Combining MILP Modeling with Algebraic Bias Evaluation for Linear Mask Search: Improved Fast Correlation Attacks on SNOW

Xinxin Gong[1], Yonglin Hao[1*] and Qingju Wang[2]

[1]State Key Laboratory of Cryptology, Beijing, 100878, China.
[2]SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg.

*Corresponding author(s). E-mail(s): haoyonglin@yeah.net;
Contributing authors: xinxgong@126.com; qjuwang@gmail.com;

Abstract

The Mixed Integer Linear Programming (MILP) technique has been widely applied in the realm of symmetric-key cryptanalysis. In this paper, we propose a new bitwise breakdown MILP modeling strategy for describing the linear propagation rules of modular addition-based operations. We apply such new techniques to cryptanalysis of the SNOW stream cipher family and find new linear masks: we use the MILP model to find many linear mask candidates among which the best ones are identified with particular algebraic bias evaluation techniques. For SNOW 3G, the correlation of the linear mask we found is the highest on record: such results are highly likely to be optimal according to our analysis. For SNOW 2.0, we find new masks matching the correlation record and many new sub-optimal masks applicable to improving correlation attacks. For SNOW-V/Vi, by investigating both bitwise and truncated linear masks, we find all linear masks having the highest correlation, and prove the optimum of the corresponding truncated patterns under the "fewest active S-box preferred" strategy. By using the newly found linear masks, we give correlation attacks on the SNOW family with improved complexities. We emphasize that the newly proposed uniform MILP-aided framework can be potentially applied to analyze LFSR-FSM structures composed of modular addition and S-box as non-linear components.

Keywords: SNOW Family Stream ciphers, Bitwise Breakdown, MILP Modeling, Fast Correlation Attack

1 Introduction

MILP-Aided Symmetric-Key Cryptanalysis. Ever since the introduction by Mouha *et al.* [1], the MILP-aided method has enjoyed great success in the realm of symmetric-key cryptanalysis. MILP models have been used in searching for differential and linear [2–4] characteristics, and integral distinguishers [5, 6] for block ciphers, constructing distinguishers on hash functions [7], and launching cube attacks on stream ciphers [8–12], etc. They are also used in FCA on bit-oriented stream ciphers [13]. The MILP-aided method constructs a model $\mathcal{M}$ capturing the cryptographic property propagation rules of the targeted primitive. Then, $\mathcal{M}$ is solved with some off-the-shelf solvers like Gurobi [14]. Finally, the targeted characteristics, such as linear masks, are extracted from the solution of $\mathcal{M}$.

For describing linear propagation rules, MILP models are quite efficient when coping with simple operation such as the exclusive-OR ($\oplus$), branching and S-boxes of ≤ 4 bits. However, the MILP modeling of complicated nonlinear operations such as the modular 2^m addition (denoted as $\boxplus_m$) and the 8-bit S-box transformation (`SB`) can be quite challenging. The $\boxplus_m$ operation has a linear mask searching space of $O(2^{2m})$ which is computationally infeasible to exhaust since $m = 32, 64$ are typical choices in symmetric-key primitives. As to `SB`, though several optimizations [15–17] are made so far, an 8-bit S-box still requires over 2400 linear constraints making the $\mathcal{M}$ hard to solve. On the other hand, 8-bit S-boxes and modular additions are basic building blocks of many symmetric-key cryptographic primitives. Therefore, applying the MILP modeling to the cryptanalysis of such primitives can be quite challenging.

The SNOW Family. The symmetric-key cryptographic primitives such as block ciphers, stream ciphers, hash functions play an important role in the cyber world protecting the security of information, where stream ciphers stand out for their high efficiency and low resource consumptions. Many stream ciphers are designed and selected as international standards for world-wide applications, among which are three main members of the SNOW stream cipher family namely SNOW 2.0, SNOW 3G and SNOW-V/Vi. SNOW 2.0 [18], proposed by Ekdahl and Johansson, was selected as an standard stream cipher in ISO/IEC 18033-4. SNOW 3G [19], designed in 2006 by ETSI/SAGE, serves as the core of 3GPP Confidentiality and Integrity Algorithms UEA 2 & UIA2 for UMTS and LTE networks. It is currently in use in 3G/4G mobile communication systems. Both SNOW 2.0 and SNOW 3G enjoy a 128-bit security. In response to the new requirements from 3GPP encryption, SNOW-V [20] and SNOW-Vi [21] are proposed recently. Since the close similarity between two designs[1], we refer to both of them as SNOW-V/Vi hereafter. Targeting at standards of 5G and beyond, SNOW-V/Vi has extremely high software performance and been claimed to have a 256-bit security.

