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. This process can be repeated round by round until we arrive at round r_s, namely

$$\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle = \sum_{\boldsymbol{t}^{r_s+1}, \dots, \boldsymbol{t}^{r_e-1}} \mathsf{Expr}\left\langle \prod_{j=r_s}^{r_e-1} \mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle, \boldsymbol{x}^{r_s} \right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{r_s}, \boldsymbol{t}^{r_s} \rangle,$$

where the summation is over all $\boldsymbol{t}^{r_s+1}, \dots, \boldsymbol{t}^{r_e-1}$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}) \xrightarrow{\mathcal{C}} \cdots \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e-1}, \boldsymbol{t}^{r_e-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. Note that $\mathsf{Sol}\langle \boldsymbol{t}^{r_s}, \boldsymbol{t}^{r_s} \rangle$ is trivially 1, so we actually get an approach to compute $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle$ based on core monomial trails. We next formalize this approach as a theory, where the role of each core monomial trail is explicitly specified.

Definition 2 (Contribution of Core Monomial Trail). *Let the values of flag masks be defined as in Lemma 1. For any core monomial trail ℓ written as*

$$\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}) \xrightarrow{\mathcal{C}} \cdots \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}),$$

we define the contribution of this trail as

$$\mathsf{Contr}_{\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}}\langle l \rangle = \prod_{j=r_s}^{r_e-1} \mathsf{Sol}_{\boldsymbol{\alpha}^{j,\delta}, \boldsymbol{\alpha}^{j,1_c}}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle.$$

If $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}$ are clear from the context, we write $\mathsf{Contr}\langle l \rangle$ *for simplicity.*

The contribution of a core monomial trail specifies the algebraic information carried by the trail. Collecting the contributions of all core monomial trails enables us to solve a complex instance efficiently. More precisely,

Proposition 2 (Solving SR Problem by Core Monomial Trails). *Let the values of flag masks be defined as in Lemma 1. Given an instance of the SR problem denoted by* $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$, *if* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, *the solution of this instance is* 0; *otherwise, we can calculate the solution of this instance by*

$$\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle = \sum_{\ell \in \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})} \mathsf{Expr}\left\langle \mathsf{Contr}\langle \ell\rangle, \boldsymbol{x}^{r_s}\right\rangle.$$

Proof. We prove this proposition by fixing r_s and performing induction on r_e. When $r_e = r_s + 1$, the proposition clearly holds according to Definition 2 and Lemma 2. Assuming the proposition holds for $r_e < m$, we are going to prove that it also holds for $r_e = m$.

If $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})$, then either $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m}), \boldsymbol{x}^{m-1}\right\rangle = 0$, meaning that $\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = 0$, or $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})$ for each $\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})$ satisfy $\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})$. For the latter case, we set $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$ and calculate $\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle$ according to Lemma 3 as

$$\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = \sum_{\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{m-1}\rangle, \boldsymbol{x}^{r_s}\right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{m-1}, \boldsymbol{t}^{r_s}\rangle. \tag{6}$$

According to the induction hypothesis, $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})$ leads to $\mathsf{Sol}\langle \boldsymbol{t}^{m-1}, \boldsymbol{t}^{r_s}\rangle = 0$, thus for the latter case we also have $\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = 0$.

If $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, similarly we set $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\boldsymbol{0}$ and obtain Eq. (6). According to the induction hypothesis we made at the beginning,

$$\mathsf{Sol}\langle \boldsymbol{t}^{m-1}, \boldsymbol{t}^{r_s}\rangle = \sum_{\ell \in \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})} \mathsf{Expr}\left\langle \mathsf{Contr}\langle \ell\rangle, \boldsymbol{x}^{r_s}\right\rangle. \tag{7}$$

Define the set S as

$$S = \{(\boldsymbol{t}^{m-1}, \ell) \mid \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m}), \ell \in \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})\}.$$

Combining Eqs. (6) and (7), we have

$$\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = \sum_{(\boldsymbol{t}^{m-1}, \ell) \in S} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{m-1}\rangle \cdot \mathsf{Contr}\langle \ell\rangle, \boldsymbol{x}^{r_s}\right\rangle, \tag{8}$$

where the summation is over all the pairs $(\boldsymbol{t}^{m-1}, \ell) \in S$. Notice that Eqn (8) is equivalent to

$$\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = \sum_{\ell' \in \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})} \mathsf{Expr}\left\langle \mathsf{Contr}\langle \ell'\rangle, \boldsymbol{x}^{r_s}\right\rangle,$$

which proves the proposition. □

Corollary 1. *Let the values of flag masks be defined as in Lemma 1. Then,* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ *if* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

Algorithm 1: Iterative variable substitutions

```
1  Procedure VariableSubstitution($\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$):
2      Initialize an integer $j = r_s$ and an empty hash table $T$
3      Initialize $r_e - r_s$ empty vectors $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \ldots, \boldsymbol{c}^{r_e-1}$ of variables
4      while $j < r_e$ do
5          for each possible pair $(\boldsymbol{t}^j, \boldsymbol{t}^{j+1})$ satisfying $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ do
6              Calculate $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$
7              Substitute $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$ by a new variable $c$ such that
                $c \cdot \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$
8              Add $c$ to the end of $\boldsymbol{c}^j$, and store the substitution by letting
                $T[c] = \mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$
9          Increment $j$ by one                                   // $j = j + 1$
10     return $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \ldots, \boldsymbol{c}^{r_e-1}, T$
```

Proof. We prove the contrapositive of this corollary. Since $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})| = 0$. According to Proposition 2, $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle = 0$. As stated by Lemma 1, this holds if and only if $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \nrightarrow \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. □

As mentioned earlier, each round function $\boldsymbol{f}^j, 0 \le j \le r$ in Eq. (1) performs a simple transformation and its ANF has been determined, which means we can calculate the contribution of a core monomial trail efficiently. Hence, Proposition 2 provides a feasible CMP-based way to solve $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ even if the concrete expression of $\mathsf{Expr}\,\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s}\rangle$ is not available, thus completing the refinement of the CMP theory.

