

Cube Attacks with Elimination Strategy: Key Recovery Attacks on Trivium

Haoran Li^{1,2}, Shichang Wang³, Meicheng Liu^{1,2}, Willi Meier⁴ and Dongdai Lin^{1,2}(✉)

¹ State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, People's Republic of China

{lihaoran, liumeicheng, ddlin}@iie.ac.cn

² School of Cyber Security, University of Chinese Academy of Sciences, Beijing, People's Republic
of China

³ School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

shichang.wang@ntu.edu.sg

⁴ University of Applied Sciences and Arts Northwestern Switzerland, Switzerland

willimeier48@gmail.com

Abstract. In this paper, we present a novel framework for cube attacks named *cube attacks with elimination strategy*. The core idea is to find specific key conditions and cubes such that their superpolies under these key conditions can be efficiently computed. By recovering these *conditional superpolies*, we can solve the corresponding equation system and thereby retrieve key information. If a sufficient number of such key conditions can be found, the attack can be extended to a larger key space. To apply this framework in practical attacks, we propose the following techniques. First, we propose a *nested coefficient solver* that combines variable substitution and symbolic computation to efficiently recover superpolies, and present the *conditional monomial prediction* technique to rapidly recover conditional superpolies. Second, by combining numeric mapping with monomial prediction techniques, we propose an automated cube search algorithm that is capable of generating a large number of good cubes for attacks. Finally, we develop two kinds of testing methods, which are used to efficiently extract substantial key information from large-scale equation systems.

To illustrate the power of our techniques, we apply them to Trivium. As a result, for 840 rounds of Trivium, a practical key recovery attack is mounted with complexity below 2^{55} and a success rate 77.8%. For 845 rounds, we present a practical key recovery attack with complexity below 2^{56} and a success rate 98.1% for $2^{80} \times 59.1\%$ keys. For 855 rounds, we present a theoretical weak-key recovery attack for 2^{74} keys. To the best of our knowledge, these are the best practical and theoretical attacks on Trivium. The improvements in the number of rounds reached are 13 for practical attacks and 4 for theoretical attacks, respectively.

Keywords: Cube Attack · Superpoly · Monomial Prediction · Numeric Mapping · Cube Search · Trivium

1 Introduction

Cube attack was proposed at Eurocrypt 2009 by Dinur and Shamir [DS09], which is a powerful cryptanalytic technique against symmetric ciphers. This attack method abstracts the target cipher as a multivariate black-box polynomial, where the inputs are public variables (such as initialization vectors or plain text) and secret variables (the key), and

the output is the keystream or cipher text. The core idea of the cube attack lies in carefully selecting a subset of public variables (called the *cube*) and computing the output sum (i.e., the cube sum) of the cipher over this cube. This process extracts equations involving the key variables with significantly reduced degrees. Over the past decade, cube attacks have developed rapidly, and this method and its variants have become critical analytical tools for evaluating the resistance of ciphers, especially lightweight ciphers (such as Trivium [DCP08], Grain [HJM⁺21], Acorn [Wu16], Ascon [DEMS21]), against algebraic cryptanalysis.

Trivium [DCP08] is a bit-oriented synchronous stream cipher based on NFSRs, which is one of the eSTREAM hardware-oriented finalists and an International Standard under ISO/IEC 29192-3:2012. Because of its simple structure and high level of security, Trivium has attracted extensive attention. Cube attack is the most effective method for attacking the initialization mechanism of the Trivium cipher, and in recent years, research advancements in cube attacks have always been applied to Trivium.

Recovering superpolies is a crucial step in cube attack. In traditional cube attacks, the attacker treated the superpoly as a black box, using methods such as linear or quadratic testing to recover some simple superpolies. At Crypto 2017, Todo *et al.* [TIHM17] first applied Conventional Bit-based Division Property (CBDP) to cube attacks. By constructing an MILP model, they were able to filter some key variables that do not appear in the superpolies, thereby reducing the complexity of recovering the superpolies. At Crypto 2018, Wang *et al.* [WHT⁺18] proposed the flag technique and improved the division-property-based cube attack. At Asiacrypt 2019, Wang *et al.* [WHG⁺19] proposed the pruning techniques and fast propagation of Bit-based Division Property using Three subset (BDPT) to recover the exact superpoly for the first time. However, this method is not always effective. At Eurocrypt 2020, Hao *et al.* [HLM⁺20] proposed the three-subset division property without unknown subsets (3SDPwoU) and recovered the exact superpolies by counting the number of three-subset trails. At Asiacrypt 2020, Hu *et al.* [HSWW20] proposed monomial prediction (MP) technique and established the equivalence with 3SDPwoU from a pure algebraic perspective. At Asiacrypt 2021, Hu *et al.* [HST⁺21] proposed a new framework named nested monomial prediction, which can recover massive superpolies. At Asiacrypt 2022, He *et al.* [HHPW22] proposed Non-zero Bit-based Division Property (NBDP) and Core Monomial Prediction (CMP) techniques, both of which result in simpler MILP models. Combining the divide-and-conquer strategy with these two new techniques, they can recover superpolies more effectively. At Asiacrypt 2023, Wang *et al.* [WQW23] proposed the variable substitution technique to reduce computational complexity of the superpoly recovery. At Eurocrypt 2024, He *et al.* [HHLW24] proposed a purely CMP-based approach and meet-in-the-middle framework to significantly improve the efficiency of superpoly recovery.

Besides, searching for proper cubes is also important in cube attacks, especially in practical attacks, since a good cube can significantly reduce the complexity. In [SMB17], Sarkar, Maitra and Baksi heuristically obtain several cubes for Trivium with more than 800 rounds (maximum 829). At Eurocrypt 2018, Liu *et al.* [LYWL18] found many cubes with high-bias superpolies based on the numeric mapping technique, which allowed them to theoretically attack 835-round Trivium without recovering the superpolies. At Asiacrypt 2021, Ye *et al.* [YT21] proposed a steep-sharp greedy cube searching algorithm, successfully recovering numerous linear superpolies, and mounted a practical key-recovery attack on 805-round Trivium. At FSE 2022, Sun *et al.* [Sun21] introduced the concept of valuable cube, defined as a cube whose superpoly contains at least one balanced secret variable. They proposed a cube searching algorithm based on division property and mount a practical attack on 808-round Trivium. Subsequently, [CT22] and [LHHW23] improved the above cube searching techniques, achieving attacks on 820 and 825-round Trivium, respectively. At Asiacrypt 2023, Wang *et al.* [WQW23] introduced the vector numeric

mapping technique, finding numerous cubes with simpler ANFs, which allowed them to practically attack 830-round Trivium. Recently, Zhang *et al.* [ZS24] executed a practical attack on 832-round Trivium by employing the cube search algorithm proposed in [YT21] along with a guess-and-determine based equation-solving approach.

Contributions. In this paper, we introduce a novel framework for cube attacks, named *cube attacks with elimination strategy*. The general idea is to identify specific key conditions such that complex superpolies become easier to recover. By recovering multiple superpolies under the same key condition and solving their equations, we can recover the key. Applying this process across different key conditions enables attacks over larger key spaces. Using this new framework, we mount practical key-recovery attacks on 840-round and 845-round Trivium, and a theoretical weak-key recovery attack on 855-round Trivium. For the practical attacks, we omit cube sum computation and key enumeration. Both are time-consuming but their complexity can be computed precisely (previous practical attacks omit the two steps as well). The remaining time cost corresponds to equation solving. To validate this part, we perform experiments with randomly selected keys, enabling us to evaluate both the equation-solving time and the number of key bits that can be recovered. The main contributions of this paper are summarized as follows.

Conditional Monomial Prediction. To recover conditional superpolies, we propose the *conditional monomial prediction* technique, which incorporates conditions into the monomial prediction model to accurately recover conditional superpolies. To accelerate this process, we develop a nested conditional coefficient solver that integrates variable substitution with symbolic computation. By constructing a conditional term expander based on the Non-zero Bit-based Division Property (NBDP) alongside this solver, we achieve significant speedups in recovering conditional superpolies. Furthermore, we propose an automatic key condition searching algorithm.

Automated Cube Search Algorithm. We introduce a fast automated cube search algorithm consisting of initial cube search based on numeric mapping that replaces inefficient manual selection, mother cube construction algorithm by calculating the number of valuable terms and subcube selection algorithm with fixed cube variables. This new search algorithm successfully identifies numerous cubes which can be used for attacks.

Improved Testing method. We develop two testing methods based on variable substitution for recovering keys from large-scale superpoly equation systems. The core idea of the methods is to guess low-degree equations of key bits and incorporate these guessed equations into the system of superpoly equations. By determining whether the equation system has no solution, the values of the low-degree equations are obtained. Subsequently, by solving the equation system through partial key enumeration, the entire key can be recovered.

To demonstrate the power of our method, we conduct key recovery attacks on Trivium reduced to 840, 845, and 855 rounds. For 840-round Trivium, under the framework of traditional cube attacks, we successfully execute practical attacks on **77.8%** of the tested keys with a complexity below 2^{55} . For 845-round Trivium, we employ the cube attack with elimination strategy, achieving practical attacks on **98.1%** of the tested keys with a complexity lower than 2^{56} in a weak-key space of size $2^{80} \times 59.1\%$. For 855-round Trivium, we have successfully recovered a balanced conditional superpoly, which enables a weak-key attack for 2^{74} keys. The previous key recovery attacks on Trivium and our results are summarized in Table 1.

In short, our work not only extends the known boundary of cube attacks on Trivium, but also contributes to the theoretical understanding of cube attacks through the proposed elimination strategy. This strategy systematically identifies key-dependent conditions under which complex superpolies become tractable. In this sense, our work deepens the theoretical understanding of the cube attack process and offer new insights into how key-dependent conditions influence the superpolies across rounds. From the description of

our attack framework, it can be seen that we do not impose many requirements on ciphers. Therefore, theoretically, our attack method is applicable to other NFSR-based ciphers.

Remark 1. Recently, a paper was posted on ePrint [HHL25] that constitutes a weak-key attack on 852-round Trivium. Our work is completely independent of theirs and was not inspired by their research. Specifically, there are two main differences.

- Different condition selection methods: They pre-select a set of conditions and traverse all conditions, selecting them by estimating the algebraic degree of the conditional superpoly. We compute a random part of superpoly and select the key-related expressions that appear most frequently as conditions.
- Different methods for recovering conditional superpolies: They need to precompute many linear constraints and incorporate them into the core monomial prediction model from [HHLW24]. We only need to add precise but few constraints to the superpoly recovery model proposed in this paper, this is because the conditions we select are all generated by "eliminable variables".

In addition to the above two aspects, we also propose new cube search and equation-solving techniques, which are also important in cube attacks.

Organization. The remaining parts of this paper are organized as follows. In Section 2, we introduce some basic concepts and background information. In Section 3, we present the cube attack with elimination strategy along with conditional monomial prediction. In Section 5, we introduce the fast cube searching algorithm. In Section 6, we revisit the existing equation-solving methods and provide our improvements. In Section 7, we mount attacks against 840-round, 845-round and 855-round Trivium. Finally, in Section 8, we draw our conclusions.

All our results and the source code are available at https://github.com/QHCrypto/FSE26_Cube_Attack_Elimination

2 Preliminaries

2.1 Boolean Functions

Let $\mathbb{F}_2^n$ be an n -dimensional vector space over the binary field $\mathbb{F}_2$. A function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is called a Boolean function in n variables. We write it as $f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$, which can be uniquely represented as a multi-variable polynomial over $\mathbb{F}_2$

$$f(\mathbf{x}) = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} a_{\mathbf{u}} \mathbf{x}^{\mathbf{u}}, a_{\mathbf{u}} \in \mathbb{F}_2.$$

where $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{x}^{\mathbf{u}} = \prod_{i=1}^n x_i^{u_i}$. We call it the algebraic normal form (ANF) of f , and $\mathbf{x}^{\mathbf{u}}$ is called a monomial of f if $a_{\mathbf{u}} = 1$. The algebraic degree of f is denoted by $\deg(f) := \max \{wt(\mathbf{u}) \mid a_{\mathbf{u}} \neq 0\}$, where $wt(\mathbf{u})$ is the Hamming Weight of $\mathbf{u}$.

2.2 Numeric Mapping

The numeric mapping was introduced by Liu *et al.* [Liu17] to estimate the algebraic degree of composite Boolean functions. Let $f(\mathbf{x}) = \bigoplus_{\mathbf{u}=(u_1, \dots, u_n) \in \mathbb{F}_2^n} a_{\mathbf{u}} \prod_{i=1}^n x_i^{u_i}$ be a Boolean function in n variables. The numeric mapping, denoted by DEG , is defined as

$$\begin{aligned} \text{DEG}: \mathcal{B}_n \times \mathbb{Z}^n &\rightarrow \mathbb{Z} \\ (f, \mathbf{D}) &\mapsto \max_{a_{\mathbf{u}} \neq 0} \left\{ \sum_{i=1}^n u_i d_i \right\}, \end{aligned}$$

Table 1: Summary of cube attacks against Trivium

Rounds	Cube size	#Cubes	Data	Time	Practical	Reference
672	12	63	$2^{18.6}$	2^{17}	✓	[DS09]
709	22-23	80	2^{23}	$2^{29.14}$	✓	[MS11]
767	28-31	35	2^{31}	2^{45}	✓	[DS09]
784	30-33	42	2^{33}	2^{39}	✓	[FV13]
799	32-37	18	2^{38}	2^{62}		[FV13]
805	32-38	42	2^{38}	$2^{41.4}$	✓	[YT21]
806	34-37	16	$2^{38.64}$	2^{64}		[YT21]
806	34-37	29	2^{39}	2^{39}	✓	[Sun21]
808	39-41	37	2^{44}	$2^{44.58}$	✓	[Sun21]
810	40-42	39	2^{44}	$2^{44.17}$	✓	[LHHW23]
815	44-46	35	2^{47}	$2^{47.32}$	✓	[CT22]
820	48-51	30	2^{53}	$2^{53.17}$	✓	[CT22]
820	38-45	1912	2^{45}	$< 2^{46}$	✓	[ZS24]
820	38	8192	2^{51}	2^{52}	✓	[WQW23]
825	49-52	31	2^{53}	$2^{53.09}$	✓	[LHHW23]
825	41	4096	2^{53}	2^{54}	✓	[WQW23]
830	41	8192	2^{54}	2^{55}	✓	[WQW23]
832	43-49	1373	2^{49}	$< 2^{50}$	✓	[ZS24]
832	72	1	2^{72}	2^{79}		[TIHM17, WHG ⁺ 19]
835	35	41	2^{35}	2^{75}		[LYWL18]
840	78	1	2^{78}	$2^{79.6}$		[HLM ⁺ 20]
840	75	3	$2^{76.6}$	$2^{77.8}$		[HSWW20]
840	47-62	6	2^{62}	$2^{76.32}$		[HST ⁺ 21]
840*	46-53	817	2^{54}	$< 2^{55}$	✓	Section 7.1
841	78	1	2^{78}	$2^{79.6}$		[HLM ⁺ 20]
841	76	2	2^{77}	$2^{78.6}$		[HSWW20]
841	56-76	3	2^{76}	2^{78}		[HST ⁺ 21]
842	78	1	2^{78}	$2^{79.6}$		[HLM ⁺ 20]
842	76	2	2^{77}	$2^{78.6}$		[HSWW20]
842	56-76	3	2^{76}	2^{78}		[HST ⁺ 21]
843	78	2	2^{78}	$2^{79.6}$		[Sun21]
843	56-76	5	2^{56}	2^{77}		[HST ⁺ 21]
844	54-55	2	2^{56}	2^{78}		[HST ⁺ 21]
845	54-55	2	2^{56}	2^{78}		[HST ⁺ 21]
845*	41-50	5893	$2^{55.43}$	$< 2^{56}$	✓	Section 7.2
846	51-54	6	2^{51}	2^{79}		[HHPW22]
847	52-53	2	2^{52}	2^{79}		[HHPW22]
848	52	1	2^{52}	2^{79}		[HHPW22]
849	44	2	2^{44}	2^{79}		[HHLW24]
850	44	1	2^{44}	2^{79}		[HHLW24]
851	44	1	2^{44}	2^{79}		[HHLW24]
852 [†]	44	1	2^{44}	2^{77}		[HHL25]
855[‡]	45	1	2^{45}	2^{73}		Section 7.3

* This attack is effective against 2^{80} keys with success rate 77.8%.

* This attack is effective against $59.1\% \times 2^{80}$ keys with success rate 98.1%.

[†] This attack is effective against 2^{78} keys.

[‡] This attack is effective against 2^{74} keys.

where $\mathcal{B}_n$ is the set of Boolean functions in n variables, $\mathbf{D} = (d_1, \dots, d_n) \in \mathbb{Z}^n$ is a vector of degrees. Let g_i ($1 \leq i \leq n$) be Boolean functions on m variables, and define $\deg(G) = (\deg(g_1), \dots, \deg(g_n))$ for $G = (g_1, g_2, \dots, g_n)$. The numeric degree of the composite function $h = f \circ G$ is defined as $\text{DEG}(f, \deg(G))$, denoted by $\text{DEG}(h)$ for short. The algebraic degree of h is always upper bounded by its numeric degree. This property allows iterative estimation of the algebraic degrees of output bits and internal states in NFSR-based cryptosystems using numeric mapping.

Furthermore, given an index set $c \subset [1, m]$, the numerical degree of h with respect to the variables with index in c is defined as $\text{DEG}_c(h) = \text{DEG}(f, \deg(G)|_c)$, where $\deg(G)|_c = (\deg(g_1|_c), \dots, \deg(g_n|_c))$, $\deg(g_i|_c)$ denotes the algebraic degree of variables with index in c .

2.3 Cube Attack

The cube attack was proposed by Dinur and Shamir [DS09]. Given a cryptographic algorithm, its output bit can be regarded as a Boolean function in the secret variables $\mathbf{k} = (k_1, k_2, \dots, k_n)$ and the public variables $\mathbf{v} = (v_1, v_2, \dots, v_m)$, expressed as $f(\mathbf{k}, \mathbf{v})$. Given an index set $c = \{i_1, \dots, i_\lambda\} \subseteq \{1, 2, \dots, m\}$, take a boolean vector $\mathbf{c} = (c_1, \dots, c_m)$, such that $c_i = 1$ if and only if $i \in c$. We call it a cube indicator vector. $f(\mathbf{k}, \mathbf{v})$ can be represented uniquely as

$$f(\mathbf{k}, \mathbf{v}) = \mathbf{v}^{\mathbf{c}} \cdot f_c(\mathbf{k}, \mathbf{v}) \oplus q(\mathbf{k}, \mathbf{v}),$$

where any monomials of q can not be divided by $\mathbf{v}^{\mathbf{c}}$. Typically, we refer to the index set c as the cube, $v_{i_1}, \dots, v_{i_\lambda}$ as the cube variables, and f_c as the superpoly. For f , different values can be assigned to the cube variables to obtain the different derived polynomials. In this paper, the superpolies we refer to are specifically those with all non-cube variables set to zero. The idea of cube attack is that given a fixed value of secret variables and non-cube public variables, the value of the superpoly can be obtained by traversing all possible values of the cube variables and summing all the derived polynomials, that is

$$f_c(\mathbf{k}, \mathbf{v}) = \bigoplus_{(v_{i_1}, v_{i_2}, \dots, v_{i_\lambda}) \in \mathbb{F}_2^\lambda} f(\mathbf{k}, \mathbf{v}).$$

The goal of the cube attack is to find certain cubes such that their corresponding superpolies are linear or of low degree in the secret variables, and then solve the system of superpoly equations to recover the key.

2.4 Monomial Prediction

At Asiacrypt 2020, Hu *et al.* [HSWW20] proposed the monomial prediction technique which can precisely recover superpolies. They also established an equivalence with division property. Below, we give an introduction to the monomial prediction technique.

Consider a vector Boolean function composed of r rounds: $\mathbf{f} = \mathbf{f}_{r-1} \circ \dots \circ \mathbf{f}_0$. Let $\mathbf{x}_0$ be the input of $\mathbf{f}_0$, and denote the output of $\mathbf{f}_i$ as $\mathbf{x}_{i+1}$. For any vector $\mathbf{u}$, define the monomial function $\pi_{\mathbf{u}}(\mathbf{x}) = \mathbf{x}^{\mathbf{u}}$. For a given monomial $\pi_{\mathbf{u}_{i+1}}(\mathbf{x}_{i+1})$ at round $i+1$, it can be expressed as a polynomial of $\mathbf{x}_i$. We write $\pi_{\mathbf{u}_i}(\mathbf{x}_i) \rightarrow \pi_{\mathbf{u}_{i+1}}(\mathbf{x}_{i+1})$ if the monomial $\pi_{\mathbf{u}_i}(\mathbf{x}_i)$ appears in the ANF of $\pi_{\mathbf{u}_{i+1}}(\mathbf{x}_{i+1})$. Then from round 0 to round r , we call

$$\pi_{\mathbf{u}_0}(\mathbf{x}_0) \rightarrow \pi_{\mathbf{u}_1}(\mathbf{x}_1) \rightarrow \dots \rightarrow \pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$$

a monomial trail from $\pi_{\mathbf{u}_0}(\mathbf{x}_0)$ to $\pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$, denoted by $\pi_{\mathbf{u}_0}(\mathbf{x}_0) \rightsquigarrow \pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$. If no such trail exists, we write $\pi_{\mathbf{u}_0}(\mathbf{x}_0) \not\rightsquigarrow \pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$.

