IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 2, pp. 190–221. DOI:10.46586/tosc.v2024.i2.190-221

Dynamic Cube Attacks against Grain-128AEAD

Chen Liu and Tian Tian

Information Engineering University, Zhengzhou 450001, China
LC_LiuChen@126.com, tiantian_d@126.com

Abstract. In this paper, we revisit the division property based dynamic cube attack on the full Grain-128 presented by Hao et al. at FSE 2020 and demonstrate that their attack on the full Grain-128 is invalid, that is, no key information could be successfully recovered. The theoretical framework for the dynamic cube attack provided by Hao et al. is correct, but the technique for building the MILP model in the dynamic cube attack has flaws. Besides, strong evidence indicates that their bias estimation method is not applicable to Grain-128AEAD and Grain-128. Accordingly, we introduce the three-subset division property without unknown subset (3SDP/u) into dynamic cube attacks and present a correct MILP modeling technique. In addition, we propose a heuristic technique called Polynomial Approximation with regard to Bias (PAB) to evaluate the bias in superpolies in the dynamic cube attack, which can provide a more accurate bias evaluation for high-dimension cubes. As a result, we implemented the dynamic cube attack based on 3SDP/u on 190-round Grain-128AEAD, and we could recover 3 key bits with a complexity $2^{103.44}$ and the success probability was evaluated to be 99.68%. For Grain-128, some zero-sum distinguishers of cube size 80 are given for the first time.

Keywords: Dynamic Cube Attacks · Division Property · MILP · Grain-128AEAD · Grain-128

1 Introduction

Since the cube attack was first introduced by Dinur and Shamir in [DS09], it has become an important cryptanalysis tool for stream ciphers. In cube attacks, the first output bit of a stream cipher can be regarded as a black-box polynomial $f(\boldsymbol{k}, \boldsymbol{v})$ with respect to the secret key variables $\boldsymbol{k}$ and public IV variables $\boldsymbol{v}$. Given a non-empty subset I of IV indices, $f(\boldsymbol{k}, \boldsymbol{v})$ can be rewritten uniquely as

$$f(\boldsymbol{k}, \boldsymbol{v}) = \boldsymbol{v}_I \cdot p_I(\boldsymbol{k}, \boldsymbol{v}) \oplus q_I(\boldsymbol{k}, \boldsymbol{v}),$$

where $\boldsymbol{v}_I = \prod_{i \in I} v_i$ and each term of $q_I(\boldsymbol{k}, \boldsymbol{v})$ is not divisible by $\boldsymbol{v}_I$. The IV variables in $\{v_i \mid i \in I\}$ are called cube variables. In the preprocessing phase, by assigning all possible combinations of 0/1 values to the cube variables, $2^{|I|}$ polynomials are obtained from f, and their symbolic sum p_I is called the superpoly of I in f. Generally, the non-cube variables are set to 0-constant in specific cube attacks, and in this case, p_I is a polynomial in $\boldsymbol{k}$. Cube attacks aim to find proper sets I such that the superpolies $p_I(\boldsymbol{k})$ are balanced polynomials with low-degree, which can provide 1-bit secret key information theoretically. The equations on the secret key variables could be built by inquiring about the value of $p_I(\boldsymbol{k})$ online. Finally, some key information could be restored by solving these equations. Additionally, a distinguisher variant of cube attacks, referred to as *cube tester*, was first

This work was supported by the National Natural Science Foundation of China under Grant No. 62372464.

Received: 2024-03-01 Accepted: 2024-05-01 Published: 2024-06-18

introduced in [ADMS09], which is used to detect non-random properties of a superpoly of a carefully selected cube such as constantness, linearity, and bias.

The dynamic cube attack, a variant of cube attacks proposed in [DS11], serves as a potent cryptanalytic technique extensively applied in the evaluation of security in the Grain family [DS11, DGP+11, RBMA16, HJL+20]. Unlike cube attacks, dynamic cube attacks recover secret key information by exploiting distinguishers on superpolies, such as bias and constantness. The basic idea of dynamic cube attacks is to simplify or even nullify the ANFs of several critical intermediate state bits by imposing dynamic constraints on some IV variables called dynamic variables, which aims to simplify the ANF of the output function and induce non-random properties to the corresponding superpoly. The dynamic constraints typically depend on some cube variables and key expressions. By guessing the values of key expressions (or secret key variables) involved in the constraints, the correct guesses can be obtained by detecting the non-randomness of the corresponding superpoly. Ideally, under a correct key guess, the corresponding superpoly will show a kind of non-randomness; conversely, under wrong key guesses, the corresponding superpolies behave randomly. By using the dynamic cube attack, Dinur and Shamir proposed a weak-key attack on the full version of Grain-128 in [DS11]. Subsequently, Dinur et al. further proposed a key-recovery attack on the full version of Grain-128 in [DGP+11] and experimentally verified the correct guesses for a dozen of keys. By adjusting the strategy of imposed conditions, Rahimi et al. presented a key-recovery attack against 100-round Grain v1 in [RBMA16].

In [HJL+20], Hao et al. argued that a theoretically reliable key-recovery attack should evaluate not only the **non-randomness** for the correct key guess but also the **randomness** for the wrong ones. Based on this idea, they pointed out that previous dynamic cube attacks are **unreliable**, as these attacks typically overlooked the empirical or theoretical validation of the randomness of the superpoly under wrong key guesses.

To solve this problem, Hao et al. applied the division property to dynamic cube attacks and proposed a new dynamic cube attack (which we call division property based dynamic cube attack) along with the corresponding MILP[1] modeling technique. By applying the division property and the MILP modeling technique, large cubes can be exploited to construct zero-sum distinguishers in dynamic cube attacks, which is very useful for distinguishing correct guesses from incorrect guesses. To achieve this goal, a "qualified" cube must satisfy the following requirements: for the correct key guess, the cube should introduce a zero-sum property (i.e., the bias of the corresponding superpoly is 2^{-1}); for each wrong key guess, the superpoly should be of high degree with a bias smaller than 2^{-1}. However, using high-dimensional cubes makes detecting the bias through experimental testing infeasible since the computation complexity exceeds the available computational capabilities. Therefore, to evaluate biases in superpolies of high-dimensional cubes, Hao et al. drew links between the bias and the division property through a specific algebraic structure and a heuristic assumption. Based on the links, they proposed a theoretical method for estimating the biases of superpolies. Strictly speaking, it is necessary to evaluate the biases of the superpolies for all possible wrong guesses. But, it is impractical to do this due to a large number of wrong key guesses. Therefore, the authors proposed a method to pick up a special wrong key guess that is expected to have a bias that is closest to the bias of the correct key guess. Thus, the dynamic cube attack is deemed to be successful if the correct key guess can be distinguished from the special wrong key guess. Finally, the authors in [HJL+20] applied the division property based dynamic cube attack to the full Grain-128 and recovered about 3-bit key information with a complexity of $2^{97.86}$ and a success probability of 99.83%. It is currently the best theoretical key recovery attack for the full Grain-128.

[1]Short for Mixed Integer Linear Programming, which is widely applied to various cryptanalysis techniques

Motivation. In recent years, several cube attacks have been proposed against Grain-128AEAD [HLM+20, HLM+21, HST+21, HHPW22]. These attacks aim to recover a superpoly for more initialization rounds. But, as the number of initialization rounds increases, the recovered superpolies become highly complex and exhibit significant bias. Such superpolies are considered "bad" in cube attacks because, theoretically, they can only provide key information of less than 1 bit, leading to an attack with a computation complexity close to that of the exhaustive search attack. As a comparison, the dynamic cube attack makes use of biased superpolies without recovering their full algebraic normal forms that are typically complex, and it has theoretical complexity significantly lower than that of the exhaustive search attack when applied to Grain-128. Besides, some dynamic cube attacks against Grain-128 could even experimentally recover some key bits. Thus, the dynamic cube attack seems to be a potentially powerful attack against Grain-128AEAD as well as Grain-128.

When we revisited the dynamic cube attack against the full version of Grain-128 proposed in [HJL+20], we found that the attack has some issues which results in that the key-recovery result against the full Grain-128 is invalid. The primary issue is that the MILP modeling technique for the dynamic cube attack did not consider the structural characteristics of the Grain family, making the established MILP models contain many redundant division trails. Therefore, the cubes constructed by the modeling technique and deemed "qualified" failed to meet the requirements for distinguishing between correct and wrong key guesses. Another issue is that the significant gap between the estimated and actual biases could also affect the effectiveness of the attack. The method for estimating bias proposed in [HJL+20] relies on a heuristic assumption. The accuracy of the estimated bias obtained using this method depends on the validity of this assumption. However, based on many experiments, we found that the assumption is not universally applicable to the Grain family of ciphers as the estimated bias using the method in [HJL+20] is significantly smaller than the actual bias in our experiments.

Contributions. In this paper, we focus on improving the dynamic cube attack against Grain-128AEAD. As mentioned above, the inaccuracies in the MILP modeling technique and the bias evaluation method lead to the key-recovery result in [HJL+20] against the full Grain-128 being invalid. Our main contribution is to present a more robust dynamic cube attack against the Grain family of ciphers both in implementation and theory. In specific, our contribution consists of the following three aspects.

Firstly, we introduce a heuristic technique called Polynomial Approximation with respect to Bias (PAB), which can provide a more accurate bias evaluation for a superpoly of a high-dimensional cube. Compared to the method proposed in [HJL+20], the PAB technique leverages the algebraic properties of the Grain family, enabling more precise bias estimation. Secondly, we apply the 3SDP/u to dynamic cube attacks for the first time and propose a more accurate MILP modeling method based on the 3SDP/u, which takes the structural characteristics of the Grain family into consideration. Thirdly, a method was proposed to analyze the impact of wrong key guesses on superpolies from an algebraic perspective. This method aims to identify all possible wrong key guesses that are least distinguishable from the correct key guess. Unlike the method in [HJL+20], our approach has more theoretical foundation to support its correctness.

As an illustration, we apply our methods to Grain-128AEAD and Grain-128 and obtain the following results.

1. We present a key-recovery attack against 190-round Grain-128AEAD with theoretical analysis of the success probability, which is the first dynamic cube attack against Grain-128AEAD. In our attack on Grain-128AEAD, we identify the nullification strategy by employing an automated search algorithm. It only nullifies two internal state bits by guessing 6 key bits. As a result, our attack provides 3 bits of key information with a complexity of $2^{103.44}$ and a theoretically estimated success probability

of 99.68%.

We summarized the previous and our attack results for Grain-128AEAD in Table 1. Compared with Hao et al.'s cube attack in [HLM+21], our attack can be regarded as a more "practical" key recovery attack since we can independently recover 3 key bits. In [HLM+21], it is explained from the perspective of entropy that 15 recovered superpolies with high biases can provide 5.03 bits of key information, which is a "theoretical" analysis and how to solve theses 5.03 bits of key information in practice is unknown.

2. Based on the new modeling technique, we further discuss the feasibility of Hao et al.'s dynamic cube attack against the full Grain-128. Through a large number of experiments, we claim that within the framework of dynamic cube attacks proposed in [HJL+20], it is challenging to execute an effective key-recovery attack against Grain-128.

3. We compare the accuracy of our bias estimation method with the previous method given in [HJL+20] by performing experiments on the reduced-round Grain-128AEAD. It can be seen from Table 10 that the accuracy of our new bias evaluation method is much better than the previous method for Grain-128AEAD. Besides we also give the main reasons for the inaccuracies in estimating bias with the previous method.

All our results and the source code are available in our git repository.

Table 1: Summary of the attack results for round-reduced Grain-128AEAD

Attack type	round r	Key information available	Computation complexity	Ref.
Dynamic cube attacks	190	3	$2^{103.44}$	Section 6.1
Cube attacks	190	5.03	$2^{98.91}$	[HLM+21]
Cube attacks	191	128[1]	$2^{126.26}$	[HST+21]
Cube attacks	192	128[2]	2^{127}	[HHPW22]

[1]They recovered two massive superpolies with a high-bias, which are considered to be able to provide 1.7-bit key information from the perspective of entropy.
[2]They recovered one massive superpoly, which is proved balanced through experimental testing and is considered capable of providing 1-bit key information.

2 Preliminaries

2.1 Notations

Let n be a positive integer. An n-variable Boolean function $f(x_0, x_1, \ldots, x_{n-1})$ is a mapping from $\mathbb{F}_2^n$ into $\mathbb{F}_2$, and we also denote it by $f(\boldsymbol{x})$ for simplicity. It can be uniquely written as

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{\mu}\in\mathbb{F}_2^n} a_{\boldsymbol{\mu}}\boldsymbol{x}^{\boldsymbol{\mu}}, a_{\boldsymbol{\mu}} \in \mathbb{F}_2, \tag{1}$$

where $\boldsymbol{x}^{\boldsymbol{\mu}} = \prod_{i=0}^{n-1} x_i^{\mu_i} \triangleq \pi_{\boldsymbol{\mu}}(\boldsymbol{x})$, which is called the Algebraic Normal Form (ANF) of $f(\boldsymbol{x})$. A Boolean function of the form $\boldsymbol{x}^{\boldsymbol{\mu}}$ for some vector $\boldsymbol{\mu} \in \mathbb{F}_2^n$ is called a monomial. In particular, $1 = \boldsymbol{x}^0$ is a monomial. By Eq.(1), if $a_{\boldsymbol{\mu}} = 1$, then we say the monomial $\boldsymbol{x}^{\boldsymbol{\mu}}$ appears in $f(\boldsymbol{x})$, denoted by $\boldsymbol{x}^{\boldsymbol{\mu}} \to f$; otherwise denote $\boldsymbol{x}^{\boldsymbol{\mu}} \nrightarrow f$. The algebraic degree of $f(\boldsymbol{x})$ is defined by

$$\deg(f) = \max\{wt(\boldsymbol{\mu}) | \boldsymbol{x}^{\boldsymbol{\mu}} \to f\},$$

where $wt(\boldsymbol{\mu})$ is the Hamming Weight of $\boldsymbol{\mu}$, and is denoted by $\deg(f)$. A vector of length 2^n consisting of all the outputs of $f(\boldsymbol{x})$ is called the truth table of f. The quantity ε_f such that

$$\Pr(f(\boldsymbol{x}) = 0) \triangleq \frac{|\{\boldsymbol{x} \in \mathbb{F}_2^n \mid f(\boldsymbol{x}) = 0\}|}{2^n} = \frac{1}{2} + \varepsilon_f,$$

is called the bias of $f(\boldsymbol{x})$, which describes the 0/1 distribution of the truth table of $f(\boldsymbol{x})$. It can be seen that $-\frac{1}{2} \leq \varepsilon_f < \frac{1}{2}$. If $\varepsilon_f = 0$, then the number of 0's is equal to that of 1's in the truth table of f, and in this case, f is called a balanced Boolean function. If $\varepsilon_f > 0$, then the number of 0's is larger than that of 1's, and in this case, the output of f is more likely to be 0 for a random input. In particular, if ε_f is close to $\frac{1}{2}$, then f is close to being the zero constant function.

Beside Boolean functions, we need to introduce another type of polynomial ring. Let $\mathbb{Z}[x_0, \ldots, x_{n-1}]$ be the polynomial ring in n variables over the integer ring $\mathbb{Z}$ and $(x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1})$ be the ideal in $\mathbb{Z}[x_0, \ldots, x_{n-1}]$ generated by $\{x_i^2 - x_i \mid 0 \leq i \leq n-1\}$. Then the quotient ring $\mathbb{Z}[x_0, \ldots, x_{n-1}]/(x_0^2 - x_0, \ldots, x_{n-1}^2 - x_{n-1})$ is a commutative ring with identity, and we denote it by $\bar{\mathbb{Z}}[\boldsymbol{x}]$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$. Since $x_i^2 = x_i$ in $\bar{\mathbb{Z}}[\boldsymbol{x}]$, it follows that every polynomial $f(\boldsymbol{x}) \in \bar{\mathbb{Z}}[\boldsymbol{x}]$ can be written uniquely as

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{\mu} \in \mathbb{F}_2^n} \alpha_{\boldsymbol{\mu}} \boldsymbol{x}^{\boldsymbol{\mu}}, \alpha_{\boldsymbol{\mu}} \in \mathbb{Z}.$$

2.2 Frequency test to Boolean functions

Let $f(\boldsymbol{x})$ be an n-variable random Boolean function, that is, its truth table contains as many zeroes as ones. This implies that $\Pr(f(\boldsymbol{x}) = 1) = \Pr(f(\boldsymbol{x}) = 0) = \frac{1}{2}$ for a uniformly distributed discrete random variable $\boldsymbol{x}$ over $\mathbb{F}_2^n$. Let $\mathbb{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a non-empty subset of $\mathbb{F}_2^n$. Assume f is random on $\mathbb{X}$ (that is, $f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_N)$ are random samples from the truth table of f). Denote the number of 1's in the N-tuple $(f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_N))$ by S. Then the central limit theorem implies that for a sufficiently large N, the distribution of

$$y = \frac{S - N/2}{\sqrt{N/4}} = \frac{2S - N}{\sqrt{N}}$$

is approximately the standard normal distribution $\mathcal{N}(0, 1)$. Generally, when N is greater than 25 or 30, this approximation will be good [HTZ13, Chapter 5.6]. Next, we devise a test for random Boolean functions. Choose a significance level $0 < \alpha \leq 0.05$. If $\Phi(|y|) \leq 1 - \frac{\alpha}{2}$, then we say the function f passes the frequency test for randomness with a significance level α, where Φ is the distribution function of $\mathcal{N}(0, 1)$, that is,

$$\Phi(|y|) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{|y|} e^{-\frac{u^2}{2}} \mathrm{d}u.$$

Otherwise, we say $f(\boldsymbol{x})$ fails the frequency test for randomness with the significance level α and in this case we also say f is **biased**.

Finally, we discuss the sample size N in the above frequency test to detect a Boolean function f' with bias $\varepsilon_{f'} > 0$. The bias implies that $\Pr(f'(\boldsymbol{x}) = 1) = \frac{1}{2} - \varepsilon_{f'}$ for a random input $\boldsymbol{x} \in \mathbb{F}_2^n$. Find the number u_α such that $\Phi(u_\alpha) = 1 - \frac{\alpha}{2}$. If we want f' to fail the frequency test for randomness, then the sample size N should be the solution of

$$\frac{N(\frac{1}{2} - \varepsilon_{f'}) - \frac{N}{2}}{\sqrt{\frac{N}{4}}} < -u_\alpha,$$

that is,

$$N > \frac{u_\alpha^2}{(2\varepsilon_{f'})^2}. \tag{2}$$

Thus, we can detect that f' is biased by the frequency test for randomness with the significance level α when the size of $\mathbb{X}$ is at least $(2\varepsilon_{f'})^{-2} \cdot u_\alpha^2$. To make the assertion for a Boolean function being biased more reliable, in the following we set $N = 2 \cdot (2\varepsilon_{f'})^{-2} \cdot u_\alpha^2$.