Since both the SR problem and MP are concerned with the presence of monomials, it would be necessary to investigate the relationship between the CMP-based approach in Proposition 2 and the usual MP-based method in Theorem 2. In fact, we can equivalently relate CMP and MP by means of variable substitution.

Equivalence Between CMP and MP. Let the values of flag masks be defined as in Lemma 1. We set $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\mathbf{0}$. Given an instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$, we illustrate how to solve this instance based on MP. First, theoretically we can perform iterative variable substitutions following Algorithm 1. The basic idea behind Algorithm 1 is, if we regard $\mathsf{Sol}_{\boldsymbol{\alpha}^{j,\delta},\boldsymbol{\alpha}^{j,1_c}}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$ as a single bit c for the transition $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, then we have $c \cdot \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, thus establishing the equivalence between CMP and MP.

In Line 5–8, we describe a generic approach to perform variable substitution for a round function $\boldsymbol{f}^j$, but this may not always be practical to directly implement as it requires enumerating an exponential number of monomials. Corresponding to the propagation rules of CMP, we propose several rules to replace Line 5–8 to optimize the substitution process.

Algorithm 2: Perform variable substitution on $\boldsymbol{f}^j$ that consists of multiple S-boxes

```
1 for each S-box $S_i, 0 \le i < m$ do
2     for each possible pair $(\boldsymbol{t}^j, \boldsymbol{t}^{j+1})$ satisfying that $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ only relates to the input $\delta$ bits of $S_i$, $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only relates to the output $\delta$ bits of $S_i$, and $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ do
3         Calculate $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$
4         Substitute $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$ by a new variable $c$ such that $c \cdot \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$
5         Add $c$ to the end of $\boldsymbol{c}^j$, and store the substitution by letting $T[c] = \mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$
```

Rule 1 (COPY) *Assume the round function $\boldsymbol{f}^j$ is the basic operation* COPY *with $x^j[0] \xrightarrow{COPY} (x^{j+1}[0], \dots, x^{j+1}[m-1])$. We do not need to perform variable substitution because $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle = 1$ if $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$.*

Rule 2 (AND) *Assume the round function $\boldsymbol{f}^j$ is the basic operation* AND *with $(x^j[0], x^j[1], \dots, x^j[m-1]) \xrightarrow{AND} x^{j+1}[0]$. If $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \dots, x^j[m-1].F\}$ and $x^{j+1}[0].F = \delta$, we substitute a new variable c for $\prod_{\substack{0 \le i < m \\ x^j[i].F = 1_c}} x^j[i]$, add c to the end of $\boldsymbol{c}^j$, and store the substitution by letting $T[c] = \prod_{\substack{0 \le i < m \\ x^j[i].F = 1_c}} x^j[i]$; otherwise we do not perform variable substitution.*

Rule 3 (XOR) *Assume the round function $\boldsymbol{f}^j$ is the basic operation* XOR *with $(x^j[0], x^j[1], \dots, x^j[m-1]) \xrightarrow{XOR} x^{j+1}[0]$. If $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \dots, x^j[m-1].F\}$, we substitute a new variable c for $\sum_{\substack{0 \le i < m \\ x^j[i].F = 1_c}} x^j[i]$, add c to the end of $\boldsymbol{c}^j$, and store the substitution by letting $T[c] = \sum_{\substack{0 \le i < m \\ x^j[i].F = 1_c}} x^j[i]$; otherwise we do not perform variable substitution.*

Rule 4 (S-boxes) *Let $\boldsymbol{f}^j$ be a function that consists of m S-boxes, denoted by $S_0, \dots, S_{m-1}$. We can set $\boldsymbol{c}^j$ and T following Algorithm 2.*

Utilizing the new variables $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \dots, \boldsymbol{c}^{r_e-1}, T$ returned by Algorithm 1, we are able to establish a one-to-one correspondence between core monomial trails and monomial trails.

Proposition 3. *There is a core monomial trail written as*

$$\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}) \xrightarrow{\mathcal{C}} \cdots \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$$

if and only if there is a monomial trail written as

$$\begin{aligned}&\pi(\boldsymbol{c}^{r_s}\|\cdots\|\boldsymbol{c}^{r_e-1}\|\boldsymbol{x}^{r_s},\boldsymbol{w}^{r_s}\|\cdots\|\boldsymbol{w}^{r_e-1}\|\boldsymbol{t}^{r_s})\\&\to\pi(\boldsymbol{c}^{r_s+1}\|\cdots\|\boldsymbol{c}^{r_e-1}\|\boldsymbol{x}^{r_s+1},\boldsymbol{w}^{r_s+1}\|\cdots\|\boldsymbol{w}^{r_e-1}\|\boldsymbol{t}^{r_s+1})\\&\to\cdots\\&\to\pi(\boldsymbol{c}^{r_e-1}\|\boldsymbol{x}^{r_e-1},\boldsymbol{w}^{r_e-1}\|\boldsymbol{t}^{r_e-1})\\&\to\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e}).\end{aligned}\tag{9}$$

Moreover, by substituting each variable c in $\boldsymbol{c}^j, r_s \le j < r_e$ back with $T[c]$, we can get a polynomial of $\boldsymbol{x}^{r_s},\dots,\boldsymbol{x}^{r_e-1}$ from $\pi(\boldsymbol{c}^{r_s}\|\cdots\|\boldsymbol{c}^{r_e-1},\boldsymbol{w}^{r_s}\|\cdots\|\boldsymbol{w}^{r_e-1})$, which is exactly the contribution of the core monomial trail.