The set of all trails from $\pi_{\mathbf{u}_0}(\mathbf{x}_0)$ to $\pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$ is denoted by $\pi_{\mathbf{u}_0}(\mathbf{x}_0) \bowtie \pi_{\mathbf{u}_{r-1}}(\mathbf{x}_{r-1})$.

Theorem 1 (Monomial prediction [HSWW20]). *We have $\pi_{\mathbf{u}_0}(\mathbf{x}_0) \rightarrow \pi_{\mathbf{u}_r}(\mathbf{x}_r)$ if and only if the number of monomial trails from $\pi_{\mathbf{u}_0}$ to $\pi_{\mathbf{u}_r}$ is odd:*

$$|\pi_{\mathbf{u}_0}(\mathbf{x}_0) \bowtie \pi_{\mathbf{u}_r}(\mathbf{x}_r)| \equiv 1 \pmod{2}.$$

Theorem 2 (Superpoly recovery [HSWW20]). *Let f be a Boolean function with variables $\mathbf{v}$ and $\mathbf{k}$, and $f = f_{r-1} \circ f_{r-2} \circ \dots \circ f_0(\mathbf{v}, \mathbf{k})$. Given a cube c , after setting all public non-cube variables to be 0, the superpoly of c is*

$$\text{Coe}(f, \mathbf{v}^c) = \bigoplus_{|\mathbf{k}^w \mathbf{v}^c \bowtie f| \equiv 1 \pmod{2}} \mathbf{k}^w,$$

where $\text{Coe}(f, \mathbf{v}^c)$ denotes the coefficient polynomial of f with respect to the monomial $\mathbf{v}^c$.

Propagation rules and MILP models. Many symmetric cipher can be decomposed into a sequence of the basic operations XOR, AND, and COPY. To construct an MILP model for the monomial trail of a Boolean function, one only needs to model these three basic operations. After building the MILP model, one can use the Gurobi optimizer [Gur24] to solve it. Please refer to Appendix C for details.

At Asiacrypt 2021, Hu *et al.* [HST⁺21] proposed a framework for nested monomial prediction, which improved upon previous superpoly recovery algorithms. At Asiacrypt 2022, He *et al.* [HHPW22] introduced the concept of valuable terms, refining the earlier framework into two processes: Choose a middle round r_m , use **Term Expander** to recover all the valuable terms, and then use **Coefficient Solver** to compute their coefficient of $\mathbf{v}^c$ for valuable terms, summing the results.

Definition 1 (Valuable terms [HHPW22]). For a Boolean function $f(\mathbf{v}, \mathbf{k}) = f_{r-1} \circ f_{r-2} \circ \dots \circ f_0(\mathbf{v}, \mathbf{k})$, denote the input and output of f_i by $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$, respectively. For $0 \leq r_m < r$, a term $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$ is called a valuable term of $\text{Coe}(f, \mathbf{v}^c)$ at round r_m if it satisfies the following conditions:

- (1) $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \rightarrow f$.
- (2) $\exists \mathbf{k}^w$ such that $\mathbf{k}^w \mathbf{v}^c \rightsquigarrow \pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$.

Additionally, He *et al.* introduced two novel techniques: the non-zero bit-based division property (NBDP) and the core monomial prediction (CMP). These techniques were respectively applied to **Term Expander** and **Coefficient Solver**, resulting in a significant acceleration of superpoly recovery. For further details, see [HHPW22].

2.5 Coefficient Solver with Variable Substitution

At Asiacrypt 2023, Wang *et al.* [WQW23] proposed a new coefficient solver. This coefficient solver aims to apply variable substitutions to the key, yielding a superpoly with simple ANF.

For a Boolean function $f(\mathbf{v}, \mathbf{k}) = f_{r-1} \circ f_{r-2} \circ \dots \circ f_0(\mathbf{v}, \mathbf{k})$ with inputs $\mathbf{v} \in \mathbb{F}_2^n$ and $\mathbf{k} \in \mathbb{F}_2^m$, where $f_i : \mathbb{F}_2^t \rightarrow \mathbb{F}_2^t$, $i = 0, \dots, r-2$. Denote the input and output of f_i by $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$, respectively. Let $\overrightarrow{f_{r_m}}$ denote $f_{r_m-1} \circ \dots \circ f_0$, then $\mathbf{y}_{r_m} = \overrightarrow{f_{r_m}}(\mathbf{v}, \mathbf{k})$.

Assume the j -th component of $\overrightarrow{f_{r_m}}$ is

$$\overrightarrow{f_{r_m}}[j] = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} h_{\mathbf{u}}^{r_m}(\mathbf{k})[j] \cdot \mathbf{v}^{\mathbf{u}}$$

Here, we denote the coefficient of $\mathbf{v}^{\mathbf{u}}$ in $\overrightarrow{f_{r_m}}[j]$ by $h_{\mathbf{u}}^{r_m}(\mathbf{k})[j]$.

To simplify the ANF of the superpoly, they proposed an iterative variable substitution algorithm:

1. Set $\mathbf{z}_0 = \mathbf{k}$.
2. Compute the first round function ($i = 1$).

$$\vec{f}_1 = \vec{f}_0(\mathbf{v}, \mathbf{k}) = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} h_{\mathbf{u}}^1(\mathbf{k}) \mathbf{v}^{\mathbf{u}} = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} h_{\mathbf{u}}^1(\mathbf{z}_0) \mathbf{v}^{\mathbf{u}}.$$

Set a new variable $\mathbf{z}_1 = \mathbf{z}_0$. For each $j = 0, \dots, l-1$ and $\mathbf{u} \in \mathbb{F}_2^n$, if $h_{\mathbf{u}}^1(\mathbf{z}_0)[j]$ is not a monomial, define a new variable $z = h_{\mathbf{u}}^1(\mathbf{z}_0)[j]$ and append z to $\mathbf{z}_1$. Thus, $\mathbf{z}_1$ is updated as an expression of $\mathbf{z}_0$.

3. For $i = 2, 3, \dots, r_m$, repeat the above process to derive the vector Boolean function $\mathbf{f}'_{r_m}$ such that $\mathbf{y}_{r_m} = \mathbf{f}'_{r_m}(\mathbf{v}, \mathbf{z}_{r_m})$, whose ANF in $\mathbf{v}$ and $\mathbf{z}_{r_m}$ can be written as

$$\mathbf{f}'_{r_m}[j] = \bigoplus_{\mathbf{u}} a_{\mathbf{u},j} \mathbf{z}_{r_m}^{\mathbf{b}_{\mathbf{u},j}} \mathbf{v}^{\mathbf{u}},$$

where $\mathbf{f}'_{r_m}[j]$ represents j -th component of $\mathbf{f}'_{r_m}$, and $a_{\mathbf{u},j} \in \mathbb{F}_2$ and $\mathbf{b}_{\mathbf{u},j} \in \mathbb{F}_2^{|\mathbf{z}_{r_m}|}$ are both determined by $\mathbf{u}$ and j .

In fact, solving $\text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c)$ in $\mathbf{z}_{r_m}$ by monomial prediction is equivalent to finding all possible monomial trails $\mathbf{z}_{r_m}^{\mathbf{b}} \mathbf{v}^c \rightsquigarrow \pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$. One can construct an MILP model to describe all feasible trails.

2.6 Description of Trivium

Trivium [DCP08] is an NFSR-based stream cipher that was one of the eSTREAM hardware-oriented finalists. Its initial state is represented by a 288-bit state $(s_0, s_1, \dots, s_{287})$. In the initialization phase, the 80-bit key is loaded to the first register and the 80-bit IV is loaded to the second register. The other state bits are set to 0 except the last three bits in the third register. Namely, the initial state bits are given by

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

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

```

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

```

After the initialization, one key stream bit is produced as $z = s_{65} \oplus s_{92} \oplus s_{161} \oplus s_{176} \oplus s_{242} \oplus s_{287}$ during each iteration. This paper studies round reduced variants of Trivium with initialization state updates reduced to 840, 845, and 855 rounds, respectively.

3 Cube Attacks with Elimination Strategy

Recent research shows that for higher-round ciphers, even when we can recover the superpoly, the solving time remains very long, making it difficult to recover a large number of superpolies for attacks within an acceptable time. Furthermore, even after spending substantial time to recover these superpolies, their ANFs are often extremely complex (e.g. more than 1000000 monomials). Currently, there is no effective method to extract more than 1 bit key information from such complex polynomials, rendering practical attacks infeasible.

In this section, we propose a new framework for cube attacks based on an elimination strategy, which addresses the above challenges and enables practical attacks on higher-round ciphers with feasible complexity.

For Boolean functions such as those in Trivium after many iterations, their ANFs are highly complex, often involving a large number of monomials. We can describe a superpoly as

$$p_c(\mathbf{k}) = \sum_{i \in \Lambda} g_i(\mathbf{k}) \cdot h_i(\mathbf{k}) + r(\mathbf{k}). \quad (1)$$

Here, g_i is a simple polynomial in $\mathbf{k}$ with few monomials, h_i is a Boolean polynomial with complex ANF, Λ is an index set, and r is a Boolean polynomial with less complex ANF. For polynomials of this form, if we set $g_i = 0$, then $p_c = r$. Thus, for keys satisfying $g_i = 0$ for all i , we can take advantage of the simpler polynomial r instead of the complex polynomial p to recover key information. To formally characterize such polynomials, we introduce the following definition.

Definition 2 (Conditional Superpoly). Given a superpoly $p_c(\mathbf{k}) = \sum_{i \in \Lambda} g_i \cdot h_i + r$, we call r the conditional superpoly of p_c with respect to the set $G = \{g_i\}_{i \in \Lambda}$, denoted as $p_{c,G}(\mathbf{k})$. In the following, we call G the conditional set.

Given a conditional set G , if we can find a set of conditional superpolies with relatively simpler ANFs with respect to G , we can recover the key information in this key subspace $\text{Zero}(G)$ by solving the system of equations for these conditional superpolies, where $\text{Zero}(G) := \{\mathbf{k} \mid g_i(\mathbf{k}) = 0, \forall g_i \in G\}$. Below, we outline the main steps of this framework.

Offline Phase: First, find several pairs of cube and its conditional set $(\mathcal{C}_{G_1}, G_1), \dots, (\mathcal{C}_{G_n}, G_n)$. Then compute the conditional superpolies for these cubes with respect to G_i , denoted by $\mathcal{P}_{\mathcal{C}_{G_i}}$.

Online Phase: In this phase, with the key fixed, we compute the value of superpolies for $\mathcal{C}_{G_i}$ by calculating the cube sum. For any cube $c \in \mathcal{C}_{G_i}$, we denote the conditional superpoly with respect to c and the conditional set G_i by p_{c,G_i} , the value of cube sum be b_c . Assign the values of these superpolies to their corresponding conditional superpolies, and establish equation systems for each conditional polynomial set:

$$\mathbb{E}_i = \bigcup_{c \in \mathcal{C}_{G_i}} \{p_{c,G_i} = b_c\} \cup \bigcup_{g \in G_i} \{g = 0\}. \quad (2)$$

By using a testing method to solve these equation systems separately, we obtain a corresponding equation set of simple equations, denoted as $\mathbb{S}_i$. Without loss of generality, let $m_1 \geq m_2 \geq \dots \geq m_n$. Assuming the key length is L , we begin by enumerating possible keys using the m_1 independent equations obtained by solving $\mathbb{E}_1$, requiring 2^{L-m_1} enumeration steps. If the correct key is not found, we proceed to utilize the m_2 equations derived from solving $\mathbb{E}_2$ for further 2^{L-m_2} enumeration. This process continues iteratively until the correct key is recovered. If our equation-solving method is accurate, any key in the weak-key space $\bigcup_{i=1}^n \text{Zero}(G_i)$ can be recovered through this process.

Complexity. Assume the complexity of calculating all the cube sums is T_c , and the time used for solving the equations is T_i . For a key $\mathbf{k} \in \bigcup_{i=1}^n \text{Zero}(G_i)$, without loss of

generality, suppose $m_1 \geq m_2 \geq \dots \geq m_n$. The time complexity is bounded by

$$T_c + \sum_{i=1}^t 2^{L-m_i} + T_i,$$

where t is the smallest index such that $\mathbf{k} \in \text{Zero}(G_t)$.

Discussion. Some readers may draw connections between our attack framework and conditional cube attacks [HWX⁺17]. To clarify these differences, we can divide the key into condition-dependent and condition-independent parts. Conditional cube attacks determine whether key conditions are satisfied by computing cube sums, thereby extracting information of the condition-dependent part of the key. In contrast, our attack focuses on weak keys that satisfy specific conditions and recovers the conditional superpoly to obtain information of the condition-independent part of the key.

In fact, to ensure that $r(\mathbf{k})$ remains less complex, we select g_i from the polynomials related to the key that appear in the early iterations of the round function. Specifically, we choose an intermediate round number r_c and select some g_i as certain $\mathbf{z}_{r_c}[i]$, where $\mathbf{z}_{r_c}$ is a vector Boolean function in terms of $\mathbf{k}$ defined in Section 2.5. In the following, we denote a conditional set as $\mathcal{Z} = \{\mathbf{z}_{r_c}[i]\}$, the superpoly after variable substitution as $p_c(\mathbf{z}_{r_c})$, the conditional superpoly of p_c with respect to $\mathcal{Z}$ as $p_{c,\mathcal{Z}}(\mathbf{z}_{r_c})$.

To implement the attack, three key problems must be addressed.

Problem 1. *How to find a set $\mathcal{Z} = \{\mathbf{z}_{r_c}[i]\}$ that ensures $p_{c,\mathcal{Z}}(\mathbf{z}_{r_c})$ has a less complex ANF without recovering $p_c(\mathbf{z}_{r_c})$.*

Problem 2. *After obtaining set $\mathcal{Z}$, how to select a cube set $\mathcal{C}_{\mathcal{Z}}$ such that their conditional superpolies have less complex ANFs.*

Problem 3. *After selecting a set $\mathcal{Z}$, how to efficiently and accurately recover the conditional superpoly $p_{c,\mathcal{Z}}(\mathbf{z}_{r_c})$ without computing the full superpoly $p_c(\mathbf{z}_{r_c})$.*

In the following, we start with Problem 3.

3.1 Conditional Monomial Prediction

In this section, we propose a method to incorporate conditions into the superpoly recovery model. We begin with the following definition.

Definition 3 (Eliminable). For a cube indicator vector $\mathbf{c}$, intermediate round r_c , and substituted variable $\mathbf{z}_{r_c}$, let $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})$ denote the vector Boolean function after r_c rounds with all non-cube variables set to 0. We say variable $\mathbf{z}_{r_c}[i]$ is *eliminable* with respect to $\mathbf{c}$ at round r_c if and only if the following holds: for every component $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})[j]$ in which $\mathbf{z}_{r_c}[i]$ appears, the expression $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})[j]$ must equal $\mathbf{z}_{r_c}[i]$ itself, that is, $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})[j] = \mathbf{z}_{r_c}[i]$.

We call such $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})[j]$ the *target elimination* state with respect to $\mathbf{z}_{r_c}[i]$ and $\mathbf{c}$. Specially, a conditional set where all elements are eliminable is defined as an eliminable set.

Below, we provide a simple example to better illustrate the meaning of "eliminable".

Example 1. Suppose $\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})$ is a vector Boolean function with 4 components, satisfying:

$$\begin{aligned} \mathbf{y}_{r_c}[0] &= \mathbf{z}_{r_c}[1] + \mathbf{v}[0], \\ \mathbf{y}_{r_c}[1] &= \mathbf{z}_{r_c}[1] + \mathbf{v}[1] \cdot \mathbf{z}_{r_c}[3], \\ \mathbf{y}_{r_c}[2] &= \mathbf{z}_{r_c}[2] + \mathbf{v}[2], \\ \mathbf{y}_{r_c}[3] &= \mathbf{z}_{r_c}[3] + \mathbf{v}[3]. \end{aligned}$$

If we select the cube as $\{0, 2\}$ and set the non-cube variables to 0, then $\mathbf{y}_{r_c}[3] = \mathbf{z}_{r_c}[3]$, and $\mathbf{z}_{r_c}[3]$ appears only in $\mathbf{y}_{r_c}[3]$. Therefore, $\mathbf{z}_{r_c}[3]$ is eliminable with respect to the cube at round r_c , whereas $\mathbf{z}_{r_c}[1]$ and $\mathbf{z}_{r_c}[2]$ are not.

From this definition, we can directly derive the following property.

Property 1. For rounds $r_0 < r_1$ and $i < |\mathbf{z}_{r_0}|$, if $\mathbf{z}_{r_1}[i]$ is an eliminable variable at round r_0 , then $\mathbf{z}_{r_1}[i]$ is not necessarily eliminable at round r_1 .

This occurs because, from round r_0 to r_1 , some state components $\mathbf{y}_{r_0}(\mathbf{z}_{r_1}, \mathbf{v})[j] = \mathbf{z}_{r_1}[i]$ may disappear in $\mathbf{y}_{r_1}$. Generally, there are more eliminable variables at lower rounds.

In the term expansion of monomial prediction, denote $\mathbb{S}(\mathbf{y}_{r_m})$ as a valuable terms set. It is obvious that any valuable term $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$ in $\mathbb{S}(\mathbf{y}_{r_m})$ can be expressed as a sum of several monomials of $\mathbf{y}_{r_c}$ at round $r_c < r_m$, denoted by $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) = \sum_{j=0}^n \pi_{\mathbf{u}_{r_c}^j}(\mathbf{y}_{r_c})$, where n is the number of monomials in the expansion. Given an eliminable variable $\mathbf{z}_{r_c}[i]$, let $\{\mathbf{y}_{r_c}(\mathbf{z}_{r_c}, \mathbf{v})[i_t]\}_{t=0}^{\lambda_i}$ denote the set of all target elimination states with respect to $\mathbf{z}_{r_c}[i]$, where λ_i is the number of such states.

If every $\pi_{\mathbf{u}_{r_c}^j}(\mathbf{y}_{r_c})$ contains at least one target elimination state as a factor, then

$$\begin{aligned} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_c}[i]=0} &= \sum_{j=0}^n \text{Coe}(\pi_{\mathbf{u}_{r_c}^j}(\mathbf{y}_{r_c}), \mathbf{v}^c) |_{\mathbf{z}_{r_c}[i]=0} \\ &= \sum_{j=0}^n \text{Coe}(\pi_{\mathbf{u}_{r_c}^j}(\mathbf{y}_{r_c}), \mathbf{v}^c) |_{\mathbf{y}_{r_c}[i_t]=0} \\ &= 0. \end{aligned} \tag{3}$$

Consequently, during the term expansion phase, we can add constraints $\forall i, t, \mathbf{u}_{r_c}[i_t] = 0$ to the **TermExpander** to get rid of the trail

$$\mathbf{k}^w \mathbf{v}^c \rightsquigarrow \pi_{\mathbf{u}_{r_c}^j}(\mathbf{y}_{r_c}) \rightsquigarrow \pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}).$$

This allows us to eliminate valuable terms that become zero under these constraints.

Definition 4 (Conditional Valuable Terms). For a Boolean function $f(\mathbf{v}, \mathbf{k}) = f_{r-1} \circ \mathbf{f}_{r-2} \circ \dots \circ \mathbf{f}_0(\mathbf{v}, \mathbf{k})$, denote the input and output of $\mathbf{f}_i$ by $\mathbf{y}_i$ and $\mathbf{y}_{i+1}$ respectively. Given $0 \leq r_c < r_m < r$ and a set of eliminable variables $\mathcal{Z} = \{\mathbf{z}_{r_c}[i]\}_{i \in \Lambda}$, let $\bigcup_{i \in \Lambda} \{\mathbf{y}_{r_c}[i_t]\}_{t=0}^{\lambda_i}$ be its corresponding target elimination states. A term $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$ is called a conditional valuable term of $\text{Coe}(f, \mathbf{v}^c)$ at round r_m if it satisfies:

- (1) $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \rightarrow f$.
- (2) There exist $\mathbf{k}^w$ and $\mathbf{u}_{r_c}$ with $\mathbf{u}_{r_c}[i_t] |_{\forall i \in \Lambda, t \leq \lambda_i} = 0$ such that $\mathbf{k}^w \mathbf{v}^c \rightsquigarrow \pi_{\mathbf{u}_{r_c}}(\mathbf{y}_{r_c}) \rightsquigarrow \pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m})$.