All members of SNOW family use the classic structure LFSR-FSM: the linear feedback shift register (LFSR) serves as the source of pseudo-randomness

[1] They only differ in the LFSR updating function which makes no difference in our analysis.

and the nonlinear finite state machine (FSM) disrupts the linearity using 8-bit S-boxes and modular 2^{32} add as building blocks. The SNOW family members enjoy high efficiencies in both software and hardware environments.

Fast Correlation Attacks. One of the most effective cryptanalytic method for stream ciphers of LFSR-FSM structure is the fast correlation attack [22–24], referred as FCA hereafter. Same as the linear attacks on block ciphers utilizing the correlation between the plaintext $\boldsymbol{s}_0$ and the ciphertext $\boldsymbol{s}_N$, FCA on stream ciphers is based on the linear correlation between the LFSR bits $\boldsymbol{\ell}$ and output bits $\boldsymbol{z}$. Therefore, for FCA, finding linear masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ for $\boldsymbol{\ell}$ and $\boldsymbol{z}$ of stream ciphers such that $\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell} \oplus \Gamma_{\boldsymbol{z}} \cdot \boldsymbol{z} = 0$ hold with high correlations is of the great importance.

Great efforts have been made in finding high-correlation $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$, resulting in various FCA on SNOW stream ciphers [25–35]. The best FCAs for SNOW 2.0 [28] and SNOW 3G [30] have data/time/memory complexities of well over 2^{128}, while the complexities corresponding to the best FCAs on SNOW-V/Vi [33, 34] are below 2^{256}, violating the claimed 256-bit security.

Motivations. It is noticeable that most of the $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s used in FCAs on SNOW are deduced either by hand or through partial exhaustive search according to some intuitive strategy: even the SAT-based automatic search in [35][2] uses directly the truncated pattern in [34]. However, whether there are better linear masks are largely unknown. Therefore, there is an urgent demand of uniform frameworks for efficiently finding many $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s that are likely to have high correlations. With so many $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ candidates, one needs a method to compute their correlations accurately and efficiently so as to identify the masks applicable to FCAs. To sum up, the whole linear mask search process is divided into two tasks as follows:

- ***Candidate Search***: Find many $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s as candidates with a uniform framework.
- ***Correlation Computation***: For each candidate, compute the correlation accurately for further selections.

Since the discussion of optimality is the strength of MILP models, we are to propose a MILP model based framework for accomplishing the first task *Candidate Search* so as to improve the FCA results on primitives using both modular additions and large S-boxes.

Our Contributions. In this paper, we make progress in both *Candidate Search* and *Correlation Computation* aspects.

For *Candidate Search*, we propose a new bitwise breakdown strategy for modeling the linear propagation rules of modular addition-based operations with MILP models. Following our strategy, a modular 2^m addition is broken down to m bitwise additions from the least significant bit (LSB) to the most significant bit (MSB). Each bitwise addition can be regarded as a small S-box whose linear propagation rules can be captured with MILP constraints deduced

[2] In fact, this work is accomplished independently and almost in parallel with [35].

with the H-representation technique in [2]. This strategy is not only quite easy for understanding but also flexible for describing the linear propagation of consecutive modular addition-based operations and deducing optimal truncated linear masks. Combining such new modular addition descriptions with a simple S-box modeling method, we propose a uniform MILP-aided framework for finding linear mask candidates which are of independent interest.

For *Correlation Computation* of SNOW-V/Vi linear masks, we propose an accurate correlation computation algorithm for particular composition function F using the algebraic bias evaluation technique. Such a technique has been proved effective in previous cryptanalysis of SNOW 2.0 and SNOW 3G [28, 30].

Thanks to new techniques, we are able to find many linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s with currently the highest correlations (i.e., having the largest value $|\mathtt{Cor}|$) for the targets in SNOW family.

- For SNOW 3G, we find 18 masks with $|\mathtt{Cor}| > 2^{-21}$: 15 of them are new; the best three breaks the previous correlation record in [30] setting a new record of $|\mathtt{Cor}| = 2^{-20.386}$. There are another 175 masks with $2^{-22} < |\mathtt{Cor}| \leq 2^{-21}$.
- For SNOW 2.0, we find 26 masks with $|\mathtt{Cor}| > 2^{-15}$: 23 of them are new; the best 4 ties the highest correlation record of $\pm 2^{-14.411}$ originated in [28]. We compare the linear mask results in Table E8 of Section E.2.
- For SNOW-V/Vi, we investigate multiple highly qualified truncated linear mask candidates, the best of which enables us to find 8 bitwise mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s with $|\mathtt{Cor}| > 2^{-48}$ and another 127 with $2^{-49} \leq |\mathtt{Cor}| < 2^{-48}$: such results are in accordance with those in [35] while different methods are used. We also investigate linear masks following sub-optimal truncated patterns whose correlation cannot be larger than $|\mathtt{Cor}| = 2^{-50.816}$, indicating the optimum of current results.