2.3 Three-subset division property without unknown subset

The concept of the *three-subset division property without unknown subset* (shortly 3SDP/u) was proposed by Hao et al. at Eurocrypt 2020 in [HLM+20], which is widely applied to cube attacks against stream ciphers. The 3SDP/u is one of the best methods to resolve the problem of determining the presence or absence of a monomial in the ANF of a Boolean function. In [HLM+20], the definition of the 3SDP/u was not given directly, but an equivalent definition called the *modified three-subset division property* was proposed.

Definition 1 (Modified three-subset division property)**.** Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^N$. Let $\mathbb{L}$ also be a **multiset** whose elements take a value of $\mathbb{F}_2^N$. When the multiset $\mathbb{X}$ has the modified three-subset division property (shortly $\mathcal{T}_{\mathbb{L}}^{1^N}$), it fulfils the following conditions:

$$\bigoplus_{\boldsymbol{x}\in\mathbb{X}} \boldsymbol{x}^{\boldsymbol{\mu}} = \begin{cases} 1 & \text{if there is an odd number of } \boldsymbol{\mu}\text{'s in } \mathbb{L}, \\ 0 & \text{otherwise.} \end{cases}$$

Generally, an iterative stream cipher can be decomposed into a composition of basic vectorial Boolean functions: COPY, AND, and XOR. We present the propagation rules of the 3SDP/u for these basic vectorial Boolean functions below and provide their MILP models in Appendix A.

Proposition 1 (COPY)**.** *Let $\boldsymbol{x} = (x_0, x_1, ..., x_{m-1})$ and $\boldsymbol{y} = (x_0, x_0, x_1, ..., x_{m-1})$ be the input and output vector of a Copy function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\mathbb{L}'}^{1^{m+1}}$ where $\mathbb{L}'$ is computed as*

$$\mathbb{L}' \leftarrow (l_0', l_0'', l_1, \ldots, l_{m-1}), \text{ if } l_0' \vee l_0'' = l_0,$$

for all $\boldsymbol{l} \in \mathbb{L}$. Note that the operator $\vee$ represents the OR operation, and here $\mathbb{L}' \leftarrow \boldsymbol{l}$ denotes that $\boldsymbol{l}$ is inserted into the multiset $\mathbb{L}'$.

Proposition 2 (AND)**.** *Let $\boldsymbol{x} = (x_1, x_2, ..., x_m)$ and $\boldsymbol{y} = (x_1 \cdot x_2, x_3, ..., x_m)$ be the input and output vector of an And function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\mathbb{L}'}^{1^{m-1}}$ where $\mathbb{L}'$ is computed as*

$$\mathbb{L}' \leftarrow (l_0, l_2, \ldots, l_{m-1}), \text{ if } l_0 = l_1,$$

for all $\boldsymbol{l} \in \mathbb{L}$.

Proposition 3 (XOR)**.** *Let $\boldsymbol{x} = (x_1, x_2, ..., x_m)$ and $\boldsymbol{y} = (x_1 \oplus x_2, x_3, ..., x_m)$ be the input and output vector of a Xor function. Let $\mathbb{X}$ and $\mathbb{Y}$ be the input and output multisets, respectively. Assuming that $\mathbb{X}$ has $\mathcal{T}_{\mathbb{L}}^{1^m}$, $\mathbb{Y}$ has $\mathcal{T}_{\mathbb{L}'}^{1^{m-1}}$ where $\mathbb{L}'$ is computed as*

$$\mathbb{L}' \leftarrow (l_0 + l_1, l_2, \ldots, l_{m-1}), \text{ if } l_0 \cdot l_1 = 0,$$

for all $\boldsymbol{l} \in \mathbb{L}$.

To determine whether a given monomial appears in the ANF of the output function for an iterative cryptosystem with r-round initialization, the authors in [HLM+20] defined the concept of *three-subset division trail*.

Definition 2 (Three-subset division trail [HLM+20])**.** Let $\mathcal{T}_{\mathbb{L}_i}$ be the three-subset division property of the input for the ith round function. Let us consider the propagation of the three-subset division property $\{\boldsymbol{l}\} \overset{def}{=} \mathbb{L}_0 \to \mathbb{L}_1 \to \cdots \to \mathbb{L}_r$. Moreover, for any bit-vector $\boldsymbol{l}_{i+1}' \in \mathbb{L}_{i+1}$, there must exist a bit-vector $\boldsymbol{l}_i' \in \mathbb{L}_i$ such that $\boldsymbol{l}_i'$ can propagate to $\boldsymbol{l}_{i+1}'$ by the propagation rules of the 3SDP/u, denoted by $\boldsymbol{l}_i' \xrightarrow{p} \boldsymbol{l}_{i+1}'$. Furthermore, for $(\boldsymbol{l}_0, \boldsymbol{l}_1, \ldots, \boldsymbol{l}_r) \in \mathbb{L}_0 \times \mathbb{L}_1 \times \cdots \times \mathbb{L}_r$, if $\boldsymbol{l}_i$ can propagate to $\boldsymbol{l}_{i+1}$ for all $i \in \{0, 1, \ldots, r-1\}$, we call $\boldsymbol{l}_0 \xrightarrow{p} \boldsymbol{l}_1 \xrightarrow{p} \cdots \xrightarrow{p} \boldsymbol{l}_r$ an r-round three-subset division trail.

Let $f(\boldsymbol{x})$ be a basic vectorial Boolean function, and $\boldsymbol{y} = f(\boldsymbol{x})$. For an input bit-vector $\boldsymbol{\mu}$, the corresponding output bit-vector $\boldsymbol{\omega}$ can be derived using the propagation rules of the 3SDP/u. In this case, the monomial $\boldsymbol{x}^{\boldsymbol{\mu}}$ appears in the ANF of $\boldsymbol{y}^{\boldsymbol{\omega}}$ with respect to $\boldsymbol{x}$, the correctness of which is guaranteed by the propagation rules.

However, when the function $f(\boldsymbol{x})$ is not a basic vectorial Boolean function, $\boldsymbol{\mu} \xrightarrow{p} \boldsymbol{\omega}$ does not necessarily mean that the monomial $\boldsymbol{x}^{\boldsymbol{\mu}}$ appears in the ANF of $\boldsymbol{y}^{\boldsymbol{\omega}}$ with respect to $\boldsymbol{x}$. It is implied that the monomial $\boldsymbol{x}^{\boldsymbol{\mu}}$ appears in the expression of $\boldsymbol{y}^{\boldsymbol{\omega}}$ with respect to $\boldsymbol{x}$, under the condition that the expression does not consider the cancellation property of the operation xor. This is because a Boolean function can typically be decomposed into a composition of basic vectorial Boolean functions, and there are more than one possible ways for propagation according to the rule of COPY, rendering μ propagate to $\boldsymbol{\omega}$ through several different ways. Assuming that there are a total of N different trails connecting $\boldsymbol{\mu}$ and $\boldsymbol{\omega}$, it implies that there are N monomials $\boldsymbol{x}^{\boldsymbol{\mu}}$ appearing in the expression of $\boldsymbol{y}^{\boldsymbol{\omega}}$ with respect to $\boldsymbol{x}$, without taking into account the cancellation property. Further, if N is even, then the monomial $\boldsymbol{x}^{\boldsymbol{\mu}}$ does not appear in the ANF of $\boldsymbol{y}^{\boldsymbol{\omega}}$ with respect to $\boldsymbol{x}$ due to the cancellation property; otherwise, $\boldsymbol{x}^{\boldsymbol{\mu}}$ appears in the ANF of $\boldsymbol{y}^{\boldsymbol{\omega}}$. To illustrate more clearly, we provide a simple example.

Example 1. Let $\boldsymbol{x} = (x_0, x_1, x_2) \in \mathbb{F}_2^3$ and $\boldsymbol{y} = f(\boldsymbol{x}) = (x_0 \oplus x_1 \oplus x_2, x_1x_2, x_0)$. The function f can be decomposed into the following procedure, say

$$\begin{aligned}(x_0, x_1, x_2) &\xrightarrow{COPY} (x_0, x_1, x_2, x_1, x_2, x_0),\\ &\xrightarrow{XOR} (x_0 \oplus x_1 \oplus x_2, x_1, x_2, x_0),\\ &\xrightarrow{AND} (x_0 \oplus x_1 \oplus x_2, x_1x_2, x_0) \triangleq (y_0, y_1, y_2).\end{aligned}$$

To check whether $\boldsymbol{x}^{(1,1,1)}$ appears in $\boldsymbol{y}^{(1,1,1)} = x_0x_1x_2$, we need to enumerate all possible three-subset division trails connecting $\boldsymbol{x}^{(1,1,1)}$ and $\boldsymbol{y}^{(1,1,1)}$. By computation, there are in total 3 trails as follows

$$\begin{aligned}(1,1,1) &\xrightarrow{p} (1,0,0,1,1,1) \xrightarrow{p} (1,1,1),\\ (1,1,1) &\xrightarrow{p} (0,1,0,1,1,1) \xrightarrow{p} (1,1,1),\\ (1,1,1) &\xrightarrow{p} (0,0,1,1,1,1) \xrightarrow{p} (1,1,1).\end{aligned}$$

Therefore, the monomial $\boldsymbol{x}^{(1,1,1)}$ appears in the ANF of $\boldsymbol{y}^{(1,1,1)}$ with respect to $\boldsymbol{x}$.

Conversely, there are only 2 three-subset division trails connecting $\boldsymbol{x}^{(0,1,1)}$ and $\boldsymbol{y}^{(1,1,0)} = (x_0 \oplus x_1 \oplus x_2) \cdot x_1x_2 = x_0x_1x_2$ given by

$$\begin{aligned}(0,1,1) &\xrightarrow{p} (0,1,0,1,1,0) \xrightarrow{p} (1,1,0),\\ (0,1,1) &\xrightarrow{p} (0,0,1,1,1,0) \xrightarrow{p} (1,1,0).\end{aligned}$$

Therefore, the monomial $\boldsymbol{x}^{(0,1,1)}$ does not appear in the ANF of $\boldsymbol{y}^{(1,1,0)}$ with respect to $\boldsymbol{x}$.

Similarly, for an iterated stream cipher with r-round initialization $f(\boldsymbol{k}, \boldsymbol{v})$, if we want to determine whether a monomial $\pi_{\boldsymbol{l}}(\boldsymbol{s}_0)$ appears in f, it is necessary to enumerate all possible three-subset division trails connecting $\pi_{\boldsymbol{l}}(\boldsymbol{s}_0)$ and f. When the number of trails is odd, the monomial $\pi_{\boldsymbol{l}}(\boldsymbol{s}_0)$ appears in the ANF of f; otherwise, $\pi_{\boldsymbol{l}}(\boldsymbol{s}_0)$ does not appear in the ANF of f.

Based on the propagation rules of these basic functions and the concept of *three-subset division trail*, Hao et al. proposed a general modeling technique to recover the superpolies against the stream ciphers in [HLM+20, HLM+21], which is a basic tool for all superpoly recovery algorithms proposed after it. For details on the corresponding MILP models for specific stream ciphers, please refer to [HLM+20, HLM+21].

2.4 A divide-and-conquer algorithm for superpoly recovery

A divide-and-conquer algorithm for recovering superpolies was proposed in [Sun21, HST+21]. Its basic idea is to decompose the complex system that failed to be solved within a time limit, into several simpler subsystems for resolution by employing a divide-and-conquer strategy based on the algebraic expression of the internal update function. If any subsystems remain unsolved within a time limit, they are further decomposed into even simpler subsystems until all subsystems are successfully resolved.

In specific, expressing the output bit z as a polynomial of some intermediate state variables (not necessarily at the same time instance), say $z = f(s_1, s_2, \ldots, s_m)$, and then the process of recovering the superpoly of the output bit z for a given cube I could be divided into recovering the superpoly of every monomial of f for the cube I. That is, if $f(s_1, s_2 \ldots, s_m)$ has N monomials, then the MILP model to recover the superpoly of z could be decomposed into N smaller MILP models as N branches for recovering the superpolies of N monomials. The MILP models for some branches could be solved very efficiently, such as 0-constant superpolies, while the MILP models for some branches are time-consuming. For those branches with MILP models that are hard to resolve, a preset time limit is used to terminate them. Therefore, for each monomial of f, if the corresponding MILP model is solved within the time limit, then the superpoly of the monomial has been recovered; otherwise, it indicates that the superpoly of the monomial is still unknown. Then, each monomial with an unsolved MILP model is further divided into more branches by expressing it as polynomials of deeper intermediate states. The above procedure is iterated until all monomials are resolved. Finally, all the superpolies are collected and assembled into the whole superpoly of the output bit.

3 Revisiting Hao et al.'s dynamic cube attack

In this section, we will first provide a brief introduction to Hao et al.'s dynamic cube attack in Section 3.1 and then discuss the issues in their attack on the full Grain-128 in Sections 3.2 and 3.3, respectively.

3.1 Brief introduction to the dynamic cube attack

In this subsection, we briefly introduce the division property based dynamic cube attack presented by Hao et al. in [HJL+20]. For a stream cipher with n-bit key variables $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$ and m-bit public IV variables $\boldsymbol{v} = (v_0, \ldots, v_{m-1})$, the ANF of its arbitrary intermediate state bit can be regarded as a polynomial $s(\boldsymbol{x}, \boldsymbol{v})$ as follows,

$$s(\boldsymbol{x}, \boldsymbol{v}) = \sum_{\boldsymbol{\mu} \in \mathbb{F}_2^m} \alpha_{\boldsymbol{\mu}} \boldsymbol{v}^{\boldsymbol{\mu}}, \text{ where } \alpha_{\boldsymbol{\mu}} \in \mathbb{F}_2[\boldsymbol{x}].$$

Naturally, the output bit can be represented as a polynomial involving the intermediate state bits. Hence, if we can identify crucial intermediate state bits for the output bit and nullify them, then the output bit can be significantly simplified, making it vulnerable to cube testers. An intermediate state bit $s_i^{(r)}$, generated at round r and position i, can be nullified by assigning specific IV variables to a dynamic value determined by the ANF of $s_i^{(r)}$. Assume that $s_i^{(r)}$ can be expressed as

$$s_i^{(r)} = s_i^{(r)}(\boldsymbol{x}, \boldsymbol{v}) = v_l \oplus g(\boldsymbol{x}, \boldsymbol{v}),$$

where each monomial in g does not contain the variable v_l. Then, the state bit $s_i^{(r)}$ can be nullified by assigning $g(\boldsymbol{x}, \boldsymbol{v})$ to v_l. Let $\boldsymbol{v'} = (v_0, \ldots, v_{l-1}, g, v_{l+1}, \ldots, v_{m-1})$ be a vector derived from $\boldsymbol{v}$. The state bit $s_i^{(r)}$ is nullified for $s_i^{(r)} = s_i^{(r)}(\boldsymbol{x}, \boldsymbol{v'}) = 0$. The expression g is

referred to as the dynamic value and the variable v_l is referred to as the dynamic variable. If there are several crucial state bits $s_{i_1}^{(r_1)}, \ldots, s_{i_t}^{(r_t)}$ (we always assume $r_1 \leq \cdots \leq r_t$ hereafter) to be nullified, then the dynamic variables $v_{l_1}, \ldots, v_{l_t}$ need to be replaced by the corresponding dynamic values $g_1, \ldots, g_t$ recursively. Since the dynamic values are uniquely determined by the pairs in $N_S = \{(v_{l_1}, s_{i_1}^{(r_1)}), \ldots, (v_{l_t}, s_{i_t}^{(r_t)})\}$, the authors referred to the set N_S as the "nullification strategy".

Once a nullification strategy N_S and a cube I are confirmed, we can identify a set of key expressions (including key variables) that need to be guessed. By guessing the values of these key expressions, we can obtain a set of dynamic values. To-be-guessed key expressions are denoted as $\boldsymbol{G} = (g_0, g_1, \ldots, g_{\kappa-1})$ where $g_j \in \mathbb{F}_2[\boldsymbol{x}]$ for $j \in \{0, \ldots, \kappa - 1\}$. There are a total of 2^κ different key guesses, denoted by $\boldsymbol{G}_\mu = (g_0 \oplus \mu[0], \ldots, g_{\kappa-1} \oplus \mu[\kappa - 1])$, where $\mu \in \{0, \ldots, 2^\kappa - 1\}$ and $\mu[i]$ represents the ith bit of μ. Obviously, $\mu = 0$ is the correct key guess. For a predefined nullification strategy, it is necessary to find a batch of cubes that satisfy the following criteria: the corresponding superpolies should be 0-constant polynomials (i.e., the bias is 2^{-1}) under the correct key guess; under any wrong key guess, the corresponding superpolies should be high-degree polynomials with a bias less than 2^{-1}. Such cubes can effectively differentiate between correct and wrong key guesses, thus recovering key information.

There are two important contributions in [HJL+20] for the dynamic cube attack. One is applying the conventional bit-based division property (CBDP) with the "flag" technique to the dynamic cube attack and proposing a new MILP modeling method. The MILP model established by this method can be used to estimate the algebraic degree of the superpoly for a given cube, provided that the nullification strategy is predefined and the assignment of the non-cube IV is fixed. The models can not only describe the zero-sum (or bias) for the correct key guess but also analyze the randomness in the wrong guesses. This modeling method forms the basis for division property based dynamic cube attacks. See [HIJ+19, HJL+20] for more details about the propagation rules of CBDP and its MILP modeling techniques.

The other is drawing links between the bias and the division property. To achieve this, two new concepts were proposed: *split set* and *minimal split set*, along with an assumption. They found that the bias of a superpoly is closely related to the split set and further proved that the minimal split set can draw lower bounds on the bias in superpolies under the given assumption. Based on these two new concepts and the assumption, a new method was proposed to theoretically estimate the bias for superpolies of large cubes.

Definition 3. ([HJL+20]) Let $p_I(\boldsymbol{k})$ be a superpoly. Let Λ be a subset of secret key indices. It defines a key class $W_\Lambda \subset \mathbb{F}_2^n$ as follows:

$$W_\Lambda = \{\boldsymbol{x} \in \mathbb{F}_2^n | x_i = 0 \text{ for all } i \in \Lambda\}.$$

Such Λ is referred to as a **split set** if the superpoly $p_I(\boldsymbol{k})$ satisfies $p_I(\boldsymbol{x}) \equiv 0$ for all $\boldsymbol{x} \in W_\Lambda$. Further, we call this Λ a **minimal split set** if, for all Λ' with size $|\Lambda'| < |\Lambda|$ and the corresponding key class $W_{\Lambda'}$, there is $p_I(\boldsymbol{x}) \not\equiv 0$ for $\boldsymbol{x} \in W_{\Lambda'}$.