We can construct a MILP model **Con-NBDP-MPModel** to recover conditional valuable terms $\mathbb{S}_c(\mathbf{y}_{r_m})$ in the following. For a monomial $\pi_{\mathbf{u}_r}(\mathbf{y}_r)$ of f , cube indicator vector $\mathbf{c}$, a middle round r_m , we first build a model $\mathcal{M} = \text{NBDP-MPModel}(\mathbf{u}_r, r, r_m, \mathbf{c})$. For all indices i_t such that $\mathbf{y}_{r_c}[i_t]$ is a target elimination state, we construct model $\mathcal{M}'$ by adding constrains $\mathbf{u}_{r_c}[i_t] = 0$ to model $\mathcal{M}$. Then the solutions of $\mathcal{M}'$ are all conditional valuable terms. The details of **Con-NBDP-MPModel** are provided in Appendix F, Algorithm 7. Correspondingly, based on **Con-NBDP-MPModel**, we propose the **Con-TermExpander**($\mathbb{S}(\mathbf{y}_r), \mathbf{c}, r, r_c, r_m, \mathcal{Z} = \{\mathbf{z}_{r_c}[i]\}_{i \in \Lambda}$), which is described in Appendix F, Algorithm 8.

To recover conditional superpolies, the next step is to calculate the coefficient of the valuable terms. Similarly, we propose a nested solver **Nest-Con-CoeSolver**($\mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_m})$)

, $\mathbf{c}, r_m, r_s, \mathcal{Z}$) with a basic coefficient solver **Con-CoeSolver0**($\mathbf{u}_{r_m}, \mathbf{c}, t_{r_m}, \mathbf{z}_{r_m} = \mathbf{g}'(\mathbf{z}_{r_s}), \mathbf{y}'_{r_m}, q = 0, \mathcal{Z}$). To eliminate the variables in $\mathcal{Z}$, we remove monomials containing $\mathbf{z}_{r_c}[i]$ from both $\mathbf{g}$ and $\mathbf{y}_{r_m}$ in the pre-computation phase, yielding refined functions: $\mathbf{g}'(\mathbf{z}_{r_s}) = \mathbf{g}(\mathbf{z}_{r_s})|_{\mathbf{z}_{r_c}[i]=0, i \in \Lambda}$, $\mathbf{y}'_{r_m} = \mathbf{y}_{r_m}|_{\mathbf{z}_{r_c}[i]=0, i \in \Lambda}$. The details of the above two algorithms are provided in the Algorithms 10 and 9. In the following, we give the proof of the soundness of this conditional superpoly recovery algorithm.

Theorem 3 (Soundness). *Suppose that $p_{\mathbf{c}}(\mathbf{z}_{r_s})$ is a superpoly recovered by **TermExpander** and **Nest-CoeSolver**, and $\mathcal{Z} = \{\mathbf{z}_{r_c}[i]\}_{i \in \Lambda}$ is an eliminable set at round r_c . The polynomial recovered by **Con-TermExpander** and **Nest-Con-CoeSolver** is identical to $p_{\mathbf{c}, \mathcal{Z}}(\mathbf{z}_{r_s})$. Specially, when $\mathcal{Z} = \emptyset$, the conditional superpoly recovery algorithm degenerates to the superpoly recovery algorithm.*

Proof. Denote $\mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_m}) \subseteq \mathbb{S}(\mathbf{y}_{r_m})$ as the conditional valuable terms set at r_m -round, respectively. Suppose $\bigcup_{i \in \Lambda} \{\mathbf{y}_{r_c}[i_t]\}_{t=0}^{\lambda_i}$ be its corresponding target elimination states at r_c -round. We have

$$\begin{aligned} p_{\mathbf{c}, \mathcal{Z}}(\mathbf{z}_{r_s}) &= \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_s}[i]=0} \\ &= \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_s}[i]=0} \\ &\quad \oplus \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_{\mathcal{Z}}^c(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_s}[i]=0}, \end{aligned} \quad (4)$$

where $\mathbb{S}_{\mathcal{Z}}^c(\mathbf{y}_{r_m})$ is the complement of $\mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_m})$. By Equation (3),

$$\begin{aligned} &\bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_{\mathcal{Z}}^c(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_s}[i]=0} = 0 \\ &= \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_{\mathcal{Z}}^c(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_c}[i]=0} = 0 \\ &= \bigoplus_{\exists i_t s.t. \mathbf{u}_{r_c}[i_t]=0} \text{Coe}(\pi_{\mathbf{u}_{r_c}}(\mathbf{y}_{r_c}), \mathbf{v}^c) |_{\mathbf{z}_{r_c}[i]=0} = 0 \\ &= 0. \end{aligned} \quad (5)$$

Then,

$$p_{\mathbf{c}, \mathcal{Z}}(\mathbf{z}_{r_s}) = \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^c) |_{\mathbf{z}_{r_s}[i]=0}, \quad (6)$$

where the right side of Equation (6) is the output of **Nest-Con-CoeSolver**. $\square$

Note that, in order to keep the conditional superpoly in a simple form, r_s is generally not chosen to be very low. However, for higher values of r_s , the number of eliminable variables is smaller compared to lower rounds. Therefore, we select the conditional rounds r_c smaller than the output round r_s .

3.2 Condition Selection Algorithm

In this section, we focus on Problem 1, which is selecting a small eliminable set $\mathcal{Z} = \{\mathbf{z}_{r_s}[i]\}_{i \in \Lambda}$ to reduce the scale of $p_{\mathbf{c}, \mathcal{Z}}(\mathbf{z}_{r_s})$ as much as possible. We propose a greedy algorithm to achieve this goal. Its core concept is to iteratively compute partial superpolies and select the eliminable variables that appear most frequently in their terms. Next, we add conditions to eliminate these variables and compute the partial conditional superpoly.

This process repeats until a termination condition is met. Finally, we select suitable subsets from the obtained eliminable set. This algorithm consists of two phases: The first phase involves generating a candidate set $\mathcal{Z}_0$. The second phase involves selecting a suitable subset from $\mathcal{Z}_0$. The main steps are as follows.

1. Initialize the set $\mathcal{Z}_0 = \emptyset$, $count = 0$, select round parameters $r_{b_0} > r_{b_1}$, and thresholds n, m, s, l, T, T_0, T_1 .
2. Initialize the set $\mathcal{Z}_1 = \emptyset$, $count = count + 1$. If $count > l$, output the set $\mathcal{Z}_0$. For a given cube indicator vector $\mathbf{c}$, use the conditional term expander to compute the set of valuable terms $\mathbb{S}_{\mathcal{Z}_0}(\mathbf{y}_{r_{b_1}})$ at round r_{b_1} . Randomly select T_1 valuable terms from this set and add them to $\mathbb{S}$. Use the conditional coefficient solver to compute the conditional superpoly of $\mathbb{S}$.
3. For the superpoly obtained in Step 2, identify the n most frequent eliminable variables and add them to the sets $\mathcal{Z}_0$ and $\mathcal{Z}_1$.
4. Use the conditional term expander to compute the number of conditional valuable terms $CVTr_{r_{b_0}}$ at round r_{b_0} with respect to eliminable set $\mathcal{Z}_1$. If $CVTr_{r_{b_0}} < T_0$, return to Step 2. Otherwise, compute the set of conditional valuable terms $\mathbb{S}_{\mathcal{Z}_1}$ at round r_{b_0} . Randomly select T_1 valuable terms from this set and add them to $\mathbb{S}$. Use the conditional coefficient solver to compute the conditional superpoly corresponding to these valuable terms.
5. For the superpoly obtained in Step 4, identify the n most frequently eliminable variables and add them to the sets $\mathcal{Z}_0$ and $\mathcal{Z}_1$, then return to Step 4. If no eliminable variable exists in the superpoly, return to Step 2.

Through the above algorithm, we generate a candidate set in which each variable is eliminable. Next, we select an appropriate subset from this candidate set.

6. For the set $\mathcal{Z}_0$ from the above algorithm, compute $CVTr_{r_{b_0}}$ for each variable in $\mathcal{Z}_0$ and select the m variables with the smallest $CVTr_{r_{b_0}}$ to update $\mathcal{Z}_0$.
7. Enumerate and evaluate subsets of $\mathcal{Z}_0$ with size less than s , compute their $CVTr_{r_{b_1}}$, and output all subsets satisfying $CVTr_{r_{b_1}} \leq T$.

The complete algorithm is given in Algorithm 14. Below, we give a toy example to clarify this process.

Example 2. Suppose we have a toy cipher $(\mathbf{v}, \mathbf{z}) \xrightarrow{f_0} \mathbf{x} \xrightarrow{f_1} \mathbf{y}$, where

$$\mathbf{x} = \mathbf{f}_0(\mathbf{v}, \mathbf{z}) = (\mathbf{v}[0] + \mathbf{z}[0], \mathbf{v}[1] + \mathbf{z}[1], \mathbf{z}[2], \mathbf{v}[0]\mathbf{z}[3] + \mathbf{v}[1]\mathbf{z}[4], \mathbf{z}[5], \mathbf{z}[6], \mathbf{z}[7]),$$

and the output function

$$\mathbf{y} = \mathbf{f}_1(\mathbf{x}) = \mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[4] + \mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[2] + \mathbf{x}[1]\mathbf{x}[3]\mathbf{x}[5] + \mathbf{x}[0]\mathbf{x}[3]\mathbf{x}[6] + \mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[2]\mathbf{x}[4]\mathbf{x}[5].$$

We select the cube as $\{0, 1\}$, then all the monomials of $\mathbf{y}$ are valuable terms, the corresponding superpoly is

$$\mathbf{z}[2] + \mathbf{z}[5] + \mathbf{z}[3]\mathbf{z}[6] + \mathbf{z}[4]\mathbf{z}[7] + \mathbf{z}[2]\mathbf{z}[5]\mathbf{z}[6].$$

Our goal is to obtain eliminable sets of size 2. We set parameters $n = 2, m = 3, s = 3, l = 2, T_1 = 2, T_0 = 4, T = 3$ and initialize an empty set $\mathcal{Z}_0$. Here, $l = 2$ serves as the upper bound for the number of loops.

Loop 1:

- We randomly select $T_1 = 2$ valuable terms: $\mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[4]$ and $\mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[4]$.
- The partial superpoly is: $\mathbf{z}[2] + \mathbf{z}[5]$.

- Since $z[2]$ and $z[5]$ are the $n = 2$ most frequently eliminable variables in this partial superpoly, we add them to the set $\mathcal{Z}_0$ and an empty set $\mathcal{Z}_1$.
- The number of conditional valuable terms with respect to $\mathcal{Z}_1$ is 3, because the corresponding target elimination states $\mathbf{x}[2]$ and $\mathbf{x}[4]$ are set to 0.
- Since $3 < T_0$, this loop ends.

Loop 2:

- We proceed to randomly select $T_1 = 2$ valuable terms: $\mathbf{x}[1]\mathbf{x}[3]\mathbf{x}[5]$ and $\mathbf{x}[0]\mathbf{x}[3]\mathbf{x}[6]$.
- The partial superpoly is: $z[3]z[6] + z[4]z[7]$.
- Since $z[6]$ and $z[7]$ are the $n = 2$ most frequently eliminable variables in this partial superpoly, we add them to $\mathcal{Z}_0$ and $\mathcal{Z}_1$.
- The number of conditional valuable terms with respect to $\mathcal{Z}_1$ is 2, because the corresponding target elimination states $\mathbf{x}[5]$ and $\mathbf{x}[6]$ are set to 0.
- Since $2 < T_0$, this loop ends.

We now get a candidate set $\mathcal{Z}_0 = \{z[2], z[5], z[6], z[7]\}$. To find eliminable subsets of size 2:

- Compute conditional valuable terms for each variable in $\mathcal{Z}_0$: 3, 3, 3, 4 respectively.
- Select the $m = 3$ variables with smallest counts: $z[2], z[5], z[6]$. Update $\mathcal{Z}_0$ accordingly.
- Enumerate and evaluate subsets of $\mathcal{Z}_0$ with size $< s = 3$, output those with conditional valuable terms $\leq T = 3$.

Finally, we obtain 3 eliminable sets: $\{z[2], z[5]\}$, $\{z[2], z[6]\}$ and $\{z[5], z[6]\}$.

3.3 Selection of Pairs

Our ultimate goal is to select a set of pairs $(\mathcal{Z}, \mathcal{C}_{\mathcal{Z}})$ such that each cube in $\mathcal{C}_{\mathcal{Z}}$ yields a conditional polynomial with respect to $\mathcal{Z}$ that has a less complex ANF. Given a candidate cube set, computing the eliminable set for each cube is computationally infeasible when the candidate cube set is large. Therefore, we propose the following method to select pairs.

1. Randomly select m cubes from the candidate cube set $\mathcal{C}$ and compute their eliminable sets by Algorithm 14. Count the frequency of these eliminable sets and select the n most frequent sets, denoted $\mathcal{Z}_1, \dots, \mathcal{Z}_n$.
2. For each set $\mathcal{Z}_i$, iterate through all cubes in $\mathcal{C}$ and select those cubes for which every variable in $\mathcal{Z}_i$ is eliminable, adding them to the set $\mathcal{C}_{\mathcal{Z}_i}$.
3. For each cube in $\mathcal{C}_{\mathcal{Z}_i}$, choose a number of middle rounds r_b and compute the number of conditional valuable terms CVT_{r_b} . Select a threshold T and remove cubes with $CVT_{r_b} > T$, thereby obtaining the final pair $(\mathcal{Z}_i, \mathcal{C}_{\mathcal{Z}_i})$.

4 The Nested Coefficient Solver

In this section, we propose a nested coefficient solver, which can be regarded as a combination of symbolic computation, monomial prediction with variable substitution, and term expansion based on NBDP. We will subsequently provide theoretical explanations for its improved efficiency and demonstrate the framework's superiority through experiments.

First, let us review the coefficient solver proposed in [WQW23]. Given a valuable terms set $\mathbb{S}(\mathbf{y}_{r_m})$ at round r_m , to compute the superpoly $p_c(\mathbf{z}_{r_s})$, where $\mathbf{z}_{r_s}$ is generated by the $r_s(< r_m)$ -round variable substitution mentioned in Section 2.5, the structure of their coefficient solver can be equivalently expressed as

$$\mathbb{S}(\mathbf{y}_{r_m}) \xrightarrow{\text{TermExpander}} \mathbb{S}(\mathbf{y}_{r_s}) \xrightarrow{\text{MP}} p_c(\mathbf{z}_{r_s}).$$

That is, get $\mathbb{S}(\mathbf{y}_{r_s})$ from $\mathbb{S}(\mathbf{y}_{r_m})$ by the **TermExpander** proposed in [HHPW22], then recover superpoly $p_c(\mathbf{z}_{r_s})$ of cube c via monomial prediction (MP) on $\mathbb{S}(\mathbf{y}_{r_s})$.

By the process of the variable substitution, we observe that for any $r_m > r_s$, $\mathbf{z}_{r_m}$ can be completely represented by $\mathbf{z}_{r_s}$. Specially, for $i = 0, \dots, |\mathbf{z}_{r_s}| - 1$, $\mathbf{z}_{r_s}[i] = \mathbf{z}_{r_m}[i]$. Therefore, we construct the coefficient solver **CoeSolver1**($\mathbb{S}(\mathbf{y}_{r_m})$, $\mathbf{c}, r_m, r_s$) with the following steps.

1. For all monomials in $\mathbb{S}(\mathbf{y}_{r_m})$, compute the superpoly $p'_c(\mathbf{z}_{r_m})$ using monomial prediction.
2. Generate the variable substitution function $\mathbf{g}$ such that $\mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s})$. Then compute the superpoly $p_c(\mathbf{z}_{r_s})$ through symbolic computation:

$$p_c(\mathbf{z}_{r_s}) = p'_c(\mathbf{z}_{r_m}) = p'_c(\mathbf{g}(\mathbf{z}_{r_s})).$$

The structure of this process is illustrated below:

$$\mathbb{S}(\mathbf{y}_{r_m}) \xrightarrow{\text{MP}} p'_c(\mathbf{z}_{r_m}) \xrightarrow{\text{SC}} p_c(\mathbf{z}_{r_s}).$$

To assess the computational efficiency of the two solvers, we define the time costs for their respective phases as t_{11} , t_{12} for their solver and t_{21} , t_{22} for our solver, with the total execution times denoted by T_1 and T_2 . Here, t_{22} represents the computational overhead attributed to symbolic computation (SC). Our experiments indicate that t_{22} accounts for less than 2% of the total superpoly recovery time, making its impact negligible when comparing the overall computational efficiency of both methods.

Take $\delta = r_m - r_s$ as the difference between 2 rounds. When δ is relatively small (e.g., $\delta < 30$), if the cardinality of $\mathbb{S}(\mathbf{y}_{r_m})$ is close to that of $\mathbb{S}(\mathbf{y}_{r_s})$, then the time required for monomial prediction to recover $p'_c(\mathbf{z}_{r_m})$ becomes approximately equal to the time needed to recover $p_c(\mathbf{z}_{r_s})$.

From this analysis, the ratio of total execution times for both phases can be expressed as

$$T_1/T_2 = (t_{11} + t_{12})/(t_{21} + t_{22}) \approx (t_{11} + \lambda t_{21})/t_{21},$$

where $\lambda = |\mathbb{S}(\mathbf{y}_{r_s})|/|\mathbb{S}(\mathbf{y}_{r_m})|$. Experiments show that $t_{11} > t_{21}$ and $\lambda > 1$, which implies that for small values of δ , the time ratio satisfies $T_1/T_2 > \lambda + 1$. This demonstrates that our coefficient solver achieves higher efficiency.

However, when δ is large (e.g., $\delta = 90$), directly applying monomial prediction (MP) to certain elements in $\mathbb{S}(\mathbf{y}_{r_m})$ requires significantly more computation time. In some cases, t_{21} may even far exceed t_{11} . To improve the efficiency of our solver **CoeSolver1** when handling large δ , inspired by [HST⁺21], we propose the nested coefficient solver **Nest-CoeSolver**($\mathbb{S}(\mathbf{y}_{r_m})$, $\mathbf{c}, r_m, r_s$).

To formalize this process, we first define a basic coefficient solver **CoeSolver0**($\mathbf{u}_{r_m}, \mathbf{c}, t_{r_m}, \mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s}), \mathbf{y}_{r_m}, q = 0$). Since when both the cube and the intermediate round number r_m are given, the variable substitution $\mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s})$ and the expression for each r_m -round state $\mathbf{y}_{r_m}(\mathbf{z}_{r_m}, \mathbf{v})$ are determined. Therefore, we choose to precompute these two components as inputs, which can save significant time when executing multiple **CoeSolver0** operations.

This solver returns 1 for monomials whose coefficients are successfully solved within the time limit t_{r_m} , and returns 0 for those that cannot be solved within the time limit. For a set $\mathbb{S}(\mathbf{y}_{r_m})$, we can divide this set into 2 parts, namely

$$\mathbb{S}(\mathbf{y}_{r_m}) = \mathbb{S}_0(\mathbf{y}_{r_m}) \cup \mathbb{S}_1(\mathbf{y}_{r_m}).$$

Where

$$\begin{aligned} \mathbb{S}_0(\mathbf{y}_{r_m}) &= \{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \mid \text{CoeSolver0}(\mathbf{u}_{r_m}, \mathbf{c}, t_{r_m}, \mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s}), \mathbf{y}_{r_m}, q = 0) = 0\}, \\ \mathbb{S}_1(\mathbf{y}_{r_m}) &= \{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \mid \text{CoeSolver0}(\mathbf{u}_{r_m}, \mathbf{c}, t_{r_m}, \mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s}), \mathbf{y}_{r_m}, q = 0) = 1\}. \end{aligned}$$

For $\mathbb{S}_1(\mathbf{y}_{r_m})$, we can get a partial superpoly

$$p_{\mathbf{c}}^{r_m}(\mathbf{z}_{r_s}) = \bigoplus_{\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}_1(\mathbf{y}_{r_m})} \text{Coe}(\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}), \mathbf{v}^{\mathbf{c}}).$$

For $\mathbb{S}_0(\mathbf{y}_{r_m})$, we can choose a round r_{l_1} close to r_m , such that $r_s \leq r_{l_1} < r_m$. Then we use **TermExpander** to calculate its valuable terms $\mathbb{S}(\mathbf{y}_{r_{l_1}})$ at round r_{l_1} . For $\mathbb{S}(\mathbf{y}_{r_{l_1}})$, we iteratively apply the above procedure until one of the following termination conditions is satisfied:

1. $\exists r_{l_i}$ such that $r_s < r_{l_i} < \dots < r_{l_1} < r_m$, $\mathbb{S}_0(\mathbf{y}_{r_{l_i}}) = \emptyset$. Then we can get the superpoly

$$p_{\mathbf{c}}(\mathbf{z}_{r_s}) = p_{\mathbf{c}}^{r_m}(\mathbf{z}_{r_s}) \oplus \bigoplus_{j=1}^i p_{\mathbf{c}}^{r_{l_j}}(\mathbf{z}_{r_s}).$$

2. $\exists r_{l_i}$ such that $r_s = r_{l_{i+1}} < r_{l_i} < \dots < r_{l_1} < r_m$, $\mathbb{S}_0(\mathbf{y}_{r_{l_i}}) \neq \emptyset$. For every $\pi_{\mathbf{u}_{r_s}}(\mathbf{y}_{r_s}) \in \mathbb{S}(\mathbf{y}_{r_s})$, choose $t_{r_s} = \infty$, calculate $\text{CoeSolver0}(\mathbf{u}_{r_s}, \mathbf{c}, \mathbf{z}_{r_m} = \mathbf{g}(\mathbf{z}_{r_s}), \mathbf{y}_{r_m}, t_{r_s}, q = 0)$, we can get

$$p_{\mathbf{c}}^{r_s}(\mathbf{z}_{r_s}) = \bigoplus_{\pi_{\mathbf{u}_{r_s}}(\mathbf{y}_{r_s}) \in \mathbb{S}(\mathbf{y}_{r_s})} \text{Coe}(\pi_{\mathbf{u}_{r_s}}(\mathbf{y}_{r_s}), \mathbf{v}^{\mathbf{c}}).$$

So the superpoly is

$$p_{\mathbf{c}}(\mathbf{z}_{r_s}) = p_{\mathbf{c}}^{r_m}(\mathbf{z}_{r_s}) \oplus \bigoplus_{j=1}^i p_{\mathbf{c}}^{r_{l_j}}(\mathbf{z}_{r_s}) \oplus p_{\mathbf{c}}^{r_s}(\mathbf{z}_{r_s}).$$

The nested framework is illustrated in Fig 1, with its corresponding algorithm **Nest-CoeSolver** ($\mathbf{A}, \mathbb{S}(\mathbf{y}_{r_m}), \mathbf{c}, r_m, r_s$) provided in Algorithm 6.

