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

Improved Cube Attacks on Trivium and Kreyvium

Jiahui He[1,3][0000−0002−4033−588X], Kai Hu[1,3,4][0000−0003−3552−7200], Hao Lei[1,3], and Meiqin Wang[1,2,3](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China.
hejiahui2020@mail.sdu.edu.cn,mqwang@sdu.edu.cn,
kai.hu@sdu.edu.cn,leihao@mail.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.

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 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 [15] 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 [6,14,16,18,26,29,32,36,38]. In particular, against the Ascon cipher [17] that is selected by NIST for future standardization of the lightweight cryptography, the cube attack shows outstanding effectiveness [7,30,34,35]. 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 [44] 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 [43]. At FSE 2016, the word-based division property was refined into bit-based division property [46], 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. [52] 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 [40,39,19,49].

Exact superpoly recovery. Initially, the cube attack treats the target cipher as a black box [15,18,33], 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 [46] was introduced into the cube attack [45], so that those secret variables that are not involved in the superpoly could be efficiently identified. In the same year, Liu et al. [31] 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 [50], 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. [51] 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 [21], 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 [25]. The MILP model of MP can be further optimized if we represent the propagation of MP as a directed graph [12]. 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. [24] 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 [51] 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 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 [4,1]) and Kreyvium (designed for Fully Homomorphic Encryption [10]). 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 [13], 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 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

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}	[15]
	709	80	22-23	Cube	2^{23}	$2^{29.14}$	[33]
	767	35	28-31	Cube	2^{31}	2^{45}	[15]
	784	42	30-33	Cube	2^{33}	2^{39}	[18]
	799	18	32-37	Cube	2^{38}	2^{62}	[18]
	802	8	34-37	Cube	2^{37}	2^{72}	[53]
	805	42	32-38	Cube	2^{38}	$2^{41.4}$	[54]
	805	28	28	Correlation Cube	2^{28}	2^{73}	[32]
	806	16	34-37	Cube	$2^{38.64}$	2^{64}	[54]
	806	29	34-37	Cube	2^{39}	2^{39}	[42]
	808	37	39-41	Cube	2^{44}	$2^{44.58}$	[42]
	810	39	40-42	Cube	2^{44}	$2^{44.17}$	[28]
	815	35	44-46	Cube	2^{47}	$2^{47.32}$	[11]
	820	30	48-51	Cube	2^{53}	$2^{53.17}$	[11]
	820†	2^{13}	38	Correlation Cube	2^{51}	2^{60}	[47]
	825	31	49-52	Cube	2^{53}	$2^{53.09}$	[28]
	825†	2^{12}	41	Correlation Cube	2^{53}	2^{60}	[47]
	830†	2^{13}	41	Correlation Cube	2^{54}	2^{60}	[47]
	832	1	72	Cube	2^{72}	2^{79}	[45,51]
	835	41	35	Correlation Cube	2^{35}	2^{75}	[32]
	840	1	78	Cube	2^{78}	$2^{79.6}$	[21]
	840	3	75	Cube	$2^{76.6}$	$2^{77.8}$	[25]
	840	6	47-62	Cube	2^{62}	$2^{76.32}$	[24]
	841	1	78	Cube	2^{78}	$2^{79.6}$	[21]
	841	2	76	Cube	2^{77}	$2^{78.6}$	[25]
	841	3	56-76	Cube	2^{76}	2^{78}	[24]
	842	1	78	Cube	2^{78}	$2^{79.6}$	[21]
	842	2	76	Cube	2^{77}	$2^{78.6}$	[25]
	842	3	56-76	Cube	2^{76}	2^{78}	[24]
	843	2	78	Cube	2^{78}	$2^{79.6}$	[42]
	843	5	56-76	Cube	2^{56}	2^{77}	[24]
	844	2	54-55	Cube	2^{56}	2^{78}	[24]
	845	2	54-55	Cube	2^{56}	2^{78}	[24]
	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	$\leq 2^{119}$	$\leq 2^{127}$	[20,21,42,45,50]
	894	1	119	Cube	2^{119}	2^{127}	[24]
	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	**1**	**124**	**Cube**	$\mathbf{2^{124}}$	$\mathbf{2^{127}}$	**Sect. 5.3**
	899	**1**	**124**	**Cube**	$\mathbf{2^{124}}$	$\mathbf{2^{127}}$	**Sect. 5.3**

* #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 [47] work for only $2^{79.8}$, $2^{79.7}$ and $2^{79.3}$ of the keys in the key space, respectively.

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 $\boldsymbol{x}^{\boldsymbol{u}} \nrightarrow f$.

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 $\boldsymbol{x}^{\boldsymbol{u}} \nrightarrow \boldsymbol{y}^{\boldsymbol{v}}$. The ANF of g is denoted by $\mathsf{Expr}\left\langle \boldsymbol{y}^{\boldsymbol{v}}, \boldsymbol{x} \right\rangle$, which represents a Boolean polynomial of $\boldsymbol{x}$ determined by $\boldsymbol{y}^{\boldsymbol{v}}$. For a polynomial p of $\boldsymbol{y}$, $\mathsf{Expr}\left\langle p, \boldsymbol{x} \right\rangle$ is defined as the summation of $\mathsf{Expr}\left\langle \boldsymbol{y}^{\boldsymbol{v}}, \boldsymbol{x} \right\rangle$ 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 [15] 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

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

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 $\boldsymbol{f} : \mathbb{F}_2^{n_0} \to \mathbb{F}_2^{n_r}$ be a composite vectorial Boolean function built by composition from a sequence of vectorial Boolean functions $\boldsymbol{f}^i : \mathbb{F}_2^{n_i} \to \mathbb{F}_2^{n_{i+1}}, 0 \leq i \leq r-1$, 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 $0 \leq r_s < r_e \leq r$, 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 $j, r_s \leq j \leq r_e - 1$ we have $\pi(\boldsymbol{x}^j, \boldsymbol{u}^j) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{u}^{j+1})$, we write the connection of these transitions as

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

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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ 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 ([25, Proposition 1]). *Use the notations defined above. We have* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{u}^{r_e})$ *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 [25]). *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 [41,37,8], 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 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 $\pi(\boldsymbol{x}^r, \boldsymbol{u}^r) \to z$. 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) $\pi(\boldsymbol{x}^{r_m}, \boldsymbol{u}^{r_m}) \to \pi(\boldsymbol{x}^r, \boldsymbol{u}^r)$, (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 [50]. 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 [50], 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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{u}^{r_s}) \to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, 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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\not\to} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

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 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

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

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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}) \cdot \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, 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 $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s} \right\rangle$ (e.g., when the vectorial Boolean function mapping $\boldsymbol{x}^{r_s}$ to $\boldsymbol{x}^{r_e}$ is simple). However, when $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s} \right\rangle$ 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 Eqn. (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.,*

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

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 $\mathbf{0}$, 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 Eqn. (2) after setting $\boldsymbol{x}^{r_s}[\boldsymbol{\alpha}^{r_s,0_c}]$ to $\mathbf{0}$. 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 $j, r_s < j \leq r_e$ 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* $\mathbf{0}$*, 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*

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

For each $j, 0 < j \leq r$*, 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* $S^r = \{\boldsymbol{u}^r \mid \pi(\boldsymbol{x}^r, \boldsymbol{u}^r) \to z, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,0_c} = \mathbf{0}\}$*. 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. Since $z = \sum_{\pi(\boldsymbol{x}^r, \boldsymbol{u}^r) \to z} \pi(\boldsymbol{x}^r, \boldsymbol{u}^r)$ and $\boldsymbol{x}^0[\boldsymbol{\gamma}^{0,0_c}]$ is set to 0, we have $z = 0$ if $|S^r| = 0$, and $z = \sum_{\boldsymbol{u}^r \in S^r} \pi(\boldsymbol{x}^r, \boldsymbol{u}^r)$ otherwise. Hence, the proposition holds

in the case $|S^r| = 0$. When $|S^r| \neq 0$, we can calculate $\mathsf{Coe}\langle z, t_I\rangle$ as

$$\mathsf{Coe}\langle z, t_I\rangle = \sum_{\boldsymbol{u}^r \in S^r} \mathsf{Coe}\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}), t_I\rangle \cdot \mathsf{Expr}\left\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,1_c}), \boldsymbol{k}\right\rangle. \quad (5)$$

For each $\pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta})$ where $\boldsymbol{u}^r \in S^r$, if $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}), \boldsymbol{x}^0\right\rangle = 0$, then $\mathsf{Sol}\langle \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}, \boldsymbol{t}^0\rangle$ is 0 for any $\boldsymbol{t}^0$. Otherwise, according to Lemma 2, we have

$$\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}), \boldsymbol{x}^0\right\rangle = \sum_{\pi(\boldsymbol{x}^0,\boldsymbol{t}^0) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta})} \mathsf{Sol}\langle \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}, \boldsymbol{t}^0\rangle \cdot \pi(\boldsymbol{x}^0, \boldsymbol{t}^0).$$

Then,

$$\mathsf{Coe}\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}), t_I\rangle = \sum_{\pi(\boldsymbol{x}^0,\boldsymbol{t}^0) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta})} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}, \boldsymbol{t}^0\rangle, \boldsymbol{k}\right\rangle \cdot \mathsf{Coe}\langle \pi(\boldsymbol{x}^0, \boldsymbol{t}^0), t_I\rangle.$$

For each $\boldsymbol{t}^0$ satisfying $\pi(\boldsymbol{x}^0, \boldsymbol{t}^0) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta})$, we have $\boldsymbol{t}^0 \preceq \boldsymbol{\gamma}^{0,\delta}$ and $\pi(\boldsymbol{x}^0, \boldsymbol{t}^0)$ only involves those bits of $\boldsymbol{x}^0$ that are loaded with cube variables, therefore if $\mathsf{Coe}\langle \pi(\boldsymbol{x}^0, \boldsymbol{t}^0), t_I\rangle$ is not 0, it must be 1, which is equivalent to $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^0, \boldsymbol{t}^0), \boldsymbol{v}\right\rangle = t_I$. Hence,

$$\mathsf{Coe}\langle \pi(\boldsymbol{x}^r, \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}), t_I\rangle = \sum_{\boldsymbol{t}^0 \in S^0} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{u}^r \wedge \boldsymbol{\gamma}^{r,\delta}, \boldsymbol{t}^0\rangle, \boldsymbol{k}\right\rangle. \quad (6)$$

Combining Eqn. (6) with Eqn. (5), the proposition is proved. □

In Eqn. (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* $\mathbf{0}$*, 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*

$$\mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle = \sum_{\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^j \rangle, \boldsymbol{x}^{r_s} \right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^j, \boldsymbol{t}^{r_s} \rangle, \tag{7}$$

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 $\mathbf{0}$, $\boldsymbol{x}^j[\boldsymbol{\alpha}^{j,0_c}]$ is also set to $\mathbf{0}$ 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. □