Assumption 1. *([HJL+20]) For a minimal split set Λ and its corresponding key class W_Λ, we assume*

$$\Pr(p_I(\boldsymbol{x}) = 0 | \boldsymbol{x} \in \overline{W}_\Lambda) = 2^{-1}, \text{ where } \overline{W}_\Lambda \triangleq \mathbb{F}_2^n \backslash W_\Lambda.$$

For a superpoly $p_I(\boldsymbol{k})$, let Λ be a minimal split set of $p_I(\boldsymbol{k})$. Then it follows from Definition 3 and Assumption 1 that

$$\Pr(p_I(\boldsymbol{k}) = 0) = 2^{-1} + 2^{-(|\Lambda|+1)},$$

and so the bias of $p_I(\boldsymbol{k})$ is set to be $2^{-(|\Lambda|+1)}$. Note that the accuracy of such bias evaluation depends on the rationality of Assumption 1.

Although Hao et al. provided a reliable framework for dynamic cube attacks in theory, the implementation of their attack against the full Grain-128 is not in accordance with their theory, which makes the key-recovery attack invalid.

3.2 Inaccuracy in the MILP models for dynamic cube attacks

The MILP modeling technique for dynamic cube attacks proposed in [HJL+20] can be utilized to evaluate the algebraic degree of the superpoly, provided that the nullification strategy and the cube are predefined. This modeling technique is essential for constructing qualified cubes, and its accuracy determines the effectiveness of the results of dynamic cube attacks. Upon analyzing this modeling technique, we discovered that the specific modeling process did not consider the algebraic structure of Grain family of ciphers. This oversight led to many redundant division trails in their models, which may result in an inaccurate estimation of the algebraic degree of a superpoly.

Through further experiments, we discovered that the qualified cubes used in [HJL+20] do not meet the specified requirements. Specifically, the superpolies corresponding to these cubes are 0-constant polynomials under both correct and wrong key guesses, making it impossible to correctly distinguish between correct and wrong key guesses. To illustrate the issues in this modeling technique, we will first describe the propagation of division property from an algebraic perspective.

The propagation of division property from an algebraic perspective. Generally, an iterative stream cipher $f(\boldsymbol{k}, \boldsymbol{v})$ can be further decomposed into a composition of simple vectorial Boolean functions as follows,

$$f(\boldsymbol{k}, \boldsymbol{v}) = F_{out} \circ \boldsymbol{F}_r \circ \cdots \circ \boldsymbol{F}_0(\boldsymbol{k}, \boldsymbol{v}), \tag{3}$$

where $\boldsymbol{F}_i : \mathbb{F}_2^N \to \mathbb{F}_2^N$ $(i \in \{1, 2, \ldots, r\})$, $\boldsymbol{F}_0 : \mathbb{F}_2^{n+m} \to \mathbb{F}_2^N$, and $F_{out} : \mathbb{F}_2^N \to \mathbb{F}_2$. Denote $\boldsymbol{s}^{(i)} \in \mathbb{F}_2^N$ as the output of $\boldsymbol{F}_i$ for $i \in \{0, \ldots, r\}$. We refer to Eq.(3) as a composite expression of $f(\boldsymbol{k}, \boldsymbol{v})$. Obviously, the composite expression of f is not unique. From an algebraic perspective, the 3SDP/u actually describes how a given monomial is propagated in a composite expression of $f(\boldsymbol{k}, \boldsymbol{v})$.

Let us consider the propagation of the 3SDP/u as follows,

$$\{\boldsymbol{l}\} \xrightarrow{\boldsymbol{F}_0} \mathbb{L}_0 \xrightarrow{\boldsymbol{F}_1} \mathbb{L}_1 \xrightarrow{\boldsymbol{F}_2} \cdots \xrightarrow{\boldsymbol{F}_r} \mathbb{L}_r \xrightarrow{\boldsymbol{F}_{out}} \mathbb{L}_{out}, \tag{$\mathcal{L}$}$$

where $\boldsymbol{l} = (\boldsymbol{\mu}, \boldsymbol{\nu}) \in \mathbb{F}_2^{n+m}$. For the sequence of multisets $\mathcal{L}$, if the multiset $\mathbb{L}_{out}$ is nonempty, there must exist a three-subset division trail, denoted by

$$\boldsymbol{l} \xrightarrow{p} \boldsymbol{l}_0 \xrightarrow{p} \cdots \xrightarrow{p} \boldsymbol{l}_r \xrightarrow{p} 1, \tag{ι}$$

where $\boldsymbol{l}_i \in \mathbb{L}_i$ for $i \in \{0, \ldots, r\}$. According to the propagation rules of 3SDP/u, $\boldsymbol{l}_i \xrightarrow{p} \boldsymbol{l}_{i+1}$ means that the monomial $\pi_{\boldsymbol{l}_i}(\boldsymbol{s}^{(i)})$ appears in the polynomial $\pi_{\boldsymbol{l}_{i+1}}(\boldsymbol{s}^{(i+1)}) \in \bar{\mathbb{Z}}[\boldsymbol{s}^{(i)}]$. Similarly, $\boldsymbol{l}_{i-1} \xrightarrow{p} \boldsymbol{l}_i \xrightarrow{p} \boldsymbol{l}_{i+1}$ means that the monomial $\pi_{\boldsymbol{l}_{i-1}}(\boldsymbol{s}^{(i-1)})$ can appear in the expansion of $\pi_{\boldsymbol{l}_{i+1}}(\boldsymbol{l}_{i+1})$ with respect to $\boldsymbol{s}^{(i-1)}$. The trail ι describes a propagation of the monomial $\pi_{\boldsymbol{l}}(\boldsymbol{k}, \boldsymbol{v})$ in a composite expression of $f(\boldsymbol{k}, \boldsymbol{v})$, and indicates that the monomial $\pi_{\boldsymbol{l}}(\boldsymbol{k}, \boldsymbol{v})$ appears in the expansion of f with respect to $\boldsymbol{k}$ and $\boldsymbol{v}$. However, the expansion of different composite expressions of f with respect to $\boldsymbol{k}$ and $\boldsymbol{v}$ may vary. This means that in other composite expressions of f, the three-subset division trails connecting the monomial $\pi_{\boldsymbol{l}}(\boldsymbol{k}, \boldsymbol{v})$ and f may not necessarily exist. We will illustrate this point through a simple example.

Example 2. Let $\boldsymbol{x} = (x_0, x_1, x_2, x_3) \in \mathbb{F}_2^4$. Let $f(\boldsymbol{x}) = x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0 \oplus x_3 \in \mathbb{F}_2[\boldsymbol{x}]$. The function f can be represented as $f_1 = F_o \circ \boldsymbol{F}_2 \circ \boldsymbol{F}_1(\boldsymbol{x})$, where

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{F}_1(\boldsymbol{x}) = (x_0, x_1, x_2, x_3, x_3), \\ \boldsymbol{z} &= \boldsymbol{F}_2(\boldsymbol{y}) = (y_0, y_0 + y_1 + y_4, y_2, y_0 + y_2 + y_4), \end{aligned}$$

and $F_o(\boldsymbol{z}) = z_1 z_3$. It is clear that the monomial $z_1 z_3$ can be expanded as the polynomial of $\boldsymbol{x}$ in $\bar{\mathbb{Z}}[\boldsymbol{x}]$ as follows,

$$\begin{aligned} z_1 z_3 &= (y_0 + y_1 + y_4) \cdot (y_0 + y_2 + y_4) && (\in \bar{\mathbb{Z}}[\boldsymbol{y}]) \\ &= y_0 y_1 + y_0 y_2 + 2y_0 y_4 + y_1 y_2 + y_1 y_4 + y_2 y_4 + y_0 + y_4 && (\in \bar{\mathbb{Z}}[\boldsymbol{y}]) \\ &= x_0 x_1 + x_0 x_2 + 2x_0 x_3 + x_1 x_2 + x_1 x_3 + x_2 x_3 + x_0 + x_3 && (\in \bar{\mathbb{Z}}[\boldsymbol{x}]) \end{aligned}$$

Let $f_2 = F_o \circ \boldsymbol{F}_2' \circ \boldsymbol{F}_1'(\boldsymbol{x})$ be another composite expression of f, where

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{F}_1'(\boldsymbol{x}) = (x_0, x_1, x_2, x_3, x_0 + x_3), \\ \boldsymbol{z} &= \boldsymbol{F}_2'(\boldsymbol{y}) = (y_0, y_1 + y_4, y_2, y_2 + y_4). \end{aligned}$$

The expandsion of $z_1 z_3$ can be obtained as follows,

$$\begin{aligned} z_1 z_3 &= (y_1 + y_4) \cdot (y_2 + y_4) = y_1 y_2 + y_1 y_4 + y_2 y_4 + y_4 && (\in \bar{\mathbb{Z}}[\boldsymbol{y}]) \\ &= x_1 x_2 + x_1 \cdot (x_0 + x_3) + x_2 \cdot (x_0 + x_3) + (x_0 + x_3) && (\in \bar{\mathbb{Z}}[\boldsymbol{x}]) \end{aligned}$$

For the first composite expression f_1, there is an even number of division trails connecting $x_0 x_3$ to f. In contrast, no three-subset division trail connects $x_0 x_3$ to f for the second composite expression f_2. Therefore, if we set x_0 and x_3 as cube variables and construct the MILP model based on the composite expression f_1, the estimated algebraic degree of the superpoly in f is not accurate due to redundant trails in the MILP model.

Therefore, when constructing the MILP model for a stream cipher $f(\boldsymbol{k}, \boldsymbol{v})$, the usage of an inadequate composite expression for f can result in numerous redundant trails within the corresponding MILP model. The propagation rules of CBDP are similar to those of 3SDP/u, so the propagation of CBDP faces the same issues as 3SDP/u. This issue is the primary reason for the failure of the dynamic cube attack against the full Grain-128 proposed in [HJL+20].

Inaccuracy in the original MILP models. The original modeling technique for dynamic cube attacks is to model round by round. Therefore, the r-round Grain-128 $G(\boldsymbol{k}, \boldsymbol{v})$ is first decomposed into a composition of the round functions as Eq.(3).

Let $(\boldsymbol{s}^{(i+1)}, \boldsymbol{b}^{(i+1)}) = \boldsymbol{F}_{i+1}(\boldsymbol{s}^{(i)}, \boldsymbol{b}^{(i)})$ for $i \in \{0, \ldots, r-1\}$ and $(\boldsymbol{s}^{(0)}, \boldsymbol{b}^{(0)}) = \boldsymbol{F}_0(\boldsymbol{k}, \boldsymbol{v})$. In any round i, only two state bits, s_{i+128} and b_{i+128}, are updated respectively by $z_i \oplus f_i \oplus s_i$ and $z_i \oplus g_i \oplus s_i$, and the rest of the state bits are shifted. The additional binary variables f_i, g_i and z_i can be represented as the polynomial of $\boldsymbol{s}^{(i)}$ and $\boldsymbol{b}^{(i)}$.

In the process of modeling the function $\boldsymbol{F}_{i+1}$, the additional binary variables f_i, g_i, and z_i are first defined, and the algebraic relation between them and $\boldsymbol{s}^{(i)}$ as well as $\boldsymbol{b}^{(i)}$ are added to the model as constraints. Then, the expressions of $\boldsymbol{s}^{(i+1)}$ and $\boldsymbol{b}^{(i+1)}$ with respect to $\boldsymbol{s}^{(i)}$, $\boldsymbol{b}^{(i)}$, f_i, g_i and z_i are added to the model. However, the MILP model established by this method may have redundant division trails, since $z_i \oplus s_i$ is involved in both two update functions of s_{i+128} and b_{i+128}, which makes the following equations hold,

$$\begin{aligned} s_{i+128} b_{i+128} &= (f_i + z_i + s_i) \cdot (g_i + z_i + s_i) && (\in \bar{\mathbb{Z}}[\boldsymbol{s}^{(i)}, \boldsymbol{b}^{(i)}, f_i, g_i, z_i]) \\ &= f_i \cdot g_i + (f_i + g_i) \cdot (z_i + s_i) \\ &\quad + 2z_i \cdot s_i + z_i + s_i. && (\in \bar{\mathbb{Z}}[\boldsymbol{s}^{(i)}, \boldsymbol{b}^{(i)}, f_i, g_i, z_i]) \end{aligned}$$

Another issue with the original MILP models is that the imposed constraints, based on the nullification strategy, do not consider the specific structure of Grain-128. We will use the nullification strategy in [HJL+20] as an example to illustrate this issue.

For the nullification strategy $N_S = \{(v_{30}, b_{158})\}$, the IV variable v_{30} is first set as a dynamic variable, and its dynamic value is determined by the ANF of b_{158} under

a specific key guess. For the correct key guess, we have the equations $b_{158} = 0$ and $v_{30} = g_{30} + z_{30}$, which are added to the MILP model as the constraints. Then, the other state bit s_{158} is updated by the expression $z_{30} + f_{30} + s_{30}$ in the MILP model. The modeling process described above is algebraically sound, but it may result in many division trails in the MILP model that should not exist. This is because the MILP model does not consider the cancellation property. Due to the nullification strategy N_S, the equations $v_{30} = z_{30} + g_{30}$ and $s_{30} = v_{30}$ are satisfied in the MILP model, thereby, the equation $s_{158} = z_{30} + f_{30} + s_{30} = 2 \cdot z_{30} + f_{30} + g_{30}$ also satisfying in the MILP model, which makes many redundant trails in this model.

In [HJL+20], the presence of numerous redundant trails in the model led to inaccurate results, causing the cubes identified as "qualified" to be unable to distinguish between correct and wrong key guesses. This directly rendered the key-recovery attack against the full Grain-128 invalid.

3.3 Empirical analysis of the impact caused by wrong key guesses

Evaluating the bias of the superpoly for each wrong key guess in practical scenarios presents challenges due to the exponential increase in the quantity of wrong key guesses as the number of key expressions to be guessed rises. This task necessitates solving numerous MILP models, which is a time-consuming process and incurs substantial computational costs. Constrained by the MILP modeling technique and computational capacity, Hao et al. introduced a resolution, which is to identify a specific wrong key guess that is considered to have potentially significant bias and is closest to the bias under the correct key guess.

For a predefined nullification strategy, $N_S = \{(l_1, s_{i_1}^{(r_1)}), \ldots, (l_t, s_{i_t}^{(r_t)})\}$, Hao et al. considered the wrong key guess satisfying the conditions $s_{i_t}^{(r_t)} = 1$ and $s_{i_j}^{(r_j)} = 0$ $(1 \leq j \leq t-1)$ as special, since under such a wrong key guess, the intermediate state bit $s_{i_t}^{(r_t)}$ has the simplest ANF, i.e., $s_{i_t}^{(r_t)} = 1$, and the other intermediate state bits that require nullification are successfully nullified. By contrast, the ANFs of the intermediate state bits that required nullification become more complex under other wrong key guesses.

Intuitively, as the complexity of the ANFs of the intermediate state bits that require nullification increases, the complexity of the superpoly may also increase, potentially resulting in a smaller bias of the superpoly. Therefore, the special wrong key guess is considered to have the most significant bias among all wrong key guesses. Based on the above view, if a batch of cubes can successfully differentiate between the correct key guess and such a specific wrong key guess, it implies that the batch of cubes can successfully differentiate between the correct key guess and all wrong key guesses. This solution not only reduces the requirements for modeling techniques but also greatly reduces the computational workload.

However, the validity of this viewpoint lacks inferences drawn from theoretical analysis; instead, it is based on personal experiences and intuition, which may lead to the division property based dynamic cube attack encountering similar issues as the traditional dynamic cube attack; that is, the constructed cubes may not guarantee successful differentiation between the correct key guess and all wrong key guesses.

4 More accurate bias evaluation for high-dimension cubes

Due to computational constraints, evaluating biases for high-dimension cubes has always been an extremely challenging problem. Hao et al. in [HJL+20] first established connections between the bias phenomenon and the division property, introducing a bias evaluation method based on the minimal split set. However, this method has significant limitations because its accuracy depends on Assumption 1. If Assumption 1 is unsuitable for a certain

cryptosystem, then this method cannot provide a relatively accurate estimate of bias. In [HJL+20], this method was applied in the division property based cube attack against the full Grain-128. However, based on our experiments, we found that Assumption 1 is not universally true for the Grain family of ciphers.

In this section, we present a new technique called Polynomial Approximation with regard to Bias (PAB) to evaluate the bias of the superpolies for the dynamic cube attack. The basic idea is that when dealing with a complex unknown polynomial, we identify and leverage its various algebraic properties to construct a simpler, known polynomial. By leveraging these properties, the constructed polynomial closely reflects the bias of the original one. Consequently, the original polynomial can be evaluated by detecting bias of the constructed polynomial. Prior to presenting the new method for evaluating bias in Section 4.1, we will first describe a trivial method for calculating the actual bias in Section 4.2.

4.1 A trivial method for calculating practical bias

Identifying biases of superpolies of high-dimension cubes through experimental testing is not feasible in cube attacks due to the computational complexity involved, which far exceeds current computational capabilities. In current cube attacks against stream ciphers, the dimensions of cubes that can be experimentally tested are mostly around 40 dimensions, not exceeding 50 dimensions.

With the *three-subset division property without unknown set* (3SDP/u) and its MILP modeling technique proposed in [HLM+20, HLM+21], it is possible to recover the exact ANF of a superpoly. Subsequently, by combining 3SDP/u with the divide-and-conquer strategy, a series of state-of-the-art methods were proposed in [Sun21, HST+21, HHPW22], which demonstrate high efficacy in the massive superpoly recovery. Therefore, we can use the accurately recovered superpoly to calculate its bias directly. Fortunately, recovering the corresponding superpoly in dynamic cube attacks is more accessible than in other variants of the cube attack, as a nullification strategy is applied to simplify the superpoly of the output bit, which makes enumerating all feasible solutions in the MILP model easier.

For a predetermined nullification strategy, if the superpoly under a specific wrong key guess has been successfully recovered, we can use experimental testing to detect its bias directly. Let $p_I(\boldsymbol{k})$ represent the recovered superpoly of a given cube I. The process of detecting bias of p_I is simple and follows these steps.