Table 1: Comparison of FCAs on SNOW 2.0, SNOW 3G and SNOW-V/Vi

Cipher	Ref.	Data	Time	Memory
SNOW 2.0	[28]	$2^{159.62}$	$2^{162.88}$	$2^{162.32}$
	This paper	$2^{156.75}$	$2^{163.66}$	$2^{163.36}$
SNOW 3G	[29]	$2^{176.56}$	$2^{176.92}$	$2^{176.56}$
	[30]	$2^{172.42}$	$2^{174.98}$	$2^{174.17}$
	This paper	$2^{170.81}$	$2^{174.95}$	$2^{174.13}$
	This paper	$2^{169.52}$	$2^{175.85}$	$2^{175.07}$
SNOW-V/Vi	[34]	$2^{237.50}$	$2^{246.53}$	$2^{238.77}$
	[35]†	$2^{239.30}$	$2^{247.22}$	$2^{239.32}$
	[35]†	$2^{237.81}$	$2^{246.06}$	$2^{238.17}$
	[35]†	$2^{236.87}$	$2^{247.79}$	$2^{239.88}$
	[35]†	$2^{234.88}$	$2^{246.40}$	$2^{238.51}$
	This paper	$2^{231.76}$	$2^{247.38}$	$2^{239.48}$

†: In [35], the complexity for finding all pairs of vectors colliding on some bits from N vectors is under-estimated as $\mathcal{O}(\sqrt{2}N)$. We unify this estimation as $\mathcal{O}(N \log_2 N)$, which is more accurate.

With the new highly qualified linear masks, we can directly propose new FCAs with improved data complexities for SNOW 2.0 and SNOW 3G. For

SNOW-V/Vi, we give a FCA using multiple linear masks and thus obtain new trade-off points. We compare our FCAs with the current best ones in Table 1 showing that ours have the lowest data complexities.

Organization of the Paper. Section 2 provides the necessary background and notations used throughout the paper. The idea of bitwise breakdown strategy is described in Section 3, followed by the MILP modeling in Section 4. With the help of an algebraic bias evaluation of composition functions in Section 5, we apply the MILP modeling technique to the *Candidate Search* task of three targets of SNOW in Section 6, obtaining masks of the highest correlation. The selected linear masks are then used for new FCAs in Section 7. We summarize the whole paper in Section 8.

2 Preliminaries

2.1 Notations and Definitions

We first introduce some notations and definitions used throughout this paper. The binary field is denoted by $\mathbb{F}_2$ and its m-dimensional extension field is denoted by $\mathbb{F}_2^m$ whose elements are m-dimensional binary vector of the form $\boldsymbol{a} = (a_0, a_1, \ldots, a_{m-1}) \in \mathbb{F}_2^m$: a_0 is LSB and a_{m-1} is MSB. The bitwise linear mask of $\boldsymbol{a} \in \mathbb{F}_2^m$ is naturally denoted as $\Gamma_{\boldsymbol{a}} \in \mathbb{F}_2^m$. One instance of $\boldsymbol{a}$ with static value can also be represented as HEX numbers: $\boldsymbol{a} = (0,1,0,0,0,1,1,1) \Leftrightarrow \boldsymbol{a} =$ `0xe2`. Besides, a matrix $M \in \mathbb{F}_2^{m \times n}$ represents a 0-1 matrix of size $m \times n$. The bitwise XOR is denoted by "$\oplus$" and the AND of $a, b \in \mathbb{F}_2$ is represented as $a \cdot b$ or simply ab. For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^m$, the standard inner product over $\mathbb{F}_2^m$ is defined as $\boldsymbol{a} \cdot \boldsymbol{b} = \bigoplus_{i=0}^{m-1} a_i b_i$. The addition modulo 2^m is $\boxplus_m$ and we may use $\boxplus_m^n$ for n consecutive modular additions: specifically, the case of $n = 1$ in Eq. (1) is denoted as $\boxplus_m$ and referred as the *ordinary* addition while the case of $n = 2$ in Eq. (2) is referred as the *consecutive* addition hereafter.

$$\boxplus_m : \boldsymbol{z} = \boldsymbol{x} \boxplus_m \boldsymbol{y}, \tag{1}$$

$$\boxplus_m^2 : \boldsymbol{z} = \boldsymbol{x} \boxplus_m \boldsymbol{y} \boxplus_m \boldsymbol{w}. \tag{2}$$

The summation and multiplication over real numbers are simply denoted as "+" and "∗".