In fact, this nested solver works by setting a time limit to automatically identify which monomials cannot have their coefficients solved quickly. These tough monomials are then broken down into simpler monomials from lower rounds, making them easier to be solved. For monomials that can be solved within the time limit, our method saves time by avoiding unnecessary term expandings. Together, these improvements make our coefficient solver much more efficient.

In the following, we discuss the selection of parameters r_m , r_s , r_{l_j} , and time constraints $t_{r_{l_j}}$. Since our coefficient solver first computes the ANF of $y_{r_m}(\mathbf{z}_{r_s}, \mathbf{v})$ through symbolic computation, r_m cannot be too large. For Trivium, we select $r_m \leq 200$. The choice of round parameter r_s is determined by our subsequent superpoly utilization strategy. In this work, we set $r_s = 83$ or 93 to ensure the ANF of superpolies remains favorable for equation solving. Intermediate rounds r_{l_j} are determined by $r_{l_j} = \max(r_s, r_{l_{j-1}} - 30)$ and the time limit $t_{r_{l_j}}$ are set to be 1 second.

To demonstrate the advantages of our coefficient solver, we selected several cubes for comparative experiments. Compared with the current fastest method proposed by

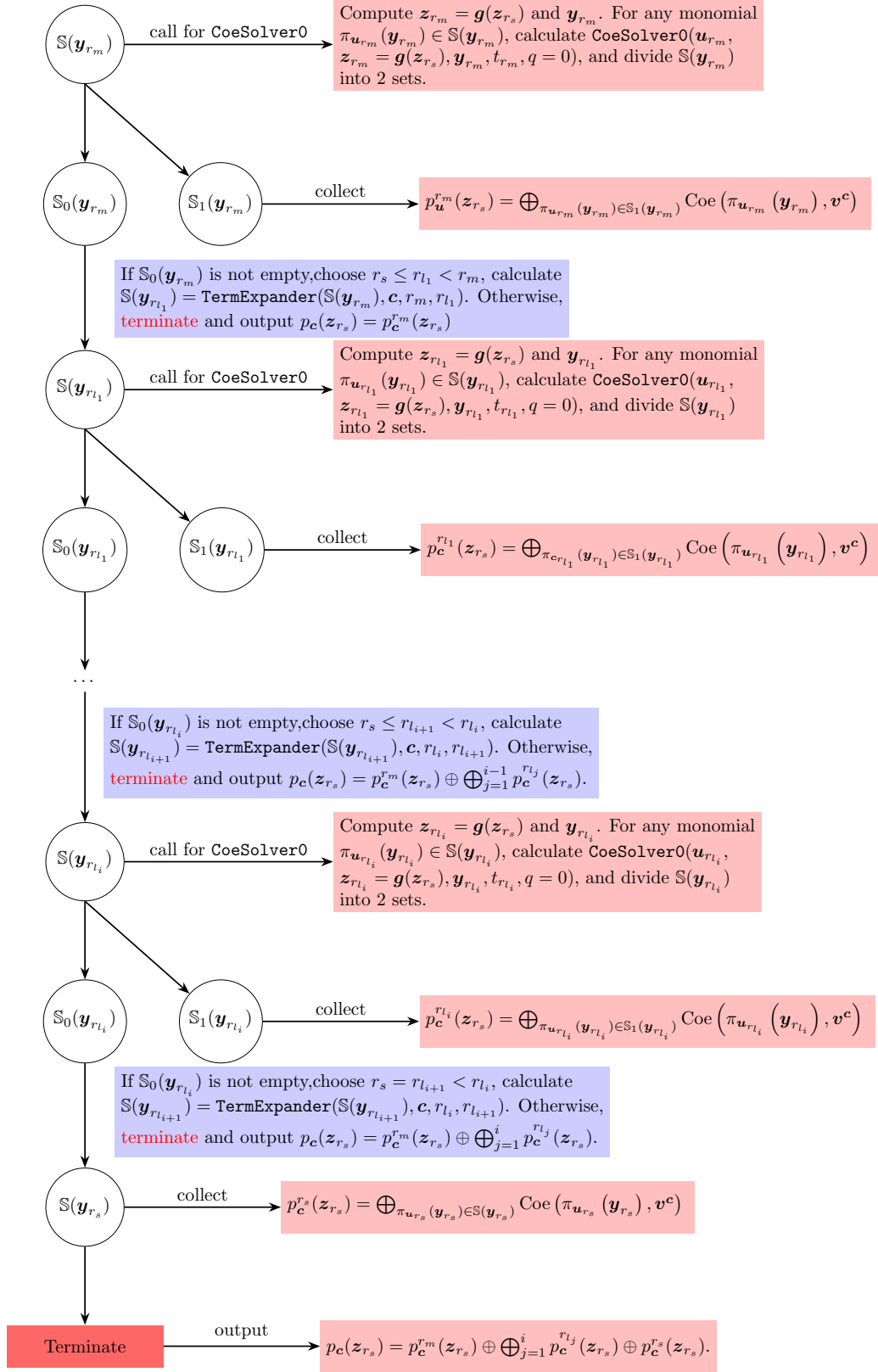


Figure 1: The procedure of nested coefficient solver

He *et al.* [HHLW24] at Eurocrypt 2024, the solving speed of our method is 4-11 times faster than that of theirs, while the coefficient solver reaches speeds about 4-37 times greater. Our solver achieves superior performance as the superpolies become more complex. Furthermore, the size of superpoly is significantly reduced after variable substitution. In particular, the size of superpoly with cube I_3 is reduced by a factor of more than 10000 after variable substitution, which will greatly accelerates the subsequent equation-solving phase. The cube indices are listed at Table 5.

Table 2: Comparison of different algorithms with same cubes

Round	Cube	Alg.	#monomials	#Coe time(s)*	#Total time(s)
840	I_1	[HHLW24]	6963	37	651
		Our method	122	9	100
840	I_2	[HHLW24]	11179848	281	412
		Our method	2976	26	98
840	I_3	[HHLW24]	19544913	2524	2728
		Our method	1853	68	247
843	I_1	[HHLW24]	359466	541	1240
		Our method	2336	58	260

* The coefficient solver in [HHLW24] can be equivalently regarded as the phase after the forward expansion begins.

5 Cube Searching

In this section, we introduce an automated cube search algorithm. Our approach extends the mother cube construction framework introduced by Ye *et al.* at Asiacrypt 2021 [YT21]. Their method comprises three steps:

1. **Initial cube selection:** Identifying candidate cubes for next step.
2. **Mother cube construction:** Iteratively adding variables to the initial cube to construct a larger cube.
3. **Subcube derivation:** Generating subcubes by removing variables from the mother cubes.

Subsequent studies [CT22, LHHW23] improved specific techniques within this framework, enabling the detection of superpolies with simple forms in lower-round Trivium (e.g., 820-round and 825-round). However, these methods face computational limitations for higher rounds (e.g., 840-round and 845-round). To overcome these challenges, we develop significant improvements to each component of the original framework. Our methods enable more efficient searching for effective cubes in higher rounds.

5.1 Selection of Initial Cubes

Previously, initial cube selection relied on identifying cubes with simple superpolies in early rounds. However, this process lacked a systematic algorithmic approach, relying on heuristic testing. Such manual selection becomes increasingly impractical for higher rounds. To overcome this technical bottleneck, we propose an automated initial cube search algorithm based on numeric mapping. We begin by presenting the property of the numeric mapping.

Property 2. Given a cube c , for any subcube $c' \subseteq c$, $\text{DEG}_{c'}(f) \leq \text{DEG}_c(f)$, where f is a Boolean function.

Based on this property, we develop the following strategy for initial cube selection. Given the cube dimension m and threshold D , the algorithm outputs an m -dimensional cube set $\mathcal{C}$ such that for any $c \in \mathcal{C}$, $\text{DEG}_c(f) \leq D$. The procedure is as follows:

1. Initialize a candidate m -dimensional cube set $\mathcal{I}$ and an empty set $\mathcal{C} = \emptyset$.
2. Randomly select a cube $c \in \mathcal{I}$ and compute $\text{DEG}_c(f)$. If $\text{DEG}_c(f) \leq D$, add c to $\mathcal{C}$, remove c from $\mathcal{I}$, and proceed to Step 4. Otherwise, proceed to Step 3.
3. Generate c' by randomly removing one variable from c and compute $\text{DEG}_{c'}(f)$. If $\text{DEG}_{c'}(f) \leq D$, remove all cubes containing c from $\mathcal{I}$ and proceed to Step 4. Otherwise, set $c \leftarrow c'$ and repeat Step 3.
4. If $\mathcal{I} \neq \emptyset$, go to Step 2. Otherwise, output the final set $\mathcal{C}$.

The core idea of this algorithm is to iteratively identify subcubes with both small dimension and high numeric degree for any given cube. By removing all mother cubes containing these bad subcubes from the candidate set, this approach significantly reduces the verification process for undesirable cubes.

Furthermore, by Property 2, this algorithm can precisely select all cubes with their numeric degree lower than D from the candidate set. Compared to exhaustively computing all cubes in $\mathcal{I}$, this algorithm can significantly reduce computational effort. For instance, when we apply it to 840-round Trivium with a candidate set of approximately 2^{30} 36-dimensional cubes and threshold $D = 36$, the algorithm can obtain the qualified cube set $\mathcal{C}$ within 571 seconds, whereas an exhaustively approach would require 597 hours ($2^{30} \times 0.002 \approx 2147484$ seconds). It should be explicitly noted that this algorithm requires pre-selection of a large candidate cube set. For example, in our attack on the 845-round algorithm, we select non-adjacent cube sets with dimension 37 as candidate cube set, where the scale of this set is 2^{25} .

5.2 Selection of Mother Cubes and Subcubes

To search for cubes, we need to estimate the complexity of superpolies. We observe that the existing methods including Möbius transform, division property, and superpoly recovery, are all computationally intensive. Therefore, we choose to calculate the number of valuable terms (VT_{r_m}) at an intermediate round r_m to estimate the complexity of superpoly, which is a more efficient way. Generally, cubes with smaller VT_{r_m} values yield simpler superpolies. To construct a mother cube, we propose Algorithm 1. Using this method, we can get a

Algorithm 1 Mother Cube Construction

Input: initial cube c , round r_m , threshold M , dimension d

Output: a cube with dimension $\leq d$ and $VT_{r_m} \leq M$

- 1: **while** dimension of $c < d$ **do**
 - 2: For each non-cube variable v_i , compute VT_{r_m} of $c \cup \{v_i\}$ via **TermExpander** and select c_1 with minimal $VT_{r_m}^{c_1}$
 - 3: **if** $VT_{r_m}^{c_1} \leq M$ **or** $VT_{r_m}^{c_1} \leq VT_{r_m}^c$ **then**
 - 4: **if** $\dim(c_1) = d$ **and** $VT_{r_m}^{c_1} \leq M$ **then**
 - 5: **return** c_1
 - 6: **else**
 - 7: $c \leftarrow c_1$
 - 8: **end if**
 - 9: **else**
 - 10: **return** c
 - 11: **end if**
 - 12: **end while**
-

mother cube with dimension $\leq d$ and $VT_{r_m} \leq M$.

To select subcubes, while previous methods employed an exhaustive search (verifying all m possible $(m - 1)$ -dimensional subcubes for an m -dimensional cube), would consume a large number of unnecessary computing resources. We introduce the concept of *fixed subcube* to reduce these unnecessary computations. That is, given a mother cube $c = \{v_{i_0}, \dots, v_{i_t}\}$, we select $c_f \subseteq c$ whose elements remain fixed during the subcube selection, where c_f is the fixed subcube of c . The subcube selection algorithm is described in Algorithm 12.

In particular, the selection of fixed subcubes can be accomplished through the following two strategies:

- **Initial-Cube Preservation:** For a constructed mother cube, we select the initial cube used in its construction as the fixed subcube. This approach maintains the core structure that leads to the mother cube's favorable properties.
- **Isolated Variable Preservation:** We identify all isolated variables within the cube as fixed elements. Formally, for a cube $c = \{v_{i_0}, \dots, v_{i_t}\}$, a variable $v_j \in c$ is considered *isolated* if it satisfies:

$$v_{j-1} \notin c \text{ and } v_{j+1} \notin c \quad (7)$$

Remark 2. The second strategy is specific to Trivium, as its round function involves the multiplication of two adjacent state bits. Moreover, the first strategy represents a more aggressive approach for Trivium, preserving the complete initial cube structure. In practical applications, we select the first strategy for high-dimension cubes, second strategy for low-dimension cubes.

6 New Testing Method

In this section, we propose two methods for solving a complex equation system.

6.1 Improved Testing Method

Recently, Zhang *et al.* [ZS24] proposed a *testing method* to solve the equations of numerous superpolies. The core idea involves iteratively guessing a single key bit ($k_i = b$) and verifying the guesses by adding them into the equation system and solving it. If the system has no solution, then one can determine that $k_i = b \oplus 1$. However, this method requires that the ANF of the superpolies cannot be too complex, and it can only verify equations of the form $k_i = b$. It fails to extract information from low-degree equations involving key variables. For example, consider the equation

$$(k_1 \oplus k_2)(k_3 \oplus k_4) = 1. \quad (8)$$

If we follow their approach by sequentially guessing the values of $k_1, \dots, k_4$, we would fail to recover any information. However, if we rewrite this equation as

$$\begin{aligned} g_1 g_2 &= 1 \\ g_1 &= k_1 \oplus k_2 \\ g_2 &= k_3 \oplus k_4 \end{aligned}$$

Then we can guess the values of g_1 and g_2 to derive the equations $k_1 \oplus k_2 = 1$ and $k_3 \oplus k_4 = 1$. Since these two equations are linearly independent, we only need to enumerate the possible values of k_1 and k_2 to recover $\mathbf{k}$. Therefore, this method allows us to extract 2 bits of key information from Equation (8).

Furthermore, previous methods did not utilize cases where solutions exist. However, we can utilize these cases by the following procedure. More precisely, for a system of superpoly equations $\mathbb{E}$, we first add the condition $k_0 = b_0$, if the system has a solution, we obtain a

solution of $\mathbf{k}$, denoted by $\mathbf{b}$; if the system has no solution, we can obtain the solution by adding the condition $k_0 = b_0 \oplus 1$. That is, we get a record $\mathbf{b}$ in which each variable with the value $k_i = b_i$ satisfying that $\mathbb{E} \cup \{k_i = b_i\}$ is solvable for each i . After obtaining $\mathbf{b}$, we only need to check whether $\mathbb{E} \cup \{k_i = b_i \oplus 1\}$ has a solution in subsequent guesses, which reduces the call times of solving the equation system. Furthermore, after several attempts at solving the equations, e.g., for Trivium, we observed the following phenomenon:

Observation 1. *Variables that appear frequently in the equations likely have unique values in the solutions.*

Based on the above discussion, we present the following *improved testing method* for superpolies, where $\mathbf{z}'_{r_s}$ is a permutation of $\mathbf{z}_{r_s}$, ordered by descending frequency of appearance in the equation system $\mathbb{E}$.

Algorithm 2 Improved testing method for superpolies

Input: Superpoly equations $\mathbb{E} = \{p_0(\mathbf{z}'_{r_s}) = b_0, \dots, p_m(\mathbf{z}'_{r_s}) = b_m\}$, where $b_i \in \{0, 1\}$. A variable substitution function $\mathbf{z}'_{r_s} = \mathbf{g}'(\mathbf{k})$.

Output: A equation system to $\mathbf{z}$, e.g., $\{\mathbf{z}'_{r_s}[j_1] = 1, \dots, \mathbf{z}'_{r_s}[j_w] = 1\}$.

- 1: $R \leftarrow \emptyset$ ▷ A set of equations recording the determined $\mathbf{z}'_{r_s}[i]$.
 - 2: $U \leftarrow \{\mathbf{z}'_{r_s}[0], \dots, \mathbf{z}'_{r_s}[t]\}$ ▷ The set containing all variables
 - 3: $\mathbb{G} = \bigcup_{i=0}^t \{\mathbf{z}'_{r_s}[i] = \mathbf{g}'(\mathbf{k})[i]\}$
 - 4: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \{\mathbf{z}'_{r_s}[0] = 0\}$ by a solver
 - 5: **if** there is no solution **then**
 - 6: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \{\mathbf{z}'_{r_s}[0] = 1\}$ by a solver
 - 7: Record the solution values as $\mathbf{b}$.
 - 8: **else**
 - 9: Record the solution values as $\mathbf{b}$.
 - 10: **end if**
 - 11: **for** $\mathbf{z}'_{r_s}[j] \in U$ **do**
 - 12: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \{\mathbf{z}'_{r_s}[j] = \mathbf{b}[j] \oplus 1\}$ by a solver
 - 13: **if** there is no solution **then**
 - 14: $R \leftarrow R \cup \{\mathbf{z}'_{r_s}[j] = \mathbf{b}[j]\}$
 - 15: **end if**
 - 16: **end for**
-

6.2 Aggressive Testing Method

Clearly, the above method can accurately recover deterministic equations. However, for more complex equation systems, though the method remains precise, its solving speed decreases significantly. In some cases, it takes more than one day on an ordinary PC to verify a guess. After several attempts at solving the equations generated by conditional superpolies we recovered, we have the following observations.

Observation 2. *For more complex equations, finding a solution is significantly faster than proving the system is unsolvable.*

Observation 3. *The solving time decreases substantially after assigning values to high-frequency variables in the equation system.*

Thus, by sacrificing some accuracy, we propose the *aggressive testing method* (Algorithm 3) to enable efficient solving of highly complex superpoly equation systems.

Detailed explanation of the Algorithm 3. In fact, the equations added to the solution set R in the second phase are not necessarily correct theoretically. In Algorithm 3,

$\mathbb{B}$ represents the set of variables that appear with high frequency in the equation system, we can partition $\mathbb{B}$ into two subsets:

$$\begin{aligned}\mathbb{B}_1 &= \{z'_{r_s}[i] \in \mathbb{B} \mid \mathbb{E} \cup \mathbb{G} \cup \{z'_{r_s}[i] = \mathbf{b}[i] \oplus 1\} \text{ is unsolvable}\}, \\ \mathbb{B}_2 &= \{z'_{r_s}[i] \in \mathbb{B} \mid \mathbb{E} \cup \mathbb{G} \cup \{z'_{r_s}[i] = \mathbf{b}[i] \oplus 1\} \text{ is solvable}\}.\end{aligned}$$

Obviously, the elements in subset $\mathbb{B}_1$ can be exactly identified by our algorithm. And for any element $z'_{r_s}[i] \in \mathbb{B}_2$, the equation system

$$\mathbb{E} \cup \mathbb{G} \cup \{z'_{r_s}[i] = \mathbf{b}[i] \oplus 1\} \cup \bigcup_{\substack{j \neq i \\ z'_{r_s}[j] \in \mathbb{B}}} \{z'_{r_s}[j] = \mathbf{b}_l[j]\} \quad (9)$$

is equal to

$$\mathbb{E} \cup \mathbb{G} \cup \{z'_{r_s}[i] = \mathbf{b}[i] \oplus 1\} \cup \bigcup_{\substack{j \neq i \\ z'_{r_s}[j] \in \mathbb{B}_2}} \{z'_{r_s}[j] = \mathbf{b}_l[j]\}. \quad (10)$$

We can then proceed to partition $\mathbb{B}_2$ into the following two subsets:

$$\begin{aligned}\mathbb{B}_2^1 &= \{z'_{r_s}[i] \in \mathbb{B}_2 \mid (10) \text{ is solvable}\}, \\ \mathbb{B}_2^2 &= \{z'_{r_s}[i] \in \mathbb{B}_2 \mid (10) \text{ is unsolvable}\}.\end{aligned}$$

Elements in $\mathbb{B}_2^1$ will be correctly removed in the algorithm, while elements in $\mathbb{B}_2^2$ will be wrongly added to R , which is the reason why this method is inaccurate.