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

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

where the summation is over all $\boldsymbol{t}^{r_s+1}, \ldots, \boldsymbol{t}^{r_e-1}$'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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, 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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})$, 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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})$ 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 $\mathbf{0}$ and calculate $\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle$ according to Lemma 3 as

$$\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s}\rangle = \sum_{\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{m}, \boldsymbol{t}^{m})} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{m-1}\rangle, \boldsymbol{x}^{r_s}\right\rangle \cdot \mathsf{Sol}\langle \boldsymbol{t}^{m-1}, \boldsymbol{t}^{r_s}\rangle. \tag{8}$$

According to the induction hypothesis, $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})$ 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 $\mathbf{0}$ and obtain Eqn. (8). According to the induction hypothesis we made at the beginning,

$$\mathsf{Sol}\langle \boldsymbol{t}^{m-1}, \boldsymbol{t}^{r_s}\rangle = \sum_{\ell \in \pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})} \mathsf{Expr}\left\langle \mathsf{Contr}\langle \ell\rangle, \boldsymbol{x}^{r_s}\right\rangle. \tag{9}$$

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 Eqn. (8) and Eqn. (9), we have

$$\mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{r_s} \rangle = \sum_{(\boldsymbol{t}^{m-1}, \ell) \in S} \mathsf{Expr}\left\langle \mathsf{Sol}\langle \boldsymbol{t}^{m}, \boldsymbol{t}^{m-1} \rangle \cdot \mathsf{Contr}\langle \ell \rangle, \boldsymbol{x}^{r_s} \right\rangle, \tag{10}$$

where the summation is over all the pairs $(\boldsymbol{t}^{m-1}, \ell) \in S$. Notice that Eqn (10) 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* $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. □

As mentioned earlier, each round function $\boldsymbol{f}^j, 0 \le j \le r$ in Eqn. (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 $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}), \boldsymbol{x}^{r_s} \right\rangle$ 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 $\pi(\boldsymbol{x}^{j}, \boldsymbol{t}^{j}) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, then we have $c \cdot \pi(\boldsymbol{x}^{j}, \boldsymbol{t}^{j}) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, 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 1: Iterative variable substitutions

Procedure `VariableSubstitution`($\mathsf{SR}_{\boldsymbol{\alpha}^{r_s},\delta,\boldsymbol{\alpha}^{r_s},1_c}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$)**:**
- Initialize an integer $j = r_s$ and an empty hash table T
- Initialize $r_e - r_s$ empty vectors $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \ldots, \boldsymbol{c}^{r_e-1}$ of variables
- **while** $j < r_e$ **do**
  - **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**
    - Calculate $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$
    - Substitute $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$ by a new variable c such that $c \cdot \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$
    - Add c to the end of $\boldsymbol{c}^j$, and store the substitution by letting $T[c] = \mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j\rangle$
  - Increment j by one // $j = j + 1$
- **return** $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \ldots, \boldsymbol{c}^{r_e-1}, T$

Rule 1 (COPY) *Assume the round function* $\boldsymbol{f}^j$ *is the basic operation COPY with* $x^j[0] \xrightarrow{COPY} (x^{j+1}[0], \ldots, x^{j+1}[m-1])$. *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* $(x^j[0], x^j[1], \ldots, x^j[m-1]) \xrightarrow{AND} x^{j+1}[0]$. *If* $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \ldots, x^j[m-1].F\}$ *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* $(x^j[0], x^j[1], \ldots, x^j[m-1]) \xrightarrow{XOR} x^{j+1}[0]$. *If* $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \ldots, x^j[m-1].F\}$, *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* $S_0, \ldots, S_{m-1}$. *We can set* $\boldsymbol{c}^j$ *and* T *following Algorithm 2.*

Utilizing the new variables $\boldsymbol{c}^{r_s}, \boldsymbol{c}^{r_s+1}, \ldots, \boldsymbol{c}^{r_e-1}, T$ 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*

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

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

for *each S-box* $S_i, 0 \leq i < m$ **do**
  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* δ *bits of* S_i, $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ *only relates to the output* δ *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**
    Calculate $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$
    Substitute $\mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$ by a new variable c such that $c \cdot \pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \to \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$
    Add c to the end of $\boldsymbol{c}^j$, and store the substitution by letting $T[c] = \mathsf{Sol}\langle \boldsymbol{t}^{j+1}, \boldsymbol{t}^j \rangle$

if and only if there is a monomial trail written as

$$\begin{aligned}
&\pi(\boldsymbol{c}^{r_s} \| \cdots \| \boldsymbol{c}^{r_e-1} \| \boldsymbol{x}^{r_s}, \boldsymbol{w}^{r_s} \| \cdots \| \boldsymbol{w}^{r_e-1} \| \boldsymbol{t}^{r_s}) \\
&\to \pi(\boldsymbol{c}^{r_s+1} \| \cdots \| \boldsymbol{c}^{r_e-1} \| \boldsymbol{x}^{r_s+1}, \boldsymbol{w}^{r_s+1} \| \cdots \| \boldsymbol{w}^{r_e-1} \| \boldsymbol{t}^{r_s+1}) \\
&\to \cdots \\
&\to \pi(\boldsymbol{c}^{r_e-1} \| \boldsymbol{x}^{r_e-1}, \boldsymbol{w}^{r_e-1} \| \boldsymbol{t}^{r_e-1}) \\
&\to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e}).
\end{aligned} \tag{11}$$

Moreover, by substituting each variable c *in* $\boldsymbol{c}^j, r_s \leq j < r_e$ *back with* $T[c]$, *we can get a polynomial of* $\boldsymbol{x}^{r_s}, \ldots, \boldsymbol{x}^{r_e-1}$ *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 $j, r_s \leq j < r_e$, 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 $\pi(\boldsymbol{c}^{r_s} \| \boldsymbol{x}^{r_s}, \boldsymbol{w}^{r_s} \| \boldsymbol{t}^{r_s}) \to \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}), \pi(\boldsymbol{c}^{r_s+1} \| \boldsymbol{x}^{r_s+1}, \boldsymbol{w}^{r_s+1} \| \boldsymbol{t}^{r_s+1}) \to \pi(\boldsymbol{x}^{r_s+2}, \boldsymbol{t}^{r_s+2}), \ldots, \pi(\boldsymbol{c}^{r_e-1} \| \boldsymbol{x}^{r_e-1}, \boldsymbol{w}^{r_e-1} \| \boldsymbol{t}^{r_e-1}) \to \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ yields an equivalent monomial trail described in the proposition. □

We can therefore also solve $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s}, \delta, \boldsymbol{\alpha}^{r_s}, 1_c}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle$ by extracting all monomials trails of the form (11) 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 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 [24,25], 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, Eqn. (7) 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.

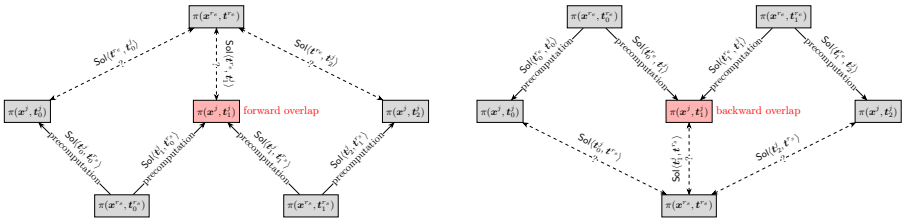

Fig. 1: Forward and backward overlaps

The forward and backward overlaps can be illustrated by 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 Eqn. (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 Eqn. (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 $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\nrightarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$, 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 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.

4.4 Optimizing the Implementation of Proposition 2

According to Proposition 2, one straightforward implementation of solving 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 to construct an MILP model using the propagation models of CMP and then extract all core monomial trails from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ to $\pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ via model solution enumeration, as shown in Algorithm 5. This procedure can be prohibitively time-intensive if there is a vast number (e.g., more than 100 000) of core monomial trails, so we propose two optimizations to speed up it.

Optimization 1: equivalent instances. We begin by defining a special relationship between instances of the SR problem. Specifically, we say that multiple instances are *equivalent* if their solutions can be derived from each other. Having established this notion of equivalent instances, the following lemma presents a constructive approach for obtaining additional instances that stand in this relationship to a given instance.

Lemma 4 (Equivalent instances). *Given one instance A of the SR problem denoted by $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ with $\boldsymbol{t}^{r_s} \prec \boldsymbol{\alpha}^{r_s,\delta}$, we arbitrarily choose a binary vector $\boldsymbol{\beta}^{r_s,\delta}$ such that $\boldsymbol{t}^{r_s} \preceq \boldsymbol{\beta}^{r_s,\delta} \prec \boldsymbol{\alpha}^{r_s,\delta}$. Further, letting $\boldsymbol{\beta}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$, $\boldsymbol{\beta}^{r_s,0_c} = \neg(\boldsymbol{\beta}^{r_s,\delta} \vee \boldsymbol{\beta}^{r_s,1_c})$, and for each $j, r_s < j \leq r_e$ $\boldsymbol{\beta}^{j,\delta}, \boldsymbol{\beta}^{j,1_c}, \boldsymbol{\beta}^{j,0_c}$ be calculated from $\boldsymbol{\beta}^{r_s,\delta}, \boldsymbol{\beta}^{r_s,1_c}, \boldsymbol{\beta}^{r_s,0_c}$ according to the operation rules of flags, we can construct another instance B, denoted by $\mathsf{SR}_{\boldsymbol{\beta}^{r_s,\delta},\boldsymbol{\beta}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,\delta}, \boldsymbol{t}^{r_s}\rangle$. If*

$\boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,0_c} = \mathbf{0}$ *and* $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,1_c}), \boldsymbol{x}^{r_s} \right\rangle \neq 0$ *after setting* $\boldsymbol{x}^{r_s}[\boldsymbol{\beta}^{r_s,0_c}]$ *to* $\mathbf{0}$*, instances A and B are equivalent; otherwise the solution of instance A is* 0*.*

Proof. After setting $\boldsymbol{x}^{r_s}[\boldsymbol{\beta}^{r_s,0_c}]$ to $\mathbf{0}$, it follows from the definition of the SR problem that

$$\begin{aligned} &\mathsf{Sol}_{\boldsymbol{\alpha}^{r_s},\delta,\boldsymbol{\alpha}^{r_s},1_c}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle \\ =\ & C \cdot \mathsf{Sol}_{\boldsymbol{\beta}^{r_s},\delta,\boldsymbol{\beta}^{r_s},1_c}\langle \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,\delta}, \boldsymbol{t}^{r_s} \rangle, \end{aligned}$$

where

$$C = \begin{cases} \mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,1_c}), \boldsymbol{x}^{r_s} \right\rangle & \text{if } \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,0_c} = \mathbf{0}\ , \\ 0 & \text{otherwise}\ . \end{cases}$$

If $\boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,0_c} = \mathbf{0}$ and $\mathsf{Expr}\left\langle \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,1_c}), \boldsymbol{x}^{r_s} \right\rangle \neq 0$, C is not 0 and can be calculated easily, indicating that instances A and B are equivalent; otherwise C is 0, resulting in the solution of instance A being 0. □

If an instance denoted by $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s},\delta,\boldsymbol{\alpha}^{r_s},1_c}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle$ satisfies $\boldsymbol{\alpha}^{r_s,\delta} = \boldsymbol{t}^{r_s}$, we refer to it as an *optimal instance*. According to Lemma 4, a non-optimal instance either has the solution 0 or can be transformed into an equivalent optimal instance. Compared to the non-optimal instances, optimal instances can be solved more efficiently due to reduced number of core monomial trails required when applying Proposition 2.