1. Identify the key variables included in p_I, denoted as $\mathbb{K} = \{k_{i_1}, k_{i_2}, \ldots, k_{i_t}\}$.
2. Randomly assign values to the key variables in $\mathbb{K}$ multiple times, denoted as $\kappa_1, \ldots, \kappa_N$, as N different inputs for the superpoly p_I.
3. compute the bias $\varepsilon_N = 2^{-1} - N^{-1} \cdot \sum_{i=1}^{N} p_I(\kappa_i)$.

As described in Section 2.2, if the obtained value ε_N satisfies Eq.(2), it can be regarded as the practical bias of p_I, and this bias can be detected with the significance level α using N random inputs of p_I. If N does not satisfy Eq.(2), choose a new N' such that

$$N' \geq \frac{u_\alpha^2}{2\varepsilon_N^2},$$

and repeat the aforementioned process.

4.2 Polynomial Approximation with regard to Bias

While the process of recovering a massive superpoly is currently quite effective, it still involves significant computational and time expenses. Additionally, as the number of

initialization rounds of a cryptosystem increases, the superpoly simplified by the nullification strategy will also become more complex, making it difficult to recover its exact ANF. Therefore, a method is required to provide accurate bias evaluation with fewer computational requirements and faster processing times.

In this part, we will introduce a new method for evaluating bias based on a new technique, called Polynomial Approximation with regard to Bias. The new technique is not to recover the entire superpoly but to recover a portion of it and identify specific algebraic properties of the remaining portion, thereby constructing a polynomial with a bias similar to that in the complete superpoly, which we refer to as an approximate polynomial of the superpoly. Accordingly, the bias of the superpoly can be evaluated by detecting the bias of the approximate polynomial.

An r-round iterative stream cipher $f(\boldsymbol{k}, \boldsymbol{v})$ can be decomposed into a composition of round functions as Eq.(3). By choosing a proper positive number $r_0 \in \{0, \ldots, r-1\}$, $f(\boldsymbol{k}, \boldsymbol{v})$ can be represented as follows:

$$\begin{aligned} f(\boldsymbol{k}, \boldsymbol{v}) &= F_o \circ \boldsymbol{F}_r \circ \cdots \circ \boldsymbol{F}_{r_0+1}(\boldsymbol{s}^{(r_0)}), \\ &= \bigoplus_{\boldsymbol{\omega} \in \Omega^{(r_0)}} \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)}), \end{aligned}$$

where $\Omega^{(r_0)} \triangleq \{\boldsymbol{\omega} \in \mathbb{F}_2^N | \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)}) \to f\}$, and $\boldsymbol{s}^{(r_0)} = \boldsymbol{F}_{r_0} \circ \cdots \circ \boldsymbol{F}_0(\boldsymbol{k}, \boldsymbol{v})$. For a specified cube I, we denote the superpoly of I in $\pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)})$ as $p_{\boldsymbol{\omega}}^{(r_0)}$, and try to recover these superpolies. The method of superpoly recovery is to construct the corresponding MILP model $\mathcal{M}_{\boldsymbol{\omega}}^{r_0}$ with a time limit for each monomial $\pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)})$ in f. If the model $\mathcal{M}_{\boldsymbol{\omega}}^{r_0}$ is not solved within the time limit, the procedure will be forcibly terminated. Depending on whether the model $\mathcal{M}_{\boldsymbol{\omega}}^{r_0}$ determined by $\boldsymbol{\omega} \in \Omega^{(r_0)}$ can return all feasible solutions within the time limit, the function f can be represented as

$$f = \bigoplus_{\boldsymbol{\omega} \in \Omega_s^{(r_0)}} \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)}) \oplus \bigoplus_{\boldsymbol{\omega} \in \Omega_u^{(r_0)}} \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_0)}) \triangleq f_s^{(r_0)} \oplus f_u^{(r_0)},$$

where

$$\begin{aligned} \Omega_s^{(r_0)} &= \{\boldsymbol{\omega} \in \Omega^{(r_0)} | \mathcal{M}_{\boldsymbol{\omega}}^{r_0} \text{ is solved within the time limit and } p_{\boldsymbol{\omega}}^{(r_0)} \text{ has been recovered } \}, \\ \Omega_u^{(r_0)} &= \{\boldsymbol{\omega} \in \Omega^{(r_0)} | \mathcal{M}_{\boldsymbol{\omega}}^{r_0} \text{ is unsolved within the time limit and } p_{\boldsymbol{\omega}}^{(r_0)} \text{ is still unknown } \}. \end{aligned}$$

Obviously, the superpoly $p_s^{(r_0)}$ in $f_s^{(r_0)}$ has been recovered, namely $p_s^{(r_0)} = \bigoplus_{\boldsymbol{\omega} \in \Omega_s^{(r_0)}} p_{\boldsymbol{\omega}}^{(r_0)}$. Furthermore, if the set $\Omega_u^{(r_0)}$ is empty, the superpoly p_I will be successfully recovered. However, recovering the superpoly is a challenging task, making it almost impossible for the set $\Omega_u^{(r_0)}$ to be empty, and our primary goal is to handle the monomials determined by the set $\Omega_u^{(r_0)}$.

Each monomial $\boldsymbol{\omega} \in \Omega_u^{(r_0)}$ undergoes a detection process to ascertain whether its superpoly is divisible by the variables $k_0, \ldots, k_{n-1}$. Assuming that the superpoly $p_{\boldsymbol{\omega}}^{(r_0)}$ is divisible by the variables $\{k_{i_1}, \ldots, k_{i_t}\}$, it can be represented as follows:

$$p_{\boldsymbol{\omega}}^{(r_0)} = \boldsymbol{k}^{\boldsymbol{\mu}} \cdot q_{\boldsymbol{\omega}}^{(r_0)},$$

where $\boldsymbol{\mu} (\in \mathbb{F}_2^n)$ satisfys $\boldsymbol{k}^{\boldsymbol{\mu}} = \prod_{j=1}^{t} k_{i_j}$. It is noteworthy to highlight that the polynomial $q_{\boldsymbol{\omega}}^{(r_0)}$ still remains unknown. If the superpoly in each monomial $\boldsymbol{\omega} \in \Omega_u^{(r_0)}$ has such a representation, i.e., $p_{\boldsymbol{\omega}}^{(r_0)} = \boldsymbol{k}^{\boldsymbol{\mu}_{\boldsymbol{\omega}}} \cdot q_{\boldsymbol{\omega}}^{(r_0)}$, then the entire superpoly p_I can be represented as

$$p_I = p_s^{(r_0)} \oplus \bigoplus_{\boldsymbol{\omega} \in \Omega_u^{(r_0)}} \boldsymbol{k}^{\boldsymbol{\mu}_{\boldsymbol{\omega}}} \cdot q_{\boldsymbol{\omega}}^{(r_0)}.$$

Otherwise, we will choose another positive integer $r_1 \in \{0, \ldots, r_0 - 1\}$, and then expand the polynomial $f_u^{(r_0)}$ into a polynomial $f^{(r_1)}$ in terms of $s^{(r_1)}$ as follows,

$$f^{(r_1)} = \bigoplus_{\boldsymbol{\omega} \in \Omega^{(r_1)}} \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_1)}),$$

where $\Omega^{(r_1)} = \{\boldsymbol{\omega} \in \mathbb{F}_2^N | \pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_1)}) \to f_u^{(r_0)}\}$. We repeat the process of dealing with $\Omega^{(r_0)}$ along with the number of rounds reducing $(0 < r_t < \cdots < r_1 < r_0)$ until we obtain the following equation, i.e.,

$$p_I = p_s \oplus \bigoplus_{\boldsymbol{\omega} \in \Omega_u^{(r_t)}} \boldsymbol{k}^{\boldsymbol{\mu}_{\boldsymbol{\omega}}} \cdot q_{\boldsymbol{\omega}}^{(r_t)}, \tag{4}$$

where $p_s = \bigoplus_{i=0}^{t} p_s^{(r_i)}$. The worst-case scenario of this process is that $\Omega_u^{(r_t)}$ is an empty set, indicating that the entire superpoly has been recovered.

For each $\boldsymbol{\omega} \in \Omega_u^{(r_t)}$, the superpoly $p_{\boldsymbol{\omega}}^{(r_t)}$ in $\pi_{\boldsymbol{\omega}}(\boldsymbol{s}^{(r_t)})$ is considered a relatively complex polynomial due to the inability to enumerate all potential three-subset division trails within a time limit. The polynomial $q_{\boldsymbol{\omega}}^{(r_t)}$ serves as the predominant component of polynomial $p_{\boldsymbol{\omega}}^{(r_t)}$, thus indicating that polynomial $q_{\boldsymbol{\omega}}^{(r_t)}$ possesses a level of complexity equivalent to that of polynomial $p_{\boldsymbol{\omega}}^{(r_t)}$, making it difficult to recover the ANF of $q_{\boldsymbol{\omega}}^{(r_t)}$. Hence, it is crucial to determine how to provide an approximate polynomial that maintains a bias similar to the bias of $q_{\boldsymbol{\omega}}^{(r_t)}$ when constructing an approximate polynomial of the superpoly p_I. According to the equation $p_{\boldsymbol{\omega}}^{(r_t)} = \boldsymbol{k}^{\boldsymbol{\mu}} \cdot q_{\boldsymbol{\omega}}^{(r_t)}$, the part $\boldsymbol{k}^{\boldsymbol{\mu}}$ in $p_{\boldsymbol{\omega}}^{(r_t)}$ that is most correlated with the key variables $\{k_0, \ldots, k_{n-1}\}$ has been extracted. Therefore, we consider the correlation between $q_{\boldsymbol{\omega}}^{(r_t)}$ and the key variables notably feeble.

Given this understanding, it is possible to introduce a new variable $x_{\boldsymbol{\omega}} \in \mathbb{F}_2$ as substitutes for the unidentified polynomial $q_{\boldsymbol{\omega}}^{(r_t)}$, leading to the derivation of a new polynomial p_I' expressed in terms of $\boldsymbol{k}$ and $\{x_{\boldsymbol{\omega}}\}_{\boldsymbol{\omega} \in \Omega_u^{(r_t)}}$ as follows,

$$p_I' = p_s \oplus \bigoplus_{\boldsymbol{\omega} \in \Omega_u^{(r_t)}} \boldsymbol{k}^{\boldsymbol{\mu}_{\boldsymbol{\omega}}} \cdot x_{\boldsymbol{\omega}}. \tag{5}$$

It can be seen that p_I' is a simple and identified polynomial whose bias can be detected by the method described in Section 4.1. However, there is a significant gap between the practical bias p_I' and p_I, and the reasons mainly lie in two aspects.

1. **The polynomial $q_{\boldsymbol{\omega}}^{(r_t)}$ has a bias that is not equal to 0.** Although the polynomial $q_{\boldsymbol{\omega}}^{(r_t)}$ is extremely complex, it does not mean that it is balanced. Therefore, the new variable $x_{\boldsymbol{\omega}}$ used to replace $q_{\boldsymbol{\omega}}^{(r_t)}$ should not be randomly distributed over $\mathbb{F}_2$, implying that $\Pr(x_{\boldsymbol{\omega}} = 1) \neq \Pr(x_{\boldsymbol{\omega}} = 0)$.
2. **Too many new variables were introduced.** There are correlations among the polynomials $\{q_{\boldsymbol{\omega}}^{(r_t)}\}_{\boldsymbol{\omega} \in \Omega_u^{(r_t)}}$ and variables $\boldsymbol{k}$; however, the newly introduced variables are not correlated with each other or with the original variables. This implies that the more new variables we add, the smaller the bias of p_I' will be, which can be proven by the piling-up lemma.

To mitigate these issues, we introduced a new algebraic property of polynomials, referred to as *the minimum algebraic degree*. Based on this property, we proposed a heuristic method to estimate the probability of each new variable taking the value of 1.

Definition 4 (The minimal algebraic degree)**.** The Boolean function f can be represented as follows,

$$f(\boldsymbol{x}) = \bigoplus_{\boldsymbol{\mu} \neq (0,\ldots,0) \in \mathbb{F}_2^n} a_{\boldsymbol{\mu}} \boldsymbol{x}^{\boldsymbol{\mu}} \oplus c, \text{ where } a_{\boldsymbol{\mu}} \text{ and } c \in \mathbb{F}_2.$$

We denote the minimal algebraic degree of the function f as $\deg_m(f)$, defined as

$$\deg_m(f) = (-1)^c \cdot \min\{wt(\boldsymbol{\mu}) | \boldsymbol{x}^{\boldsymbol{\mu}} \to f\}.$$

A heuristic method for probability estimation. For each $\boldsymbol{\omega} \in \Omega_u^{(r_t)}$, we heuristically use the minimal algebraic degree of $q_{\boldsymbol{\omega}}^{(r_t)}$, denoted by $d_{\boldsymbol{\omega}}$, to define the probability of $x_{\boldsymbol{\omega}}$ taking the value 1, i.e.,

$$\Pr(x_{\boldsymbol{\omega}} = 1) = \begin{cases} 2^{-d_{\boldsymbol{\omega}}}, & \text{if } d_{\boldsymbol{\omega}} > 0, \\ 1 - 2^{d_{\boldsymbol{\omega}}}, & \text{otherwise.} \end{cases}$$

The heuristic method is based on a simple property: for a monomial $\prod_{j=1}^{d} x_{i_j}$, when $x_{i_1}, \ldots, x_{i_d}$ are random variables over $\mathbb{F}_2$, it has the probability given by

$$\Pr\Big(\prod_{j=1}^{d} x_{i_j} = 1\Big) = 2^{-d}.$$

As the value of d increases, the probability of the monomial taking the value 1 decreases. Intuitively, when a polynomial has a high minimal algebraic degree, the probability of it taking the value 1 may be very small. Furthermore, if there are correlations among monomials in this polynomial, it may bring this intuition closer to the actual scenario. In our experiments, we observed that for the Grain family, particularly for Grain-128AEAD and Grain-128, there is a high correlation among the monomials in their superpolies, and their superpolies generally have high minimal algebraic degrees. Therefore, for a superpoly with a high correlation between monomials and with d-minimal algebraic degree, we regard it as a monomial of degree d. Besides, for the new variable x used to substitute the superpoly, the probability of it taking value 1 is required to be 2^{-d}.

For the sake of simplicity, we will only focus on the case where the minimum algebraic degree is greater than 0 hereafter, which also aligns with the algebraic characteristics of the Grain family.

A set of rules for polynomial transformations. We provide a set of polynomial transformation rules designed to integrate unknown polynomials, aiming to reduce the introduction of new variables. The specific rules are listed as follows:

rule 1: If there are polynomials $\boldsymbol{k}^{\boldsymbol{\mu}} \cdot q_{\boldsymbol{\omega}_1}^{(r_t)}, \ldots, \boldsymbol{k}^{\boldsymbol{\mu}} \cdot q_{\boldsymbol{\omega}_l}^{(r_t)}$ involved in Eq.(4), we use a new symbol polynomial g to represent the sum $(q_{\boldsymbol{\omega}_1}^{(r_t)} \oplus \cdots \oplus q_{\boldsymbol{\omega}_l}^{(r_t)})$, and these polynomials can be replaced by $\boldsymbol{k}^{\boldsymbol{\mu}} \cdot g$. The minimal algebraic degree of g can be obtained by $\deg_m(g) = \min\{d_{\boldsymbol{\omega}_1}, \ldots, d_{\boldsymbol{\omega}_l}\}$.

rule 2: If there are such polynomials $\boldsymbol{k}^{\boldsymbol{\mu}_1} \cdot q_{\boldsymbol{\omega}_1}^{(r_t)}$ and $\boldsymbol{k}^{\boldsymbol{\mu}_2} \cdot q_{\boldsymbol{\omega}_2}^{(r_t)}$ in Eq.(4) satisfying that $\boldsymbol{k}^{\boldsymbol{\mu}_2}$ is divisible by $\boldsymbol{k}^{\boldsymbol{\mu}_1}$, we use a new symbol polynomial g to represent the sum $(q_{\boldsymbol{\omega}_1}^{(r_t)} \oplus \frac{\boldsymbol{k}^{\boldsymbol{\mu}_2}}{\boldsymbol{k}^{\boldsymbol{\mu}_1}} \cdot q_{\boldsymbol{\omega}_2}^{(r_t)})$, and thereby the polynomials can be replaced by $\boldsymbol{k}^{\boldsymbol{\mu}_1} \cdot g$. The minimal algebraic degree of g can be obtained by $\deg_m(g) = \min\{d_{\boldsymbol{\omega}_1}, d_{\boldsymbol{\omega}_2} + (wt(\boldsymbol{\mu}_2) - wt(\boldsymbol{\mu}_1))\}$.

rule 3: If there are at least three polynomials $(k_{i_1} \cdots k_{i_j}) \cdot k_{l_1} \cdot q_{\boldsymbol{\omega}_1}^{(r_t)}$, $(k_{i_1} \cdots k_{i_j}) \cdot k_{l_2} \cdot q_{\boldsymbol{\omega}_2}^{(r_t)}$, $(k_{i_1} \cdots k_{i_j}) \cdot k_{l_3} \cdot q_{\boldsymbol{\omega}_3}^{(r_t)}$ in Eq.(4), we use a new symbol polynomial g to represent the sum $(k_{l_1} \cdot q_{\boldsymbol{\omega}_1}^{(r_t)} \oplus k_{l_2} \cdot q_{\boldsymbol{\omega}_2}^{(r_t)} \oplus k_{l_3} \cdot q_{\boldsymbol{\omega}_3}^{(r_t)})$, and thereby these polynomials can be replaced by $(k_{i_1} \cdots k_{i_j}) \cdot g$. The minimal algebraic degree of g can be obtained by $\deg_m(g) = 1 + \min\{d_{\boldsymbol{\omega}_1}, d_{\boldsymbol{\omega}_2}, d_{\boldsymbol{\omega}_3}\}$.

Based on the polynomial transformation rules, a simplified form of Eq.(4) can be derived as follows,

$$p_I = p_s \oplus \bigoplus_{i=1}^{t} \boldsymbol{k}^{\boldsymbol{\mu}_i} \cdot g_i.$$

Besides, the minimal algebraic degree of g_i has been obtained, denoted by d_i. By replacing the polynomials $g_1, \ldots, g_t$ with new variables $x_1, \ldots, x_t \in \mathbb{F}_2$, we can obtain an approximate polynomial of the superpoly p_I, i.e.,

$$p'_I = p_s \oplus \bigoplus_{i=1}^{t} \boldsymbol{k}^{\boldsymbol{\mu}_i} \cdot x_i.$$

Note that the new variables satisfy the conditions: $\Pr(x_i = 1) = 2^{-d_i}$, for $i \in \{1, \ldots, t\}$.

The construction of this approximate polynomial involves leveraging the known part and the algebraic properties of the undetermined part of the superpoly. It makes the constructed polynomial closer to the superpoly in terms of algebraic properties, further making the bias of the approximate polynomial more similar to the bias of the superpoly.