However, our experiments show that for large-scale systems (e.g., with over 500 equations), $\mathbb{B}_1$ constitutes the majority of cases. Moreover, within $\mathbb{B}_2$, $\mathbb{B}_2^1$ dominates overwhelmingly. This occurs because high-frequency variables appear in nearly all equations, making their solutions highly likely to be unique. When uniqueness fails, it indicates that these variables may have been eliminated during equation simplification, resulting in low correlation with other variables. Thus, the existence of solutions to Equation (10) is logically equivalent to the existence of solutions to $\mathbb{E} \cup \mathbb{G} \cup \{z'_{r_s}[i] = \mathbf{b}[i] \oplus 1\}$, which means that $\mathbb{B}_2^2 = \emptyset$. In fact, for almost all tested equation systems in our experiments, $\mathbb{B}_2^2 = \emptyset$. This demonstrates that our algorithm achieves significant efficiency improvements at the cost of negligible accuracy loss. In subsequent attacks, we will accordingly select distinct equation-solving algorithms based on the complexity level of the equation systems.

To solve the superpoly equations, we employ the CryptoMiniSat solver [SNC09]. In fact, our improved algorithm not only yields low-degree equations of secret key, but also make the superpoly equation system more tractable. This is because the variable substitutions performed during superpoly recovery significantly reduce the scale of superpolies, thereby simplifying the corresponding CNF (conjunctive normal form) of the equations and consequently accelerating the solving process.

7 Applications to Trivium

In this section, we conduct attacks on 840-, 845-, and 855-round Trivium using distinct approaches. Experiments are conducted using the Gurobi Solver (version 10.0.1) on a workstation equipped with high-speed processors (totally 256 CPU cores with 512 threads).

7.1 Practical Attacks on 840-Round Trivium

Selection of Cubes and Calculation of superpolies. For the initial cube sets, we selected cubes of dimensions 36 and 37 that contain no adjacent indexes. The number of such cubes is $\sum_{n=36}^{37} \binom{81-n}{n} \approx 2^{30}$. For dimension $36 \leq m \leq 37$, setting parameters

Algorithm 3 Aggressive Testing Method

Input: Superpoly equations $\mathbb{E} = \{p_0(\mathbf{z}'_{r_s}) = b_0, \dots, p_m(\mathbf{z}'_{r_s}) = b_m\}$, where $b_i \in \{0, 1\}$. A variable substitution function $\mathbf{z}'_{r_s} = \mathbf{g}'(\mathbf{k})$. Integer l, n . Time limit T_0, T_1 .

Output: A equation system to $\mathbf{z}$, e.g., $\{\mathbf{z}'_{r_s}[j_1] = 1, \dots, \mathbf{z}'_{r_s}[j_w] = 1\}$.

- 1: $R \leftarrow \emptyset$ ▷ A set of equations recording the determined $\mathbf{z}'_{r_s}[i]$.
- 2: $U \leftarrow \{\mathbf{z}'_{r_s}[0], \dots, \mathbf{z}'_{r_s}[t]\}$ ▷ The set containing all variables
- 3: $\mathbb{G} = \bigcup_{i=0}^t \{\mathbf{z}'_{r_s}[i] = \mathbf{g}'(\mathbf{k})[i]\}$
- 4: $\mathbb{A} = \{\mathbf{z}'_{r_s}[0], \dots, \mathbf{z}'_{r_s}[l-1]\}$
- 5: $\mathbb{B} = \{\mathbf{z}'_{r_s}[0], \dots, \mathbf{z}'_{r_s}[n-1]\}$
- 6: **Phase 1: Find an initial solution by enumerating all the possible values of variables in $\mathbb{A}$**
- 7: **for** $\mathbf{b}_l \in \{0, 1\}^l$ **do**
- 8: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \bigcup_{i=0}^{l-1} \{\mathbf{z}'_{r_s}[i] = \mathbf{b}_l[i]\}$ by a solver
- 9: **if** there is a solution under the time limit T_0 **then**
- 10: Record the solution values as $\mathbf{b}$.
- 11: **break**
- 12: **end if**
- 13: **end for**
- 14: **if** there is no solution for all $\mathbf{b}_l \in \{0, 1\}^l$ **then**
- 15: **exit**
- 16: **end if**
- 17: **Phase 2: Identify fixed equations**
- 18: **for** $i < n$ **do**
- 19: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \{\mathbf{z}'_{r_s}[i] = \mathbf{b}[i] \oplus 1\} \cup \bigcup_{\mathbf{z}'_{r_s}[j] \in \mathbb{B}}^{j \neq i} \{\mathbf{z}'_{r_s}[j] = \mathbf{b}_l[j]\}$ by a solver
- 20: **if** there is no solution under the time limit T_1 **then**
- 21: $R \leftarrow R \cup \{\mathbf{z}'_{r_s}[i] = \mathbf{b}[i]\}$
- 22: **else**
- 23: $\mathbb{B} \leftarrow \mathbb{B} \setminus \{\mathbf{z}'_{r_s}[i]\}$
- 24: **end if**
- 25: **end for**
- 26: **Phase 3: Solve remaining variables**
- 27: **for** $\mathbf{z}'_{r_s}[j] \in U$ **do**
- 28: Solve the system $\mathbb{E} \cup \mathbb{G} \cup R \cup \{\mathbf{z}'_{r_s}[j] = \mathbf{b}[j] \oplus 1\}$ by a solver
- 29: **if** there is no solution **then**
- 30: $R \leftarrow R \cup \{\mathbf{z}'_{r_s}[j] = \mathbf{b}[j]\}$
- 31: **end if**
- 32: **end for**

$T = m$ in Algorithm 11 and set the output function in numeric mapping $f = s_{65}$, we obtained 101 cubes. From these cubes, we select a 37-dimension cube as the initial cube. Let the dimension upper bound of the mother cube be 58 and 54, the intermediate round $r_m = 510$, and the upper bound of VT_{r_m} be 17. Using Algorithm 1, we construct 2 mother cubes. With the minimum subcube dimension set to 52 and 46, upper bound of VT_{r_m} be 19, we obtain 2653 and 817 subcubes via Algorithm 12. Choose the middle round $r_m = 180$, the output variable round $r_s = 83$, time limit $t = 1$ second, intermediate rounds $r_{l_j} = \max(r_s, r_{l_{j-1}} - 30)$. Our superpoly recovery algorithm can successfully recover them within 2 days. The recovered superpolies are actually expressed in terms of variable $\mathbf{z}_{83} = \{z_0, \dots, z_{152}\}$. For the key variables $\mathbf{k} = \{k_0, \dots, k_{79}\}$, the structure of $\mathbf{z}_{83}$ can be

characterized as follows:

$$z_i = g_i(\mathbf{k}) = \begin{cases} k_i & i = 0, \dots, 79 \\ k_{147-i} \oplus 1 & i = 80, 81 \\ k_{53} \oplus k_{78}k_{79} & i = 82 \\ k_{135-i} \oplus k_{160-i}k_{161-i} \oplus k_{162-i} & i = 83, \dots, 135 \\ k_{24}k_{25} \oplus k_{26} \oplus k_{68} & i = 136 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} \oplus 1 & i = 137, 138 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} & i = 139, \dots, 147 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} \oplus k_{213-i} & i = 148, \dots, 152. \end{cases} \quad (11)$$

Key Recovery in the Online Phase. In the online phase, for a given key, we first compute the cube sums to get values of the superpolies. Take the 58-dimensional mother cube as an example. Let these values be denoted as $\mathbf{b} = (b_0, \dots, b_{2652})$. This allows us to establish the following system of equations in the secret key:

$$\mathbb{E}_{\mathcal{P}} = \bigcup_{j=0}^{2652} \{p_j(\mathbf{z}_{83}) = b_j\} \cup \bigcup_{i=0}^{152} \{z_i + g_i(\mathbf{k}) = 0\}. \quad (12)$$

The subsequent step is employing Algorithm 2 to solve $\mathbb{E}_{\mathcal{P}}$. Since exhaustively testing whether the systems are solvable for all keys is computationally infeasible, we instead conduct our evaluation using a randomly generated a set of thousands of keys to assess the effectiveness of the attack.

For a random key, after solving $\mathbb{E}_{\mathcal{P}}$, we obtain a set of solutions for $\mathbf{z}_{83}[j]$. Since each $\mathbf{z}_{83}[j]$ is a balanced polynomial with few monomials, we can employ a guess-and-determine approach to enumerate keys using these equations. To calculate the number of key guesses required, we utilize an automated guessing tool proposed in [HE22]. By inputting the equation system, this tool computes the minimal number of key guesses needed, we denote it as Gk_e . Let V be the number of variables in the equation system. The number of determined variables is $D = V - Gk_e$. Thus, the total number of guessed variables is $Gk = L - D$, where L represents the key length. Thus, after solving the equations, the number of keys to be enumerated is 2^{Gk} .

In our experiments, using the 58-dimensional mother cube, for **98.9%** of the tested keys, the recovered equation system contains at least 22 determined variables. This allows the key to be fully recovered with at most 2^{58} enumerations—a computationally feasible task in practical time. So the time complexity for calculating cube sum and enumeration is estimated at $2^{58} + 2^{58} = 2^{59}$. Notably, **56.8%** of these keys can be recovered directly, without requiring any key enumeration. Although our evaluation covers only a fraction of the total key space, the results strongly suggest that our novel attack method holds substantial practical relevance for recovering randomly generated keys. Furthermore, using the 54-dimensional mother cube, **77.8%** of the tested keys can be completely recovered with at most 2^{54} enumerations.

Specifically, our improved testing methods can be executed efficiently. The average execution time for solving the equation system generated by the 54-dimensional mother cube is approximately 250 seconds on a single thread. Compared to the time for computing cube sum and enumeration, this becomes negligible. Therefore, by using the 54-dimensional mother cube, we can achieve a practical attack on 840-round Trivium with success rate **77.8%** at a complexity below 2^{55} . In Appendix A, we provide an example that details the attack process.

7.2 Practical Attacks on 845-Round Trivium

Selection of Cubes and Calculation of Conditional Superpolies. For the initial cube sets, we selected cubes of dimension 37 that contain no adjacent indexes. The number of such cubes is $\binom{44}{37} \approx 2^{25}$. Setting parameters $T = 40$ in Algorithm 11, we obtained 58 cubes. From these cubes, we select 33 cubes as the initial cubes. We set the upper bound for the mother cube dimension to 54, the intermediate round to $r_m = 515$, and the upper bound of VT_{r_m} to 120. Using Algorithm 1, we constructed a total of **13** mother cubes with dimension from 49 to 54. With the minimum subcube dimension set to 41, the upper bound of VT_{r_m} be 45, we obtain 11870 subcubes.

After randomly selecting 50 cubes with small VT_{r_m} obtained above, we use the method described in Section 3.3, and configured Algorithm 14 with parameters: $r_{b_0} = 250$, $r_{b_1} = 185$, $n = 2$, $m = 10$, $s = 3$, $l = 4$, $T = 120$, $T_1 = 80$, $T_0 = 20$, this process ultimately yielded 9 eliminable sets

$$\begin{aligned} \mathcal{Z}_0 &= \{z_{54}, z_{56}, z_{58}\}, \mathcal{Z}_1 = \{z_{55}, z_{57}, z_{82}\}, \mathcal{Z}_2 = \{z_{90}, z_{103}, z_{105}\}, \\ \mathcal{Z}_3 &= \{z_{100}, z_{113}, z_{115}\}, \mathcal{Z}_4 = \{z_{114}, z_{116}, z_{118}\}, \mathcal{Z}_5 = \{z_{90}, z_{114}, z_{116}\}, \\ \mathcal{Z}_6 &= \{z_{90}, z_{116}, z_{118}\}, \mathcal{Z}_7 = \{z_{54}, z_{56}, z_{100}\}, \mathcal{Z}_8 = \{z_{88}, z_{90}, z_{103}\} \end{aligned}$$

with each set yielding over 550 cubes. Choose the middle round $r_m = 185$, output round $r_s = 93$, condition round $r_c = 69$, a time limit $t = 1$ second, intermediate rounds $r_{l_j} = \max(r_s, r_{l_{j-1}} - 30)$. Using these parameters, our conditional superpoly recovery algorithm successfully recovered the superpolies within two days. The recovered superpolies are expressed in terms of variable $\mathbf{z}_{93} = \{z_0, \dots, z_{172}\}$. The relationship between the key variables $\mathbf{k} = (k_0, \dots, k_{79})$ and the state $\mathbf{z}_{93}$ is provided in Appendix J.

Key Recovery in the Online Phase. In the online phase, we first get the value of superpolies of $\mathcal{C}_{\mathcal{Z}_0}, \dots, \mathcal{C}_{\mathcal{Z}_8}$ by calculating the cube sums. Since all cubes in these sets are contained within those 13 mother cubes, the complexity of calculating the cube sums is

$$2^{54} + 2^{53} + 2 \times 2^{52} + 4 \times 2^{51} + 2^{50} + 4 \times 2^{49} = 2^{55} + 2^{53} + 2^{51} + 2^{50} \approx 2^{55.43}.$$

Then we assign the values of these superpolies to their corresponding conditional superpolies, and establish equation systems for each conditional polynomial set to solve.

As an example, consider the set $\mathcal{Z}_0 = \{z_{54}, z_{56}, z_{58}\}$. The size of $\mathcal{C}_{\mathcal{Z}_0}$ is 752. Let these values be denoted as $\mathbf{b} = (b_0, \dots, b_{751})$. This allows us to establish the following system of equations concerning the secret key:

$$\mathbb{E}_{\mathcal{P}_{\mathcal{Z}_0}} = \bigcup_{j=0}^{751} \{p_j(\mathbf{z}_{93}) = b_j\} \cup \bigcup_{i=0}^{172} \{z_i + g_i(\mathbf{k}) = 0\} \cup \{z_{54} = z_{56} = z_{58} = 0\}. \quad (13)$$

The subsequent step is employing the aggressive testing method to solve $\mathbb{E}_{\mathcal{P}_{\mathcal{Z}_0}}, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_1}}, \dots, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_8}}$. For $\mathbb{E}_{\mathcal{P}_{\mathcal{Z}_0}}, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_3}}, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_4}}, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_5}}, \mathbb{E}_{\mathcal{P}_{\mathcal{Z}_6}}$, we set the parameter T_0 in Algorithm 3 as 100 seconds. For other systems of equations, T_0 is set to be 1000 seconds. For all systems of equations, we take $l = 4$, $n = 10$ and $T_1 = 100$ seconds. Given that exhaustively testing whether the systems are solvable for all keys is computationally infeasible, we instead conduct our evaluation using a randomly generated set of 1000 keys to assess the effectiveness of the attack.

In practice, for keys belonging to a certain weak-key space, the value of their conditional superpoly equals that of the superpoly. Therefore, we can derive the equations to be solved by assigning key values in the conditional superpoly, thereby simulating the cube summation recovery process. However, for keys not belonging to a certain weak-key space, the value of the conditional superpoly does not match that of the superpoly. In this case, we can only assign values to the conditional superpoly by computing cube sums—a highly time-consuming process. Consequently, simulating this scenario for 1000 keys is

impractical. Nevertheless, we can adopt the following implementation strategy to impose time constraints when solving such equations.

For a random key $k \in \bigcup_{i=0}^8 \text{Zero}(\mathcal{Z}_i)$, without loss of generality, suppose $k \in \bigcap_{i=0}^t \text{Zero}(\mathcal{Z}_i)$ with $8 \geq t \geq 1$. We can use the aggressive testing method to solve the 9 systems of equations in parallel. In order to make the total time complexity less than 2^{56} , the time complexity for enumeration should be less than $2^{54} + 2^{52} = 10 \times 2^{51}$. So when solving the equations, we stop the process once 33 $z_{93}[j]$ values are recovered. Let T_i denote the time required to solve these 9 equation systems until 33 $z_{93}[j]$ values are recovered, with $T_{e_0} < T_{e_1} < \dots < T_{e_8}$, where $e_0, \dots, e_8$ are permutation of $0, \dots, 8$. Here, we define a time bound T : if the time for equations solving reaches this bound with recovering less than 33 $z_{93}[j]$ values, we abort the process and count the number of determined variables. If this count exceeds 28, we will continue to perform enumerations, otherwise, we will not perform enumerations. Then the time complexity will be $2^{51} \times d + T_{e_d}$, where d is the index such that e_d is the first index which is $\leq t$. In other words, $T_{e_d} = \min_{i \leq t} \{T_i\}$. For this scenario, the upper bound of time complexity is $2^{51} \times 9 + \min_{i \leq t} \{T_i\}$. To satisfy $2^{51} \times 9 + \min_{i \leq t} \{T_i\} < 10 \times 2^{51}$, we select T as the operation time for 2^{51} executions of 845-round Trivium. In [LHHW23], the authors used a PC equipped with two RTX3090 GPUs to perform 2^{53} iterations of 825-round Trivium operations over two weeks. By extension, we can equate the computation time for 2^{51} iterations to 84 hours.

After the above analysis, we only need to calculate the corresponding equation-solving time for each key in the weak-key space to obtain a strict upper bound on the time complexity. In our experiments, **59.1%** of the 1000 randomly selected keys fall within the key space $\bigcup_{i=0}^8 \text{Zero}(\mathcal{Z}_i)$. Among these keys, the average execution time for equation solving is approximately 2 hours using 27 threads, the upper bound of $\min_{i \leq t} \{T_i\}$ is 15 hours, which is far less than 84 hours. Specially, using aggressive testing method, we can recover equation systems containing at least 29 determined variables for **98.1%** of these keys. Therefore, we can achieve a practical attack on 845-round Trivium with success rate 98.1% at a time complexity below 2^{56} for $2^{80} \times 59.1\%$ keys. More comprehensive statistical analysis of this attack is provided in Appendix B.

7.3 Cube Attacks on 855-Round Trivium

In this section, we give a weak-key attack on 855-round Trivium by recovering a single conditional superpoly with 6 conditions.

Selection of Cube and Calculation of Conditional Superpoly. For the initial cube sets, we selected cubes of dimension 37 that contain no adjacent indices. By setting parameters $T = 43$ in Algorithm 11, we obtained 16 cubes. From these cubes, we randomly select one cube as the initial cube. We then set the upper bound for the mother cube's dimension to 45, the intermediate round to $r_m = 515$, and the upper bound of VT_{r_m} to 210. Using Algorithm 1, we constructed one 45-dimensional cube. Next, we configured Algorithm 14 with parameter $r_{b_0} = 250$, $r_{b_1} = 185$, $n = 2$, $m = 10$, $s = 6$, $l = 4$, $T = 120$, $T_1 = 80$, $T_0 = 20$. This configuration yielded an eliminable set $z_{61}, z_{63}, z_{100}, z_{102}, z_{124}, z_{126}$, resulting in the 6 conditions:

$$z_{61} = z_{63} = z_{100} = z_{102} = z_{124} = z_{126} = 0.$$

Choose the middle round $r_m = 225$, output variable round $r_s = 83$, condition round $r_c = 69$, time limit $t = \infty$, intermediate rounds $r_{l_j} = \max(r_s, r_{l_{j-1}} - 30)$. Our conditional superpoly recovery algorithm recovered the corrected conditional superpoly within 37 hours.¹ The final

¹We would like to thank Jiahui He for pointing out that the source code used to obtain the previously published 855-round result allocated space for only 880 of the 882 auxiliary z -variables, yielding an incorrect superpoly. The array size has been corrected, and the 855-round results reported in this paper have been recomputed accordingly.

result includes the contributions recovered from previously failed coefficient computations. The recovered superpoly is expressed in terms of variables $z_{83} = \{z_0, \dots, z_{152}\}$ and contains 1233115 monomials after combining like terms. Substitution into the physical key variables $\mathbf{k}$ produces 4162761127 monomial occurrences before combining like terms. Further details are provided in Table 3.

To validate the correctness of this superpoly, we compared its values with the exact cube sums computed using a RTX4090 GPU for 50 random weak keys. All 50 comparisons matched.

Table 3: Details of the 855-round conditional superpoly

Round	Cube	#conditions	#monomials	#Key variables	Balance
855	I_4	6	4162761127	78	0.49981

* The balance is estimated from 100,000 distinct weak keys; 49,981 evaluations returned 1 and 50,019 returned 0.

* The monomial and key-variable counts refer to the unmerged expansion in $\mathbf{k}$, before applying the six physical-key constraints.