Proposition 4. *Assuming each round function* $\boldsymbol{f}^j$ *is a basic operation, let instances A and B be defined as in Lemma 4. If* $\boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,0_c} = \mathbf{0}$ *(this condition is weaker than the condition that instances A and B are equivalent), solving instance B using Proposition 2 requires no more core monomial trails than solving instance A.*

Proof. The proof of this proposition is provided in Sup.Mat. C. □

Optimization 2: partial XOR and two-step trail extraction. For 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 extract the core monomial trails in $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ in two steps, by dividing the propagation rule of XOR into two parts, which are referred to as *partial XOR* and the *complement of partial XOR*. These two parts and the corresponding MILP models are illustrated in Sup.Mat. F.1.

In the first step, we construct an MILP model $\mathcal{M}_0$ following the propagation models of CMP, but for the basic operation XOR we use the partial XOR model (Model 8) instead of the full XOR model (Model 6). We then find all solutions of $\mathcal{M}_0$ by an MILP solver and obtain a subset of $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. The construction of $\mathcal{M}_0$ is provided in Algorithm 6.

In the second step, we aim to capture the missing trails in $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ due to the use of partial XOR. To this end, we use Rule 13 and Model 9 to capture the transition $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \overset{\mathcal{C}}{\to} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ that is allowed by

the full XOR rule but not by the partial XOR rule. Following Algorithm 7, we can construct an MILP model $\mathcal{M}_1$ and all solutions of $\mathcal{M}_1$ exactly correspond to the core monomial trails in $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ that are not identified in the first step.

Since a CMP transition characterizable by the complete XOR rule will probably also be characterizable by the partial XOR rule, we anticipate the number of $\mathcal{M}_0$ solutions will surpass the number of $\mathcal{M}_1$ solutions. This suggests the primary computational difficulty lies in solving $\mathcal{M}_0$, which is less complex than the MILP model used in the direct implementation of Proposition 2. Our experimental results confirmed that if $\mathcal{M}_0$ is solved within a time limit, then usually $\mathcal{M}_1$ is also solvable within the same time limit.

The final implementation of Proposition 2 utilizes both optimizations together as follows: When given an instance, we first check if it is optimal. If not, then by Lemma 4 it either has the solution 0 or can be reduced to an optimal instance. If it is already an optimal instance, we solve it with the two-step optimization.

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 [21,25,22]. 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 [9]. It has been selected as one of the Profile 2 Algorithms in the eSTREAM portfolio [1] and standardized by ISO/IEC as International Standard

29192-3 [4], cementing its status as a trusted cryptographic primitive. The internal state of TRIVIUM is represented by a 288-bit state $\boldsymbol{s} = (s[0], s[1], \ldots, s[287])$ divided into three registers. At the initialization phase, the 80-bit secret key $\boldsymbol{K}$ is loaded to the first register, and the 80-bit initialization vector $\boldsymbol{IV}$ is loaded into the second register. The other state bits are set to 0 except the last three bits in the third register. Namely, the initial state is represented as

$$\begin{aligned}
&(s[0], s[1], \ldots, s[92]) \leftarrow (K[0], K[1], \ldots, K[79], 0, \ldots, 0)\\
&(s[93], s[94], \ldots, s[176]) \leftarrow (IV[0], IV[1], \ldots, IV[79], 0, \ldots, 0)\\
&(s[177], s[178], \ldots, s[287]) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)
\end{aligned}$$

The update function is given by the following pseudo-code:

$$\begin{aligned}
&t_1 \leftarrow s[65] \oplus s[90] \cdot s[91] \oplus s[92] \oplus s[170]\\
&t_2 \leftarrow s[161] \oplus s[174] \cdot s[175] \oplus s[176] \oplus s[263]\\
&t_3 \leftarrow s[242] \oplus s[285] \cdot s[286] \oplus s[287] \oplus s[68]\\
&(s[0], s[1], \ldots, s[92]) \leftarrow (t_3, s[0], s[1], \ldots, s[91])\\
&(s[93], s[94], \ldots, s[176]) \leftarrow (t_1, s[93], s[94], \ldots, s[175])\\
&(s[177], s[178], \ldots, s[287]) \leftarrow (t_2, s[177], s[178], \ldots, s[286])
\end{aligned}$$

After 1152 rounds of initialization, one key stream bit $z = s[65] \oplus s[92] \oplus s[161] \oplus s[176] \oplus s[242] \oplus s[287]$ is produced by every update function.

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 Algorithm 8.

Superpoly verification for up to 848 rounds of Trivium. The cube indices of TRIVIUM used for verification are listed in Table 5. For each cube listed in 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 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 [47], the authors in [48] 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 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 termi-

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 hours	0.50	337 087 128 231	80	32
I_4	849	52 hours	0.50	189 293 249 301	80	32
I_3	850	81 hours	0.50	3 291 633 158 676	80	34
I_3	851	600 hours	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.

nated even after two months (1440 hours). 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, whose specification is provided in 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 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 days. In contrast, our MITM framework only took about 9 days to recover this superpoly, reducing the time to $\frac{1}{5}$ of the original.

5.3 Superpoly Recovery for Kreyvium

In 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 Algorithm 10.

Table 4: Details related to the superpolies recovered for Kreyvium

I	Rounds	Time Cost	Balancedness	#Monomials	#Key Bits	Degree
I_1	896	180 hours	0.50	29	30	2
I_2	896	58 hours	0	0	0	0
I_2	897	67 hours	0.50	83	60	3
I_3	898	182 hours	0.50	102	75	4
I_3	899	272 hours	0.50	4793	121	7

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 days 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 Table 4.

6 Key Recovery Attack

Since the 851-round superpoly of Trivium contains a large number of monomials and involves full key bits, we encounter another crucial challenge in the cube attack, namely how to extract the information of the secret key from such a massive superpoly.

Assume the 851-round superpoly is denoted by p and leads to an equation $p(\boldsymbol{k}) = a$ in the online phase, where a is constant 0 or 1. A straightforward idea is that we guess 79 secret key bits and reduce $p(\boldsymbol{k}) = a$ to an equation of the remaining one secret key bit, then we solve this equation and check if our guess is correct by performing an encryption call. This idea works only when evaluating p once is faster than performing one encryption call, but this is obviously impossible for a massive p. Another typical idea is the Möbius transform, with which we can compute the truth table of a Boolean function from its ANF more efficiently. The standard Möbius transform is introduced in Sup.Mat. H. We assume the standard Möbius transform requires $n \cdot 2^n$ bit operations and 2^n-bits memory.

Memory-efficient Möbius transform [13]. We want to evaluate a polynomial $F(x[0], \ldots, x[n-1])$ of degree d on the space $\{0, 1\}^n$ using the Möbius transform. Assume that d is not too large and the polynomial is represented by a binary vector of size $\binom{n}{\downarrow d}$, where $\binom{n}{\downarrow d} = \sum_{i=0}^{d} \binom{n}{i}$. A fast algorithm for computing the Möbius transform is based on the decomposition

$$F(x[0], \ldots, x[n-1]) = x[0] \cdot F_0(x[1], \ldots, x[n-1]) + F_1(x[1], \ldots, x[n-1]). \quad (12)$$

Thus, we can evaluate F recursively by first evaluating $F_1(x[1], \ldots, x[n-1])$ (i.e., $F(\boldsymbol{x})$ for $x[0] = 0$), and then calculating and evaluating $F_0(x[1], \ldots, x[n-1]) + F_1(x[1], \ldots, x[n-1])$ (i.e., $F(\boldsymbol{x})$ for $x[0] = 1$). Denoting the memory complexity required to evaluate a d-degree polynomial of n variables by $M(n, d)$, we have $M(n, d) = M(n-1, d) + \binom{n}{\downarrow d}$ and $M(n, n) = 2^n$. Therefore, the total memory complexity is less than $n \cdot \binom{n}{\downarrow d}$, and the time complexity in bit operations is bounded by $\binom{n}{\downarrow d} + 2 \cdot \binom{n-1}{\downarrow d} + \cdots + 2^{n-d-1} \cdot \binom{d+1}{\downarrow d} + 2^{n-d} \cdot d \cdot 2^d$, which is below $n \cdot 2^n$.

Thanks to the memory-efficient Möbius transform proposed by Dinur at EUROCRYPT 2021 [13], we can further reduce the memory complexity. Specifically, we choose a parameter m. At the top m levels of the recursion, we recursively evaluate the input polynomial on all 2^m values of $x[0], \ldots, x[m-1]$ independently. At the bottom levels, we switch to the in-place implementation of the standard Möbius transform to evaluate the polynomial on all values of $x[m], \ldots, x[n-1]$. The memory required for the independent evaluations at the top m levels is bounded by $2 \cdot \binom{n}{\downarrow d}$, while for the in-place Möbius transform, 2^{n-m}-bits memory is sufficient. In [13], the parameter m is chosen as $m \approx n - \log \binom{n}{\downarrow d}$, so that the final memory complexity is bounded by $3 \cdot \binom{n}{\downarrow d}$. The complexity analysis of the memory-efficient Möbius transform is general, but in practice we are dealing with a sparse superpoly whose number of monomials is known, hence it may be possible to further reduce the memory complexity.

Key recovery for 851-round Trivium. We focus on the equation $p(\boldsymbol{k}) = a$ established by the 851-round superpoly $p(\boldsymbol{k})$. According to our tests without considering the monomial cancellation, the superpoly $p(\boldsymbol{k})$ contains approximately 20 129 749 853 208 monomials and its degree is bounded by 36, we therefore roughly estimate that the actual number of monomials in $p(\boldsymbol{k})$ is $\frac{20\,129\,749\,853\,208}{10} \approx 2^{40.87}$.

Instead of representing $p(\boldsymbol{k})$ by a binary vector of size $\binom{80}{\downarrow 36}$, we represent it as an array of size $2^{40.87}$. Each element in the array is 80 bits in size and represents a monomial in $p(\boldsymbol{k})$, so the array takes up a total of $2^{40.87} \times 80 \approx 2^{47.19}$ bits of memory. Similar to the memory-efficient Möbius transform, we choose a parameter m and solve the equation $p(\boldsymbol{k}) = a$ as follows:

1. Based on the formula (12), we first recursively evaluate $p(\boldsymbol{k})$ on all 2^m values of $k[0], \ldots, k[m-1]$ independently.
2. For each value of $k[0], \ldots, k[m-1]$ that reduces $p(\boldsymbol{k})$ to a polynomial p' of $k[m], \ldots, k[79]$:
   (a) We switch to the in-place implementation of the standard Möbius transform to build the truth table of p'.
   (b) If the value of p' is equal to a on a value of $k[m], \ldots, k[79]$, then together with the value of $k[0], \ldots, k[m-1]$ we get a candidate solution for $p(\boldsymbol{k}) = a$. We can check if this candidate solution is correct by performing an additional encryption call.