Proof. For each $j, r_s \le j < r_e$, according to the process of iterative variable substitutions, $\pi(\boldsymbol{x}^j,\boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1},\boldsymbol{t}^{j+1})$ if and only if there is an unique $\boldsymbol{w}^j$ such that $\pi(\boldsymbol{c}^j\|\boldsymbol{x}^j,\boldsymbol{w}^j\|\boldsymbol{t}^j)\to\pi(\boldsymbol{x}^{j+1},\boldsymbol{t}^{j+1})$. Combining all these MP transitions $\pi(\boldsymbol{c}^{r_s}\|\boldsymbol{x}^{r_s},\boldsymbol{w}^{r_s}\|\boldsymbol{t}^{r_s})\to\pi(\boldsymbol{x}^{r_s+1},\boldsymbol{t}^{r_s+1}),\pi(\boldsymbol{c}^{r_s+1}\|\boldsymbol{x}^{r_s+1},\boldsymbol{w}^{r_s+1}\|\boldsymbol{t}^{r_s+1})\to\pi(\boldsymbol{x}^{r_s+2},\boldsymbol{t}^{r_s+2}),\dots,\pi(\boldsymbol{c}^{r_e-1}\|\boldsymbol{x}^{r_e-1},\boldsymbol{w}^{r_e-1}\|\boldsymbol{t}^{r_e-1})\to\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})$ yields an equivalent monomial trail described in the proposition. □

We can therefore also solve $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1c}}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^{r_s}\rangle$ by extracting all monomials trails of the form (9) after performing the iterative variable substitutions, which gives an equivalent MP-based interpretation for the CMP-based approach in Proposition 2. For ease of understanding, we give a concrete example in [21, Sup. Mat. D].

4.3 MITM Framework

At this point, we have a complete process for computing the superpoly $\mathsf{Coe}\langle z, t_I\rangle$, i.e., we first reduce the superpoly recovery to the instances of the SR problem using Proposition 1 and then solve the instances by the CMP-based approach in Proposition 2. The remaining question is how to further improve the efficiency of this process. A similar issue has been also encountered in previous work related to MP [23,24], and it is resolved by a divide-and-conquer strategy, which can be further used in a nested fashion. Inspired by this, we have also tailored a nested divide-and-conquer framework for solving the SR problem.

Divide-and-Conquer: Forward Expansion and Backward Expansion. As a natural corollary of Proposition 2, Eq. (5) in Lemma 3 can be enhanced to

$$\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^{r_s}\rangle=\sum_{\boldsymbol{t}^j}\mathsf{Expr}\left\langle\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^j\rangle,\boldsymbol{x}^{r_s}\right\rangle\cdot\mathsf{Sol}\langle\boldsymbol{t}^j,\boldsymbol{t}^{r_s}\rangle,$$

where the summation is over all $\boldsymbol{t}^j$'s that satisfy both $\pi(\boldsymbol{x}^{r_s},\boldsymbol{t}^{r_s})\xrightarrow{\mathcal{C}}\pi(\boldsymbol{x}^j,\boldsymbol{t}^j)$ and $\pi(\boldsymbol{x}^j,\boldsymbol{t}^j)\xrightarrow{\mathcal{C}}\pi(\boldsymbol{x}^{r_e},\boldsymbol{t}^{r_e})$. This reveals the following two perspectives for computing $\mathsf{Sol}\langle\boldsymbol{t}^{r_e},\boldsymbol{t}^{r_s}\rangle$ by the divide-and-conquer strategy:

- *Forward expansion*: We choose a j between r_s and r_e, and determine all $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$. For each such $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, we precompute $\mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s}\rangle$. If $\mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s}\rangle$ is not 0 (i.e., $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$), we store $\mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s}\rangle$ and then focus on computing $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j\rangle$.
- *Backward expansion*: We choose a j between r_s and r_e, and determine all $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$'s that satisfy $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. For each such $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, we precompute $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j\rangle$. If $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j\rangle$ is not 0 (i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$), we store $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j\rangle$ and then focus on computing $\mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s}\rangle$.

Typically, we will choose j close to r_e in the backward expansion and j close to r_s in the forward expansion, so that the precomputation takes only a small amount of time.

One obvious advantage of these two expansions is that both of them split a complex instance into multiple simpler instances that are easier to deal with. Furthermore, if not using any expansion, solving $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ would require enumerating $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})|$ core monomial trails, but with backward expansion the total number of enumerated core monomial trails will be (ignoring the core monomial trails needed for the precomputation)

$$\sum_{\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)|,$$

which is not larger than $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})|$, so the backward expansion reduces the number of required core monomial trails, and the same is also true for forward expansion.