Let n, m be two positive integers such that m divides n and $d = \frac{n}{m}$. For $\boldsymbol{x} \in \mathbb{F}_2^n$, and its bitwise linear mask $\Gamma_{\boldsymbol{x}} \in \mathbb{F}_2^n$, we can write $\boldsymbol{x}$ as $\boldsymbol{x} = (\boldsymbol{x}_0 \parallel \cdots \parallel \boldsymbol{x}_{d-1})$, where $\boldsymbol{x}_i \in \mathbb{F}_2^m$ for $i = 0, \ldots, d-1$ and $\boldsymbol{x}_0$ is the least significant part[3]. Corresponding to the *bitwise* linear mask $\Gamma_{\boldsymbol{x}}$, the *truncated* linear mask is defined naturally as $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_{d-1}}) \in \mathbb{F}_2^d$, where

$$T_{\boldsymbol{x}_i} = \begin{cases} 0, & \text{if } \Gamma_{\boldsymbol{x}_i} = \boldsymbol{0} \, (\in \mathbb{F}_2^m) \\ 1, & \text{otherwise} \end{cases}$$

[3]Note that $\Gamma_{\boldsymbol{x}} \cdot \boldsymbol{x} = \bigoplus_{i=0}^{d-1} \Gamma_{\boldsymbol{x}_i} \cdot \boldsymbol{x}_i$, where $\Gamma_{\boldsymbol{x}_i} \in \mathbb{F}_2^m$ is the bitwise linear mask of $\boldsymbol{x}_i \in \mathbb{F}_2^m$.

For $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_2^n$, $\boldsymbol{z} = \boldsymbol{x} \boxplus_m \boldsymbol{y} \in \mathbb{F}_2^n$ is acquired by d parallel operations of the addition modulo 2^m as: $\boldsymbol{z}_i = \boldsymbol{x}_i \boxplus_m \boldsymbol{y}_i \in \mathbb{F}_2^m$ for $i = 0, \ldots, d-1$. For SNOW-V/Vi, $\boxplus_{32}$ is carried out on 128-bit states by the parallel applications of four additions modulo 2^{32} over each sub-word.

The AES round function, denoted as AES^R, is used in the FSM updating function of SNOW-V/Vi, which is composed of SubBytes (SB), ShiftRows (SR) and MixColumns (MC) as $AES^R(\cdot) = \texttt{MC} \circ \texttt{SR} \circ \texttt{SB}(\cdot)$ with no AddRoundKey operation included here. Note that the branch number of MC in each column is 5. Let $\boldsymbol{s}$ be the 128-bit input to $AES^R(\cdot)$, then $\boldsymbol{s}$ is divided bytewisely and mapped to the state array of the AES round function as follows:

$$\boldsymbol{s} = (\boldsymbol{s}_0 \| \ldots \| \boldsymbol{s}_{15}) = \begin{pmatrix} \boldsymbol{s}_0 & \boldsymbol{s}_4 & \boldsymbol{s}_8 & \boldsymbol{s}_{12} \\ \boldsymbol{s}_1 & \boldsymbol{s}_5 & \boldsymbol{s}_9 & \boldsymbol{s}_{13} \\ \boldsymbol{s}_2 & \boldsymbol{s}_6 & \boldsymbol{s}_{10} & \boldsymbol{s}_{14} \\ \boldsymbol{s}_3 & \boldsymbol{s}_7 & \boldsymbol{s}_{11} & \boldsymbol{s}_{15} \end{pmatrix}$$

When analyzing the truncated linear masks, we constantly use the *bytewise* truncation: for $\boldsymbol{x} \in \mathbb{F}_2^n$, its truncated linear mask $T_{\boldsymbol{x}}$ is always of the length $\frac{n}{8}$. Therefore, for the bitwise linear mask $\Gamma_{\boldsymbol{s}}$ of $\boldsymbol{s} \in \mathbb{F}_2^{128}$ in SNOW-V/Vi, the corresponding bytewise-truncated linear mask $T_{\boldsymbol{s}}$ has 16 entries.