The memory required for the independent evaluations is bounded by $2 \cdot 2^{47.19} = 2^{48.19}$ bits, while the in-place implementation of the Möbius transform

requires 2^{80-m}-bits memory, so we choose $m = 32$ and thus our memory complexity is slightly more than 2^{49} bits. The final time complexity is bounded by $2^{32} \cdot (2^{40.87} + 48 \times 2^{48}) \approx 2^{85.58}$ bit operations plus 2^{79} encryption calls, which is slightly more than 2^{79} TRIVIUM calls.

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: 2021-03-23.
2. Gurobi Optimization. https://www.gurobi.com.
3. Gurobi Optimization Reference Manual. https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.1/refman.pdf.
4. ISO/IEC 29192-3:2012: Information technology Security techniques Lightweight cryptography part 3: Stream ciphers. https://www.iso.org/standard/56426.html.
5. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *Int. J. Wirel. Mob. Comput.*, 5(1):48–59, 2011.
6. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and trivium. In Orr Dunkelman, editor, *FSE 2009*, volume 5665 of *LNCS*, pages 1–22. Springer, 2009.

7. Jules Baudrin, Anne Canteaut, and Léo Perrin. Practical cube attack against nonce-misused ascon. *IACR Trans. Symmetric Cryptol.*, 2022(4):120–144, 2022.
8. Christina Boura and Daniel Coggia. Efficient MILP modelings for sboxes and linear layers of SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):327–361, 2020.
9. Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 244–266. Springer, 2008.
10. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. *J. Cryptol.*, 31(3):885–916, 2018.
11. Cheng Che and Tian Tian. An experimentally verified attack on 820-round trivium. In Yi Deng and Moti Yung, 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*, pages 357–369. Springer, 2022.
12. Stéphanie Delaune, Patrick Derbez, Arthur Gontier, and Charles Prud'homme. A simpler model for recovering superpoly on trivium. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography - 28th International Conference, SAC 2021, Virtual Event, September 29 - October 1, 2021, Revised Selected Papers*, volume 13203 of *Lecture Notes in Computer Science*, pages 266–285. Springer, 2021.
13. Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021.
14. Itai Dinur, Pawel Morawiecki, Josef Pieprzyk, Marian Srebrny, and Michal Straus. Cube attacks and cube-attack-like cryptanalysis on the round-reduced keccak sponge function. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 733–761. Springer, 2015.
15. Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.
16. Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 167–187. Springer, 2011.
17. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.
18. Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 502–517. Springer, 2013.
19. Yuki Funabiki, Yosuke Todo, Takanori Isobe, and Masakatu Morii. Improved integral attack on HIGHT. In *ACISP 2017*, pages 363–383, 2017.

20. Yonglin Hao, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. Links between division property and other cube attack variants. *IACR Trans. Symmetric Cryptol.*, 2020(1):363–395, 2020.
21. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 466–495. Springer, 2020.
22. Jiahui He, Kai Hu, Bart Preneel, and Meiqin Wang. Stretching cube attacks: Improved methods to recover massive superpolies. In Shweta Agrawal and Dongdai Lin, 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*, pages 537–566. Springer, 2022.
23. Martin Hell, Thomas Johansson, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEAD - A lightweight AEAD stream cipher. *NIST Lightweight Cryptography, Round*, 3, 2019.
24. Kai Hu, Siwei Sun, Yosuke Todo, Meiqin Wang, and Qingju Wang. Massive superpoly recovery with nested monomial predictions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 392–421. Springer, 2021.
25. Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020.
26. Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round keccak sponge function. In Jean-Sébastien Coron and Jesper Buus Nielsen, 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*, pages 259–288, 2017.
27. Antoine Joux. *Algorithmic cryptanalysis*. CRC press, 2009.
28. Hao Lei, Jiahui He, Kai Hu, and Meiqin Wang. More balanced polynomials: Cube attacks on 810- and 825-round trivium with practical complexities. *IACR Cryptol. ePrint Arch.*, page 1237, 2023.
29. Zheng Li, Wenquan Bi, Xiaoyang Dong, and Xiaoyun Wang. Improved conditional cube attacks on keccak keyed modes with MILP method. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 99–127. Springer, 2017.
30. Zheng Li, Xiaoyang Dong, and Xiaoyun Wang. Conditional cube attack on round-reduced ASCON. *IACR Trans. Symmetric Cryptol.*, 2017(1):175–202, 2017.
31. Meicheng Liu. Degree evaluation of NFSR-based cryptosystems. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10403 of *LNCS*, pages 227–249. Springer, 2017.

32. Meicheng Liu, Jingchun Yang, Wenhao Wang, and Dongdai Lin. Correlation cube attacks: From weak-key distinguisher to key recovery. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 715–744. Springer, 2018.
33. Piotr Mroczkowski and Janusz Szmidt. The cube attack on stream cipher Trivium and quadraticity tests. *Fundam. Informaticae*, 114(3-4):309–318, 2012.
34. Raghvendra Rohit, Kai Hu, Sumanta Sarkar, and Siwei Sun. Misuse-free key-recovery and distinguishing attacks on 7-round ascon. *IACR Trans. Symmetric Cryptol.*, 2021(1):130–155, 2021.
35. Raghvendra Rohit and Santanu Sarkar. Diving deep into the weak keys of round reduced ascon. *IACR Trans. Symmetric Cryptol.*, 2021(4):74–99, 2021.
36. Md. Iftekhar Salam, Harry Bartlett, Ed Dawson, Josef Pieprzyk, Leonie Simpson, and Kenneth Koon-Ho Wong. Investigating cube attacks on the authenticated encryption stream cipher ACORN. In Lynn Batten and Gang Li, editors, *ATIS 2016*, volume 651 of *Communications in Computer and Information Science*, pages 15–26, 2016.
37. Yu Sasaki and Yosuke Todo. New algorithm for modeling s-box in MILP based differential and division trail search. In Pooya Farshim and Emil Simion, editors, *SecITC 2017*, volume 10543 of *LNCS*, pages 150–165. Springer, 2017.
38. Ling Song, Jian Guo, Danping Shi, and San Ling. New MILP modeling: Improved conditional cube attacks on keccak-based constructions. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2018.
39. Ling Sun, Wei Wang, and Meiqin Wang. Automatic search of bit-based division property for ARX ciphers and word-based division property. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 128–157. Springer, 2017.
40. Ling Sun, Wei Wang, and Meiqin Wang. Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. *IET Information Security*, 14(1):12–20, 2020.
41. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, 2014.
42. Yao Sun. Automatic search of cubes for attacking stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2021(4):100–123, 2021.
43. Yosuke Todo. Integral cryptanalysis on full MISTY1. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015*, volume 9215 of *LNCS*, pages 413–432, 2015.
44. Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.
45. Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Ho-

vav Shacham, editors, *CRYPTO 2017*, volume 10403 of *LNCS*, pages 250–279. Springer, 2017.
46. Yosuke Todo and Masakatu Morii. Bit-based division property and application to Simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016.
47. Jianhua Wang, Lu Qin, and Baofeng Wu. Correlation cube attack revisited: Improved cube search and superpoly recovery techniques. Cryptology ePrint Archive, Paper 2023/1408, 2023. https://eprint.iacr.org/2023/1408.
48. Jianhua Wang, Baofeng Wu, and Zhuojun Liu. Improved degree evaluation and superpoly recovery methods with application to trivium. *CoRR*, abs/2201.06394, 2022.
49. Qingju Wang, Lorenzo Grassi, and Christian Rechberger. Zero-sum partitions of PHOTON permutations. In Nigel P. Smart, editor, *CT-RSA 2018*, volume 10808 of *LNCS*, pages 279–299. Springer, 2018.
50. Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10991 of *LNCS*, pages 275–305. Springer, 2018.
51. SenPeng Wang, Bin Hu, Jie Guan, Kai Zhang, and Tairong Shi. MILP-aided method of searching division property using three subsets and applications. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019*, volume 11923 of *LNCS*, pages 398–427. Springer, 2019.
52. Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 648–678. Springer, 2016.
53. Chen-Dong Ye and Tian Tian. A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers. In Willy Susilo and Guomin Yang, editors, *ACISP 2018*, volume 10946 of *LNCS*, pages 172–187. Springer, 2018.
54. Chen-Dong Ye and Tian Tian. A practical key-recovery attack on 805-round trivium. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 187–213. Springer, 2021.

Appendix

A Propagation Rules and Models for MP

In this section, we give the concrete propagation rules and models for MP. We consider a round function $\boldsymbol{f}^j, 0 \leq j < r$, whose input and output are $\boldsymbol{x}^j \in \mathbb{F}_2^{n_j}$ and $\boldsymbol{x}^{j+1} \in \mathbb{F}_2^{n_{j+1}}$, respectively.

A.1 Propagation Rules

Rule 5 (COPY [21]) *Let $\boldsymbol{f}^j$ be the basic operation COPY ($n_{j+1} = n_j + m - 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0], \ldots, x^j[0], x^j[1], \ldots, x^j[n_j - 1])$, where the first m bits of $\boldsymbol{x}^{j+1}$ are generated from $x^j[0]$ by COPY. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

- $t^{j+1}[m + k - 1] = t^j[k] \ \forall \ 1 \leq k \leq n_j - 1$.
- $t^{j+1}[0] \vee t^{j+1}[1] \vee \cdots \vee t^{j+1}[m-1] = t^j[0]$.

Rule 6 (AND [21]) *Let $\boldsymbol{f}^j$ be the basic operation AND ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \wedge x^j[1] \wedge \cdots \wedge x^j[m - 1], x^j[m], \ldots, x^j[n_j - 1])$. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

- $t^{j+1}[k - m + 1] = t^j[k] \ \forall \ m \leq k \leq n_j - 1$.
- $\forall \ k \in \{0, 1, \ldots, m-1\}$, $t^{j+1}[0] = t^j[k]$.

Rule 7 (XOR [21]) *Let $\boldsymbol{f}^j$ be the basic operation XOR ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \oplus x^j[1] \oplus \cdots \oplus x^j[m - 1], x^j[m], \ldots, x^j[n_j - 1])$. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \rightarrow \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

- $t^{j+1}[k - m + 1] = t^j[k] \ \forall \ m \leq k \leq n_j - 1$.
- $t^{j+1}[0] = t^j[0] + t^j[1] + \cdots + t^j[m-1]$.