Forward and Backward Overlaps. We further discuss the rationale behind the forward and backward expansion. Now consider that we want to solve two instances denoted by $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}^{r_s}\rangle$ simultaneously with the flag masks taking the same values as in Lemma 3. If we use the backward expansion from round r_e to round j separately on the two instances, the total number of enumerated core monomial trails will be

$$\sum_{\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_0^{r_e})} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)| + \sum_{\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_1^{r_e})} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)|,$$

where the left summation is for $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and the right summation is for $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}^{r_s}\rangle$. However, if there exists a $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ satisfying both $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_0^{r_e})$ and $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_1^{r_e})$, we can observe that the core monomial trails in $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can only be enumerated once for both $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}^{r_s}\rangle$, and we say that $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}^{r_s}\rangle$ have a *backward overlap at* $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$. Therefore, in total we actually only need to enumerate $\sum_{\boldsymbol{t}^j} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)|$ trails, where the summation is over all $\boldsymbol{t}^j$'s that satisfy one of the following three conditions:

1. $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_0^{r_e}), \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_1^{r_e})$;
2. $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_0^{r_e}), \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_1^{r_e})$;
3. $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_0^{r_e}), \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}_1^{r_e})$.

Condition 3 is exactly the condition that should be satisfied by those $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$'s where $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}^{r_s}\rangle$ have backward overlaps. For m instances denoted by $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}^{r_s}\rangle, \ldots, \mathsf{SR}\langle \boldsymbol{t}_{m-1}^{r_e}, \boldsymbol{t}^{r_s}\rangle$, if there are any two of them that have a backward overlap at $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, we say these m instances have a backward overlap at $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$.

Similarly, for two instances denoted by $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}_0^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}_1^{r_s}\rangle$ with the values of flag masks determined as in Lemma 3, we say $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}_0^{r_s}\rangle$ and $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}_1^{r_s}\rangle$ have a *forward overlap at* $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ if the $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ satisfies both $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}_0^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ and $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}_1^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$. The concept of forward overlaps can be extended to multiple instances in the same way as backward overlaps. Naturally, the more backward (resp. forward) overlaps occurs at round j, the more effective we consider the backward (reps. forward) expansion to be.

The forward and backward overlaps can be illustrated by [21, Fig. 1], where we use sold lines to indicate the parts that are precomputed and dashed lines to indicate the parts that are to be computed. The monomial highlighted in red indicates where the forward or backward overlap occurs.

Computing the Superpoly with MITM Framework. As mentioned in Proposition 1, computing the superpoly, i.e., $\mathsf{Coe}\langle z, t_I\rangle$, reduces to solving concrete instances of the form $\mathsf{SR}_{\boldsymbol{\gamma}^{0,\delta}, \boldsymbol{\gamma}^{0,1_c}}\langle \boldsymbol{t}^r, \boldsymbol{t}^0\rangle$ with $\boldsymbol{\gamma}^{0,\delta}, \boldsymbol{\gamma}^{0,1_c}$ determined by Eq. (3). Assuming $\pi(\boldsymbol{x}^0, \boldsymbol{t}^0) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^r, \boldsymbol{t}^r)$, then we can solve $\mathsf{SR}_{\boldsymbol{\gamma}^{0,\delta}, \boldsymbol{\gamma}^{0,1_c}}\langle \boldsymbol{t}^r, \boldsymbol{t}^0\rangle$ by applying the forward expansion and backward expansion interchangeably and recursively as follows, where the forward depth r_0 and the backward depth r_1 represent the number of rounds affected each time we use the forward and backward expansion, respectively.

1. Initialize $\boldsymbol{M}^{0,\delta} = \boldsymbol{\gamma}^{0,\delta}, \boldsymbol{M}^{0,1_c} = \boldsymbol{\gamma}^{0,1_c}, \boldsymbol{M}^{0,0_c} = \neg(\boldsymbol{\gamma}^{0,\delta} \vee \boldsymbol{\gamma}^{0,1_c})$ according to Eq. (3). For each $j, 0 < j \leq r$, calculate the values of $\boldsymbol{M}^{j,\delta}, \boldsymbol{M}^{j,1_c}, \boldsymbol{M}^{j,0_c}$ as $\boldsymbol{\gamma}^{j,\delta}, \boldsymbol{\gamma}^{j,1_c}, \boldsymbol{\gamma}^{j,0_c}$ according to the operation rules of flags.
2. Prepare a hash table P whose key is an instance and value is a Boolean polynomial of $\boldsymbol{x}^0$ and initialize P as $P[\mathsf{SR}\langle \boldsymbol{t}^r, \boldsymbol{\gamma}^{0,\delta}\rangle] = 1$. Initialize $r_s = 0, r_e = r$. Prepare a binary variable d to represent the direction of the expansion and initialize $d = 1$. Initialize a Boolean polynomial $p = 0$ to store the results.
3. If $r_e < B$, we flip the value of d. Prepare an empty hash table P_e of the same type as P to store the new instances generated by the expansion.
4. If $d = 0$, we use forward expansion. Namely, for each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ as a key of P:
   (a) Determine all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$.
   (b) For each such $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$, compute $\mathsf{Sol}\langle \boldsymbol{t}^{r_s+r_0}, \boldsymbol{t}^{r_s}\rangle$ using Proposition 2, and if $\mathsf{Sol}\langle \boldsymbol{t}^{r_s+r_0}, \boldsymbol{t}^{r_s}\rangle$ is not 0, we consider two cases: if the instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s+r_0}\rangle$ is already a key of P_e, we update P_e

by $P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s+r_0}\rangle] = P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s+r_0}\rangle] + \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_s+r_0}, \boldsymbol{t}^{r_s}\rangle, \boldsymbol{x}^0\right\rangle \cdot P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$; otherwise we add the instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s+r_0}\rangle$ to P_e by letting $P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s+r_0}\rangle] = \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_s+r_0}, \boldsymbol{t}^{r_s}\rangle, \boldsymbol{x}^0\right\rangle \cdot P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$.

   (c) Let $P = P_e$ and update r_s by $r_s = r_s + r_0$.