Therefore, we can compute the actual bias ε' of the approximate polynomial p'_I by the method described in Section 4.1. Moreover, we use the bias ε' of the approximate polynomial p'_I to evaluate the bias of the superpoly p_I.

5 A reliable implementation of dynamic cube attacks

In this section, we introduce a reliable implementation of dynamic cube attacks against the Grain family of ciphers, which includes an algebraic analysis of the impact caused by wrong key guesses and a more accurate MILP modeling technique for dynamic cube attacks.

5.1 Algebraic analysis of the impact caused by wrong key guesses

In dynamic cube attacks, it is necessary to evaluate the biases of the superpolies for all possible wrong key guesses. However, it is impractical to do this due to the need to solve a large number of MILP models, which is time-consuming and entails a significant computational cost. Therefore, we also need to identify specific wrong key guesses that may have significant biases and are closest to the bias of the correct guess. We refer to these key guesses as "bad" wrong key guesses.

For a predefined nullification strategy N_S and a given cube I, based on the ANFs of the intermediate state bits that need to be nullified, we can identify a set of dynamic variables $v_{d_1}, \ldots, v_{d_t}$, and their respective dynamic values $f_{d_1}(\boldsymbol{k}, \boldsymbol{v}), \ldots, f_{d_t}(\boldsymbol{k}, \boldsymbol{v})$. It is worth noting that, through simple algebraic substitutions, it can be ensured that none of $f_{d_1}, \ldots, f_{d_t}$ contain any dynamic variables.

The first output bit $z = f(\boldsymbol{k}, \boldsymbol{v})$ can be uniquely decomposed into the following form, i.e.,

$$f(\boldsymbol{k}, \boldsymbol{v}) = q_{\boldsymbol{0}}(\boldsymbol{k}, \boldsymbol{v}) \oplus \bigoplus_{\boldsymbol{\mu} \neq (0,\ldots,0) \in \mathbb{F}_2^t} \prod_{i=1}^{t} (v_{d_i} \oplus f_{d_i})^{\mu_i} \cdot q_{\boldsymbol{\mu}}(\boldsymbol{k}, \boldsymbol{v}), \tag{6}$$

where there are no dynamic variables involved in the polynomial $q_{\boldsymbol{\mu}}(\boldsymbol{k}, \boldsymbol{v})$, for each vector $\boldsymbol{\mu} \in \mathbb{F}_2^t$. Based on Eq.(6), we can analyze the impact of different wrong key guesses on the biases of the corresponding superpolies. Taking the specific strategy in [HJL+20] as an example, we will provide a simple explanation.

For the nullification strategy $N_S = \{(v_{30}, b_{158})\}$, the dynamic variable v_{30} is required to satisfy the following equation,

$$v_{30} = g_0 \oplus k_{42}v_{38} \oplus k_{125}v_{72} \oplus v_{43}v_{50} \oplus v_{90}. \tag{7}$$

For Eq.(7), if the variables v_{38} and v_{72} are treated as non-cube variables with fixed values, guessing the values of k_{42} and k_{125} is meaningless, due to the equation $v_{30} = g_0' \oplus v_{43}v_{50} \oplus v_{90}$. In this situation, the constructed cubes can not provide 3-bit key information. Therefore, the variables v_{38} and v_{72} should be set to cube variables.

The function $f(\boldsymbol{k}, \boldsymbol{v})$ can be decomposed into the expression as Eq.(6), i.e.,

$$f(\boldsymbol{k}, \boldsymbol{v}) = q \cdot (v_{30} \oplus g_0 \oplus k_{42}v_{38} \oplus k_{125}v_{72} \oplus v_{43}v_{50} \oplus v_{90}) \oplus h,$$

where q and h are the polynomials with respect to the key variables $\boldsymbol{k}$ and the cube variables $\boldsymbol{v}_I$.

Let μ be a positive integer. Denote the guess values of (g_0, k_{42}, k_{125}) by $(g_0 \oplus \mu[0], k_{42} \oplus \mu[1], k_{125} \oplus \mu[2])$. Each value of μ corresponds to a unique key guess, and we refer to μ as the key-guess pattern. Since the value of v_{30} is determined by the guess values of (g_0, k_{42}, k_{125}), the function f can be further represented as follows,

$$f(\boldsymbol{k}, \boldsymbol{v}) = q \cdot (\mu[0] \oplus \mu[1] \cdot v_{38} \oplus \mu[2] \cdot v_{72}) \oplus h.$$

Since the superpoly under the key-guess pattern $\mu = 0$ is required to be a 0-constant polynomial, the corresponding superpoly in h should be a 0-constant polynomial. The polynomial q can be uniquely decomposed as follows,

$$q = q_0 \oplus v_{38} \cdot q_1 \oplus v_{72} \cdot q_2 \oplus v_{38}v_{72} \cdot q_3,$$

where the IV variables v_{38}, v_{72} are not involved in the polynomials q_0, q_1, q_2, q_3. Therefore, the function f can be further represented as follows,

$$\begin{aligned} f(\boldsymbol{k}, \boldsymbol{v}) &= (q_0 \oplus v_{38} \cdot q_1 \oplus v_{72} \cdot q_2 \oplus v_{38}v_{72} \cdot q_3) \cdot (\mu[0] \oplus \mu[1] \cdot v_{38} \oplus \mu[2] \cdot v_{72}) \oplus h \\ &= \mu[0] \cdot v_{38}v_{72} \cdot q_3 \oplus \mu[0] \cdot (q_0 \oplus v_{38} \cdot q_1 \oplus v_{72} \cdot q_2) \\ &\oplus \mu[1] \cdot v_{38}v_{72} \cdot (q_2 \oplus q_3) \oplus \mu[1] \cdot (q_0 \oplus v_{38} \cdot q_1) \\ &\oplus \mu[2] \cdot v_{38}v_{72} \cdot (q_1 \oplus q_3) \oplus \mu[2] \cdot (q_0 \oplus v_{72} \cdot q_2) \oplus h \\ &\triangleq v_{38}v_{72} \cdot (\mu[0] \cdot q_3 \oplus \mu[1] \cdot q_2 \oplus \mu[1] \cdot q_3 \oplus \mu[2] \cdot q_1 \oplus \mu[2] \cdot q_3) \oplus h' \end{aligned}$$

Due to the variables v_{42} and v_{72} set to cube variables, the superpoly in f only comes from $v_{38}v_{72} \cdot (\mu[0] \cdot q_3 \oplus \mu[1] \cdot q_2 \oplus \mu[1] \cdot q_3 \oplus \mu[2] \cdot q_1 \oplus \mu[2] \cdot q_3)$. Let p_1, p_2, p_3 be the corresponding superpoly in the polynomials $v_{38}v_{72} \cdot q_0$, $v_{38}v_{72} \cdot q_1$, $v_{38}v_{72} \cdot q_2$ respectively. Therefore, the superpoly p in f also can be represented as follows,

$$p = (\mu[0] \oplus \mu[1] \oplus \mu[2]) \cdot p_3 \oplus \mu[1]p_2 \oplus \mu[2]p_1.$$

We show the superpoly under different key-guess patterns in Table 2.

Table 2: The superpoly under different key-guess patterns.

$(\mu[0], \mu[1], \mu[2])$	the superpoly p	$(\mu[0], \mu[1], \mu[2])$	the superpoly p
$(0, 0, 0)$	0	$(1, 0, 0)$	p_3
$(0, 1, 0)$	$p_2 \oplus p_3$	$(1, 1, 0)$	p_2
$(0, 0, 1)$	$p_1 \oplus p_3$	$(1, 0, 1)$	p_1
$(0, 1, 1)$	$p_1 \oplus p_2$	$(1, 1, 1)$	$p_1 \oplus p_2 \oplus p_3$

For the Grain family of ciphers, the output bit can be expressed as a polynomial with respect to the secret key and public IV variables. As the initialization rounds

increase, the superpolies in the polynomial representing the output bit become incredibly complex. Typically, the linear combination of different superpolies is considered even more complicated, as there are no significant cancellations of monomials. Therefore, we can come to the following conclusion.

Let $\boldsymbol{c} = (c_1, c_2, c_3) \in \mathbb{F}_2^3$, and $p_{\boldsymbol{c}} = c_1 \cdot p_1 \oplus c_2 \cdot p_2 \oplus c_3 \cdot p_3$ be a linear combination of the superpolies p_1, p_2 and p_3. Denote the bias of a polynomial p as $\varepsilon(p)$. For each $\boldsymbol{c} \in \mathbb{F}_2^3$, the bias $\varepsilon(p_{\boldsymbol{c}})$ satisfies the following condition: $\varepsilon(p_{\boldsymbol{c}}) \leq \min\{\varepsilon(p_i) | c_i = 1 \text{ for } i \in \{1, 2, 3\}\}$.

As shown in Table 2, when the key-guess mode is $\mu = 1$, 3, or 5, the associated superpolies may exhibit more significant biases. Therefore, it is necessary to identify the biases of the superpolies under key-guess modes $\mu = 1$, 3, and 5, as it is uncertain which one exhibits a greater bias. By employing the nulling strategy N_S, if we can construct a series of qualified cubes that successfully distinguish between the key guessing mode $\mu = 0$ and key guessing modes $\mu = 1$, 3, and 5, then we believe that these cubes can be used to distinguish between correct and wrong key guesses. Leveraging these cubes, we can execute a reliable dynamic cube attack against Grain-128. Compared to the analysis in [HJL+20], our method relies more on theoretical deduction, which is more reasonable.

Theoretically, this analysis method is a general and effective approach to identify the potential "bad" wrong key guesses in dynamic cube attacks for most stream ciphers. However, when the imposed conditions are complicated, the analysis becomes too intricate to find the possible "bad" key guesses.

5.2 A new MILP model for dynamic cube attacks

In this subsection, we introduce a new MILP modeling technique for dynamic cube attacks targeting the Grain family.

5.2.1 Specifications of Grain-128AEAD and Grain-128

The Grain family of ciphers include Grain v1, Grain-128, Grain-128a, and Grain-128AEAD. The Grain family of ciphers consists of two registers: an NFSR $\boldsymbol{b} = (b_0, \ldots, b_{n-1})$ and an LFSR $\boldsymbol{s} = (s_0, \ldots, s_{n-1})$, along with two feedback functions g and f for the NFSR and the LFSR.

In the initialization phase, the n-bit key and the m-bit IV are first loaded into $\boldsymbol{b}$ and $\boldsymbol{s}$, respectively, while the other state bits are set to constant values. Then, the cipher is clocked $2n$ times without generating a keystream. Simultaneously, the output produced from certain bits of the NFSR and the LFSR is fed to the cipher, as illustrated in Figure 1.

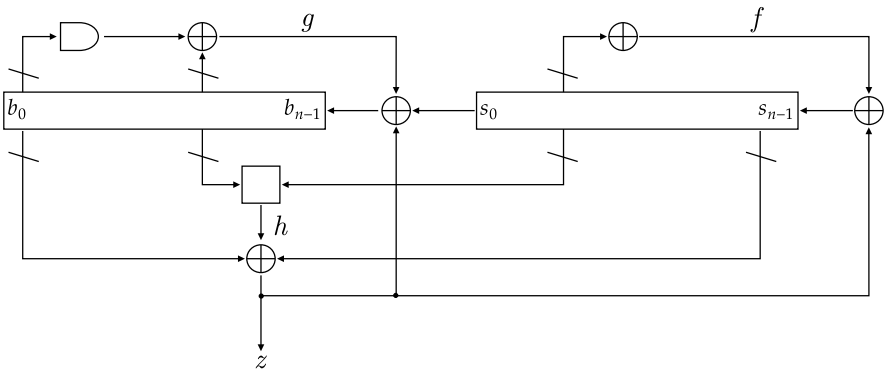

Figure 1: Structure of the Grain family

Grain-128AEAD and Grain-128 are both members of the Grain family designed to

provide 128-bit security. Grain-128AEAD is one of the ten finalist candidates of the NIST LWC standardization process. Grain-128 is the preliminary version of Grain-128AEAD, whose specification is simpler than Grain-128AEAD. Their internal state at time t consists of an LFSR and an NFSR, both with a length of 128 bits, denoted by $\boldsymbol{s}^{(t)} = (s_t, \ldots, s_{t+127})$ and $\boldsymbol{b}^{(t)} = (b_t, \ldots, b_{t+127})$, respectively.

For Grain-128AEAD, the initial states can be represented as follows:

$$\begin{aligned}\boldsymbol{b}^{(0)} &= (b_0, b_1, \ldots, b_{127}) = (k_0, k_1, \ldots, k_{127}),\\ \boldsymbol{s}^{(0)} &= (s_0, s_1, \ldots, s_{127}) = (v_0, v_1, \ldots, v_{95}, 1, \ldots, 1, 0).\end{aligned}$$

Let z_t be an output of the pre-output function as time t, and it is computed as

$$z_t = h(\boldsymbol{s}^{(t)}, \boldsymbol{b}^{(t)}) \oplus s_{t+93} \oplus \bigoplus_{j \in \mathbb{A}} b_{t+j},$$

where $\mathbb{A} = \{2, 15, 36, 45, 64, 73, 89\}$, and $h(\boldsymbol{s}^{(t)}, \boldsymbol{b}^{(t)})$ is defined as

$$h(\boldsymbol{s}^{(t)}, \boldsymbol{b}^{(t)}) = b_{t+12}s_{t+8} \oplus s_{t+13}s_{t+20} \oplus b_{t+95}s_{t+42} \oplus s_{t+60}s_{t+79} \oplus b_{t+12}b_{t+95}s_{t+94}.$$

Moreover, the functions $g(\boldsymbol{b}^{(t)})$ and $f(\boldsymbol{s}^{(t)})$ are defined as follows,

$$\begin{aligned}f(\boldsymbol{s}^{(t)}) &= s_t \oplus s_{t+7} \oplus s_{t+38} \oplus s_{t+70} \oplus s_{t+81} \oplus s_{t+96},\\ g(\boldsymbol{b}^{(t)}) &= b_{t+0} \oplus b_{t+26} \oplus b_{t+56} \oplus b_{t+91} \oplus b_{t+96} \oplus b_{t+3}b_{t+67} \oplus b_{t+11}b_{t+13}\\ &\quad \oplus b_{t+17}b_{t+18} \oplus b_{t+27}b_{t+59} \oplus b_{t+40}b_{t+48} \oplus b_{t+61}b_{t+65} \oplus b_{t+68}b_{t+84}\\ &\quad \oplus b_{t+88}b_{t+92}b_{t+93}b_{t+95} \oplus b_{t+22}b_{t+24}b_{t+25} \oplus b_{t+70}b_{t+78}b_{t+82}.\end{aligned}$$

The internal state bits s_{t+128} and b_{t+128} are computed by

$$\begin{aligned}s_{t+128} &= z_t \oplus f(\boldsymbol{s}^{(t)}),\\ b_{t+128} &= z_t \oplus s_t \oplus g(\boldsymbol{b}^{(t)}).\end{aligned}$$

In the initialization phase, the state is updated 256 times without producing an output. After the initialization, the update function is adjusted so that z_t is not fed to the state but rather used as a pre-output keystream. Grain-128AEAD inherits many specifications from Grain-128a proposed in [ÅHJM11]. Assuming that the first bit of the pre-output keystream can be observed, there is no difference between Grain-128AEAD and Grain-128a. Hereinafter, we study the round-reduced Grain-128AEAD under the above assumption.

For Grain-128, the feedback function for the NFSR is more sparse and is specified as

$$\begin{aligned}g(\boldsymbol{b}^{(t)}) &= b_{t+0} \oplus b_{t+26} \oplus b_{t+56} \oplus b_{t+91} \oplus b_{t+96} \oplus b_{t+3}b_{t+67} \oplus b_{t+11}b_{t+13}\\ &\quad \oplus b_{t+17}b_{t+18} \oplus b_{t+27}b_{t+59} \oplus b_{t+40}b_{t+48} \oplus b_{t+61}b_{t+65} \oplus b_{t+68}b_{t+84}.\end{aligned}$$

Moreover there is a small tweak in the h function as

$$h(\boldsymbol{s}^{(t)}, \boldsymbol{b}^{(t)}) = b_{t+12}s_{t+8} \oplus s_{t+13}s_{t+20} \oplus b_{t+95}s_{t+42} \oplus s_{t+60}s_{t+79} \oplus b_{t+12}b_{t+95}s_{t+95},$$

where s_{t+95} is used to instead of s_{t+94}. The others are identical to Grain-128AEAD.

5.2.2 A new modeling technique

Let $\boldsymbol{s}^{(r)} = (s_r, \ldots, s_{r+n-1})$ and $\boldsymbol{b}^{(r)} = (b_r, \ldots, b_{r+n-1})$ represent the intermediate states of the Grian family of cipher after the r-th round of iteration. In the initialization phase, the internal state bits s_{r+n} and b_{r+n} can also be updated by

$$\begin{aligned}s_{t+n} &= z'_t \oplus f',\\ b_{t+n} &= z'_t \oplus g,\end{aligned}$$

where $z'_t \triangleq z_t \oplus s_t$ and $f = f' \oplus s_t$. The purpose of updating the intermediate state bits with the functions z', f', and g is to avoid the issue mentioned in Section 3.2.

For a given nullification strategy $N_S = \{(v_{l_j}, s_{n-1}^{(r_j)})\}_{j\in J_s} \cup \{(v_{t_j}, b_{n-1}^{(r_j)})\}_{j\in J_b}$ and a predefined cube I, the dynamic variables and their dynamic values can be determined. For a pair $(v_d, s_{n-1}^{(r_d)}) \in N_S$ and a key-guess pattern μ, denote $\varphi_d^{(0)}$ as the dynamic value of the dynamic variables v_d under the key-guess pattern $\mu = 0$. Let $\delta_d^{(\mu)}$ be the ANF of the to-be-nullified state bit $s_{n-1}^{(r_d)}$ under a key-guess pattern μ. Hence, the dynamic values of v_d can be represented as $\varphi_d^{(\mu)} = \varphi_d^{(0)} \oplus \delta_d^{(\mu)}$, and we have the following equation:

$$\delta_d^{(\mu)} = s_{n-1}^{(r_d)} = z'(\boldsymbol{s}^{(r_d-1)}, \boldsymbol{b}^{(r_d-1)}) \oplus f'(\boldsymbol{s}^{(r_d-1)}) = v_d \oplus \varphi_d^{(0)}. \tag{8}$$

Obviously, only one of z' and f contains the dynamic variable v_d. If the dynamic variable v_d comes from z' and $r_d \notin R_b$, then the constraint $b_{n-1}^{(r_d)} = z'(\boldsymbol{s}^{(r_d-1)}, \boldsymbol{b}^{(r_d-1)}) \oplus g(\boldsymbol{b}^{(r_d-1)})$ has been replaced in the model with the constraint $b_{n-1}^{(r_d)} = g(\boldsymbol{b}^{(r_d-1)}) + f'(\boldsymbol{s}^{(r_d-1)}) + \delta_d^{(\mu)}$. While these constraints are algebraically equivalent due to Eq.(8), this modification in the MILP model helps reduce unnecessary division trails within the model, as described in Section 3.2. For the pair $(v_d, b_{n-1}^{(r_d)}) \in N_S$ that satisfies similar conditions, we use the identical processing method to replace the original constraint with the new one.