A.2 Propagation Models

Model 1 (COPY [21]) *Let $\boldsymbol{f}^j$ be the basic operation COPY ($n_{j+1} = n_j + m - 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0], \ldots, x^j[0], x^j[1], \ldots, x^j[n_j - 1])$, where the first m bits of $\boldsymbol{x}^{j+1}$ are generated from $x^j[0]$ by COPY. Let $a, b_0, b_1, \ldots, b_{m-1}$ be the MILP variables corresponding to $x^j[0], x^{j+1}[0], \ldots, x^{j+1}[m - 1]$ such that $a \xrightarrow{COPY} (b_0, b_1, \ldots, b_{m-1})$ is a propagation trail of COPY. The following inequalities are sufficient to describe all the valid trails for COPY:*

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

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

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

Model 2 (AND [21]) *Let $\boldsymbol{f}^j$ be the basic operation AND ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \wedge x^j[1] \wedge \cdots \wedge x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. Let $a_0, a_1, \ldots, a_{m-1}, b$ be the MILP variables corresponding to $x^j[0], \ldots, x^j[m-1], x^{j+1}[0]$ such that $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{AND} b$ is a propagation trail of AND. The following inequalities are sufficient to describe all the valid trails for AND:*

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

Model 3 (XOR [21]) *Let $\boldsymbol{f}^j$ be the basic operation XOR ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \oplus x^j[1] \oplus \cdots \oplus x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. Let $a_0, a_1, \ldots, a_{m-1}, b$ be the MILP variables corresponding to $x^j[0], \ldots, x^j[m-1], x^{j+1}[0]$ such that $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{XOR} b$ is a propagation trail of XOR. The following inequalities are sufficient to describe all the valid trails for XOR:*

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

B Propagation Rules and Models for CMP

In this section, we give the concrete propagation rules and models for CMP. We consider a round function $\boldsymbol{f}^j, 0 \le j < r$, whose input and output are $\boldsymbol{x}^j \in \mathbb{F}_2^{n_j}$ and $\boldsymbol{x}^{j+1} \in \mathbb{F}_2^{n_{j+1}}$, respectively. Here, to emphasize that our theory is independent of how round functions are defined, we also give the propagation rule of CMP when passing through S-boxes. Note that the following propagation rules are based on the premise that all 0_c bits have been set to constant 0. When all state bits become δ bits, the CMP propagation will degenerate to MP propagation.

B.1 Propagation Rules

Rule 8 (COPY [22]) *Let $\boldsymbol{f}^j$ be the basic operation COPY ($n_{j+1} = n_j + m - 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0], \ldots, x^j[0], x^j[1], \ldots, x^j[n_j - 1])$, where the first m bits of $\boldsymbol{x}^{j+1}$ are generated from $x^j[0]$ by COPY. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ that only relates to the δ bits of round j, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

– $t^{j+1}[m+k-1] = t^j[k] \ \forall\ 1 \le k \le n_j - 1$.
– $t^{j+1}[0] \vee t^{j+1}[1] \vee \cdots \vee t^{j+1}[m-1] = t^j[0]$.

Rule 9 (AND [22]) *Let $\boldsymbol{f}^j$ be the basic operation AND ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \wedge x^j[1] \wedge \cdots \wedge x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ that only relates to the δ bits of round j, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

– $t^{j+1}[k-m+1] = t^j[k] \ \forall\ m \le k \le n_j - 1$.
– $t^{j+1}[0] = t^j[0] \vee t^j[1] \vee \cdots \vee t^j[m-1]$.
– $\forall\ k \in \{0, 1, \ldots, m-1\}$, *if* $x^j[k].F = \delta$, $t^{j+1}[0] = t^j[k]$.

Rule 10 (XOR [22]) *Let $\boldsymbol{f}^j$ be the basic operation XOR ($n_{j+1} = n_j - m + 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0] \oplus x^j[1] \oplus \cdots \oplus x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. Given the monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ that only relates to the δ bits of round j, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when $\boldsymbol{t}^{j+1}$ satisfies*

– $t^{j+1}[k-m+1] = t^j[k] \ \forall\ m \le k \le n_j - 1$.
– *If* $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \ldots, x^j[m-1]\}$, $t^{j+1}[0] \ge t^j[0] + t^j[1] + \cdots + t^j[m-1]$; *otherwise* $t^{j+1}[0] = t^j[0] + t^j[1] + \cdots + t^j[m-1]$.

Note that when $\{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \ldots, x^j[m-1]\}$, we allow $t^j[0] = 0, t^j[1] = 0, \ldots, t^j[m-1] = 0$ to propagate to $t^{j+1}[0] = 1$. This may seem strange at first glance, but it does fit the definition of CMP.

Rule 11 (S-boxes) *Let $\boldsymbol{f}^j$ be a function that consists of m S-boxes, denoted by $S_0, \ldots, S_{m-1}$. For each S-box S_i, we construct a table P_i by precomputation to store all the pairs $(\boldsymbol{t}^j, \boldsymbol{t}^{j+1})$ that satisfy*

– *$\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ only relates to the input δ bits of S_i; $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only relates to the output δ bits of S_i.*
– *$\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$.*

Now given a monomial $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ that only relates to the δ bits of round j, we decompose it into the product of m monomials, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) = \prod_{i=0}^{m-1} \pi(\boldsymbol{x}^j, \boldsymbol{t}_i^j)$, such that $\pi(\boldsymbol{x}^j, \boldsymbol{t}_i^j)$ only relates to the input of S_i. Then, $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ can propagate to $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$, i.e., $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ only when we can also decompose $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ into the product of m monomials, i.e., $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1}) = \prod_{i=0}^{m-1} \pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}_i^{j+1})$, such that $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}_i^{j+1})$ only relates to the output of S_i, and the pair $(\boldsymbol{t}_i^j, \boldsymbol{t}_i^{j+1})$ can be found in the table P_i.

B.2 Propagation Models

Model 4 (COPY [22]) *Let $\boldsymbol{f}^j$ be the basic operation COPY ($n_{j+1} = n_j + m - 1$) with $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ and $\boldsymbol{x}^{j+1} = (x^j[0], \ldots, x^j[0], x^j[1], \ldots, x^j[n_j -$*

1]), *where the first* m *bits of* $\boldsymbol{x}^{j+1}$ *are generated from* $x^j[0]$ *by COPY. Let* $a, b_0, b_1, \ldots, b_{m-1}$ *be the MILP variables corresponding to* $x^j[0], x^{j+1}[0], \ldots, x^{j+1}[m-1]$ *such that* $a \xrightarrow{COPY} (b_0, b_1, \ldots, b_{m-1})$ *is a propagation trail of COPY. The following inequalities are sufficient to describe all the valid trails for COPY:*

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

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

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

Model 5 (AND [22]) *Let* $\boldsymbol{f}^j$ *be the basic operation AND* ($n_{j+1} = n_j - m + 1$) *with* $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ *and* $\boldsymbol{x}^{j+1} = (x^j[0] \wedge x^j[1] \wedge \cdots \wedge x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. *Let* $a_0, a_1, \ldots, a_{m-1}, b$ *be the MILP variables corresponding to* $x^j[0], \ldots, x^j[m-1], x^{j+1}[0]$ *such that* $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{AND} b$ *is a propagation trail of AND. The following inequalities are sufficient to describe all the valid trails for AND:*

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

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

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{m-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b = a_0 \vee a_1 \vee \cdots \vee a_{m-1}; \\ \mathcal{M}.con \leftarrow b = a_i \text{ if } x^j[i].F = \delta, \forall\ i \in \{0, 1, \ldots, m-1\}. \end{cases}$$

Model 6 (XOR [22]) *Let* $\boldsymbol{f}^j$ *be the basic operation XOR* ($n_{j+1} = n_j - m + 1$) *with* $\boldsymbol{x}^j = (x^j[0], x^j[1], \ldots, x^j[n_j - 1])$ *and* $\boldsymbol{x}^{j+1} = (x^j[0] \oplus x^j[1] \oplus \cdots \oplus x^j[m-1], x^j[m], \ldots, x^j[n_j - 1])$. *Let* $a_0, a_1, \ldots, a_{m-1}, b$ *be the MILP variables corresponding to* $x^j[0], \ldots, x^j[m-1], x^{j+1}[0]$ *such that* $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{XOR} b$ *is a propagation trail of XOR. The following inequalities are sufficient to describe all the valid trails for XOR:*

$$\begin{cases} \mathcal{M}.var \leftarrow b, a_0, a_1, \ldots, a_{m-1} \text{ as binary}; \\ \mathcal{M}.con \leftarrow b \geq a_0 + \cdots + a_{m-1}, \text{ if } \{1_c, \delta\} \subseteq \{x^j[0].F, x^j[1].F, \ldots, x^j[m-1].F\}; \\ \mathcal{M}.con \leftarrow b = a_0 + \cdots + a_{m-1}, \text{ otherwise.} \end{cases}$$

Algorithm 3: Generate MILP model for $\boldsymbol{f}^j$ that consists of multiple S-boxes

for *each S-box* $S_i, 0 \leq i < m$ **do**
  Prepare an empty table P_i
  for *each possible pair* $(\boldsymbol{t}^j, \boldsymbol{t}^{j+1})$ *that satisfies* $\pi(\boldsymbol{x}^j, \boldsymbol{t}^j)$ *only relates to the input* δ *bits of* S_i, $\pi(\boldsymbol{x}^{j+1}, \boldsymbol{t}^{j+1})$ *only relates to the output* δ *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**
    Add the pair $(\boldsymbol{t}^j, \boldsymbol{t}^{j+1})$ to P_i
  Generate linear inequalities with respect to $a_0, \ldots, a_{n_j-1}, b_0, \ldots, b_{n_{j+1}-1}$ using mathematical tools (e.g., Sagemath) to constrain the pairs in P_i
  Add the linear inequalities to $\mathcal{M}.con$

Model 7 (S-boxes) *Let* $\boldsymbol{f}^j$ *be a function that consists of* m *S-boxes, denoted by* $S_0, \ldots, S_{m-1}$. *Let* $a_0, \ldots, a_{n_j-1}$ *and* $b_0, \ldots, b_{n_{j+1}-1}$ *be the MILP variables corresponding to* $\boldsymbol{x}^j$ *and* $\boldsymbol{x}^{j+1}$, *respectively. We can model the CMP propagation through* $\boldsymbol{f}^j$ *by adding linear constraints following Algorithm 3.*

C Proof of Proposition 4

Proof. In the proof we will consider the propagation of CMP under different values of the flag masks, so in order to avoid confusion, we refer to $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}$ taking the values $\boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{\alpha}^{r_s,1_c}$ as case 1 and $\boldsymbol{M}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c}$ taking the values $\boldsymbol{\beta}^{r_s,\delta}, \boldsymbol{\beta}^{r_s,1_c}$ as case 2, which correspond to instances A and B, respectively. To be accurate, we are going to prove

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})| \text{ in case 1} \geq |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e} \wedge \boldsymbol{\beta}^{r_e,\delta})| \text{ in case 2.}$$

Assuming the values of flag masks at each round $j, r_s < j \leq r_e$ in case 1 are calculated as $\boldsymbol{\alpha}^{j,\delta}, \boldsymbol{\alpha}^{j,1_c}, \boldsymbol{\alpha}^{j,0_c}$ according to the operation rules of flags, we notice that $\boldsymbol{\alpha}^{j,\delta} \succeq \boldsymbol{\beta}^{j,\delta}, \boldsymbol{\alpha}^{j,1_c} \preceq \boldsymbol{\beta}^{j,1_c}, \boldsymbol{\alpha}^{j,0_c} \preceq \boldsymbol{\beta}^{j,0_c}$.