5. If $d = 1$, we use backward expansion. Namely, for each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ as a key of P:
   (a) Determine all $\pi(\boldsymbol{x}^{r_e-r_1}, \boldsymbol{t}^{r_e-r_1})$'s that satisfy $\pi(\boldsymbol{x}^{r_e-r_1}, \boldsymbol{t}^{r_e-r_1}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.
   (b) For each such $\pi(\boldsymbol{x}^{r_e-r_1}, \boldsymbol{t}^{r_e-r_1})$, compute $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-r_1}\rangle$ using Proposition 2, and if $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-r_1}\rangle$ is not 0, we consider two cases: if the instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e-r_1}, \boldsymbol{t}^{r_s}\rangle$ is already a key of P_e, we update P_e by $P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e-r_1}, \boldsymbol{t}^{r_s}\rangle] = P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e-r_1}, \boldsymbol{t}^{r_s}\rangle] + \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-r_1}\rangle, \boldsymbol{x}^0\right\rangle \cdot P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$; otherwise we add the instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e-r_1}, \boldsymbol{t}^{r_s}\rangle$ to P_e by letting $P_e[\mathsf{SR}\langle \boldsymbol{t}^{r_e-r_1}, \boldsymbol{t}^{r_s}\rangle] = \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_e-r_1}\rangle, \boldsymbol{x}^0\right\rangle \cdot P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$.
   (c) Let $P = P_e$ and update r_e by $r_e = r_e - r_1$.
6. For each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ as a key of P, if $P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$ is 0, we remove $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ from the keys of P.
7. If the size of P is not larger than N, we jump to Step 3; otherwise we start to solve the instances in P and prepare an empty hash table P_u of the same type as P to store the unsolved instances that will be generated later.
8. For each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ as a key of P, we solve it using Proposition 2 within a time limit $\tau^{r_e-r_s}$. If the instance is solved within the time limit, we obtain $\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ and update p by $p = p + \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle, \boldsymbol{x}^0\right\rangle \cdot P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$; if the instance is determined to have the solution 0, we discard it; if the instance is not solved within the time limit, we add the pair to P_u by letting $P_u[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle] = P[\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle]$.
9. If the size of P_u is 0, then p is returned as the final result; otherwise we let $P = P_u$ and jump back to Step 3 to continue expanding and solving instances in P.

As the above process advances, the gap between r_s and r_e decreases progressively, hence we call it a *meet-in-the-middle (MITM)* framework. The parameters $B, N, r_0, r_1, \tau^{r_e-r_s}$ appearing in the process depends on the structure of a cipher, so we will give their values on the spot when applying the framework to a specific cipher. In particular, the time limit $\tau^{r_e-r_s}$ increases as the gap between r_s and r_e shrinks to ensure that more time resources are allocated to solving sufficiently simple instances rather than complex ones. We also set very small values for the forward depth r_0 and the backward depth r_1 (e.g., $r_0 = 5$ and $r_1 = 20$ for Trivium) so that solving instances during the expansion process, namely Step 4b and 5b, can be completed quickly. The way we update P_e in Step 4b and 5b can be thought of as a direct reflection of the role of (forward or backward) overlaps.

For each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ as a key of P in Step 8, it is very likely $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, resulting in the instance being determined to have the solution 0 and then discarded. Therefore, we can adjust Step 4a to directly identify all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that satisfy both $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$

and $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, which can be implemented using the callback interface provided by the Gurobi solver. A similar adjustment can be made to Step 5a. Details on how to use callbacks for expansion are discussed in [21, Sup.Mat. E].

Forward or Backward. We compare the forward expansion and backward expansion strategies heuristically to explain why we set a parameter B in the MITM framework. In Step 3, we have to determine which expansion strategy to adopt for the hash table P. It can be predicted that some instances in P may have forward overlaps at round $r_s + r_0$, while others may have backward overlaps at round $r_e - r_1$. Nevertheless, we observe that for each instance $\mathsf{SR}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle$ in P, the Hamming weight of $\boldsymbol{t}^{r_e}$ is much greater than that of $\boldsymbol{t}^{r_s}$. For example, when $r_s = 25, r_e = 289$ for 849-round Trivium with the cube indices chosen as I_3 in Table 2, $wt(\boldsymbol{t}^{r_s})$ is approximately 40, but $wt(\boldsymbol{t}^{r_e})$ is only about 20. This means, if we assume two instances denoted by $\mathsf{SR}\langle \boldsymbol{t}_0^{r_e}, \boldsymbol{t}_0^{r_s} \rangle$ and $\mathsf{SR}\langle \boldsymbol{t}_1^{r_e}, \boldsymbol{t}_1^{r_s} \rangle$ in P are independently stochastic, i.e., the binary vectors $\boldsymbol{t}_0^{r_e}, \boldsymbol{t}_0^{r_s}, \boldsymbol{t}_1^{r_e}, \boldsymbol{t}_1^{r_s}$ are independent and random, then it is more likely that these two instances will have a forward overlap at round $r_s + r_0$ than that they will have a backward overlap at round $r_e - r_1$, and therefore we believe that the forward expansion lessens the amount of trails that must be enumerated to a greater degree compared to the backward expansion.

However, high Hamming weights also present some challenges. One critical limitation of the forward expansion is that the size of the hash table P grows dramatically as the forward depth r_0 increases. While the backward expansion faces a similar issue as the backward depth r_1 rises, P expands at a smaller rate compared to the forward expansion. For this reason, the backward depth r_1 is set greater than the forward depth r_0. If we use the difference between r_s and r_e to evaluate the difficulty of each instance in P, then the backward expansion closes the gap between r_s and r_e much faster than the forward expansion, and the resulting new instances are simpler.