Compared with the original MILP modeling method proposed in [HJL+20] for dynamic cube attacks, our new modeling technique can achieve a more accurate MILP model. First, it is known that 3SDP/u is more accurate than the CBDP, and Algorithm 2 is based on 3SDP/u while the original model is based on CBDP. Second, the specific algebraic structure of the Grain family is taken into consideration in Algorithm 2 to avoid generating numerous redundant trails. Our modeling technique is demonstrated in Algorithms 1 and 2, focusing on the modeling ideas while omitting some technical details. The detailed modeling process has been uploaded to the GitHub repository as code.

Algorithm 1: A New MILP Model for the Grain family

procedure GrainFamilyInit(Model $\mathcal{M}$, Cube I, Nullification strategy $N_S = \{(v_{l_j}, s_{n-1}^{(r_j)})\}_{j\in J_s} \cup \{(v_{t_j}, b_{n-1}^{(r_j)})\}_{j\in J_b}$, Key-guess pattern μ)
  $\mathcal{M}.var \leftarrow \boldsymbol{bvar} = (svar[0], \ldots, svar[n-1])$ as binary variables
  $\mathcal{M}.var \leftarrow \boldsymbol{svar} = (bvar[0], \ldots, bvar[n-1])$ as binary variables
  $\mathcal{M}.var \leftarrow \boldsymbol{kvar} = (kvar[0], \ldots, kvar[n-1])$ as binary variables
  $\mathcal{M}.var \leftarrow \boldsymbol{vvar} = (vvar[0], \ldots, vvar[m-1])$ as binary variables
  $\mathcal{M}.obj \leftarrow \max\{\sum_{i=0}^{n-1} kvar[i]\}$
  $\mathcal{M}.con \leftarrow vvar[i] = 1$ for $i \in I$
  $\mathcal{M}.con \leftarrow vvar[j] = 0$ for $j \notin I \cup \{l_j\}_{j\in J_s} \cup \{t_j\}_{j\in J_b}$
  According to N_S and μ, deduce the ANFs of the dynamic values corresponding to the dynamic variables, i.e., $\{v_{l_j}, f_{l_j}^{(\mu)}\}$ for $j \in J_s$ and $\{v_{t_j}, f_{l_j}^{(\mu)}\}$ for $j \in J_b$.
  for $j \in J_s$ **do**
    $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, vvar[l_j], f_j^{\mu}, \boldsymbol{kvar}, \boldsymbol{vvar})$
  for $j \in J_b$ **do**
    $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, vvar[t_j], f_j^{\mu}, \boldsymbol{kvar}, \boldsymbol{vvar})$
  $\mathcal{M}.con \leftarrow bvar[i] = kvar[i]$ for $i \in \{0, \ldots, n-1\}$
  $\mathcal{M}.con \leftarrow svar[i] = vvar[i]$ for $i \in \{0, \ldots, m-1\}$ ▷ for Grain-128AEAD, there is a another constraint, i.e., $\mathcal{M}.con \leftarrow svar[n-1] = 0$
  return $(\mathcal{M}, \boldsymbol{svar}, \boldsymbol{bvar}, \boldsymbol{kvar}, \boldsymbol{vvar})$
end procedure

In Algorithm 1, we added the algebraic relationship between dynamic variables and dynamic values as constraints into the model. The procedure ModelPoly adds polynomials as constraints into the model. Algorithm 2 illustrates the application of algebraic structures for the Grain family in our new modeling technique. The procedures ModelFuncF,

Algorithm 2: A New MILP Model for the Grain family

procedure GrainFamilyModel(Cube I, Nullification strategy $N_S = \{(v_{l_j}, s_{n-1}^{(r_j)})\}_{j\in J_s} \cup \{(v_{t_j}, b_{n-1}^{(r_j)})\}_{j\in J_b}$, Key-guess pattern μ)
Prepare an empty MILP model $\mathcal{M}$
Update $\mathcal{M}$ as $(\mathcal{M}, \boldsymbol{svar}, \boldsymbol{bvar}, \boldsymbol{kvar}, \boldsymbol{vvar}) \leftarrow$ GrainFamilyInit$(\mathcal{M}, I, N_S, \mu)$
According to N_S and μ, deduce the ANFs of the to-be-nullified state bits $\{s_{n-1}^{(r_j)}\}_{j\in J_s}$ and $\{b_{n-1}^{(r_j)}\}_{j\in J_b}$, denoted by $\{\varphi_j^{(\mu)}\}_{j\in J_s}$ and $\{\psi_j^{(\mu)}\}_{j\in J_b}$, respectively.
for r **from** 1 **to** R **do**
    if $r \notin \{r_j\}_{j\in J_s\cup J_b}$ **do**
        $\mathcal{M}.var \leftarrow fvar, gvar, zvar, zvar1, zvar2, newb, news$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelFuncF$(\mathcal{M}, fvar, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncG$(\mathcal{M}, gvar, \boldsymbol{bvar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncZ$(\mathcal{M}, zvar, \boldsymbol{bvar}, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow zvar = zvar1 \vee zvar2$
        $\mathcal{M}.con \leftarrow newb = zvar1 + gvar$
        $\mathcal{M}.con \leftarrow news = zvar2 + fvar$
    else if $r \in \{r_j\}_{j\in J_s\cap J_b}$ **do**
        $\mathcal{M}.var \leftarrow newb, news$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, newb, \varphi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, news, \psi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow (bvar[0], svar[0]) = (0, 0)$
    else if $r \in \{r_j\}_{j\in J_b}$ **and** v_{t_j} involved in z' **do**
        $\mathcal{M}.var \leftarrow gvar, fvar, newb, news, xvar, xvar1, xvar2$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, xvar, \varphi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow xvar = xvar1 \vee xvar2$
        $\mathcal{M}.con \leftarrow$ ModelFuncF$(\mathcal{M}, fvar, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncG$(\mathcal{M}, gvar, \boldsymbol{bvar})$
        $\mathcal{M}.con \leftarrow newb = xvar1$
        $\mathcal{M}.con \leftarrow news = xvar2 + fvar + gvar$
        $\mathcal{M}.con \leftarrow svar[0] = 0$
    else if $r \in \{r_j\}_{j\in J_b}$ **and** v_{t_j} not involved in z' **do**
        $\mathcal{M}.var \leftarrow zvar, fvar, newb, news, xvar$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, xvar, \varphi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncF$(\mathcal{M}, fvar, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncZ$(\mathcal{M}, zvar, \boldsymbol{bvar}, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow newb = xvar$
        $\mathcal{M}.con \leftarrow news = fvar + zvar$
    else if $r \in \{r_j\}_{j\in J_s}$ **and** v_{l_j} involved in z' **do**
        $\mathcal{M}.var \leftarrow fvar, gvar, newb, news, xvar, xvar1, xvar2$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, xvar, \psi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow xvar = xvar1 \vee xvar2$
        $\mathcal{M}.con \leftarrow$ ModelFuncF$(\mathcal{M}, fvar, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncG$(\mathcal{M}, gvar, \boldsymbol{bvar})$
        $\mathcal{M}.con \leftarrow newb = xvar1 + fvar + gvar$
        $\mathcal{M}.con \leftarrow news = xvar2$
        $\mathcal{M}.con \leftarrow svar[0] = 0$
    else if $r \in \{r_j\}_{j\in J_s}$ **and** v_{l_j} not involved in z' **do**
        $\mathcal{M}.var \leftarrow zvar, gvar, newb, news, xvar$ as binary variables
        $\mathcal{M}.con \leftarrow$ ModelPoly$(\mathcal{M}, xvar, \varphi_r, \boldsymbol{kvar}, \boldsymbol{vvar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncG$(\mathcal{M}, gvar, \boldsymbol{bvar})$
        $\mathcal{M}.con \leftarrow$ ModelFuncZ$(\mathcal{M}, zvar, \boldsymbol{bvar}, \boldsymbol{svar})$
        $\mathcal{M}.con \leftarrow newb = gvar + zvar$
        $\mathcal{M}.con \leftarrow news = xvar$
    $(bvar[0], \ldots, bvar[126], bvar[127]) \leftarrow (bvar[1], \ldots, bvar[127], newb)$
    $(svar[0], \ldots, svar[126], svar[127]) \leftarrow (svar[1], \ldots, svar[127], news)$
$\mathcal{M}.var \leftarrow ovar$ as binary variable
$\mathcal{M}.con \leftarrow$ ModelFuncZ$(\mathcal{M}, ovar, \boldsymbol{bvar}, \boldsymbol{svar})$
$\mathcal{M}.con \leftarrow bvar[i] = 0$ for $i \in \{0, \ldots, n-1\}$
$\mathcal{M}.con \leftarrow svar[i] = 0$ for $i \in \{0, \ldots, n-1\}$
return $(\mathcal{M}, \boldsymbol{kvar}, \boldsymbol{vvar})$
end procedure

ModelFuncG, and ModelFuncZ used in Algorithm 2 are specific implementations for the function g and the modified functions f' and z'.

The models established by Algorithm 2 can be used to estimate the algebraic degree of a superpoly and recover the superpoly for a given cube. Moreover, by adjusting the objective function as described in Algorithm 1 to $\mathcal{M}.obj \leftarrow \min\{\sum_{i=0}^{n-1} kvar[i]\}$, the modified model can be employed to estimate the lower bound of the minimal algebraic degree of the superpoly. Additionally, if the initially feasible models become infeasible by adding the constraint $\mathcal{M} \leftarrow kvar[i] = 0$ to the model, then we can determine that the superpoly is divisible by the key variable k_i. By solving n MILP models with different constraints, i.e., $\mathcal{M} \leftarrow kvar[i] = 0$ for $i \in \{0, \ldots, n-1\}$, all key variables that divide the superpoly can be identified.

6 Applications to Grain-128AEAD and Grain-128

In this section, we applied dynamic cube attacks based on the 3SDP/u to Grain-128AEAD and Grain-128. As a result, we successfully conducted a key-recovery attack on 190-round Grain-128AEAD, with the ability to recover 3 key bits with a probability exceeding 99.68%. Additionally, we provided a feasibility analysis of a dynamic cube attack against full-version Grain-128. Furthermore, we compared two different methods of bias evaluation. The solver for the MILP model we used is: Gurobi Solver (version 9.0.3), and the platform we used is: AMD Ryzen 5950X 16-core Processor 3.7 GHz, 128G RAM, and Ubuntu 22.04 LTS. The source codes and experimental results are both available in our git repository.

6.1 Dynamic cube attack on Grain-128AEAD

Through automated searching, we found a qualified nullification strategy N_S, which contains 2 pairs: $N_S = \{(v_{23}, b_{151}), (v_{24}, b_{154})\}$. We find that when b_{151} and b_{152} are nullified by setting v_{23} and v_{24} to dynamic variables, the superpoly of $I = \{0, \ldots, 95\} \backslash \{23, 24\}$ at round 190 is 0-constant polynomial. According to the ANFs of b_{151} and b_{152}, the dynamic variables v_{23} and v_{24} can be represented as follows,

$$\begin{aligned} v_{23} &= v_{36}v_{43} \oplus v_{83} \oplus \boldsymbol{k_{118}}v_{65} \oplus \boldsymbol{k_{35}}v_{31} \oplus \boldsymbol{g_0}, \\ v_{24} &= v_{37}v_{44} \oplus v_{84} \oplus \boldsymbol{k_{119}}v_{66} \oplus \boldsymbol{k_{36}}v_{32} \oplus \boldsymbol{g_1}, \end{aligned} \tag{9}$$

where g_0 and g_1 are the polynomial of key variables. Let $\kappa_1 \triangleq (g_0, k_{35}, k_{118})$ and $\kappa_2 \triangleq (g_1, k_{36}, k_{119})$. According to the expression, we need to guess the values of $\kappa \triangleq (\kappa_1, \kappa_2)$.

Let $\mu = (\mu_1, \mu_2) \in \mathbb{Z}^2$ be the key-guess pattern of κ. Under a key-guess pattern μ, the corresponding key guess is

$$\kappa_\mu \triangleq (g_0 \oplus \mu_1[0], k_{35} \oplus \mu_1[1], k_{118} \oplus \mu_1[2], g_1 \oplus \mu_2[0], k_{36} \oplus \mu_2[1], k_{119} \oplus \mu_2[2]).$$

According to the algebraic analysis method proposed in Section 5.1, the output of 190-round Grain-128AEAD can be represented as follows,

$$\begin{aligned} f(\boldsymbol{k}, \boldsymbol{v}) &= f^{(0)} \oplus f^{(1)}(\boldsymbol{k}, \boldsymbol{v}) \cdot (\mu_1[0] \oplus \mu_1[1] \cdot v_{31} \oplus \mu_1[2] \cdot v_{65}) \\ &\quad \oplus f^{(2)}(\boldsymbol{k}, \boldsymbol{v}) \cdot (\mu_2[0] \oplus \mu_2[1] \cdot v_{32} \oplus \mu_2[2] \cdot v_{66}) \\ &\quad \oplus f^{(3)}(\boldsymbol{k}, \boldsymbol{v}) \cdot (\mu_1[0] \oplus \mu_1[1] \cdot v_{31} \oplus \mu_1[2] \cdot v_{65}) \cdot (\mu_2[0] \oplus \mu_2[1] \cdot v_{32} \oplus \mu_2[2] \cdot v_{66}) \\ &= f^{(0)} \oplus (f_0^{(1)} \oplus v_{31}f_1^{(1)} \oplus v_{65}f_2^{(1)} \oplus v_{31}v_{65}f_3^{(1)}) \cdot (\mu_1[0] \oplus \mu_1[1] \cdot v_{31} \oplus \mu_1[2] \cdot v_{65}) \\ &\quad \oplus (f_0^{(2)} \oplus v_{32}f_1^{(2)} \oplus v_{66}f_2^{(2)} \oplus v_{32}v_{66}f_3^{(2)}) \cdot (\mu_2[0] \oplus \mu_2[1] \cdot v_{32} \oplus \mu_2[2] \cdot v_{66}) \\ &\quad \oplus f^{(3)} \cdot (\mu_1[0] \oplus \mu_1[1] \cdot v_{31} \oplus \mu_1[2] \cdot v_{65}) \cdot (\mu_2[0] \oplus \mu_2[1] \cdot v_{32} \oplus \mu_2[2] \cdot v_{66}). \end{aligned} \tag{10}$$

The superpolies in $v_{31}v_{65}f_i^{(1)}$ and $v_{32}v_{66}f_i^{(2)}$ are respectively denoted as $p_i^{(1)}$ and $p_i^{(2)}$, where $i \in \{1,2,3\}$. By analyzing Eq.(10), it is concluded that the key-guess patterns $\mu = (1,0)$, $(3,0)$, $(5,0)$, $(0,1)$, $(0,3)$, $(0,5)$ are considered to be "bad". The details are listed in Table 3.

Table 3: The details for selected "bad" key-guess patterns

$\mu = (\mu_1, \mu_2)$	the superpoly p^μ	$\mu = (\mu_1, \mu_2)$	the superpoly p^μ
$(1,0)$	$p_3^{(1)}$	$(0,1)$	$p_3^{(2)}$
$(3,0)$	$p_2^{(1)}$	$(0,3)$	$p_2^{(2)}$
$(5,0)$	$p_1^{(1)}$	$(0,5)$	$p_1^{(2)}$

Table 4: The qualified cubes used for attacking 190-round Grain-128 AEAD. The nullification strategy is $N_S = \{(v_{23}, b_{151}), (v_{24}, b_{152})\}$.