We prove the proposition by fixing r_s and performing induction on r_e. First, we consider $r_e = r_s + 1$. If $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1} \wedge \boldsymbol{\beta}^{r_s+1,\delta})$ in case 2, there must exist a $\boldsymbol{w}_0^{r_s}, \boldsymbol{w}_0^{r_s} \preceq \boldsymbol{\beta}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$ such that $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s} \vee \boldsymbol{w}_0^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1} \wedge \boldsymbol{\beta}^{r_s+1,\delta})$. Moreover, the assumption that $\boldsymbol{f}^{r_s}$ is a basic operation guarantees that there exists a $\boldsymbol{w}_1^{r_s}, \boldsymbol{w}_1^{r_s} \preceq \boldsymbol{\beta}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$ such that $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{w}_1^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1} \wedge \boldsymbol{\beta}^{r_s+1,1_c})$. Since $\boldsymbol{t}^{r_s+1} \wedge \boldsymbol{\beta}^{r_s+1,0_c} = \boldsymbol{0}$, we have $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s} \vee \boldsymbol{w}_0^{r_s} \vee \boldsymbol{w}_1^{r_s}) \rightarrow \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1})$, which indicates that $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1})|$ in case 1. This means the proposition holds for $r_e = r_s + 1$. Assume the proposition holds for $r_e < m$, now we want to prove it also holds for $r_e = m$.

In case 1, solving instance A requires $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m)|$ core monomial trails and

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m)| = \sum_{\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m)} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})|.$$

In case 2, solving instance B requires $|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta})|$ core monomial trails and

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta})| = \sum_{\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}\boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta})} |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}\boldsymbol{t}^{m-1})|.$$

Let

$$S_1 = \{\boldsymbol{t}^{m-1} \mid \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m) \text{ in case 1}\},$$
$$S_2 = \{\boldsymbol{t}\boldsymbol{t}^{m-1} \mid \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}\boldsymbol{t}^{m-1}) \xrightarrow{\mathcal{C}} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta}) \text{ in case 2}\}.$$

Due to the assumption that $\boldsymbol{f}^{m-1}$ is a basic operation, there exists a vector $\boldsymbol{w}_0^{m-1}, \boldsymbol{w}_0^{m-1} \preceq \boldsymbol{\beta}^{m-1,1_c}$ such that $\pi(\boldsymbol{x}^{m-1}, \boldsymbol{w}_0^{m-1}) \to \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,1_c})$. We next define a mapping φ from S_2 to S_1. Given $\boldsymbol{t}\boldsymbol{t}^{m-1}$ as a vector in S_2, we can define its image $\boldsymbol{t}^m$ as follows.

- Find a vector $\boldsymbol{w}_1^{m-1}, \boldsymbol{w}_1^{m-1} \preceq \boldsymbol{\beta}^{m-1,1_c}$ such that $\pi(\boldsymbol{x}^{m-1}, \boldsymbol{w}_1^{m-1} \vee \boldsymbol{t}\boldsymbol{t}^{m-1}) \to \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta})$.
- Since $\boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,0_c} = \boldsymbol{0}$, we have $\pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}\boldsymbol{t}^{m-1} \vee \boldsymbol{w}_0^{m-1} \vee \boldsymbol{w}_1^{m-1}) \to \pi(\boldsymbol{x}^m, \boldsymbol{t}^m)$. Hence, we define $\boldsymbol{t}^{m-1} = \varphi(\boldsymbol{t}\boldsymbol{t}^{m-1}) = (\boldsymbol{t}\boldsymbol{t}^{m-1} \vee \boldsymbol{w}_0^{m-1} \vee \boldsymbol{w}_1^{m-1}) \wedge \boldsymbol{\alpha}^{m-1,\delta}$.

The mapping is one-to-one, therefore $|S_1| \geq |S_2|$. Furthermore, notice that for each $\boldsymbol{t}^{m-1} \in S_1$ as the image of a $\boldsymbol{t}\boldsymbol{t}^{m-1} \in S_2$, $\boldsymbol{t}^{m-1} \wedge \boldsymbol{\beta}^{m-1,0_c} = \boldsymbol{0}$. According to the induction hypothesis,

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}^{m-1})| \text{ in case 1} \geq |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{m-1}, \boldsymbol{t}\boldsymbol{t}^{m-1})| \text{ in case 2.}$$

Combining $|S_1| \geq |S_2|$, we conclude that

$$|\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m)| \text{ in case 1} \geq |\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^m, \boldsymbol{t}^m \wedge \boldsymbol{\beta}^{m,\delta})| \text{ in case 2,}$$

thus proving the proposition. □

D Example for the Equivalence Between MP and CMP

Example 2. Let $\boldsymbol{x}^0 = (x^0[0], \ldots, x^0[3], x^0[4])$ and $\boldsymbol{x}^0.F = \{\delta, \delta, 1_c, 1_c, 0_c\}$. Let $\boldsymbol{x}^1 = (x^1[0], \ldots, x^1[3]) = \boldsymbol{f}^0(\boldsymbol{x}^0)$ with $x^1[0] = x^0[0]\underline{x^0[2]} + \underline{x^0[2]} + \underline{x^0[3]} + x^0[4]$, $x^1[1] = x^0[1]\underline{x^0[2]} + x^0[0]x^0[1]\underline{x^0[3]} + x^0[0]x^0[1]x^0[4]$, $x^1[2] = x^0[2], x^1[3] = x^0[3]$. Let $\boldsymbol{x}^2 = (x^2[0], x^2[1], x^2[2], x^2[3]) = \boldsymbol{f}^1(\boldsymbol{x}^1)$ with $x^2[0] = x^1[0] + \underline{x^1[2]}, x^2[1] =$

$x^1[1], x^2[2] = x^1[2], x^2[3] = x^1[3]$. Given $\pi(\boldsymbol{x}^0, \boldsymbol{t}^0) = x^0[0]x^0[1]$ and $\pi(\boldsymbol{x}^2, \boldsymbol{t}^2) = x^2[0]x^2[1]$, we want to calculate $\mathsf{Sol}\langle \boldsymbol{t}^0, \boldsymbol{t}^2 \rangle$. According to operation rules of flags, we can calculate the flags of $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, i.e.,

$$\boldsymbol{x}^1.F = \{\delta, \delta, 1_c, 1_c\}, \boldsymbol{x}^2.F = \{\delta, \delta, 1_c, 1_c\}.$$

After setting the 0_c bits of $\boldsymbol{x}^0$ (i.e., $x^0[4]$) to $\mathbf{0}$, the monomial $x^0[4]$ in $x^1[0]$ and the monomial $x^0[0]x^0[1]x^0[4]$ in $x^1[1]$ are removed. We can compute that there are two core monomial trails

$$x^0[0]x^0[1] \xrightarrow{\mathcal{C}} x^1[1] \xrightarrow{\mathcal{C}} x^2[0]x^2[1],$$
$$x^0[0]x^0[1] \xrightarrow{\mathcal{C}} x^1[0]x^1[1] \xrightarrow{\mathcal{C}} x^2[0]x^2[1].$$

Let $\boldsymbol{t}_0^1 = (0, 1, 0, 0)$ and $\boldsymbol{t}_1^1 = (1, 1, 0, 0)$ such that $\pi(\boldsymbol{x}^1, \boldsymbol{t}_0^1) = x^1[1], \pi(\boldsymbol{x}^1, \boldsymbol{t}_1^1) = x^1[0]x^1[1]$. We can calculate that

$$\mathsf{Sol}\langle \boldsymbol{t}^2, \boldsymbol{t}_0^1 \rangle = x^1[2], \mathsf{Sol}\langle \boldsymbol{t}_0^1, \boldsymbol{t}^0 \rangle = x^0[3],$$
$$\mathsf{Sol}\langle \boldsymbol{t}^2, \boldsymbol{t}_1^1 \rangle = 1, \mathsf{Sol}\langle \boldsymbol{t}_1^1, \boldsymbol{t}^0 \rangle = x^0[2] + x^0[3].$$

Hence, according to Proposition 2, we can calculate $\mathsf{Sol}\langle \boldsymbol{t}^0, \boldsymbol{t}^2 \rangle$ as

$$\mathsf{Sol}\langle \boldsymbol{t}^0, \boldsymbol{t}^2 \rangle = \mathsf{Expr}\left\langle x^0[3]x^1[2] + x^0[2] + x^0[3], \boldsymbol{x}^0 \right\rangle = x^0[3]x^0[2] + x^0[2] + x^0[3].$$

On the other hand, $\boldsymbol{f}^0$ is an S-box that maps 5 bits to 4 bits and $\boldsymbol{f}^1$ is the basic operation XOR, so we perform iterative variable substitutions by applying Rule 3 for $\boldsymbol{f}^1$ and Rule 4 for $\boldsymbol{f}^0$. For $\boldsymbol{f}^0$, we compute

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

Along with the expressions of $\boldsymbol{x}^1[0]$ and $\boldsymbol{x}^1[1]$, we generate $\boldsymbol{c}^0 = (c^0[0], \ldots, c^0[5])$ by substituting $c^0[0], \ldots, c^0[5]$ for $x^0[2], x^0[2]+x^0[3], x^0[2], x^0[3], x^0[2]+x^0[2]x^0[3]$, $x^0[2] + x^0[3]$ (underlined parts), respectively. While for $\boldsymbol{f}^1$, we generate $\boldsymbol{c}^1 = (c^1[0])$ by only substituting $c^1[0]$ for $x^1[2]$. Corresponding to the two core monomial trails, we can compute there are two monomial trails

$$c^0[3]c^1[0]x^0[0]x^0[1] \to c^1[0]x^1[1] \to x^2[0]x^2[1],$$
$$c^0[5]x^0[0]x^0[1] \to x^1[0]x^1[1] \to x^2[0]x^2[1].$$

By substituting $c^0[3], c^0[5], c^1[0]$ back with $x^0[3], x^0[2]+x^0[3], x^1[2]$, respectively, we obtain $x^0[3]x^1[2]$ from $c^0[3]c^1[0]$ and $x^0[2]+x^0[3]$ from $c^0[5]$, which are exactly the contributions of the two core monomial trails. This confirms Proposition 3.

E Expansion with Callback

What is callback? The callback function is a user interface provided by the Gurobi solver that allows the user to define a function that allows Gurobi to

interrupt the solving process at a certain point in the internal solving process and run the callback function instead, and then continue the solving process when the callback function finishes. In this paper, we configure Gurobi to jump to the callback function every time it finds a solution. In the callback function, we can add constraints called *lazy constraints* to the model. For more details, please refer to [3].

What can we do with callback? The basic components of an MILP model are MILP variables and MILP constraints. If there exist certain values that can be assigned to the MILP variables to satisfy the MILP constraints, we call these values a solution to the model. However, in some cases we may only be concerned with what values some of the MILP variables can take. For example, assume an MILP model is composed of 10 MILP variables, denoted by $s_0, \ldots, s_9$, but we only focus on the values $s_0, \ldots, s_4$ can take. In this case, we can use the callback interface to have Gurobi jump to the callback function every time a solution is found. In the callback function, we extract the values of $s_0, \ldots, s_4$ and add lazy constraints to exclude these values from the solution space. Then, Gurobi will jump back to the solving process and continue solving the model. The above process will be repeated until all possible values that $s_0, \ldots, s_4$ can take are found.