In summary, the faster closure of the gap between r_s and r_e achieved through backward expansion as opposed to forward expansion renders it most suitable when r_e is significantly large (e.g., over 350 for Trivium). In contrast, when r_e falls below a certain threshold B, forward and backward expansion can be applied interchangeably. This allows balancing the benefits and drawbacks of both strategies to optimize performance.

Ultimately, our overall process for computing $\mathsf{Coe}\langle z, t_I \rangle$ can be summarized as follows: we first reduce the superpoly recovery to the instances of the SR problem using Proposition 1 and then solve each instance using the MITM framework (i.e., Step 1 to Step 9). For the implementation of Proposition 2 in the MITM framework, more details can be taken into account for optimization, which are illustrated in [21, Sect. 4.4].

5 Applications

We apply our MITM framework to three stream ciphers that have been targeted in previous research: Trivium, Grain-128AEAD and Kreyvium. As a result, we are able to verify previous results at a much lower time cost, and also recover the exact superpolies for up to 851 rounds of Trivium and up to 899 rounds of Kreyvium. All experiments are conducted using the Gurobi Solver (version 9.1.2) on a workstation equipped with high-speed processors (totally 32 cores and 64 threads). The source code and some of the recovered superpolies are available in our git repository.

More specifically, when referring to the number of rounds for a stream cipher as r, we are considering that the initialization phase of the cipher consists of r rounds, and we assume that we have access to the output bits produced after this initialization phase. We also omit the description of how to construct an MILP model of CMP or MP for a concrete cipher, as they can be found in previous literature [20,22,24]. For the cube indices that we will mention later in this section, we also attempt to recover the superpolies by re-running the code provided by [22] on our platform, so that we can compare the time consumption of our MITM framework against the framework presented in [22].

For the cube indices used in this section, we always set the non-cube variables to constant 0, though our MITM framework works no matter what constant values the non-cube variables take.

5.1 Superpoly Recovery for Trivium

Trivium is a hardware-efficient, synchronous stream cipher designed by De Cannière and Preneel [7]. The specification of Trivium is provided in [21, Sect. 5.1].

Parameters. For the parameters required by the MITM framework, we set B, N, r_0, r_1 to $350, 50\,000, 5, 20$, respectively. The time limit $\tau^{r_e - r_s}$ is selected according to [21, Algorithm 8].

Superpoly Verification for up to 848 Rounds of Trivium. The cube indices of Trivium used for verification are listed in [21, Table 5]. For each cube listed in [21, Table 5], we verified its superpoly using our MITM framework in almost half the time it took in [22]. The verification results are provided in [21, Table 6].

Superpoly Recovery for up to 851 Rounds of Trivium. To the best of our knowledge, currently there is no dedicated method for selecting a good cube that can yield a simple superpoly. However, we notice that using the vector numeric mapping technique published in [45], the authors in [46] discovered two cubes, which we refer to as I_3 and I_4 in Table 2, whose 844-round superpolies are simpler than superpolies of any cubes previously found. This strongly suggests that the superpolies of these two cubes may still maintain manageable complexity even at higher numbers of rounds beyond 844. With this in mind, we applied the MITM framework to these two cubes and successfully recovered the superpolies

Table 2. The cube indices for TRIVIUM up to 851 rounds

I	Indices	Size
I_3	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 17, 19, 21, 24, 26, 28, 30, 32, 34, 36, 39, 41, 43, 45, 47, 49, 51, 54, 56, 58, 60, 62, 64, 66, 69, 71, 73, 75, 77, 79	44
I_4	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 17, 19, 22, 24, 26, 28, 30, 32, 34, 37, 39, 41, 43, 45, 47, 49, 52, 54, 56, 58, 60, 62, 64, 67, 69, 71, 73, 75, 77, 79	44

Table 3. Details related to the superpolies recovered for TRIVIUM

I	Rounds	Time Cost	Balancedness[§]	#Monomials[⋆]	#Key Bits[⋆]	Degree[⋆]
I_3	849	24 h	0.50	337 087 128 231	80	32
I_4	849	52 h	0.50	189 293 249 301	80	32
I_3	850	81 h	0.50	3 291 633 158 676	80	34
I_3	851	600 h	0.50	20 129 749 853 208	80	36

[§] The balancedness of each superpoly is estimated by testing 2^{15} random keys.
[⋆] An upper bound on the number of monomials, the number of involved key bits or the algebraic degree.

for up to 851 rounds of TRIVIUM. The details of the recovered superpolies are given in Table 3. Since the memory required to store the monomials contained in each superpoly exceeds the memory of workstation, we evaluated the number of monomials, the number of involved key bits and the algebraic degree without considering monomial cancellation, and thus the corresponding data in Table 3 is only an upper bound.

We also attempted to reproduce the 851-round superpoly of I_3 using the framework introduced in [22]. Unfortunately, the program had still not terminated even after two months (1440 h). This demonstrates that our MITM framework outperforms previous approaches in terms of computational efficiency and is capable of exceeding what previous work has been able to achieve.

5.2 Superpoly Recovery for Grain-128AEAD

For the cipher Grain-128AEAD that is described in [21, Sup.Mat. G.2], we set the parameters B, N, r_0, r_1 that are required by the MITM framework to $90, 15\,000, 1, 1$, respectively. The time limit $\tau^{r_e - r_s}$ is selected according to [21, Algorithm 9].