Key Recovery. To recover key information from this approximately balanced conditional superpoly, we adopted the method described in [HHLW24]. First, we represent this polynomial p as a list of size $M = 4162761127$, where each element is an 80-bit value corresponding to a monomial. The memory required to store this list is thus $80M = 333020890160 \approx 2^{38.28}$ bits. Next, we select 34 key variables $k_{33}, k_{58}, k_{59}, k_{62}, k_{36}, k_{37}, k_{38}, k_{34}, k_{50}, k_{51}, k_{52}, k_{53}, k_{54}, k_{12}, k_{13}, \dots, k_{32}$ to enumerate. Since the 6 conditions are

$$\begin{aligned} z_{61} = k_{61} = 0, \quad z_{102} = k_{33} + k_{58}k_{59} + k_{60} = 0, \quad z_{100} = k_{35} + k_{60}k_{61} + k_{62} = 0, \\ z_{63} = k_{63} = 0, \quad z_{124} = k_{11} + k_{36}k_{37} + k_{38} = 0, \quad z_{126} = k_9 + k_{34}k_{35} + k_{36} = 0, \end{aligned}$$

we can compute the values of $k_{61}, k_{63}, k_{35}, k_{60}, k_{11}, k_9$ from the guessed values of $k_{33}, k_{58}, k_{59}, k_{62}, k_{36}, k_{37}, k_{38}, k_{34}$; in particular, $k_{35} = k_{62}$ because $k_{61} = 0$. Thus, by assigning the 34 enumerated values to p , we can get a new polynomial p' with respect to the remaining $80 - 6 - 34 = 40$ key variables. Then we can construct truth table for each p' using the Möbius transform with time complexity 40×2^{40} . Suppose we obtain the equation $p(\mathbf{k}) = a$ by computing cube sum. From all entries in the truth table, we identify all key assignments for which $p' = a$ holds, and then verify the correctness of these assignments by querying the encryption oracle. Under the same storage model, the memory complexity is $2 \times (80M) + 2^{40} \approx 2^{40.68}$ bits. The time complexity is $2^{34} \times (80M + 40 \times 2^{40}) \approx 2^{79.33}$ bit operations plus approximately 2^{73} encryption queries, which is slightly more than 2^{73} Trivium calls if we treat an 855-round Trivium query as $855 \times 9 \approx 2^{12.91}$ bit operations.

8 Conclusion

In this paper, we present a novel framework of cube attacks named cube attacks with elimination strategy. To apply this framework in attacks, we develop several techniques. First, we introduce the conditional monomial prediction and propose a nested coefficient solver that significantly accelerates the previous monomial prediction algorithm. Additionally, we develop an automated cube search algorithm, enabling the discovery of a large number of cubes for attacks on a high round that were previously inaccessible. At last, we propose two variable-substitution-based testing methods that enable more efficient recovery of key information from complex equation systems. By integrating these advancements, we successfully carry out practical attacks to 840- and 845-round Trivium, achieving a 13-round improvement over prior practical attacks, and theoretical weak-key attacks to 855-round Trivium, achieving a 4-round improvement. These results demonstrate the effectiveness of our proposed techniques.

It should be noted that the core part of our framework is selecting appropriate key conditions to simplify the superpoly, thereby recovering key information from conditional superpolies. From the description of our attack framework, it can be seen that we do not impose many requirements on ciphers. Therefore, theoretically, our attack method is applicable to all ciphers that can be targeted by cube attacks (such as Ascon, as well as NFSR-based ones like Acorn and Grain). In the future, we will keep improving this framework and apply this method to these ciphers to demonstrate its power.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments. We also thank Jiahui He for helpful discussions. This work was supported by the National Key R&D Program of China (No. 2024YFA1013000), the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB0690000, the National Natural Science Foundation of China (Grant No. 62372449, 12231015 and 62122085), and the Youth Innovation Promotion Association of Chinese Academy of Sciences. This research was also partially supported by the Ministry of Education in Singapore under Grants RG102/24.

References

- [CT22] 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.
- [DCP08] Christophe De Canniere and Bart Preneel. Trivium. In *New Stream Cipher Designs: The eSTREAM Finalists*, pages 244–266. Springer, 2008.
- [DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schl  ffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.
- [DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.
- [FV13] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 502–517. Springer, 2013.
- [Gur24] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, 2024. Available at: <https://www.gurobi.com>.
- [HE22] Hosein Hadipour and Maria Eichlseder. Autoguess: A tool for finding guess-and-determine attacks and key bridges. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*,

- volume 13269 of *Lecture Notes in Computer Science*, pages 230–250. Springer, 2022.
- [HHL25] Jiahui He, Kai Hu, and Guowei Liu. Exploring core monomial prediction further: Weak-key superpoly recovery for 852-round trivium. *Cryptology ePrint Archive*, Paper 2025/1330, 2025.
- [HHLW24] Jiahui He, Kai Hu, Hao Lei, and Meiqin Wang. Massive superpoly recovery with a meet-in-the-middle framework - improved cube attacks on trivium and kreyvium. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 368–397. Springer, 2024.
- [HHPW22] 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.
- [HJM⁺21] Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, Jonathan Sönerup, and Hirotaka Yoshida. Grain-128AEADv2 – a lightweight aead stream cipher. *NIST Lightweight Cryptography*, Round, 3, 2021.
- [HLM⁺20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020.
- [HST⁺21] 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.
- [HSWW20] 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, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 446–476. Springer, 2020.
- [HWX⁺17] 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.
- [LHHW23] Hao Lei, Jiahui He, Kai Hu, and Meiqin Wang. More balanced polynomials: Cube attacks on 810- and 825-round trivium with practical complexities. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers*, volume 14201 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2023.
- [Liu17] Meicheng Liu. Degree evaluation of nfsr-based cryptosystems. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 227–249. Springer, 2017.
- [LYWL18] 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.
- [MS11] Piotr Mroczkowski and Janusz Szmidi. Corrigendum to: The cube attack on stream cipher trivium and quadraticity tests. *IACR Cryptol. ePrint Arch.*, page 32, 2011.
- [SMB17] Santanu Sarkar, Subhamoy Maitra, and Anubhab Baksi. Observing biases in the state: case studies with trivium and trivia-sc. *Des. Codes Cryptogr.*, 82(1-2):351–375, 2017.
- [SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257. Springer, 2009.
- [Sun21] Yao Sun. Automatic search of cubes for attacking stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2021(4):100–123, 2021.
- [TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube attacks on non-blackbox polynomials based on division property. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 250–279. Springer, 2017.
- [WHG⁺19] 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, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 398–427. Springer, 2019.

- [WHT⁺18] 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, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 275–305. Springer, 2018.
- [WQW23] Jianhua Wang, Lu Qin, and Baofeng Wu. Correlation cube attack revisited - improved cube search and superpoly recovery techniques. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part III*, volume 14440 of *Lecture Notes in Computer Science*, pages 190–222. Springer, 2023.
- [Wu16] Hongjun Wu. ACORN: a lightweight authenticated cipher (v3). Submission to CAESAR competition, 2016.
- [YT21] 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.
- [ZS24] Min Zhang and Yao Sun. A new practical cube attack via recovering numerous superpolys. *IACR Transactions on Symmetric Cryptology*, 2024(4):38–63, 2024.

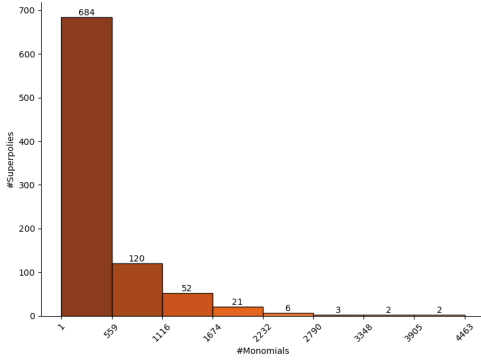
Supplementary Material

A Details for 840-round Attack

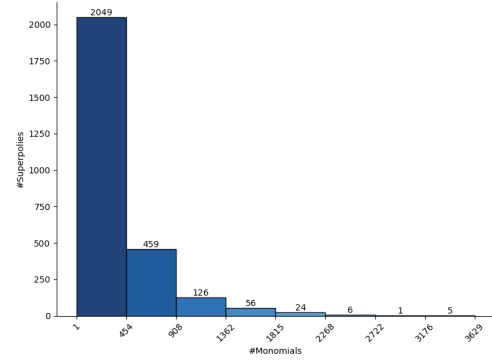
In this section, we present some statistical data of our attacks on 840-round Trivium. The mother cubes are

$$\begin{aligned} \mathcal{C}_{840}^1 &= \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, \\ &v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{27}, v_{29}, v_{30}, v_{31}, v_{32}, v_{33}, v_{34}, v_{35}, v_{36}, v_{37}, v_{38}, v_{40}, v_{42}, v_{44}, \\ &v_{46}, v_{48}, v_{50}, v_{52}, v_{55}, v_{57}, v_{59}, v_{61}, v_{63}, v_{65}, v_{67}, v_{70}, v_{72}, v_{74}, v_{76}, v_{77}, v_{78}, v_{79}\}. \\ \mathcal{C}_{840}^2 &= \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, \\ &v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{27}, v_{29}, v_{31}, v_{33}, v_{35}, v_{36}, v_{37}, v_{40}, v_{42}, v_{44}, v_{46}, v_{48}, v_{50}, v_{52}, \\ &v_{55}, v_{57}, v_{59}, v_{61}, v_{63}, v_{65}, v_{67}, v_{70}, v_{72}, v_{74}, v_{76}, v_{77}, v_{78}, v_{79}\}. \end{aligned}$$

Figure 2 illustrates the distribution of term counts for two sets of superpolies. Figure 3 displays the distribution of algebraic degrees. It can be observed that the superpolies generated by cubes selected through our cube search algorithm remain relatively small in scale.

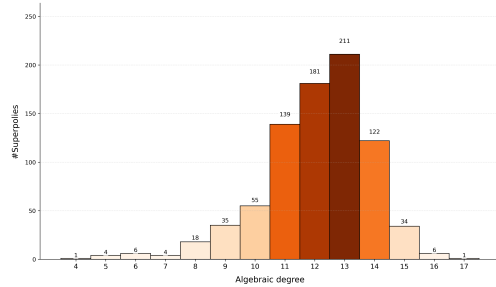


(a) The number of monomials of superpolies of the mothercube $\mathcal{C}_{840}^2$

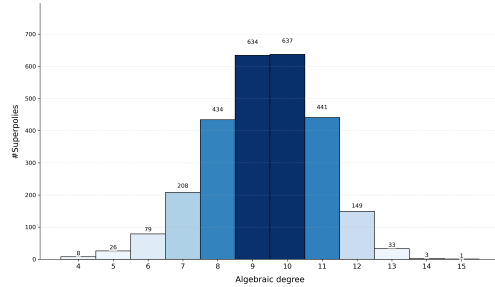


(b) The number of monomials of superpolies of the mothercube $\mathcal{C}_{840}^1$

Figure 2: The number of monomials of superpolies



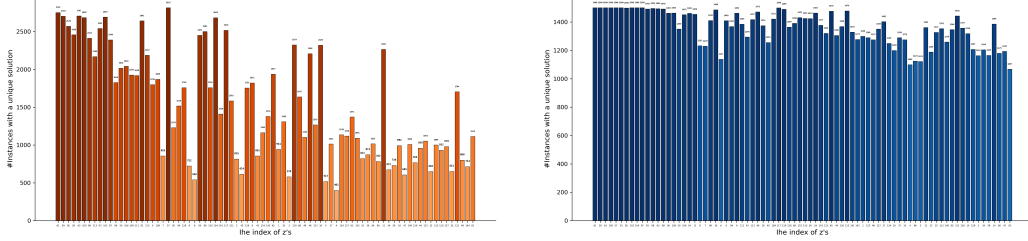
(a) The distributions of the algebraic degree of superpolies with respect to the mothercube $\mathcal{C}_{840}^2$



(b) The distributions of the algebraic degree of superpolies with respect to the mothercube $\mathcal{C}_{840}^1$

Figure 3: The distributions of the algebraic degree of superpolies

Figure 4 demonstrates the statistical frequency of variables yielding unique solutions across thousands of randomly sampled keys. The x-axis represents the index j of $z_{83}[j]$, ordered from left to right by the descending frequency of each variable's occurrence in the superpolies. Notably, variables with higher occurrence frequencies exhibit a greater number of instances with unique solutions.



(a) Unique solution count of top 80 frequent variables with respect to the mothercube $\mathcal{C}_{840}^2$

(b) Unique solution count of top 80 frequent variables with respect to the mothercube $\mathcal{C}_{840}^2$

Figure 4: The distributions of the unique solution count

In the following, we provide an example of utilizing equations with respect to $z_{83}[j]$ for enumeration.

Example 3. Given that the 80-bit secret key is represented in hexadecimal as 1B CC 95 CA 14 E1 F3 94 F7 62, where the most significant bit corresponds to k_0 and the least significant bit corresponds to k_{79} . We recovered the values of superpolies with a complexity of 2^{54} and ultimately derived 26 equations, as shown in Table 4. We selected the red portion as the determined variables, and after enumerating the values of the remaining 54 secret variables, we can get these determined variables. Therefore, we can recover the full key with time complexity of $2^{54} + 2^{54} \approx 2^{55}$.

Table 4: Determined values of equations

Equations	Det. vars	Equations	Det. vars
$z_{11} = k_{11} = 0$	k_{11}	$z_{16} = k_{16} = 0$	k_{16}
$z_{24} = k_{24} = 0$	k_{24}	$z_{26} = k_{26} = 0$	k_{26}
$z_{56} = k_{56} = 0$	k_{56}	$z_{57} = k_{57} = 0$	k_{57}
$z_{46} = k_{46} = 1$	k_{46}	$z_{59} = k_{59} = 0$	k_{59}
$z_{61} = k_{61} = 1$	k_{61}	$z_{63} = k_{63} = 1$	k_{63}
$z_{83} = k_{52} + k_{77}k_{78} + k_{79} = 0$	k_{79}	$z_{85} = k_{50} + k_{75}k_{76} + k_{77} = 1$	k_{77}
$z_{87} = k_{48} + k_{73}k_{74} + k_{75} = 0$	k_{75}	$z_{89} = k_{46} + k_{71}k_{72} + k_{73} = 0$	k_{73}
$z_{91} = k_{44} + k_{69}k_{70} + k_{71} = 0$	k_{71}	$z_{93} = k_{42} + k_{67}k_{68} + k_{69} = 1$	k_{69}
$z_{96} = k_{39} + k_{64}k_{65} + k_{66} = 1$	k_{66}	$z_{100} = k_{35} + k_{60}k_{61} + k_{62} = 0$	k_{62}
$z_{102} = k_{33} + k_{58}k_{59} + k_{60} = 0$	k_{60}	$z_{104} = k_{31} + k_{56}k_{57} + k_{58} = 1$	k_{58}
$z_{111} = k_{24} + k_{49}k_{50} + k_{51} = 0$	k_{51}	$z_{115} = k_{20} + k_{45}k_{46} + k_{47} = 1$	k_{47}
$z_{117} = k_{18} + k_{43}k_{44} + k_{45} = 0$	k_{45}	$z_{121} = k_{14} + k_{39}k_{40} + k_{41} = 1$	k_{41}

B Details for 845-round Attack

In this section, we present some statistical data of our attacks on 845-round Trivium. We first provide an explanation for the parameter configuration in Algorithm 14.

When selecting these parameters, we must balance the size of the conditional superpolies, the recovery time, and the overall size of the weak-key space. To ensure that no single

weak-key subspace is too small, we set $s = 3$ making each individual weak-key subspace of size 2^{77} . To guarantee that the attack remains effective for more than half of the keys, we select 9 weak-key subspaces of equal size. The two intermediate round numbers $r_{b_0} = 250$, $r_{b_1} = 185$, along with the bounds $T = 120$, $T_1 = 80$ and $T_0 = 20$ are chosen based on Trivium's structure and the performance of our conditional monomial prediction. Finally, the parameters $n = 2$, $m = 10$ and $l = 4$ are selected to precompute a suitably sized candidate set, enabling efficient extraction of a valid triple-subset from this collection. Next, I will explain the data presented in the figure.

First, Figure 5 illustrates the distribution of term counts for the 9 conditional superpolies set. It can be observed that these conditional superpolies remain relatively not too large in scale, most are smaller than 10,000 terms.

Figure 6 shows the accuracy of our method in solving equations for different weak-key subspaces. As shown in the results, we achieved 100% accuracy for 7 out of the 9 subspaces, with the remaining two reaching 99% and 97% respectively. In our evaluation, a solution is deemed accurate when all recovered $z_{93}[j]$ values are correct.

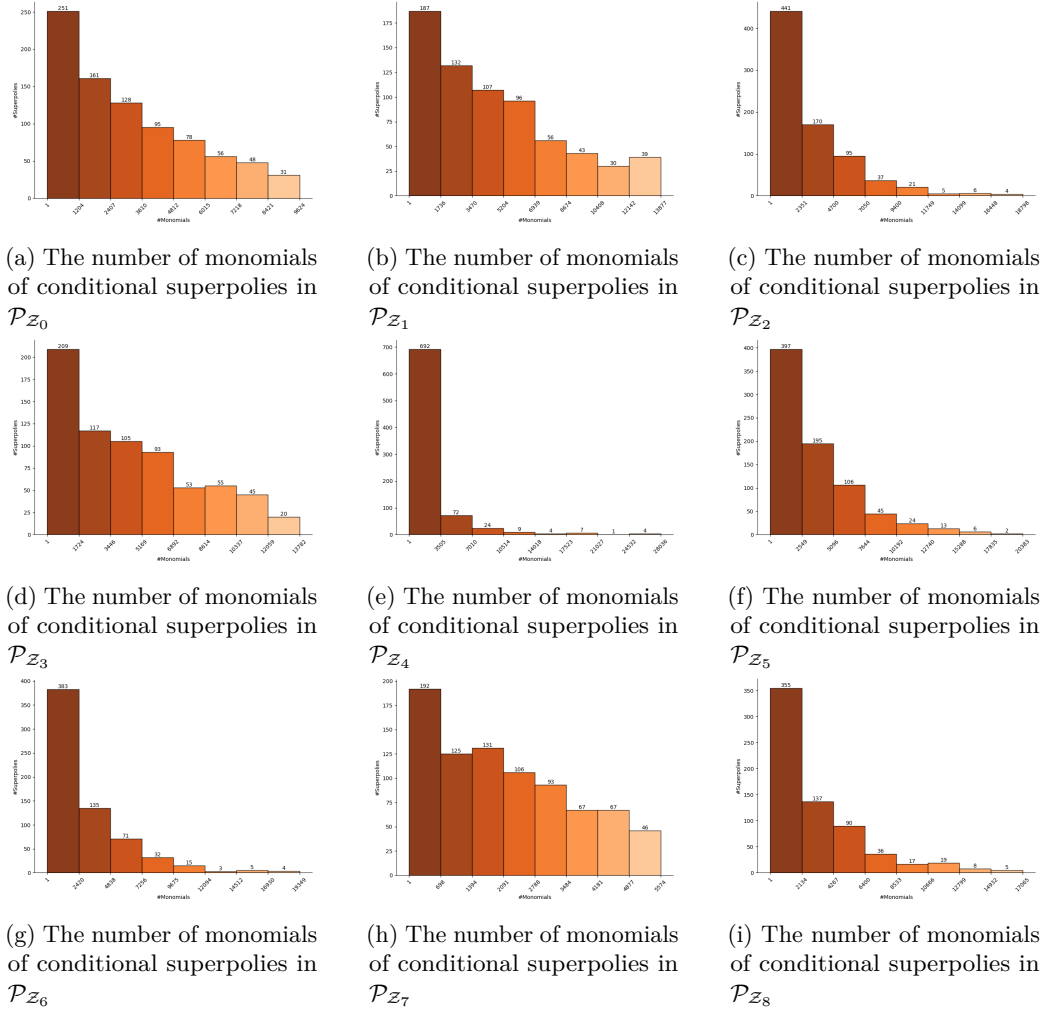


Figure 5: The number of monomials of conditional superpolies in 9 set

Our attack uses the recovery of 33 $z_{93}[j]$ s values as the termination criterion. This criterion was chosen because we aim to obtain at least 29 determined variables from as few

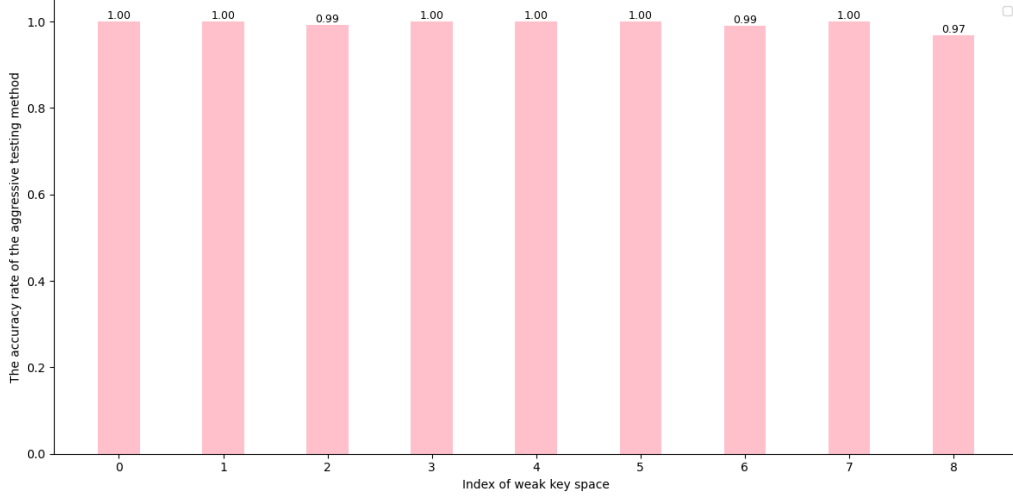


Figure 6: Equation solution accuracy in different weak key spaces

equations as possible. Since the $z_{93}[j]$ -expressions shows relatively strong independence among key variables, we set the threshold to be 33 which is slightly higher than 29. As shown in Figure 7, with our parameter settings, we successfully recovered at least 29 determined variables for the vast majority of keys. Moreover, keys that fail in one subspace frequently succeed in others. Only 11 keys failed across all subspaces, which is what enables our 98% success rate.