Expansion with callback. Considering that Step 4a and Step 5a are very similar, here we only discuss how to determine those $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'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 $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$. The corresponding MILP model is illustrated in Algorithm 4.

Benefits and drawbacks of using callback functions. We take Step 4a as an example. If we only want to compute all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$, then the computational burden will be negligible, since we only need to start from $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s})$ and derive forward following the propagation rules, which can even be done manually. The cost is that at Step 8 we have to deal with a large number of instances whose final solutions are determined to be 0. In contrast, if we directly compute all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$ and $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ at Step 4a, we save a lot of time that would otherwise be spent in Step 8, but at the cost that we are required to solve an MILP model as shown in Algorithm 4. If $r_e - r_s$ is large, such an MILP model can be very complex and difficult to solve. Hence, in practice, when $r_e - r_s$ is large, we use a non-callback mode at Step 4a, namely we only compute all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$; when $r_e - r_s$ is reduced to a relatively small number, we adjust Step 4a to adopt a callback mode, namely we compute all $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$'s that both satisfy $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$ and $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$.

Algorithm 4: Determine 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 $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$

Procedure `CallbackFunc`$(\mathcal{M}, s_0^{r_s+r_0}, \ldots, s_{n_{r_s+r_0}-1}^{r_s+r_0})$**:**
  Extract the values of $s_0^{r_s+r_0}, \ldots, s_{n_{r_s+r_0}-1}^{r_s+r_0}$ as $\boldsymbol{t}^{r_s+r_0}$
  Save $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0})$ as a monomial that satisfies 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 $\pi(\boldsymbol{x}^{r_s+r_0}, \boldsymbol{t}^{r_s+r_0}) \overset{\mathcal{C}}{\rightsquigarrow} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$
  Initialize a linear expression $L = 0$
  for $i = 0$ *to* $n_{r_s+r_0} - 1$ **do**
    if $t^{r_s+r_0}[i] = 1$ **then** $L = L + 1 - s_i^{r_s+r_0}$
    else $L = L + s_i^{r_s+r_0}$
  $\mathcal{M}.con \leftarrow L \geq 1$ `// Exclude` $\boldsymbol{t}^{r_s+r_0}$ `from the solution space`

Procedure `CallbackCMPModel`$(\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s} \rangle)$**:**
  Starting from $\boldsymbol{M}^{r_s,\delta} = \boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$, calculate the values of flag masks in each round according to the operation rules of flags
  Prepare an empty MILP model $\mathcal{M}$
  $\mathcal{M}.var \leftarrow s_i^{r_s}$ for $i \in \{0, 1, \ldots, n_{r_s} - 1\}$ as binary
  for $i = 0$ *to* $n_{r_s} - 1$ **do**
    $\mathcal{M}.con \leftarrow s_i^{r_s} = 1$ if $t^{r_s}[i] = 1$
    $\mathcal{M}.con \leftarrow s_i^{r_s} = 0$ if $t^{r_s}[i] = 0$
  for $j = r_s$ *to* $r_s + r_0 - 1$ **do**
    Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the propagation models of CMP
  Set `CallbackFunc` as a callback function, so that Gurobi will jump to it every time a solution is found
  for $j = r_s + r_0$ *to* $r_e - 1$ **do**
    Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the propagation models of CMP
  for $i = 0$ *to* $n_{r_e} - 1$ **do**
    $\mathcal{M}.con \leftarrow s_i^{r_e} = 1$ if $t^{r_e}[i] = 1$
    $\mathcal{M}.con \leftarrow s_i^{r_e} = 0$ if $t^{r_e}[i] = 0$
  return $\mathcal{M}$

F Optimizing the Implementation of Proposition 2

F.1 Partial XOR and its Complement

Rule 12 (Partial XOR [22]) *For the propagation rule of the basic operation XOR described in Rule 10, we consider a partial adoption as follows:*

- $t^{j+1}[k-m+1] = t^j[k] \ \forall \ m \leq k \leq n_j - 1$.
- $t^{j+1}[0] = t^j[0] + t^j[1] + \cdots + t^j[m-1]$.

That is, we remove the propagation from $t^j[0] = 0, t^j[1] = 0, \ldots, t^j[m-1] = 0$ *to* $t^{j+1}[0] = 1$.

Model 8 (Partial XOR [22]) *For the propagation model of the basic operation XOR described in Model 6, we consider a partial adoption as follows:*

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

Rule 13 (Complement of Rule 12 in Rule 10) *For the propagation rule of the basic operation XOR described in Rule 10, we consider a partial adoption as follows:*

- $t^{j+1}[k-m+1] = t^j[k] \ \forall \ m \leq k \leq n_j - 1$.
- $t^{j+1}[0] > t^j[0] + t^j[1] + \cdots + t^j[m-1]$.

Model 9 (Complement of Model 8 in Model 6) *On the basis of the propagation model listed in Model 6 for* $\boldsymbol{f}^j$ *as the basic operation XOR, we add an additional linear constraint as follows: we construct a linear expression* $L^j = b - \sum_{i=0}^{m-1} a_i$ *and add the constraint* $L^j = 1$ *to* $\mathcal{M}.con$.

F.2 Direct Implementation of Proposition 2

Algorithm 5: Construct an MILP model to extract the core monomial trails in $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ for the instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$

Procedure `ExtractCMPTrails`($\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$)**:**
- Starting from $\boldsymbol{M}^{r_s,\delta} = \boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$, calculate the values of flag masks in each round according to the operation rules of flags
- Prepare an empty MILP model $\mathcal{M}$
- $\mathcal{M}.var \leftarrow s_i^{r_s}$ for $i \in \{0, 1, \ldots, n_{r_s} - 1\}$ as binary
- **for** $i = 0$ *to* $n_{r_s} - 1$ **do**
  - $\mathcal{M}.con \leftarrow s_i^{r_s} = 1$ if $t^{r_s}[i] = 1$
  - $\mathcal{M}.con \leftarrow s_i^{r_s} = 0$ if $t^{r_s}[i] = 0$
- **for** $j = r_s$ *to* $r_e - 1$ **do**
  - Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the propagation models of CMP
- **for** $i = 0$ *to* $n_{r_e} - 1$ **do**
  - $\mathcal{M}.con \leftarrow s_i^{r_e} = 1$ if $t^{r_e}[i] = 1$
  - $\mathcal{M}.con \leftarrow s_i^{r_e} = 0$ if $t^{r_e}[i] = 0$
- Configure the MILP solver to find all the solutions of $\mathcal{M}$ and start to solve $\mathcal{M}$
- **if** $\mathcal{M}$ *is solved (i.e., all the solutions have been found)* **then**
  - Prepare an empty set S to store core monomial trails
  - **for** *Each solution of* $\mathcal{M}$ **do**
    - **for** $j = r_s$ *to* r_e **do**
      - Extract the values of $s_0^j, \ldots, s_{n_j-1}^j$ as $\boldsymbol{t}^j$
    - We obtain a core monomial trail written as $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\to} \pi(\boldsymbol{x}^{r_s+1}, \boldsymbol{t}^{r_s+1}) \overset{\mathcal{C}}{\to} \cdots \overset{\mathcal{C}}{\to} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ and add this trail to S
  - **return** S

F.3 Two-step Implementation of Proposition 1

Algorithm 6: Construct an MILP model $\mathcal{M}_0$ to extract a subset of $\pi(\boldsymbol{x}^{r_s}, \boldsymbol{t}^{r_s}) \overset{\mathcal{C}}{\bowtie} \pi(\boldsymbol{x}^{r_e}, \boldsymbol{t}^{r_e})$ for the instance $\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$ in the first step

Procedure `FirstStepCMPModel`($\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$)**:**

- Starting from $\boldsymbol{M}^{r_s,\delta} = \boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$, calculate the values of flag masks in each round according to the operation rules of flags
- Prepare an empty MILP model $\mathcal{M}_0$
- $\mathcal{M}_0.var \leftarrow s_i^{r_s}$ for $i \in \{0, 1, \ldots, n_{r_s} - 1\}$ as binary
- **for** $i = 0$ *to* $n_{r_s} - 1$ **do**
  - $\mathcal{M}_0.con \leftarrow s_i^{r_s} = 1$ if $t^{r_s}[i] = 1$
  - $\mathcal{M}_0.con \leftarrow s_i^{r_s} = 0$ if $t^{r_s}[i] = 0$
- **for** $j = r_s$ *to* $r_e - 1$ **do**
  - **if** $\boldsymbol{f}^j$ *is the basic operation XOR* **then**
    - Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the partial XOR model (Model 8)
  - **else**
    - Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the propagation models of CMP
- **for** $i = 0$ *to* $n_{r_e} - 1$ **do**
  - $\mathcal{M}_0.con \leftarrow s_i^{r_e} = 1$ if $t^{r_e}[i] = 1$
  - $\mathcal{M}_0.con \leftarrow s_i^{r_e} = 0$ if $t^{r_e}[i] = 0$
- **return** $\mathcal{M}_0$

Algorithm 7: Construct an MILP model $\mathcal{M}_1$ to extract the missing trails due to the use of partial XOR

Procedure `SecondStepCMPModel(`$\mathsf{SR}_{\boldsymbol{\alpha}^{r_s,\delta},\boldsymbol{\alpha}^{r_s,1_c}}\langle \boldsymbol{t}^{r_e}, \boldsymbol{t}^{r_s}\rangle$`):`

- Starting from $\boldsymbol{M}^{r_s,\delta} = \boldsymbol{\alpha}^{r_s,\delta}, \boldsymbol{M}^{r_s,1_c} = \boldsymbol{\alpha}^{r_s,1_c}$, calculate the values of flag masks in each round according to the operation rules of flags
- Prepare an empty MILP model $\mathcal{M}_1$
- $\mathcal{M}_1.var \leftarrow s_i^{r_s}$ for $i \in \{0, 1, \ldots, n_{r_s} - 1\}$ as binary
- **for** $i = 0$ *to* $n_{r_s} - 1$ **do**
  - $\mathcal{M}_1.con \leftarrow s_i^{r_s} = 1$ if $t^{r_s}[i] = 1$
  - $\mathcal{M}_1.con \leftarrow s_i^{r_s} = 0$ if $t^{r_s}[i] = 0$
- Initialize a linear expression $L = 0$
- **for** $j = r_s$ *to* $r_e - 1$ **do**
  - **if** $\boldsymbol{f}^j$ *is the basic operation XOR* **then**
    - Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the full XOR model (Model 6)
    - Construct a linear expression L^j in the same way as done in Model 9 for $\boldsymbol{f}^j$, but do not add the constraint $L^j = 1$.
    - $L = L + L^j$
  - **else**
    - Update $s_0^j, \ldots, s_{n_j-1}^j$ to $s_0^{j+1}, \ldots, s_{n_{j+1}-1}^{j+1}$ according to the propagation models of CMP
- **for** $i = 0$ *to* $n_{r_e} - 1$ **do**
  - $\mathcal{M}_1.con \leftarrow s_i^{r_e} = 1$ if $t^{r_e}[i] = 1$
  - $\mathcal{M}_1.con \leftarrow s_i^{r_e} = 0$ if $t^{r_e}[i] = 0$
- $\mathcal{M}_1.con \leftarrow L \geq 1$
- **return** $\mathcal{M}_1$

G Applications to Ciphers

G.1 Trivium

Algorithm 8: Choose the time limit $\tau^{r_e - r_s}$ for Trivium

if $r_e - r_s > 600$ **then** $\tau^{r_e - r_s} = 40$
else if $r_e - r_s > 500$ **then** $\tau^{r_e - r_s} = 80$
else if $r_e - r_s > 400$ **then** $\tau^{r_e - r_s} = 160$
else if $r_e - r_s > 300$ **then** $\tau^{r_e - r_s} = 320$
else if $r_e - r_s > 250$ **then** $\tau^{r_e - r_s} = 640$
else if $r_e - r_s > 100$ **then** $\tau^{r_e - r_s} = 1200$
else if $r_e - r_s > 20$ **then** $\tau^{r_e - r_s} = 3600$
else $\tau^{r_e - r_s} = \infty$

Table 5: Cube indices of Trivium used for verification

I	Indices	Size
I_0	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 75, 77, 79	55
I_1	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 75, 77, 79	54
I_2	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 30, 32, 34, 36, 38, 40, 42, 45, 47, 49, 51, 53, 55, 57, 60, 62, 64, 66, 68, 70, 72, 77, 75, 79	52

Table 6: Verification and comparison of previous results for TRIVIUM

I	Rounds	Status	Time Cost*([22])	Time Cost (ours)
I_0	845	Verified(✓)	20 hours	9 hours
I_1	845	Verified(✓)	9 hours	4 hours
I_2	848	Verified(✓)	11 days	5 days

* The time cost is evaluated by re-running the code provided by [22] on our platform.