Superpoly Verification for up to 192 Rounds of Grain-128AEAD. In [22], the authors recovered a 192-round superpoly with the cube indices chosen as $I = \{0, 1, 2, \ldots, 95\} \backslash \{42, 43\}$. We re-ran the code provided by them on our platform and verified the result after about 45 d. In contrast, our MITM framework only took about 9 d to recover this superpoly, reducing the time to $\frac{1}{5}$ of the original.

5.3 Superpoly Recovery for Kreyvium

In [21, Sup.Mat. G.3], we provide the specification of Kreyvium and discuss the limitations of the effectiveness of our MITM framework on Kreyvium. For the parameters required by the MITM framework, we set B, N, r_0, r_1 to $270, 15\,000, 5, 20$, respectively. The time limit $\tau^{r_e - r_s}$ is selected according to [21, Algorithm 10].

Superpoly Verification for 895-Round Kreyvium. Using the code provided by [22] on our platform, we reproduced the 895-round superpoly of the cube indices $I_0 = \{0, 1, \ldots, 127\} \backslash \{66, 72, 73, 78, 101, 106, 109, 110\}$ in about two weeks. In contrast, our MITM framework took only about 9 d to recover this superpoly.

Superpoly Recovery for up to 899 Rounds of Kreyvium. By adjusting the cube indices I_0 slightly, we finally determine three cube indices that can lead to more than 895 rounds of simple superpolies. These cube indices are referred to as $I_1 = \{0, 1, \ldots, 127\} \backslash \{66, 73, 106, 109, 110\}$, $I_2 = \{0, 1, \ldots, 127\} \backslash \{66, 73, 106, 110\}$ and $I_3 = \{0, 1, \ldots, 127\} \backslash \{66, 73, 85, 87\}$, respectively. The details of the recovered superpolies are given in [21, Table 4].

6 Key Recovery Attack

Based on the memory-efficient Möbius transform proposed by Dinur at EUROCRYPT 2021 [11], we can mount a key-recovery attack against 851-round Trivium using the superpoly with a time complexity of slightly more than 2^{79} and a memory complexity of about 2^{49} bits. Since in this part we do not bring new techniques, we put the concrete key recovery process in [21, Sect. 6].

Key Recovery for 899-Round Kreyvium. Since the 899-round superpoly of Kreyvium only involves 121 key bits, we can easily mount a key-recovery attack against 899-round Kreyvium with a time complexity of about 2^{127}.

7 Conclusion

In this paper, we analyze algebraically how core monomial trails contribute to the composition of the superpoly, based on the basic definition of core monomial prediction (CMP), thus establishing a theory of superpoly recovery that relies exclusively on CMP. This CMP-based approach can be equivalently linked to MP by means of a variable substitution technique. For a further speedup, we design a meet-in-the-middle (MITM) framework to embed our CMP-based approach. Using this framework, we are able to recover the superpolies for reduced-round versions of the ciphers Trivium and Kreyvium with 851 and 899 rounds, resulting in cube attacks that cover more rounds than previous work.

Acknowledgment. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This research is supported by the National Key Research and Development Program of China (Grant No. 2018YFA0704702), the National Natural Science Foundation of China (Grant No. 62032014, U2336207), Department of Science & Technology of Shandong Province (No. SYS202201), Quan Cheng Laboratory (Grant No. QCLZD202301, QCLZD202306). 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. eSTREAM: the ECRYPT stream cipher project (2018). https://www.ecrypt.eu.org/stream/. Accessed 23 Mar 2021
2. Gurobi Optimization. https://www.gurobi.com
3. ISO/IEC 29192-3:2012: Information technology - Security techniques - Lightweight cryptography - Part 3: stream ciphers. https://www.iso.org/standard/56426.html
4. Aumasson, J.-P., Dinur, I., Meier, W., Shamir, A.: Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 1–22. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03317-9_1
5. Baudrin, J., Canteaut, A., Perrin, L.: Practical cube attack against nonce-misused Ascon. IACR Trans. Symmetric Cryptol. **2022**(4), 120–144 (2022)
6. Boura, C., Coggia, D.: Efficient MILP modelings for sboxes and linear layers of SPN ciphers. IACR Trans. Symmetric Cryptol. **2020**(3), 327–361 (2020)
7. De Cannière, C., Preneel, B.: Trivium. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 244–266. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68351-3_18
8. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream Ciphers: a practical solution for efficient homomorphic-ciphertext compression. J. Cryptol. **31**(3), 885–916 (2018)
9. Che, C., Tian, T.: An experimentally verified attack on 820-round trivium. In: Deng, Y., Yung, M., editors, Information Security and Cryptology - 18th International Conference, Inscrypt 2022, Beijing, China, December 11-13, 2022, Revised Selected Papers, volume 13837 of Lecture Notes in Computer Science, pp. 357–369. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-26553-2_19
10. Delaune, S., Derbez, P., Gontier, A., Prud'homme, C.: A simpler model for recovering superpoly on trivium. In: AlTawy, R., Hülsing, A. (eds.) SAC 2021. LNCS, vol. 13203, pp. 266–285. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99277-4_13
11. Dinur, I.: Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 374–403. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_14
12. Dinur, I., Morawiecki, P., Pieprzyk, J., Srebrny, M., Straus, M.: Cube attacks and cube-attack-like cryptanalysis on the round-reduced Keccak sponge function. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 733–761. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_28

13. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 278–299. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_16
14. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 167–187. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21702-9_10
15. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 33 (2021)
16. Fan, H., Hao, Y., Wang, Q., Gong, X., Jiao, L.: Key filtering in cube attacks from the implementation aspect. In: Deng, J., Kolesnikov, V., Schwarzmann, A.A., editors, Cryptology and Network Security - 22nd International Conference, CANS 2023, Augusta, GA, USA, October 31 - November 2, 2023, Proceedings, vol. 14342 of Lecture Notes in Computer Science, pp. 293–317. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-7563-1_14
17. Fouque, P.-A., Vannet, T.: Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 502–517. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_26
18. Funabiki, Y., Todo, Y., Isobe, T., Morii, M.: Improved integral attack on HIGHT. ACISP **2017**, 363–383 (2017)
19. Hao, Y., Jiao, L., Li, C., Meier, W., Todo, Y., Wang, Q.: Links between division property and other cube attack variants. IACR Trans. Symmetric Cryptol. **2020**(1), 363–395 (2020)
20. Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 466–495. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_17
21. He, J., Hu, K., Lei, H., Wang, M.: Massive superpoly recovery with a meet-in-the-middle framework – improved cube attacks on trivium and kreyvium. Cryptology ePrint Archive, Paper 2024/342 (2024). https://eprint.iacr.org/2024/342
22. He, J., Hu, K., Preneel, B., Wang, M.: Stretching cube attacks: improved methods to recover massive superpolies. In: Agrawal, S., Lin, D., editors, Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part IV, volume 13794 of Lecture Notes in Computer Science, pp. 537–566. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22972-5_19
23. Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 392–421. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_14
24. Hu, K., Sun, S., Wang, M., Wang, Q.: An algebraic formulation of the division property: revisiting degree evaluations, cube attacks, and key-independent sums. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12491, pp. 446–476. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64837-4_15
25. Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round keccak sponge function. In: Coron, J.-S., Nielsen, J.B., editors, Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, volume 10211 of Lecture Notes in Computer Science, pp. 259–288 (2017)

26. Lei, H., He, J., Hu, K., Wang, M.: More balanced polynomials: cube attacks on 810- and 825-round trivium with practical complexities. IACR Cryptol. ePrint Arch., 1237 (2023)
27. Li, Z., Bi, W., Dong, X., Wang, X.: Improved conditional cube attacks on keccak keyed modes with MILP method. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 99–127. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_4
28. Li, Z., Dong, X., Wang, X.: Conditional cube attack on round-reduced ASCON. IACR Trans. Symmetric Cryptol. **2017**(1), 175–202 (2017)
29. Liu, M.: Degree evaluation of NFSR-based cryptosystems. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 227–249. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_8
30. Liu, M., Yang, J., Wang, W., Lin, D.: Correlation Cube Attacks: from weak-key distinguisher to key recovery. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 715–744. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_23
31. Mroczkowski, P., Szmidt, J.: The cube attack on stream cipher Trivium and quadraticity tests. Fundam. Informaticae **114**(3–4), 309–318 (2012)
32. Rohit, R., Kai, H., Sarkar, S., Sun, S.: Misuse-free key-recovery and distinguishing attacks on 7-round ascon. IACR Trans. Symmetric Cryptol. **2021**(1), 130–155 (2021)
33. Rohit, R., Sarkar, S.: Diving deep into the weak keys of round reduced ascon. IACR Trans. Symmetric Cryptol. **2021**(4), 74–99 (2021)
34. Salam, I., Bartlett, H., Dawson, E., Pieprzyk, J., Simpson, L., Wong, K.K.-H.: Investigating cube attacks on the authenticated encryption stream cipher ACORN. In: Batten, L., Li, G., editors, ATIS 2016, volume 651 of Communications in Computer and Information Science, pp. 15–26 (2016)
35. Sasaki, Yu., Todo, Y.: New algorithm for modeling S-box in MILP based differential and division trail search. In: Farshim, P., Simion, E. (eds.) SecITC 2017. LNCS, vol. 10543, pp. 150–165. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69284-5_11
36. Song, L., Guo, J., Shi, D., Ling, S.: New MILP Modeling: improved conditional cube attacks on keccak-based constructions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 65–95. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_3
37. Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 128–157. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_5
38. Sun, L., Wang, W., Wang, M.: MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. IET Inf. Secur. **14**(1), 12–20 (2020)
39. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (Related-key) differential characteristic search: application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_9
40. Sun, Y.: Automatic search of cubes for attacking stream ciphers. IACR Trans. Symmetric Cryptol. **2021**(4), 100–123 (2021)
41. Todo, Y.: Integral cryptanalysis on full MISTY1. In: Gennaro, R., Robshaw, M., editors, CRYPTO 2015, LNCS, vol. 9215, pp. 413–432 (2015)

42. Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_12
43. Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10403, pp. 250–279. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_9
44. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 357–377. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_18
45. Wang, J., Qin, L., Wu, B.: Correlation cube attack revisited: improved cube search and superpoly recovery techniques. Cryptology ePrint Archive, Paper 2023/1408 (2023). https://eprint.iacr.org/2023/1408
46. Wang, J., Wu, B., Liu, Z.: Improved degree evaluation and superpoly recovery methods with application to trivium. CoRR, abs/2201.06394 (2022)
47. Wang, Q., Grassi, L., Rechberger, C.: Zero-sum partitions of PHOTON permutations. In: Smart, N.P. (ed.) CT-RSA 2018. LNCS, vol. 10808, pp. 279–299. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76953-0_15
48. Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10991, pp. 275–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_10
49. Wang, S., Hu, B., Guan, J., Zhang, K., Shi, T.: MILP-aided method of searching division property using three subsets and applications. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 398–427. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_14
50. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_24
51. Ye, C., Tian, T.: A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 172–187. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_11
52. Ye, C.-D., Tian, T.: A practical key-recovery attack on 805-round trivium. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13090, pp. 187–213. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92062-3_7