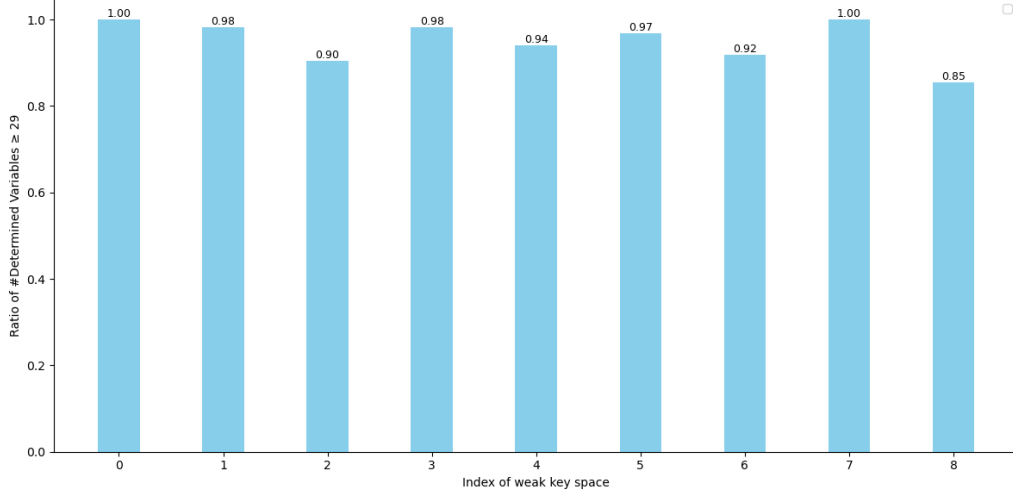


Figure 7: Proportion of cases where at least 29 determined variables are recovered in different weak key spaces

Figure 8 presents the occurrence frequency of the top 33 variables in each conditional superpoly set, along with the number of instances where these variables had unique solutions during solving. The figure reveals that the occurrence counts of these variables nearly match the total number of equations and frequently occurring variables consistently yield unique solutions in most cases .

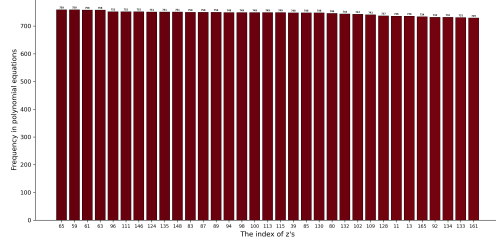
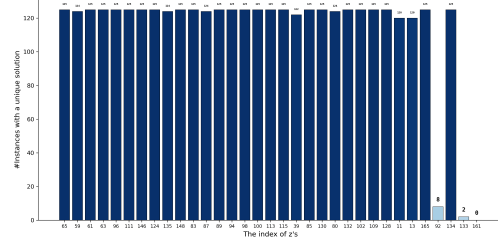
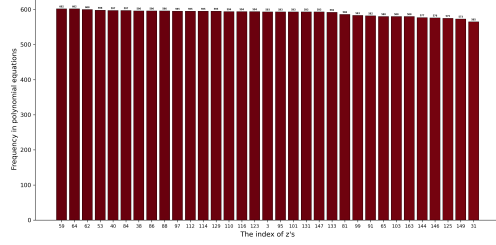
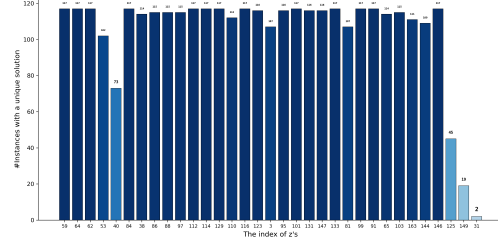
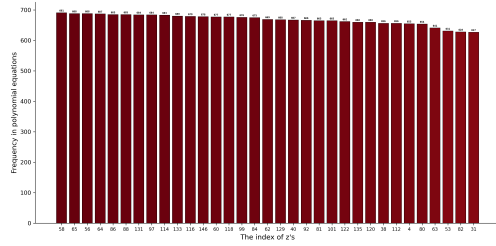
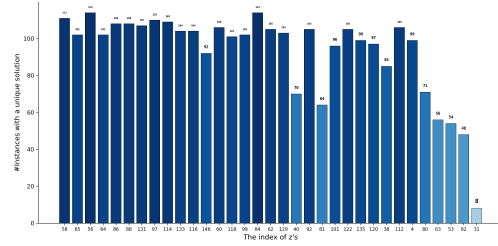
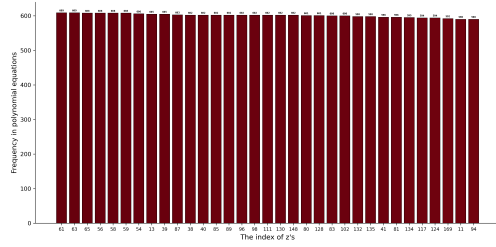
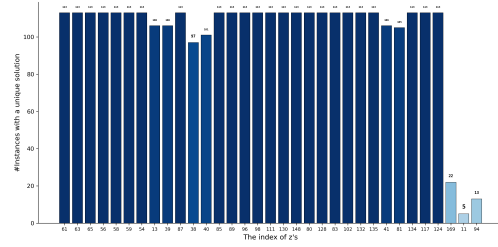
(a) The appearing times of variables in the conditional superpolies in $\mathcal{P}_{Z_0}$ (b) Unique solution count of top 33 frequent variables in $\mathcal{P}_{Z_0}$ (c) The appearing times of variables in the conditional superpolies in $\mathcal{P}_{Z_1}$ (d) Unique solution count of top 33 frequent variables in $\mathcal{P}_{Z_1}$ (e) The appearing times of variables in the conditional superpolies in $\mathcal{P}_{Z_2}$ (f) Unique solution count of top 33 frequent variables in $\mathcal{P}_{Z_2}$ (g) The appearing times of variables in the conditional superpolies in $\mathcal{P}_{Z_3}$ (h) Unique solution count of top 33 frequent variables in $\mathcal{P}_{Z_3}$

Figure 8: The distributions of the unique solution count

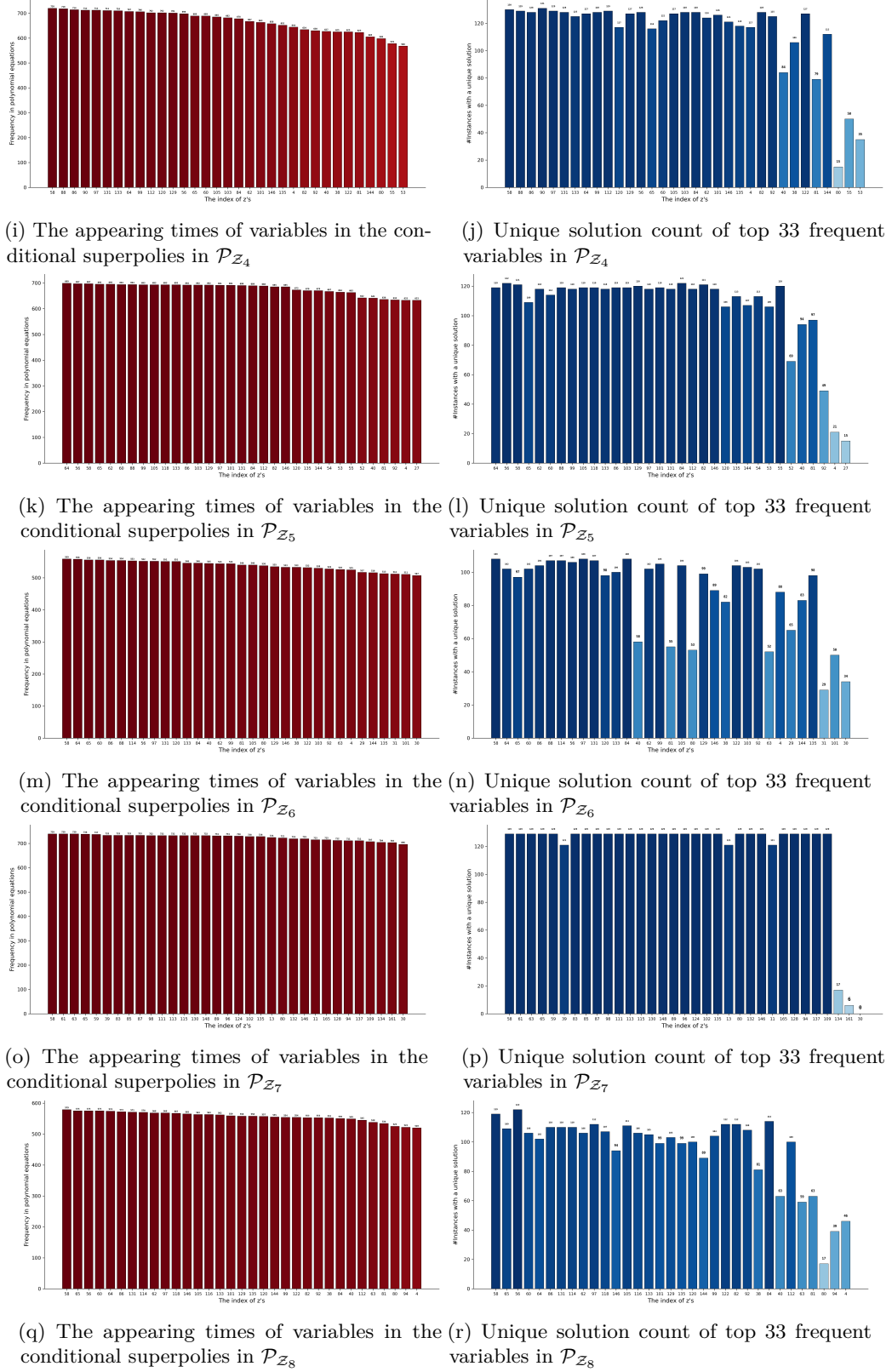


Figure 8: The distributions of the unique solution count

C MILP Models of the Propagation Trails

Model 1 (COPY). Let $a \xrightarrow{COPY} (b_0, b_1, \dots, b_{n-1})$ be a propagation trail of COPY. All trails for COPY can be fully described by the following inequalities.

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

Model 2 (AND). Let $(b_0, b_1, \dots, b_{n-1}) \xrightarrow{AND} a$ be a propagation trail of AND. All trails for AND can be fully described by the following inequalities.

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

Model 3 (XOR). Let $(b_0, b_1, \dots, b_{n-1}) \xrightarrow{XOR} a$ be a propagation trail of XOR. All trails for XOR can be fully described by the following inequalities.

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

D Numeric Mapping

Algorithm 4 Estimation of Degree of Trivium-Like Ciphers [Liu17]

Input: Given the ANFs of all internal states (A_t, B_t, C_t) with $t \leq N_0$, and the set of variables X

Output: An upper bound on algebraic degree of an N -round Trivium-like cipher f with X as input variables.

```

1: for  $\lambda$  in  $\{A, B, C\}$  do
2:   for  $t$  from  $1 - n_\lambda$  to  $0$  do
3:      $d_\lambda^{(t)} \leftarrow \deg(\lambda_0[t], X)$   $\triangleright A_i[t] = x_t, B_i[t] = y_t, \text{ and } C_i[t] = z_t$ 
4:   end for
5: end for
6:  $D^{(0)} \leftarrow (d_A^{(1-n_A)}, \dots, d_A^{(0)}, d_B^{(1-n_B)}, \dots, d_B^{(0)}, d_C^{(1-n_C)}, \dots, d_C^{(0)})$   $\triangleright$  Compute the exact algebraic degrees of the internal states for the first  $N_0$  rounds
7: for  $t$  from  $1$  to  $N_0$  do
8:   for  $\lambda$  in  $\{A, B, C\}$  do
9:      $dm_\lambda^{(t)} \leftarrow \deg(g_\lambda^{(t)}, X)$ 
10:     $d_\lambda^{(t)} \leftarrow \deg(\lambda_t[t], X)$ 
11:   end for
12:    $D^{(t)} \leftarrow (d_A^{(t-n_A+1)}, \dots, d_A^{(t)}, d_B^{(t-n_B+1)}, \dots, d_B^{(t)}, d_C^{(t-n_C+1)}, \dots, d_C^{(t)})$ 
13: end for
14: for  $t$  from  $N_0 + 1$  to  $N$  do
15:   for  $\lambda$  in  $\{A, B, C\}$  do
16:      $dm_\lambda^{(t)} \leftarrow \text{DegMul}^*(g_\lambda^{(t)})$ 
17:      $d_\lambda^{(t)} \leftarrow \max \{dm_\lambda^{(t)}, \text{DEG}(\ell_\lambda, D^{(t-1)})\}$ 
18:   end for
19:    $D^{(t)} \leftarrow (d_A^{(t-n_A+1)}, \dots, d_A^{(t)}, d_B^{(t-n_B+1)}, \dots, d_B^{(t)}, d_C^{(t-n_C+1)}, \dots, d_C^{(t)})$ 
20: end for
21: return  $\text{DEG}(f, D^{(N)})$ 
22: procedure  $\text{DEGMUL}^*(g_\lambda^{(t)})$ 
23:    $t_1 \leftarrow t - r_{\rho(\lambda)}$   $\triangleright \rho(A) = C, \rho(C) = B, \rho(B) = A$ 
24:   if  $t_1 \leq 0$  then
25:     return  $d_{\rho(\lambda)}^{(t_1)} + d_{\rho(\lambda)}^{(t_1+1)}$ 
26:   end if
27:    $t_2 \leftarrow t_1 - r_{\rho^2(\lambda)}$ 
28:    $d_1 \leftarrow \min \{d_{\rho^2(\lambda)}^{(t_2)} + dm_{\rho(\lambda)}^{(t_1+1)}, d_{\rho^2(\lambda)}^{(t_2+2)} + dm_{\rho(\lambda)}^{(t_1)}, d_{\rho^2(\lambda)}^{(t_2)} + d_{\rho^2(\lambda)}^{(t_2+1)} + d_{\rho^2(\lambda)}^{(t_2+2)}\}$ 
29:    $d_2 \leftarrow \text{DEG}(\ell_{\rho(\lambda)}, D^{(t_1)}) + dm_{\rho(\lambda)}^{(t_1)}$ 
30:    $d_3 \leftarrow \text{DEG}(\ell_{\rho(\lambda)}, D^{(t_1-1)}) + d_{\rho(\lambda)}^{(t_1+1)}$ 
31:    $d \leftarrow \max \{d_1, d_2, d_3\}$ 
32:   return  $d$ 
33: end procedure

```

E Nested Coefficient solver

Algorithm 5 CoeSolver0

Input: $c, \mathbf{u}_{r_m}, t_{r_m}, \mathbf{z}_{r_m} = g(\mathbf{z}_{r_s}), \mathbf{y}_{r_m}, q = 0$.

Output: 0 or 1

```

1: Declare an empty MILP model  $\mathcal{M}$ 
2: Let  $\mathbf{a}$  be  $n + t$  MILP variables of  $\mathcal{M}$  corresponding to the  $n + t$  components of  $\mathbf{v} \parallel \mathbf{z}_{r_m}$ 
3:  $\mathcal{M}.\text{con} \leftarrow \mathbf{a}_i = u_i$  for all  $i \in [n]$ 
4: Update  $\mathcal{M}$  according to the function  $\mathbf{y}_{r_m}$  and denote  $\mathbf{b}$  as the output state of  $\mathbf{y}_{r_m}$ 
5:  $\mathcal{M}.\text{con} \leftarrow \mathbf{b}_i = \mathbf{u}_{r_m}[i]$  for all  $i$ 
6:  $\mathcal{M}.\text{set\_timelimit}(t_{r_m})$ 
7:  $\mathcal{M}.\text{optimize}()$ 
8: if  $\mathcal{M}$  is overtime then
9:   return 0
10: end if
11: Prepare a hash table  $H$  whose key is  $t$ -bit string and value is counter
12: for each feasible solution of  $\mathcal{M}$  do
13:   Let  $\mathbf{s}$  denote the solution  $(a_n, \dots, a_{n+t-1})$ 
14:    $H[\mathbf{s}] \leftarrow H[\mathbf{s}] + 1$ 
15: end for
16: Prepare a polynomial  $q'(\mathbf{z}_{r_m}) \leftarrow 0$ 
17: for each  $\mathbf{s}$  satisfying  $H[\mathbf{s}]$  is odd do
18:    $q' \leftarrow q' \oplus \mathbf{z}_{r_m}^{\mathbf{s}}$ 
19: end for
20:  $q(\mathbf{z}_{r_s}) \leftarrow q'(g(\mathbf{z}_{r_s}))$ 
21: return 1

```

Algorithm 6 Nest-CoeSolver

```

1: procedure NEST-COESOLVER( $\mathbf{A}, \mathbb{S}(\mathbf{y}_{r_m}), \mathbf{c}, r_m, r_s$ )
2:   Input: Cipher  $\mathbf{A}$ , valuable terms  $\mathbb{S}(\mathbf{y}_{r_m})$ , cube indicator vector  $\mathbf{c}$ , round of valuable
   terms  $r_m$ , output round  $r_s$ .
3:   Output: Superpoly  $p_{\mathbf{c}}(\mathbf{z}_{r_s})$ 
4:   Initialize polynomial  $p \leftarrow 0$ 
5:   Initialize set  $\mathbb{S} \leftarrow \emptyset$ 
6:   while true do
7:     select time limit  $t_{r_m}$ 
8:     Generate  $\mathbf{z}_{r_m} = \mathbf{g}'(\mathbf{z}_{r_s}), \mathbf{y}'_{r_m}$ 
9:     for each  $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}(\mathbf{y}_{r_m})$  do
10:      if CoeSolver0( $\mathbf{c}, \mathbf{u}_{r_m}, t_{r_m}, \mathbf{z}_{r_m}, \mathbf{y}'_{r_m}, q = 0$ ) = 1 then
11:        Add  $q$  to  $p$ 
12:      else
13:        Add monomial to  $\mathbb{S}$ 
14:      end if
15:    end for
16:    if  $\mathbb{S} = \emptyset$  then
17:      return  $p$ 
18:    end if
19:    Choose  $r_l < r_m$  and compute  $\mathbb{S}(\mathbf{y}_{r_l}) = \text{TermExpander}(\mathbb{S}, r_l, r_m, \mathbf{u})$ 
20:    if  $\mathbb{S}(\mathbf{y}_{r_l}) = \emptyset$  then
21:      return  $p$ 
22:    end if
23:    Set  $r_m \leftarrow r_l, \mathbb{S}(\mathbf{y}_{r_m}) \leftarrow \mathbb{S}(\mathbf{y}_{r_l})$ 
24:  end while
25: end procedure

```

F Conditional Monomial Prediction

Algorithm 7 Con-NBDP-MPModel

- 1: **procedure** CON-NBDP-MPModel($r, \mathbf{t}^{(r)}, \mathbf{u}, r_m, r_c, \mathcal{Y} = \bigcup_{i \in \Lambda} \{\mathbf{y}_{r_c}[i_t]\}_{t=0}^\lambda$ the target elimination state)
 - 2: Declare an empty MILP model $\mathcal{M}$
 - 3: Declare $\mathbf{k}^0$ as $n + m$ MILP variables of $\mathcal{M}$ corresponding to public and secret variables
 - 4: $\mathcal{M}.con \leftarrow \mathbf{k}_i^0 = 1 \ \forall u_i = 1$
 - 5: $\mathcal{M}.con \leftarrow \mathbf{k}_i^0 = 0 \ \forall u_i = 0$
 - 6: $\mathcal{M}.con \leftarrow \mathbf{k}_{i+n}^0 = 0 \ \forall i \in \{0, 1, \dots, m-1\}$
 - 7: Update $\mathbf{k}^0$ according to the propagation rules of NBDP through r_c rounds to $\mathbf{k}^{r_c}$
 - 8: Declare $\mathbf{s}^{r_c}$ as n_{r_c} MILP variables of $\mathcal{M}$
 - 9: $\mathcal{M}.con \leftarrow \mathbf{k}_i^{r_c} = \mathbf{s}_i^{r_c} = 0 \ \forall \mathbf{y}_{r_c}[i_t] \in \mathcal{Y}$
 - 10: $\mathcal{M}.con \leftarrow \mathbf{k}_i^{r_c} = \mathbf{s}_i^{r_c} \ \forall \mathbf{y}_{r_c}[i_t] \notin \mathcal{Y}$
 - 11: Update $\mathbf{k}^{r_c}$ according to the propagation rules of NBDP through r_m rounds to $\mathbf{k}^{r_m}$
 - 12: Declare $\mathbf{s}^{r_m}$ as n_{r_m} MILP variables of $\mathcal{M}$
 - 13: $\mathcal{M}.con \leftarrow \mathbf{k}_i^{r_m} = \mathbf{s}_{i_t}^{r_m} = 0 \ \forall \mathbf{s}_i^{(r_m)}.F = 0_c$
 - 14: $\mathcal{M}.con \leftarrow \mathbf{k}_i^{r_m} \leq \mathbf{s}_i^{r_m} \ \forall \mathbf{s}_i^{(r_m)}.F \neq 0_c$
 - 15: Update $\mathbf{s}^{r_m}$ according to the propagation rules of the monomial prediction through $r - r_m$ rounds to $\mathbf{s}^r$
 - 16: $\mathcal{M}.con \leftarrow \mathbf{s}_i^r = 0 \ \forall t_i^{(r)} = 0$
 - 17: $\mathcal{M}.con \leftarrow \mathbf{s}_i^r = 1 \ \forall t_i^{(r)} = 1$
 - 18: **return** $\mathcal{M}$
 - 19: **end procedure**
-