G.2 Grain-128AEAD

Grain-128AEAD [23] is an authenticated encryption algorithm with support for associated data and has been selected as one of the ten finalist candidates of the NIST LWC standardization process. The design of Grain-128AEAD is closely based on Grain-128a [5] introduced in 2011. Once the pre-output generator has been initialized, a 64-bit shift register and a 64-bit accumulator are initialized with the pre-output keystream to generate the authentication tag later. We follow the assumption made in [21] that the first pre-output bit could be observed.

The internal state of the pre-output generator in Grain-128AEAD can be represented by two 128-bit registers, namely $\boldsymbol{b} = (b[0], b[1], \ldots, b[127])$ and $\boldsymbol{s} = (s[0], s[1], \ldots, s[127])$. To initialize the pre-output generator, the 128-bit key $\boldsymbol{K}$ is loaded into the first register and the 96-bit Nonce $\boldsymbol{N}$ is loaded into the second register. The other state bits of the second register are set to 1 except that the least one bit. Namely, the internal state bits are represented as

$$\begin{aligned}(b[0], b[1], \ldots, b[127]) &\leftarrow (K[0], K[1], \ldots, K[127])\\ (s[0], s[1], \ldots, s[127]) &\leftarrow (N[0], N[1], \ldots, N[95], 1, \ldots, 1, 0).\end{aligned}$$

During the initialization of the pre-output generator, the update function is given by the following pseudo-code:

$$\begin{aligned}
g &\leftarrow b[0] \oplus b[26] \oplus b[56] \oplus b[91] \oplus b[96] \oplus b[3]b[67] \oplus b[11]b[13] \oplus b[17]b[18]\\
&\quad \oplus b[27]b[59] \oplus b[40]b[48] \oplus b[61]b[65] \oplus b[68]b[84] \oplus b[88]b[92]b[93]b[95]\\
&\quad \oplus b[22]b[24]b[25] \oplus b[70]b[78]b[82],\\
f &\leftarrow s[0] \oplus s[7] \oplus s[38] \oplus s[70] \oplus s[81] \oplus s[96],\\
h &\leftarrow b[12]s[8] \oplus s[13]s[20] \oplus b[95]s[42] \oplus s[60]s[79] \oplus b[12]b[95]s[94],\\
z &\leftarrow h \oplus s[93] \oplus b[2] \oplus b[15] \oplus b[36] \oplus b[45] \oplus b[64] \oplus b[73] \oplus b[89],\\
(b[0], b[1], \ldots, b[127]) &\leftarrow (b[1], \ldots, b[127], g \oplus s[0] \oplus z),\\
(s[0], s[1], \ldots, s[127]) &\leftarrow (s[1], \ldots, s[127], f \oplus z).
\end{aligned}$$

After updating the state 256 times without producing an output in the initialization, z is used as a pre-output key stream instead of being fed back to the state. Hereinafter, the assume that the first bit of the pre-output key stream can be observed.

Algorithm 9: Choose the time limit $\tau^{r_e - r_s}$ for Grain-128AEAD

if $r_e - r_s > 120$ **then** $\tau^{r_e - r_s} = 60$
else if $r_e - r_s > 110$ **then** $\tau^{r_e - r_s} = 120$
else if $r_e - r_s > 100$ **then** $\tau^{r_e - r_s} = 180$
else if $r_e - r_s > 33$ **then** $\tau^{r_e - r_s} = 360$
else if $r_e - r_s > 10$ **then** $\tau^{r_e - r_s} = 360$
else $\tau^{r_e - r_s} = \infty$

G.3 Kreyvium

Kreyvium is a stream cipher that is designed for the use of fully Homomorphic encryption [10]. As a variant of Trivium, Kreyvium offers a higher security level with 128-bit key and IV, but without increasing the multiplicative depth of the corresponding circuit. The internal state of Kreyvium is composed of 5 registers. Three of them are similar to those in Trivium, while the other two registers correspond to the secret key and the IV, denoted by $\boldsymbol{K^*} = (K^*[0], \ldots, K^*[127])$ and $\boldsymbol{IV^*} = (IV^*[0], \ldots, IV^*[127])$, respectively. The initial state of Kreyvium is set as

$$
\begin{aligned}
&(s[0], s[1], \ldots, s[92]) \leftarrow (K[0], K[1], \ldots, K[92]),\\
&(s[93], s[94], \ldots, s[176]) \leftarrow (IV[0], IV[1], \ldots, IV[83]),\\
&(s[177], s[178], \ldots, s[287]) \leftarrow (IV[84], \ldots, IV[127], 1, \ldots, 1, 0),\\
&(IV^*[127], \ldots, IV^*[0]) \leftarrow (IV[0], \ldots, IV[127]),\\
&(K^*[127], \ldots, K^*[0]) \leftarrow (K[0], \ldots, K[127]).
\end{aligned}
$$

The update function is given by the following pseudo-code:

$$
\begin{aligned}
&t_1 \leftarrow s[65] \oplus s[92], t_2 \leftarrow s[161] \oplus s[176], t_3 = s[242] \oplus s[287] \oplus K^*[0],\\
&z \leftarrow t_1 \oplus t_2 \oplus t_3,\\
&t_1 \leftarrow t_1 \oplus s[90]s[91] \oplus s[170] \oplus IV^*[0],\\
&t_2 \leftarrow t_2 \oplus s[174]s[175] \oplus s[263],\\
&t_3 \leftarrow t_3 \oplus s[285]s[286] \oplus s[68],\\
&t_4 \leftarrow K^*[0], t_5 \leftarrow IV^*[0],\\
&(s[0], s[1], \ldots, s[92]) \leftarrow (t_3, s[0], s[1], \ldots, s[91]),\\
&(s[93], s[94], \ldots, s[176]) \leftarrow (t_1, s[93], s[94], \ldots, s[175]),\\
&(s[177], s[178], \ldots, s[287]) \leftarrow (t_2, s[177], s[178], \ldots, s[286]),\\
&(K^*[127], K^*[126], \ldots, K^*[0]) \leftarrow (t_4, K^*[127], \ldots, K^*[1]),\\
&(IV^*[127], IV^*[126], \ldots, IV^*[0]) \leftarrow (t_5, IV^*[127], \ldots, IV^*[1]).
\end{aligned}
$$

After 1152 rounds of initialization, the key stream bit z is produced by every update function.

Limitations of CMP on Kreyvium. Compared to Trivium, Kreyvium increases the size of the IV from 80 to 128, plus we choose cube indices with more

than 120 dimensions, so many of the 288 bits of the initial state $\boldsymbol{s}$ are flagged as δ bits if we set the flags of the initial state according to Eqn. (3). Under the influence of the update function of Kreyvium, it will not take too many rounds for all of these 288 bits to become δ bits. For example, when we choose $I_3 = \{0, 1, \ldots, 127\} \backslash \{66, 73, 85, 87\}$ for Kreyvium, it only takes 158 rounds for the 288-bit state of Kreyvium to become all δ bits, but if we choose I_3 in Table 2 for Trivium, it would take 228 rounds for all 288 bits to become δ bits.

The advantage of CMP is that only δ bits are tracked, so the fewer the δ bits in the round state, the more effective CMP is. Therefore, it can be expected that CMP will be more effective on Trivium. In other words, the reduction in superpoly recovery time by CMP is not so significant on Kreyvium as it is on Trivium.

On the other hand, since Kreyvium has a more complex update function than Trivium, the resulting MILP model is more difficult to solve. As a result, most of the time spent in our MITM framework is devoted to reducing r_e to a sufficiently small value through backward expansion, so that the MILP model is easier to solve.

Algorithm 10: Choose the time limit $\tau^{r_e - r_s}$ for Kreyvium

if $r_e - r_s > 600$ **then** $\tau^{r_e - r_s} = 40$
else if $r_e - r_s > 500$ **then** $\tau^{r_e - r_s} = 320$
else if $r_e - r_s > 400$ **then** $\tau^{r_e - r_s} = 320$
else if $r_e - r_s > 300$ **then** $\tau^{r_e - r_s} = 320$
else if $r_e - r_s > 250$ **then** $\tau^{r_e - r_s} = 640$
else if $r_e - r_s > 100$ **then** $\tau^{r_e - r_s} = 1200$
else if $r_e - r_s > 20$ **then** $\tau^{r_e - r_s} = 3600$
else $\tau^{r_e - r_s} = \infty$

H Möbius Transform

Standard Möbius transform. Given the ANF of a Boolean function $F(\boldsymbol{x})$ of n variables represented by a binary vector of 2^n entries corresponding to its 2^n coefficients, the truth table of F (also represented by a binary vector of 2^n entries) can be computed from the ANF of F via the Möbius transform over $\mathbb{F}_2^n$. The standard implementation of the Möbius transform is based on the formula

$$F(x[0], \ldots, x[n-1]) = x[0] \cdot F_0(x[1], \ldots, x[n-1]) + F_1(x[1], \ldots, x[n-1]).$$

With the knowledge of the truth tables of F_0 and F_1 (the ANFs of F_0 and F_1 can be obtained from the ANF of F directly), one can compute the truth table of F with 2^{n-1} bit operations. Denoting the time complexity of the Möbius transform on a Boolean function of n variables by $T(n)$, we have $T(n) = 2T(n-1) + 2^{n-1}$,

and hence $T(n) \leq n \cdot 2^{n-1} < n \cdot 2^n$. In this paper, we assume the standard Möbius transform requires $n \cdot 2^n$ bit operations and 2^n-bits memory. For more details, please refer to [27].