cube I	the bias $\varepsilon_{(\mu_1,\mu_2)}$ under the "bad" key-guess patterns (μ_1, μ_2)					
	$(1,0)$	$(3,0)$	$(5,0)$	$(0,1)$	$(0,3)$	$(0,5)$
$I_0 = \{0,\ldots,95\}\backslash\{23,24,95\}$	0.353606	0.206203	0.234221	0.4230747	0.399117	0.4361982
$I_1 = \{0,\ldots,95\}\backslash\{23,24,77,40\}$	0.387184	0.299633	0.294281	0.4387169	0.354754	0.4249859
$I_2 = \{0,\ldots,95\}\backslash\{23,24,71,40\}$	0.387218	0.346762	0.344177	0.4813395	0.4379272	0.4396868
$I_3 = \{0,\ldots,95\}\backslash\{23,24,68,40\}$	0.316473	0.257362	0.241364	0.4721603	0.4406729	0.4352102
$I_4 = \{0,\ldots,95\}\backslash\{23,24,67,42\}$	0.367339	0.274574	0.250311	0.4757776	0.392146	0.4511347
$I_5 = \{0,\ldots,95\}\backslash\{23,24,61,55\}$	0.367626	0.24719	0.188051	0.4761515	0.4403486	0.4232893
$I_6 = \{0,\ldots,95\}\backslash\{23,24,61,52\}$	0.38121	0.281216	0.258658	0.4892836	0.4862223	0.472414
$I_7 = \{0,\ldots,95\}\backslash\{23,24,60,41\}$	0.370648	0.289206	0.250978	0.4867525	0.4836922	0.4671307
$I_8 = \{0,\ldots,95\}\backslash\{23,24,55\}$	0.379585	0.20465	0.213818	0.4711533	0.4273214	0.4395924
$I_9 = \{0,\ldots,95\}\backslash\{23,24,50\}$	0.339352	0.235249	0.221518	0.471941	0.4487085	0.4553604
$I_{10} = \{0,\ldots,95\}\backslash\{23,24,49,41\}$	0.324727	0.18964	0.227296	0.4730148	0.4603596	0.4364719
$I_{11} = \{0,\ldots,95\}\backslash\{23,24,44,33\}$	0.380052	0.292162	0.264006	0.49561977	0.49242878	0.456008
$I_{12} = \{0,\ldots,95\}\backslash\{23,24,42,27\}$	0.387638	0.281886	0.278914	0.479454	0.4136972	0.470829
$I_{13} = \{0,\ldots,95\}\backslash\{23,24,77,41,40\}$	0.334992	0.212773	0.253929	0.4375563	0.315454	0.4094419
$I_{14} = \{0,\ldots,95\}\backslash\{23,24,71,46,40\}$	0.366632	0.309636	0.275089	0.480011	0.4891071	0.4603796
$I_{15} = \{0,\ldots,95\}\backslash\{23,24,71,60,40\}$	0.309626	0.140639	0.229797	0.388562	0.380021	0.335477
$I_{16} = \{0,\ldots,95\}\backslash\{23,24,49,46,35\}$	0.380275	0.284727	0.225223	0.49509048	0.49503422	0.4813623
$I_{17} = \{0,\ldots,95\}\backslash\{23,24,49,46,40\}$	0.341744	0.282315	0.250940	0.4590654	0.4652958	0.4487858
$I_{18} = \{0,\ldots,95\}\backslash\{23,24,49,46,41\}$	0.38393	0.253141	0.231102	0.4838247	0.4805527	0.4533014
$I_{19} = \{0,\ldots,95\}\backslash\{23,24,49,46,45\}$	0.329613	0.32315	0.206355	0.4760637	0.4771605	0.4480047
$I_{20} = \{0,\ldots,95\}\backslash\{23,24,60,49,46\}$	0.368801	0.283889	0.222698	0.4751892	0.4819527	0.4385643
$I_{21} = \{0,\ldots,95\}\backslash\{23,24,75,49,46\}$	0.380912	0.290606	0.224708	0.484374	0.4801178	0.450757
$I_{22} = \{0,\ldots,95\}\backslash\{23,24,49,41,35\}$	0.313745	0.148555	0.209636	0.49448395	0.4807119	0.4658022
$I_{23} = \{0,\ldots,95\}\backslash\{23,24,42,41,27\}$	0.389872	0.262617	0.244953	0.49575138	0.4516106	0.4830332
$I_{24} = \{0,\ldots,95\}\backslash\{23,24,76,42\}$	0.356276	0.305232	0.317458	0.49332905	0.4739418	0.4831657
$I_{25} = \{0,\ldots,95\}\backslash\{23,24,72\}$	0.345942	0.207445	0.138506	0.4693813	0.4510098	0.4379272
$I_{26} = \{0,\ldots,95\}\backslash\{23,24,72,61\}$	0.330338	0.189197	0.121749	0.4762859	0.4642897	0.4043407
$I_{27} = \{0,\ldots,95\}\backslash\{23,24,69,29\}$	0.386326	0.318699	0.258453	0.4800005	0.4519835	0.4615879
$I_{28} = \{0,\ldots,95\}\backslash\{23,24,68,46\}$	0.382341	0.299612	0.222552	0.486948	0.4850311	0.4703856
$I_{29} = \{0,\ldots,95\}\backslash\{23,24,61,32\}$	0.372087	0.294085	0.227592	0.49308872	0.4638119	0.083961
$I_{30} = \{0,\ldots,95\}\backslash\{23,24,55,41\}$	0.383193	0.254925	0.252210	0.4804106	0.4482346	0.4562445
$I_{31} = \{0,\ldots,95\}\backslash\{23,24,54\}$	0.379627	0.237605	0.28257	0.4856052	0.4630537	0.458209
$I_{32} = \{0,\ldots,95\}\backslash\{23,24,50,44\}$	0.372524	0.228606	0.240152	0.4957962	0.484704	0.464653
$I_{33} = \{0,\ldots,95\}\backslash\{23,24,50,41\}$	0.274916	0.172963	0.1509	0.4690971	0.430047	0.4502716
$I_{34} = \{0,\ldots,95\}\backslash\{23,24,49,35\}$	0.385032	0.294983	0.27245	0.493433	0.4884911	0.4795504
$I_{35} = \{0,\ldots,95\}\backslash\{23,24,46,28\}$	0.378247	0.299514	0.23912	0.4755831	0.489665	0.4692144
$I_{36} = \{0,\ldots,95\}\backslash\{23,24,41,27\}$	0.344182	0.227023	0.221251	0.49296188	0.4623575	0.4599113
$I_{37} = \{0,\ldots,95\}\backslash\{23,24,70,42,41\}$	0.347362	0.274938	0.239745	0.49620628	0.4766798	0.4822416
$I_{38} = \{0,\ldots,95\}\backslash\{23,24,68,61,42\}$	0.378164	0.283128	0.267408	0.4750986	0.4426651	0.418539
$I_{39} = \{0,\ldots,95\}\backslash\{23,24,68,46,42\}$	0.352967	0.292024	0.218067	0.4866066	0.4863405	0.4649763
$I_{40} = \{0,\ldots,95\}\backslash\{23,24,68,42,41\}$	0.388642	0.27685	0.264484	0.49326611	0.463604	0.4774561
$I_{41} = \{0,\ldots,95\}\backslash\{23,24,76,61,41\}$	0.399852	0.237556	0.280949	0.49482155	0.4734402	0.4454041
$I_{42} = \{0,\ldots,95\}\backslash\{23,24,61,60,46\}$	0.378113	0.299518	0.25487	0.4923811	0.49433804	0.4730492
$I_{43} = \{0,\ldots,95\}\backslash\{23,24,61,50,44\}$	0.366341	0.222016	0.214379	0.49053955	0.4833498	0.4406195
$I_{44} = \{0,\ldots,95\}\backslash\{23,24,61,46,45\}$	0.338963	0.378028	0.230568	0.49240112	0.4890747	0.463501
$I_{45} = \{0,\ldots,95\}\backslash\{23,24,60,46,45\}$	0.310297	0.31269	0.185132	0.479125	0.4852772	0.454525
$I_{46} = \{0,\ldots,95\}\backslash\{23,24,51,42,41\}$	0.369997	0.322962	0.241527	0.495255	0.4258976	0.4170637
$I_{47} = \{0,\ldots,95\}\backslash\{23,24,50,44,41\}$	0.374805	0.191571	0.198668	0.49560547	0.49258804	0.4706526
$I_{48} = \{0,\ldots,95\}\backslash\{23,24,71,49,35\}$	0.309337	0.223654	0.200107	0.4647303	0.4269733	0.4178019
$I_{49} = \{0,\ldots,95\}\backslash\{23,24,75,46,42\}$	0.33437	0.248691	0.195266	0.4613829	0.4739819	0.45644
$I_{50} = \{0,\ldots,95\}\backslash\{23,24,76,41,40\}$	0.354204	0.222609	0.272528	0.4736853	0.446743	0.4405336

According to the definite nullification strategy N_S and these "bad" key-guess patterns, we found some qualified cubes by constructing and solving corresponding MILP models, denoted as $I_0, \ldots, I_{50}$. Their biases under "bad" key-guess patterns were detected by conducting experimental testing on the corresponding superpolies that have been recovered. The amount of data used to detect the biases is 2^{20}, which satisfies Equation (2). This implies that the detected biases can be considered practical biases in these superpolies under "bad" key-guess patterns. Both the cubes used and their biases under "bad" key-guess patterns were listed in Table 4.

Our dynamic cube attack process is consistent with the one described in [HJL+20]. It

follows the steps outlined below.

1. Guess $\kappa_1 \triangleq (g_0, k_{35}, k_{118})$ and $\kappa_2 \triangleq (g_1, k_{36}, k_{119})$.
2. For each guess, compute the values of the superpolies of the cubes I listed in 4 under the predefined nullification strategy.
3. If any of the 51 values is non-zero, the key guess is wrong; otherwise, keep the key guess as a correct key guess condidate.

Complexity and Success Probabilities. The time and data complexities of the attack are both $2^6 \cdot \sum_{i=0}^{50} 2^{|I_i|} = 2^{103.44}$. The memory complexity is only 2^6 counters of 51-bits, which is negligible. Since the correct key guess ensures 51 0-summations, evaluating the success probability PS is equivalent to evaluating the probability for a wrong key guess to generate 51 zero summations. Therefore, for a key-guess pattern (μ_1, μ_2), the theoretically success probability $P_S^{(\mu_1,\mu_2)}$ can be evaluated as $P_S^{(\mu_1,\mu_2)} = 1 - \prod_{i=0}^{50}(2^{-1} + \varepsilon^{(\mu_1,\mu_2)})$. We list the theoretical success probabilities for selected "bad" key-guess patterns in Table 5. It can be seen from Table 5 that the probability of successfully distinguishing between the key-guess pattern $\mu = (0, 0)$ and the "bad" key-guess pattern $\mu = (0, 1)$ is the lowest compared to all other bad key-guess patterns. It is because the corresponding superpolies always have the most significant biases under the key-guess pattern $\mu = (0, 1)$, approaching 2^{-1}.

Table 5: the success probabilities for selected "bad" key-guess patterns

$\mu = (\mu_1, \mu_2)$	$(1, 0)$	$(3, 0)$	$(5, 0)$	$(0, 1)$	$(0, 3)$	$(0, 5)$
$p^{(\mu_1,\mu_2)}$	$p_3^{(1)}$	$p_2^{(1)}$	$p_1^{(1)}$	$p_3^{(2)}$	$p_2^{(2)}$	$p_1^{(2)}$
$P_S^{(\mu_1,\mu_2)}$	0.9995967	0.9999992	0.9999999	0.6840611	0.8930240	0.9540340

According to Eq.(2) and the conclusion described in Section 5.1, we can estimate the bias of the superpoly under different key-guess patterns, thus evaluating the corresponding success probability. For the key-guess patterns (μ_1, μ_2) where $\mu_1 > 0$, the bias $\varepsilon^{(\mu_1,\mu_2)}$ satisfies $\varepsilon^{(\mu_1,\mu_2)} \leq \varepsilon^{(\mu_1,0)}$, which implies that the success probability $P_S^{(\mu_1,\mu_2)}$ satisfies $P_S^{(\mu_1,\mu_2)} \geq P_S^{(\mu_1,0)}$. Therefore, $P_S^{((1,0))}$, $P_S^{((3,0))}$, and $P_S^{((5,0))}$ can be used to evaluate the lower bound for $P_S^{(\mu_1,\mu_2)}$. We have listed the estimated success probabilities for all key-guess patterns in Table 6.

Table 6: the success probabilities for all key-guess patterns

(μ_1, μ_2)	$P_S^{(\mu_1,\mu_2)}$	(μ_1, μ_2)	$P_S^{(\mu_1,\mu_2)}$
$(1, *)$	$P_S^{(1,0)}$	$(0, 1)$	$P_S^{(0,1)}$
$(2, *)$	$P_S^{(3,0)}$	$(0, 2)$	$P_S^{(0,3)}$
$(3, *)$	$P_S^{(3,0)}$	$(0, 3)$	$P_S^{(0,3)}$
$(4, *)$	$P_S^{(5,0)}$	$(0, 4)$	$P_S^{(0,5)}$
$(5, *)$	$P_S^{(5,0)}$	$(0, 5)$	$P_S^{(0,5)}$
$(6, *)$	$P_S^{(5,0)}$	$(0, 6)$	$P_S^{(0,5)}$
$(7, *)$	$P_S^{(5,0)}$	$(0, 7)$	$P_S^{(0,5)}$

We present the attack results in Table 7. We explain the contents in the table, taking the second row as an example. We can successfully identify the key-guess patterns $\{(\mu_1, \mu_2) | 0 < \mu_1 \leq 7, 0 \leq \mu_2 \leq 7\}$ with a theoretical success rate of 99.68% as the wrong key guesses. That is, out of 64 different key guesses, we can filter out 56 wrong key guesses with a success rate of 99.68%, providing $\log_2(\frac{64}{8}) = 3$ bits key information.

Table 7: The dynamic cube attack against 190-round Grain-128AEAD

Excludable key guessing patterns (μ_1, μ_2)	The success probability	Key information available
$\{(\mu_1,\mu_2)\|0<\mu_1\leq 7, 0\leq\mu_2\leq 7\}$	99.68%	$\log_2(\frac{64}{64-56}) = 3$ bits
$\{(\mu_1,\mu_2)\|0\leq\mu_1,\mu_2\leq 7\}\backslash\{(0,\mu_2)\|0\leq\mu_2\leq 3\}$	82.57%	$\log_2(\frac{64}{64-60}) = 4$ bits
$\{(\mu_1,\mu_2)\|0\leq\mu_1,\mu_2\leq 7\}\backslash\{(0,\mu_2)\|0\leq\mu_2\leq 1\}$	65.85%	$\log_2(\frac{64}{64-62}) = 5$ bits
$\{(\mu_1,\mu_2)\|0\leq\mu_1,\mu_2\leq 7\}\backslash\{(0,0)\}$	45.04%	$\log_2(\frac{64}{64-63}) = 6$ bits

6.2 Zero-sum distinguisher for Full Grain-128

In this subsection, we provide a further discussions on the feasibility of Hao et al's dynamic cube attack against the full Grain-128.

Under the same nullification strategy $N_S = \{90, b_{158}\}$, we applied our new MILP modeling method to evaluate the cubes used in [HJL+20]. We found that their superpolies are 0-constant polynomials under both correct and wrong key guesses. This indicates that these cubes were unable to distinguish between correct and wrong key guesses.

By searching for cubes with dimensions ranging from 70 to 80, we found a batch of qualified cubes that meet the requirements: under the correct key guess, their superpolies are 0-constant polynomials; under wrong key guesses, their superpolies are non-constant polynomials with high degrees. However, these cubes were also unable to distinguish between correct and wrong key guesses because their superpolies have a bias of almost 2^{-1} under wrong key guesses. Additionally, we observed that under wrong key guesses, the minimal algebraic degrees of their superpolies are generally high, ranging from a minimum of 9 to a maximum of 23. That is why their superpolies have biases very close, even equal to 2^{-1}. We presented some cubes along with their respective minimal algebraic degrees in Table 11 (available in Appendix B) and Table 8. The comprehensive results have been uploaded to our Git repository.

Table 8: The minimal algebraic degrees of the superpolies under the "bad" key-guess patterns

cube I	$\mu=1$	$\mu=3$	$\mu=5$	cube I	$\mu=1$	$\mu=3$	$\mu=5$
I_0	15	14	13	I_{25}	13	10	9
I_1	13	11	12	I_{26}	12	10	11
I_2	10	9	9	I_{27}	15	13	13
I_3	13	11	12	I_{28}	19	17	18
I_4	14	12	12	I_{29}	19	19	15
I_5	13	11	12	I_{30}	19	15	17
I_6	11	10	10	I_{31}	20	18	19
I_7	14	9	13	I_{32}	20	19	17
I_8	12	11	11	I_{33}	22	21	19
I_9	14	12	10	I_{34}	20	17	17
I_{10}	12	11	12	I_{35}	23	21	19
I_{11}	15	14	15	I_{36}	23	17	18
I_{12}	10	9	9	I_{37}	21	17	17
I_{13}	11	10	10	I_{38}	14	9	11
I_{14}	16	14	15	I_{39}	21	20	18
I_{15}	15	13	11	I_{40}	18	16	16
I_{16}	14	12	12	I_{41}	18	16	17
I_{17}	16	13	13	I_{42}	19	18	16
I_{18}	15	11	12	I_{43}	21	18	18
I_{19}	16	12	13	I_{44}	17	17	16
I_{20}	15	13	12	I_{45}	17	14	15
I_{21}	16	12	14	I_{46}	17	14	15
I_{22}	13	10	11	I_{47}	17	13	14
I_{23}	16	12	14	I_{48}	20	17	16
I_{24}	16	12	15	I_{49}	18	15	15

As the dimension of cubes decreases and the search space increases, it becomes challenging to find cubes with 0-constant superpolies under the correct key guess. To further explore the structural characteristics of Grain-128, we attempted to recover the superpolies of the full-version Grain-128 without applying any nullification strategies. As a result, we successfully recovered the superpolies of a batch of cubes with 80 dimensions, including 9 0-constant superpolies. These cubes with 0-constant superpolies can serve as the zero-sum distinguishers for the full-version Grain-128, and listed in Table 9. All the recovered superpolies have been uploaded to our git repository.

Table 9: The zero-sum distinguishers for the full-version Grain-128

$\{0,\ldots,95\}\backslash\{12,25,27,29,30,35,39,40,41,43,57,62,67,70,76,91\}$
$\{0,\ldots,95\}\backslash\{28,29,30,31,35,39,40,41,43,44,45,61,71,76,85,89\}$
$\{0,\ldots,95\}\backslash\{13,20,29,30,38,39,40,41,43,49,52,54,63,70,76,81\}$
$\{0,\ldots,95\}\backslash\{26,30,38,40,41,43,44,50,69,72,73,75,78,84,87,90\}$
$\{0,\ldots,95\}\backslash\{17,30,33,38,39,40,41,43,44,54,60,66,70,71,92,94\}$
$\{0,\ldots,95\}\backslash\{30,31,38,39,40,41,42,43,50,53,56,69,76,81,90,93\}$
$\{0,\ldots,95\}\backslash\{15,27,32,35,38,39,40,41,43,44,50,67,72,80,86,90\}$
$\{0,\ldots,95\}\backslash\{26,30,32,39,40,41,43,44,50,55,59,67,68,73,76,90\}$
$\{0,\ldots,95\}\backslash\{18,26,30,38,39,40,41,43,44,49,50,52,53,85,90,92\}$

In our experiments without any nullification strategy, we observed that all of the recovered superpolies of high-dimensional cubes exhibit significant biases close to 2^{-1}, along with having a high algebraic degree and minimal algebraic degree. Additionally, these superpolies are very sparse. Based on the above experimental results, we infer that the superpolies of high-dimensional cubes in the full Grain-128 generally exhibit significant biases, as well as high algebraic degree and minimal algebraic degree. Moreover, when a nullification strategy is applied to the full Grain-128, even under wrong key guesses, the superpolies may still be simplified, particularly for the "bad" key guesses. Besides, the biases of superpolies may be more significant, approaching 2^{-1}. Therefore, under the framework of the dynamic cube attack proposed by Hao et al., we consider that it is difficult to carry out key-recovery attacks solely by altering the nullification strategy.

6.3 Comparison of two different bias evaluation methods

In this section, we will compare the accuracy of bias estimation method based on the PAB technique given in Section 4.2 and the method based on the minimal split set given in [HJL+20] by experimental testing on the reduced-round Grain-128AEAD. The nullification strategy and cubes employed were derived from Section 6.1. We apply the two methods to the superpolies under the key-guess pattern $\mu = (5, 0)$, respectively. The results are explicitly shown in Table 10. We provide a straightforward explanation of the data in Table 10, using the data from the 30th row as an example, which is "I_{30}, 0.252210, $\{(k_{88}, 5), (k_{89}, 4)\}$, 0.242751, $\Lambda = \{46, 47, 88, 89\}$, 0.03125". For the cube I_{30}, the practical bias obtained by directly testing the recovered superpoly is 0.252210. By using the PAB technique, we can represent the superpoly as $p'_I = p_s \oplus k_{88}g_1 \oplus k_{89}g_2$, where $\deg_m(g_1) = 5$ and $\deg_m(g_2) = 4$. Furthermore, by using the corresponding approximate polynomial, the bias of the superpoly is evaluated to be 0.242751. As for the method based on the minimal split set, the minimal split set is given by $\Lambda = \{46, 47, 88, 89\}$, and the estimated bias of the superpoly is $2^{-(|\Lambda|+1)} = 0.03125$.