Algorithm 8 Conditional Term expander

```

1: procedure CON-TERMEXPANDER( $\mathbb{S}(\mathbf{y}_r)$ ,  $r$ ,  $r_c$ ,  $r_m$ ,  $\mathbf{u}$ ,  $\mathcal{Z}$ )
2:   if  $r > R$  then  $\triangleright R$  is a bound bigger than  $r_m$ 
3:     Call the term expander in NMP to expand all terms from  $\mathbb{S}(\mathbf{y}_r)$  into  $\mathbb{S}(\mathbf{y}_R)$   $\triangleright$ 
       Details of NMP are listed at [HST+21]
4:      $r \leftarrow R$ 
5:      $\mathbb{S}(\mathbf{y}_r) \leftarrow \mathbb{S}(\mathbf{y}_R)$ 
6:   end if
7:   Genrate the target elimination states  $\mathcal{Y} = \bigcup_{i \in \Lambda} \{\mathbf{y}_{r_c}[i_t]\}_{t=0}^\lambda$  with respect to  $\mathcal{Z}$  at
       round  $r_c$ 
8:   repeat
9:      $r_l \leftarrow \max(r - \varepsilon, r_m)$ 
10:    Initialize an empty set  $\mathbb{S}(\mathbf{y}_{r_l})$ 
11:    Prepare a hash table  $J_{r_l}$  whose key is a term of  $r_l$ -th round and the value is an
       integer
12:    for  $\pi_{\mathbf{t}_r}(\mathbf{y}_r) \in \mathbb{S}(\mathbf{y}_r)$  do
13:      Prepare an empty set  $\mathbb{P}_{r_l}$ 
14:       $\mathcal{M} = \text{Con-NBDP-MPModel}(r, \mathbf{t}_r, \mathbf{u}, r_l, r_c, \mathcal{Y})$   $\triangleright$  See Algorithm 7
15:      Solve the MILP model  $\mathcal{M}$  within the time limit  $\tau'$ 
16:      loop
17:        if Find a solution for  $\mathcal{M}$  then
18:          Extract the solution corresponding to  $\pi_{\mathbf{t}_r}(\mathbf{y}_r)$  as  $\pi_{\mathbf{p}_{r_l}}(\mathbf{y}_{r_l})$ 
19:          Remove  $\pi_{\mathbf{p}}(\mathbf{s}_{r_l})$  from the solution space
20:          Solve a MILP model of the monomial prediction to determine if
              $\pi_{\mathbf{p}_{r_l}}(\mathbf{y}_{r_l}) \rightarrow \pi_{\mathbf{t}_r}(\mathbf{y}_r)$ 
21:          if  $\pi_{\mathbf{p}_{r_l}}(\mathbf{y}_{r_l}) \rightarrow \pi_{\mathbf{t}_r}(\mathbf{y}_r)$  then
22:            Add  $\pi_{\mathbf{p}_{r_l}}(\mathbf{y}_{r_l})$  to  $\mathbb{P}^{(r_l)}$ 
23:          end if
24:          Continue to solve  $\mathcal{M}$ 
25:        else if  $\mathcal{M}$  has no solution then
26:          Break the loop
27:        else if  $\mathcal{M}$  reaches the time limit then
28:          Clear  $\mathbb{P}^{(r_l)}$  and call the term expander of NMP to expand  $\pi_{\mathbf{t}_r}(\mathbf{y}_r)$ 
             into  $\mathbb{P}_{r_l}$ 
29:          Break the loop
30:        end if
31:      end loop
32:      for each  $\pi_{\mathbf{t}_{r_l}}(\mathbf{y}_{r_l})$  in  $\mathbb{P}_{r_l}$  do
33:        Increase  $J_{r_l}[\pi_{\mathbf{t}_{r_l}}(\mathbf{y}_{r_l})]$  by 1
34:      end for
35:    end for
36:    for each  $\pi_{\mathbf{t}_{r_l}}(\mathbf{y}_{r_l})$  whose  $J_{r_l}[\pi_{\mathbf{t}_{r_l}}(\mathbf{y}_{r_l})]$  is an odd-number do
37:       $\mathbb{S}(\mathbf{y}_{r_l}) = \mathbb{S}(\mathbf{y}_{r_l}) \cup \pi_{\mathbf{t}_{r_l}}(\mathbf{y}_{r_l})$ 
38:    end for
39:     $\mathbb{S}(\mathbf{y}_r) \leftarrow \mathbb{S}(\mathbf{y}_{r_l})$ 
40:     $r \leftarrow r_l$ 
41:  until  $r = r_m$ 
42:  return  $\mathbb{S}(\mathbf{y}_{r_m})$ 
43: end procedure

```

Algorithm 9 Con-CoeSolver0

Input: $c, \mathbf{u}_{r_m}, t_{r_m}, \mathbf{z}_{r_m} = \mathbf{g}'(\mathbf{z}_{r_s}), \mathbf{y}'_{r_m}, q = 0$.**Output:** 0 or 1

- 1: Declare an empty MILP model $\mathcal{M}$
 - 2: Let $\mathbf{a}$ be $n + t$ MILP variables of $\mathcal{M}$ corresponding to the $n + t$ components of $\mathbf{v} \parallel \mathbf{z}_{r_m}$
 - 3: $\mathcal{M}.\text{con} \leftarrow a_i = u_i$ for all $i \in [n]$
 - 4: Update $\mathcal{M}$ according to the function $\mathbf{y}_{r_m}$ and denote $\mathbf{b}$ as the output state of $\mathbf{y}_{r_m}$
 - 5: $\mathcal{M}.\text{con} \leftarrow b_i = \mathbf{u}_{r_m}[i]$ for all i
 - 6: $\mathcal{M}.\text{set_timelimit}(t_{r_m})$
 - 7: $\mathcal{M}.\text{optimize}()$
 - 8: **if** $\mathcal{M}$ is overtime **then**
 - 9: **return** 0
 - 10: **end if**
 - 11: Prepare a hash table H whose key is t -bit string and value is counter
 - 12: **for** each feasible solution of $\mathcal{M}$ **do**
 - 13: Let $\mathbf{s}$ denote the solution $(a_n, \dots, a_{n+t-1})$
 - 14: $H[\mathbf{s}] \leftarrow H[\mathbf{s}] + 1$
 - 15: **end for**
 - 16: Prepare a polynomial $q'(\mathbf{z}_{r_m}) \leftarrow 0$
 - 17: **for** each $\mathbf{s}$ satisfying $H[\mathbf{s}]$ is odd **do**
 - 18: $q' \leftarrow q' \oplus \mathbf{z}_{r_m}^{\mathbf{s}}$
 - 19: **end for**
 - 20: $q(\mathbf{z}_{r_s}) \leftarrow q'(\mathbf{g}'(\mathbf{z}_{r_s}))$
 - 21: **return** 1
-

Algorithm 10 Nest-Con-CoeSolver

```

1: procedure NEST-CON-COESOLVER( $\mathbf{A}, \mathbb{S}(\mathbf{y}_{r_m}), \mathbf{c}, r_m, r_s, \mathcal{Z}$ )
2:   Input: Cipher  $\mathbf{A}$ , valuable terms  $\mathbb{S}(\mathbf{y}_{r_m})$ , cube indicator vector  $\mathbf{c}$ , round of valuable
   terms  $r_m$ , output round  $r_s$ .
3:   Output: Superpoly  $p_{\mathbf{c}}(\mathbf{z}_{r_s})$ 
4:   Initialize polynomial  $p \leftarrow 0$ 
5:   Initialize set  $\mathbb{S} \leftarrow \emptyset$ 
6:   while true do
7:     select time limit  $t_{r_m}$ 
8:     Generate  $\mathbf{z}_{r_m} = \mathbf{g}'(\mathbf{z}_{r_s})$ ,  $\mathbf{y}'_{r_m}$  with respect to  $\mathcal{Z}$ 
9:     for each  $\pi_{\mathbf{u}_{r_m}}(\mathbf{y}_{r_m}) \in \mathbb{S}(\mathbf{y}_{r_m})$  do
10:      if Con-CoeSolver0( $\mathbf{u}_{r_m}, \mathbf{c}, \mathbf{g}'(\mathbf{z}_{r_s}), \mathbf{y}'_{r_m}, t_{r_s}, q = 0$ ) = 1 then
11:         $p = p \oplus q$ 
12:      else
13:        Add  $\pi_{\mathbf{u}_{r_m}}$  to  $\mathbb{S}$ 
14:      end if
15:    end for
16:    if  $\mathbb{S} = \emptyset$  then
17:      return  $p$ 
18:    end if
19:    Choose  $r_l < r_m$  and compute  $\mathbb{S}(\mathbf{y}_{r_l}) = \text{Con-TermExpander}(\mathbb{S}, r_l, r_c, r_m, \mathbf{u}, \mathcal{Z})$ 
20:    if  $\mathbb{S}(\mathbf{y}_{r_l}) = \emptyset$  then
21:      return  $p$ 
22:    end if
23:    Set  $r_m \leftarrow r_l$ ,  $\mathbb{S}(\mathbf{y}_{r_m}) \leftarrow \mathbb{S}(\mathbf{y}_{r_l})$ 
24:  end while
25: end procedure

```

G Cube Indices

Table 5: The cube indices used in this paper

I	Indices	Size
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, 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_2	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15, 16, 18, 20, 23, 25, 27, 29, 31, 33, 35, 37, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 77, 78, 79	46
I_3	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 27, 29, 30, 31, 33, 35, 36, 37, 38, 40, 42, 44, 46, 48, 50, 52, 55, 57, 59, 61, 63, 65, 67, 70, 72, 74, 76, 78	53
I_4	0, 1, 2, 3, 5, 7, 9, 10, 11, 12, 13, 14, 16, 18, 20, 22, 24, 27, 29, 31, 33, 35, 37, 39, 42, 44, 46, 48, 50, 52, 54, 57, 59, 61, 63, 65, 67, 68, 69, 70, 72, 74, 76, 78, 79	45

H Cube Search Algorithms

Algorithm 11 Automated Initial Cube Selection

Input: Candidate cube set $\mathcal{I}$, dimension m , threshold D

Output: Filtered cube set $\mathcal{C}$

```

1:  $\mathcal{C} \leftarrow \emptyset$ 
2: while  $\mathcal{I} \neq \emptyset$  do
3:    $c \leftarrow \text{RandomSelect}(\mathcal{I})$ 
4:   if  $\text{NUM}(f, c) \leq D$  then
5:      $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$ 
6:      $\mathcal{I} \leftarrow \mathcal{I} \setminus \{c\}$ 
7:   else
8:     repeat  $\triangleright$  This process remove bad cubes from set  $\mathcal{I}$  by applying property 2
9:        $c' \leftarrow \text{ReduceDimension}(c, 1)$ 
10:      if  $\text{NUM}(f, c') \leq D$  then
11:         $\mathcal{I} \leftarrow \mathcal{I} \setminus \{c'' \in \mathcal{I} \mid c \subseteq c''\}$ 
12:        break
13:      end if
14:       $c \leftarrow c'$ 
15:    until  $|c| < m - 1$ 
16:   end if
17: end while
18: return  $\mathcal{C}$ 

```

Algorithm 12 Subcube Generation

Input: mother cube c with dimension d , fixed subcube c_f , minimum dimension $d_{\min}$ of subcube, round parameter r_m , threshold M .

Output: A set of subcubes $\mathcal{C}$

```

1: Initialize:
2:    $S \leftarrow \{c\}$ ,  $\mathcal{C} \leftarrow \emptyset$ ,  $K \leftarrow \emptyset$ 
3:    $\mathcal{C} \leftarrow \mathcal{C} \cup K$ ,  $S \leftarrow K$ ,  $K \leftarrow \emptyset$ 
4: while  $S \neq \emptyset$  do
5:   for  $c \in S$  do
6:     if  $\dim(c) > d_{\min}$  then
7:       for subcube  $c'$  with dimension  $\dim(c) - 1$  and  $c_f \subseteq c'$  do
8:         Compute  $VT_{r_m}$  of  $c'$ 
9:         if  $VT_{r_m} \leq M$  then
10:          add  $c'$  into  $K$ 
11:        end if
12:      end for
13:     else
14:       return  $\mathcal{C}$ 
15:     end if
16:      $\mathcal{C} \leftarrow \mathcal{C} \cup K$ 
17:   end for
18:    $S \leftarrow K$ ,  $K \leftarrow \emptyset$ 
19: end while
20: return  $\mathcal{C}$ 

```

I Variable Elimination and Subset Selection

Algorithm 13 Optimized Filter Subsets Procedure

```

1: function FILTERSUBSETS( $\mathcal{Z}_0, m, r_{b_0}$ )
2:   Initialize priority queue  $Q$  ▷ sorted by  $CVT$  in ascending order
3:   for each variable  $z \in \mathcal{Z}_0$  do
4:      $\mathbb{S}_{\{z\}}(\mathbf{y}_{r_{b_0}}) \leftarrow \text{Con-Term Expander}(\mathbb{S}(\mathbf{y}_r), \mathbf{c}, r, r_c, r_{b_0}, \{z\})$ 
5:      $CVT \leftarrow |\mathbb{S}_{\{z\}}(\mathbf{y}_{r_{b_0}})|$ 
6:      $Q.\text{push}((z, CVT))$ 
7:     if  $Q.\text{size}() > m$  then
8:        $Q.\text{pop}()$  ▷ remove the element with maximal  $CVT$ 
9:     end if
10:  end for
11:   $\mathcal{Z}_{\text{filtered}} \leftarrow \emptyset$ 
12:  while  $Q$  is not empty do
13:     $\mathcal{Z}_{\text{filtered}} \leftarrow \mathcal{Z}_{\text{filtered}} \cup \{Q.\text{top}().\text{var}\}$ 
14:     $Q.\text{pop}()$ 
15:  end while
16:  return  $\mathcal{Z}_{\text{filtered}}$ 
17: end function

```

Algorithm 14 Variable Elimination and Subset Selection

Input: $\mathcal{Z}_0 \leftarrow \emptyset$, $count \leftarrow 0$, cube indicator vector $\mathbf{c}$, monomials set $\mathbb{S}(\mathbf{y}_r)$ at function round r , output round r_s , condition round r_c .

Output: A set A .

```

1: Set parameters  $r_{b_0} > r_{b_1}$ ,  $n, m, s, l, T, T_0, T_1$ .  $A \leftarrow \emptyset$ 
2: procedure MAIN
3:   while  $count \leq l$  do
4:      $\mathcal{Z}_1 \leftarrow \emptyset$ ,  $count \leftarrow count + 1$ 
5:      $\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_1}}) = \text{Con-Term Expander}(\mathbb{S}(\mathbf{y}_r), \mathbf{c}, r, r_c, r_{b_1}, \mathcal{Z}_1)$ 
6:      $\mathbb{S} \leftarrow \text{SelectRandom}(\mathbb{S}_{\mathcal{Z}_0}(\mathbf{y}_{r_{b_1}}), T_1)$ 
7:      $superpoly \leftarrow \text{Nest-Con-CoeSolver}(\mathbb{S}, \mathbf{c}, r_{b_1}, r_s, \mathcal{Z}_1)$ 
8:      $\mathcal{Z}_0 \leftarrow \mathcal{Z}_0 \cup \text{TopNFrequent}(superpoly, n)$ 
9:      $\mathcal{Z}_1 \leftarrow \mathcal{Z}_1 \cup \text{TopNFrequent}(superpoly, n)$ 
10:     $\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_0}}) = \text{Con-Term Expander}(\mathbb{S}(\mathbf{y}_r), \mathbf{c}, r, r_c, r_{b_0}, \mathcal{Z}_1)$ 
11:     $CVT \leftarrow |\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_0}})|$ 
12:    while  $CVT > T_0$  do
13:       $\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_1}}) = \text{Con-Term Expander}(\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_0}}), \mathbf{c}, r_{b_0}, r_c, r_{b_1}, \mathcal{Z}_1)$ 
14:       $\mathbb{S} \leftarrow \text{SelectRandom}(\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_1}}), T_1)$ 
15:       $superpoly \leftarrow \text{Nest-Con-CoeSolver}(\mathbb{S}, \mathbf{c}, r_{b_1}, r_s, \mathcal{Z}_1)$ 
16:       $vars \leftarrow \text{TopNFrequent}(superpoly, n)$ 
17:      if  $vars = \emptyset$  then break
18:      end if
19:       $\mathcal{Z}_0 \leftarrow \mathcal{Z}_0 \cup vars$ 
20:       $\mathcal{Z}_1 \leftarrow \mathcal{Z}_1 \cup vars$ 
21:       $CVT \leftarrow |\mathbb{S}_{\mathcal{Z}_1}(\mathbf{y}_{r_{b_0}})|$ 
22:    end while
23:  end while
24:   $\mathcal{Z}_0 \leftarrow \text{FilterSubsets}(\mathcal{Z}_0, m, r_{b_0})$  ▷ This function is defined in 13
25:  for  $k = 1$  to  $s$  do
26:    for  $\mathcal{Z} \in \text{Subsets}(\mathcal{Z}_0, k)$  do ▷ enumerate k-subsets of  $\mathcal{Z}_0$ 
27:       $\mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_{b_0}}) = \text{Con-Term Expander}(\mathbb{S}(\mathbf{y}_r), \mathbf{c}, r, r_c, r_{b_0}, \mathcal{Z})$ 
28:      if  $|\mathbb{S}_{\mathcal{Z}}(\mathbf{y}_{r_{b_0}})| \leq T$  then
29:         $A \leftarrow A \cup \{\mathcal{Z}\}$ 
30:      end if
31:    end for
32:  end for
33:  output  $A$ 
34: end procedure
35: function SELECTRANDOM( $\mathbb{S}, t$ )
36:   return  $t$  random terms from  $\mathbb{S}$ 
37: end function
38: function TOPNFFREQUENT( $poly, n$ )
39:   return top  $n$  frequent eliminatable variables in  $poly$ 
40: end function

```

J Variable Substitution in Section 7.3

$$z_i = g_i(\mathbf{k}) = \begin{cases} k_i & i = 0, \dots, 79 \\ k_{147-i} \oplus 1 & i = 80, 81 \\ k_{53} \oplus k_{78}k_{79} \oplus 1 & i = 82 \\ k_{135-i} \oplus k_{160-i}k_{161-i} \oplus k_{162-i} & i = 83, \dots, 135 \\ k_{24}k_{25} \oplus k_{26} \oplus k_{68} & i = 136 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} \oplus 1 & i = 137, 138 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} & i = 139, \dots, 147 \\ k_{160-i}k_{161-i} \oplus k_{162-i} \oplus k_{204-i} \oplus k_{213-i} & i = 148, \dots, 152 \\ k_{64}k_{65} + z_{84} & i = 154 \\ k_6k_7 + k_8 + k_{50} + k_{59} & i = 155 \\ k_{63}k_{64} + k_{65} + z_{85} & i = 156 \\ k_5k_6 + k_7 + k_{49} + k_{58} & i = 157 \\ k_{62}k_{63} + k_{64} + z_{86} & i = 158 \\ k_4k_5 + k_6 + k_{48} + k_{57} & i = 159 \\ k_{61}k_{62} + k_{63} + z_{87} & i = 160 \\ k_3k_4 + k_5 + k_{47} + k_{56} & i = 161 \\ k_{60}k_{61} + k_{62} + z_{88} & i = 162 \\ k_2k_3 + k_4 + k_{46} + k_{55} & i = 163 \\ k_{59}k_{60} + k_{61} + z_{89} & i = 164 \\ k_1k_2 + k_3 + k_{45} + k_{54} & i = 165 \\ k_{58}k_{59} + k_{60} + z_{90} & i = 166 \\ k_0k_1 + k_2 + k_{44} + z_{82} & i = 167 \\ k_{57}k_{58} + k_{59} + z_{91} & i = 168 \\ k_1 + k_{43} + k_0k_{68} + z_{83} & i = 169 \\ k_{56}k_{57} + k_{58} + z_{92} & i = 170 \\ k_0 + k_{42} + k_{68}z_{80} + z_{84} & i = 171 \\ k_{55}k_{56} + k_{57} + z_{93} & i = 172 \end{cases}$$