It can be seen from Table 10 that the accuracy of the new bias evaluation method is much better than that of the minimal split method for Grain-128AEAD. We further investigate the reason for this phenomenon. Based on numerous experiments, we have observed that the minimal algebraic degrees of the superpolies in Grain-128AEAD are

Table 10: Bias evaluation for 190-round Grain-128AEAD under $N_S = \{(v_{23}, b_{151}), (v_{24}, b_{152})\}$ and $\mu = (5, 0)$.

Cube	Superpoly	PAB		the minimal split set	
I	Recovery	$\{(k^{\mu\omega}, d_{\omega})\}$	evaluated bias	the minimal split set Λ	evaluated bias
I_0	0.234221	$\{(k_{88}k_{89}, 3)\}$	0.236503	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_1	0.294281	$\{(k_{88}k_{89}, 3)\}$	0.289374	$\Lambda = \{47, 57, 64, 89\}$	0.03125
I_2	0.344177	$\{(k_{89}, 5)\}$	0.343991	$\Lambda = \{36, 47, 57, 89\}$	0.03125
I_3	0.241364	$\{(k_{89}, 4)\}$	0.241931	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_4	0.250311	$\{(k_{88}k_{89}, 3)\}$	0.244338	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_5	0.188051	$\{(k_{88}, 4), (k_{89}, 4)\}$	0.18449	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_6	0.258658	$\{(k_{88}k_{89}, 2)\}$	0.252482	$\Lambda = \{47, 57, 88, 89\}$	0.03125
I_7	0.250978	$\{(k_{88}k_{89}, 3)\}$	0.246790	$\Lambda = \{46, 64, 88, 89\}$	0.03125
I_8	0.213818	$\{(k_{88}, 4), (k_{89}, 4)\}$	0.207318	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_9	0.221518	$\{(k_{89}, 4)\}$	0.231354	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{10}	0.227296	$\{(k_{88}k_{89}, 4)\}$	0.225621	$\Lambda = \{46, 58, 88, 89\}$	0.03125
I_{11}	0.264006	$\{(k_{88}k_{89}, 3)\}$	0.263735	$\Lambda = \{46, 56, 88, 89\}$	0.03125
I_{12}	0.278914	$\{(k_{88}k_{89}, 4), (k_{88}k_{103}, 4)\}$	0.271873	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{13}	0.253929	$\{\}$	0.253144	$\Lambda = \{46, 47, 88, 89\}$	0.03125
I_{14}	0.275089	$\{(k_{89}, 4)\}$	0.269062	$\Lambda = \{57, 64, 88, 89\}$	0.03125
I_{15}	0.229797	$\{(k_{89}, 3)\}$	0.217179	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{16}	0.225223	$\{(k_{88}, 3)\}$	0.217594	$\Lambda = \{46, 64, 88, 89\}$	0.03125
I_{17}	0.250940	$\{(k_{89}, 5)\}$	0.247095	$\Lambda = \{58, 64, 88, 89\}$	0.03125
I_{18}	0.231102	$\{(k_{88}, 3)\}$	0.223129	$\Lambda = \{46, 64, 88, 89\}$	0.03125
I_{19}	0.206355	$\{(k_{88}, 2), (k_{89}, 5)\}$	0.189639	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{20}	0.222698	$\{(k_{88}, 3), (k_{89}, 5)\}$	0.210399	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{21}	0.224708	$\{(k_{88}, 3), (k_{89}, 5)\}$	0.216341	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{22}	0.209636	$\{(k_{89}, 4)\}$	0.204303	$\Lambda = \{46, 88, 89\}$	0.0625
I_{23}	0.244953	$\{(k_{89}, 4)\}$	0.236709	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{24}	0.317458	$\{(k_{89}, 7)\}$	0.317111	$\Lambda = \{46, 47, 48, 88\}$	0.03125
I_{25}	0.138506	$\{(k_{89}, 3)\}$	0.140588	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{26}	0.121749	$\{(k_{89}, 2)\}$	0.115462	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{27}	0.258453	$\{(k_{89}, 3)\}$	0.247838	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{28}	0.222552	$\{(k_{89}, 4)\}$	0.225904	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{29}	0.227592	$\{(k_{89}, 3)\}$	0.219513	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{30}	0.252210	$\{(k_{88}, 5), (k_{89}, 4)\}$	0.242751	$\Lambda = \{46, 47, 88, 89\}$	0.03125
I_{31}	0.28257	$\{(k_{88}k_{89}, 4)\}$	0.280329	$\Lambda = \{46, 47, 58, 88\}$	0.03125
I_{32}	0.240152	$\{(k_{89}, 4)\}$	0.243257	$\Lambda = \{46, 88, 89, 124\}$	0.03125
I_{33}	0.1509	$\{(k_{89}, 3)\}$	0.164343	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{34}	0.27245	$\{(k_{89}, 3)\}$	0.251105	$\Lambda = \{46, 47, 88, 89\}$	0.03125
I_{35}	0.23912	$\{(k_{88}k_{89}, 4)\}$	0.237271	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{36}	0.221251	$\{(k_{89}, 4)\}$	0.222968	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{37}	0.239745	$\{(k_{89}, 5)\}$	0.238529	$\Lambda = \{19, 46, 88, 89\}$	0.03125
I_{38}	0.267408	$\{(k_{88}k_{89}, 4)\}$	0.269031	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{39}	0.218067	$\{(k_{88}, 2)\}$	0.205309	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{40}	0.264484	$\{(k_{89}, 4)\}$	0.257638	$\Lambda = \{46, 58, 88, 89\}$	0.03125
I_{41}	0.280949	$\{(k_{88}k_{89}, 4)\}$	0.286235	$\Lambda = \{46, 64, 88, 89\}$	0.03125
I_{42}	0.25487	$\{(k_{89}, 3)\}$	0.242833	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{43}	0.214379	$\{(k_{89}, 3)\}$	0.216105	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{44}	0.230568	$\{(k_{89}, 3)\}$	0.22729	$\Lambda = \{47, 57, 88, 89\}$	0.03125
I_{45}	0.185132	$\{(k_{88}, 2), (k_{89}, 3)\}$	0.163287	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{46}	0.241527	$\{\}$	0.241939	$\Lambda = \{40, 46, 88, 89\}$	0.03125
I_{47}	0.198668	$\{(k_{89}, 3)\}$	0.200201	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{48}	0.200107	$\{(k_{89}, 4), (k_{88}, 3)\}$	0.18751	$\Lambda = \{46, 64, 88, 89\}$	0.03125
I_{49}	0.195266	$\{(k_{88}, 2)\}$	0.187071	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625
I_{50}	0.272528	$\{(k_{89}, 5)\}$	0.277182	$\lvert\Lambda\rvert \geq 5$	≤ 0.015625

typically very high under wrong key guesses. Therefore, for the identified minimal split set, the superpoly still exhibits a high minimal algebraic degree and significant bias over its complement space, leading to the failure of Assumption 1 on Grain-128AEAD in the method based on the minimal split set. For the full version of Grain-128, we also observed similar algebraic properties for superpolis under the wrong key guesses, which fails Assumption 1 on Grain-128.

7 Conclusion

In this paper, we explain that the dynamic cube attack proposed by Hao et al. against Grain-128 is invalid. Furthermore, we present a reliable implementation of dynamic cube attacks against the Grain family of ciphers, including wrong key-guess analysis and a more accurate MILP modeling method based on 3SDP/u. Additionally, we introduce a

new technique called Polynomial Approximation with respect to Bias (PAB), which can provide a more accurate bias evaluation. As an application, we implement a dynamic cube attack against the Grain family of ciphers. As a result, we present a key-recovery attack on 190-round Grain-128AEAD and discuss the feasibility of dynamic cube attacks on full-version Grain-128. Furthermore, through experiments, we demonstrate that the bias estimation method based on the minimal split set is unsuitable for the Grain family of ciphers. Simultaneously, we show the effectiveness and accuracy of our new method for evaluating bias. We believe this new method can play an important role in dynamic cube attacks on more rounds.

References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 1–22, Berlin, Heidelberg, 2009. Springer.

[ÅHJM11] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of grain-128 with optional authentication. *Int. J. Wirel. Mob. Comput.*, 5(1):48–59, 2011.

[DGP+11] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 327–343, Berlin, Heidelberg, 2011. Springer.

[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, Berlin, Heidelberg, 2009. Springer.

[DS11] Itai Dinur and Adi Shamir. Breaking grain-128 with dynamic cube attacks. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011.

[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, Cham, 2022. Springer.

[HIJ+19] Yonglin Hao, Takanori Isobe, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. Improved division property based cube attacks exploiting

algebraic properties of superpoly. *IEEE Trans. Computers*, 68(10):1470–1486, 2019.

[HJL+20] Yonglin Hao, Lin Jiao, Chaoyun Li, Willi Meier, Yosuke Todo, and Qingju Wang. Links between division property and other cube attack variants. *IACR Trans. Symmetric Cryptol.*, 2020(1):363–395, 2020.

[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, Cham, 2020. Springer.

[HLM+21] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset. *J. Cryptol.*, 34(3):22, 2021.

[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, Cham, 2021. Springer.

[HTZ13] Robert V. Hogg, Elliot A. Tanis, and Dale L. Zimmerm. *Probability and statistical inference (ninth Edition)*. Pearson Education, Inc., 2013.

[RBMA16] Majid Rahimi, Mostafa Barmshory, Mohammad Hadi Mansouri, and Mohammad Reza Aref. Dynamic cube attack on grain-v1. *IET Inf. Secur.*, 10(4):165–172, 2016.

[Sun21] Yao Sun. Automatic search of cubes for attacking stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2021(4):100–123, 2021.

A The MILP models of the 3SDP/u for three basic functions

We introduce the MILP models for the three basic functions COPY, AND, and XOR. Note that these models are first proposed in [HLM+20, HLM+21], and describe the generalized forms of their propagation rules, which are easily deduced from their original forms. When constructing the MILP model, we usually initialize an empty model $\mathcal{M}$, then generate some MILP variables $v_1, \ldots, v_m$ by $\mathcal{M}.var \leftarrow v_1, \ldots, v_m$, which are used in the inequalities. Next, the inequality is added to the model $\mathcal{M}.con \leftarrow v_2 + \cdots + v_m \geq v_1$. Finally, we can call an off-the-shelf MILP solver to solve the model and obtain the results, including feasible or infeasible. In this paper, the MILP tool we used is the Gurobi optimizer.

Proposition 4 (MILP model for COPY)**.** *Let $a \xrightarrow{COPY} (b_0, b_1, \ldots, b_{m-1})$ be a three-subset division trail of **COPY**. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for **COPY**.*

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

Note that the Gurobi optimizer supports the Or ($\vee$) operation.

Proposition 5 (MILP model for AND)**.** *Let $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{AND} b$ be a three-subset division trail of **AND**. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for **AND**.*

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

Proposition 6 (MILP model for XOR)**.** *Let $(a_0, a_1, \ldots, a_{m-1}) \xrightarrow{XOR} b$ be a three-subset division trail of **XOR**. The following inequalities are sufficient to describe the propagation of the modified three-subset division property for **XOR**.*

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

B The cubes used in Section 6.2

Table 11: The cubes used in Section 6.2

I_0	$\{0,\ldots,95\}\backslash\{8,25,27,28,29,30,33,35,40,41,43,44,45,49,59,66,67,69,75,76,84,87\}$
I_1	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,49,59,66,69,70,73,74,75,76,84\}$
I_2	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,49,59,66,69,73,74,75,76,84,89\}$
I_3	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,67,69,73,74,75,76,84,87\}$
I_4	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,67,69,74,75,76,84,87,92\}$
I_5	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,67,69,74,75,76,84,91,92\}$
I_6	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,69,70,73,74,75,76,84,86\}$
I_7	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,69,70,74,75,76,84,87,92\}$
I_8	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,33,35,40,41,43,44,45,59,66,69,73,74,75,76,84,86,87\}$
I_9	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,39,40,41,43,44,45,49,59,66,67,69,74,75,76,84,89\}$
I_{10}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,39,40,41,43,44,45,54,59,66,67,69,74,75,76,84,89\}$
I_{11}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,39,40,41,43,44,45,55,59,66,67,69,74,75,76,84,87\}$
I_{12}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,39,40,41,43,44,45,59,66,67,69,73,74,75,76,84,89\}$
I_{13}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,39,40,41,43,44,45,59,66,67,69,74,75,76,84,89,91\}$
I_{14}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,40,41,43,44,45,49,55,57,59,66,67,69,74,75,76,84\}$
I_{15}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,40,41,43,44,45,49,55,59,66,67,69,74,75,76,84,89\}$
I_{16}	$\{0,\ldots,95\}\backslash\{8,25,27,29,30,35,40,41,43,44,45,49,57,59,66,67,69,74,75,76,84,91\}$
I_{17}	$\{0,\ldots,95\}\backslash\{26,27,28,29,30,35,37,39,40,41,43,44,45,49,61,69,70,74,84,86,92,94\}$
I_{18}	$\{0,\ldots,95\}\backslash\{26,27,28,29,30,35,37,39,40,43,44,45,49,61,69,70,74,76,84,86,92,94\}$
I_{19}	$\{0,\ldots,95\}\backslash\{26,27,28,30,35,37,39,40,41,43,44,45,49,61,69,70,74,76,84,86,92,94\}$
I_{20}	$\{0,\ldots,95\}\backslash\{26,27,28,30,35,37,39,40,43,44,45,49,60,61,69,70,74,76,84,86,92,94\}$
I_{21}	$\{0,\ldots,95\}\backslash\{26,28,30,31,32,35,36,37,40,45,46,49,50,51,54,57,62,68,74,76,92,94\}$
I_{22}	$\{0,\ldots,95\}\backslash\{26,28,30,31,32,35,36,37,40,46,49,50,51,54,57,62,68,70,74,76,92,94\}$
I_{23}	$\{0,\ldots,95\}\backslash\{26,28,30,31,35,36,37,40,45,46,49,50,51,54,57,62,68,70,74,76,92,94\}$
I_{24}	$\{0,\ldots,95\}\backslash\{21,26,30,31,35,36,37,40,45,46,49,50,51,54,57,62,68,70,74,76,92,94\}$
I_{25}	$\{0,\ldots,95\}\backslash\{26,27,28,29,30,35,37,39,40,43,44,49,61,66,69,70,74,76,84,86,92,94\}$
I_{26}	$\{0,\ldots,95\}\backslash\{26,27,28,30,35,37,39,40,41,43,44,49,61,66,69,70,74,76,84,86,92,94\}$
I_{27}	$\{0,\ldots,95\}\backslash\{26,27,28,29,30,35,37,39,40,41,43,44,49,61,66,69,70,74,84,86,92,94\}$
I_{28}	$\{0,\ldots,95\}\backslash\{21,27,28,30,33,41,42,44,45,47,50,53,55,56,61,67,70,74,75,77,82,87\}$
I_{29}	$\{0,\ldots,95\}\backslash\{27,29,33,35,39,41,43,44,47,49,53,57,61,67,70,71,74,77,83,87,94,95\}$
I_{30}	$\{0,\ldots,95\}\backslash\{27,29,30,35,39,41,43,44,47,49,53,55,57,61,70,71,74,75,77,83,87,95\}$
I_{31}	$\{0,\ldots,95\}\backslash\{27,29,30,35,39,41,43,44,47,49,53,57,61,67,70,71,74,75,77,83,87,95\}$
I_{32}	$\{0,\ldots,95\}\backslash\{27,29,35,39,41,43,44,47,49,53,55,57,61,67,70,71,74,75,77,83,87,95\}$
I_{33}	$\{0,\ldots,95\}\backslash\{27,29,33,35,39,41,43,44,47,49,53,57,61,67,70,71,74,75,77,83,87,95\}$
I_{34}	$\{0,\ldots,95\}\backslash\{27,28,29,30,35,39,41,43,44,47,49,53,57,61,67,70,71,74,77,83,87,95\}$
I_{35}	$\{0,\ldots,95\}\backslash\{27,29,30,33,35,39,41,43,44,47,49,53,57,61,67,70,71,74,77,83,87,95\}$
I_{36}	$\{0,\ldots,95\}\backslash\{27,29,30,35,39,41,43,44,45,47,49,53,57,61,67,70,71,74,77,83,87,95\}$
I_{37}	$\{0,\ldots,95\}\backslash\{27,28,29,30,35,39,41,43,44,45,47,49,53,57,61,70,71,74,77,83,87,95\}$
I_{38}	$\{0,\ldots,95\}\backslash\{23,27,28,31,35,37,39,40,41,43,45,49,50,51,57,61,62,67,71,73,74,87\}$
I_{39}	$\{0,\ldots,95\}\backslash\{21,27,29,30,33,35,39,41,43,44,47,49,53,57,61,70,71,74,77,83,87,95\}$
I_{40}	$\{0,\ldots,95\}\backslash\{21,27,29,30,31,35,39,40,41,43,44,49,53,57,61,70,71,74,77,83,87,95\}$
I_{41}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,44,45,49,53,57,61,70,71,74,77,83,87,95\}$
I_{42}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,44,49,53,57,61,66,70,71,74,77,83,87,95\}$
I_{43}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,44,49,53,57,61,67,70,71,74,77,83,87,95\}$
I_{44}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,44,49,53,57,61,69,70,71,74,77,83,87,95\}$
I_{45}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,44,49,53,57,61,70,71,74,77,83,87,94,95\}$
I_{46}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,45,49,53,57,61,67,70,71,74,77,83,87,95\}$
I_{47}	$\{0,\ldots,95\}\backslash\{21,27,29,30,35,39,40,41,43,49,53,57,61,67,69,70,71,74,77,83,87,95\}$
I_{48}	$\{0,\ldots,95\}\backslash\{21,27,29,35,39,40,41,43,44,49,53,57,61,67,70,71,74,75,77,83,87,95\}$
I_{49}	$\{0,\ldots,95\}\backslash\{21,27,29,35,39,40,41,43,44,49,53,57,61,67,70,71,74,77,83,87,94,95\}$