In FCAs on LFSR-based stream ciphers, the output bits are regarded as general vectorial Boolean functions of internal state bits, where the correlation, denoted as Cor, is one of the most important parameters for evaluating the strength of the linear approximation and also the efficiency of FCAs.

Definition 1 For an arbitrary vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ and its input-output linear mask pair $(\Gamma_i, \Gamma_o) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$, the correlation $\texttt{Cor}_F(\Gamma_i, \Gamma_o)$ (or simply $\texttt{Cor}(\Gamma_i, \Gamma_o)$ when F is obvious from the context) is defined as

$$\texttt{Cor}_F(\Gamma_i, \Gamma_o) = \Pr_{\boldsymbol{x} \leftarrow \mathbb{F}_2^n}\{\Gamma_o \cdot F(\boldsymbol{x}) = \Gamma_i \cdot \boldsymbol{x}\} - \Pr_{\boldsymbol{x} \leftarrow \mathbb{F}_2^n}\{\Gamma_o \cdot F(\boldsymbol{x}) \neq \Gamma_i \cdot \boldsymbol{x}\}.$$

When $\texttt{Cor}_F(\Gamma_i, \Gamma_o) \neq 0$, we know that Γ_i can propagate to Γ_o following the linear propagation rule of F: such (Γ_i, Γ_o) are referred as "available" masks whose correlations are also denoted as $\texttt{Cor}(\Gamma_i \xrightarrow{F} \Gamma_o)$. Specifically, for linear function F, for each $\Gamma_i \in \mathbb{F}_2^n$, there exists $\Gamma_o \in \mathbb{F}_2^m$ making $\texttt{Cor}_F(\Gamma_i, \Gamma_o) = 1$.

2.2 Brief Descriptions of SNOW-V/Vi, SNOW 3G and SNOW 2.0

All members of SNOW family use the classic LFSR-FSM structure, where the LFSR serves as the source of pseudo-randomness and the FSM disrupts the linearity. For our targeted SNOW-V/Vi, SNOW 3G and SNOW 2.0 stream ciphers, we only provide the description of the non-linear FSM part in keystream generation phase. For more details on the designs, we refer to the original specification documents [18–21]. Specifically, we use $\boxplus$ to represent $\boxplus_{32}$ in SNOW 2.0 and SNOW 3G descriptions.

2.2.1 SNOW-V/Vi

The overall schematic of SNOW-V algorithm is shown in Fig. A1 of Appendix A. The LFSR part of SNOW-V/Vi [20, 21] consists of two LFSRs, namely LFSR-A[4] and LFSR-B, both of 16 cells of length 16, giving 512 bits in total. The FSM part consists of three 128-bit registers namely $\boldsymbol{R1}$, $\boldsymbol{R2}$ and $\boldsymbol{R3}$. At each time instance, two 128-bit states from LFSR-A and LFSR-B denoted as $\boldsymbol{T1}, \boldsymbol{T2}$ are first output, and then the FSM takes the two blocks as inputs and produces a 128-bit keystream as Eq. (3)

$$\boldsymbol{z}_t = (\boldsymbol{T1}_t \boxplus_{32} \boldsymbol{R1}_t) \oplus \boldsymbol{R2}_t \tag{3}$$

After z_t is output, the FSM is updated according to Eq. (4)

$$\begin{cases} \boldsymbol{R1}_{t+1} = \sigma(\boldsymbol{R2}_t \boxplus_{32} (\boldsymbol{R3}_t \oplus \boldsymbol{T2}_t)) \\ \boldsymbol{R2}_{t+1} = AES^R(\boldsymbol{R1}_t) \\ \boldsymbol{R3}_{t+1} = AES^R(\boldsymbol{R2}_t) \end{cases} \tag{4}$$

where $\sigma = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15]$ is a bytewise permutation. We emphasize that, for the linear approximation of the FSM of SNOW-V/Vi, we extract the function $\mathcal{F}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sigma(\boldsymbol{x} \boxplus_{32} \boldsymbol{y}) \boxplus_{32} \boldsymbol{z}$, denoted as $\mathcal{F}$-function, which is the unique building block of SNOW-V/Vi. For computing the correlations under any given linear masks, we propose an efficient algorithm in later Section 5 using the algebraic bias evaluation technique.

2.2.2 SNOW 3G

The overall schematic of SNOW 3G is shown in Fig. A2 of Appendix A. SNOW 3G [19] has a 512-bit LFSR consisting of 16 32-bit words denoted as $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{15}$. Specifically, for $i = 0, \ldots, 15$, the value of $\boldsymbol{s}_i$ at time instance t is denoted as $\boldsymbol{s}_{t+i}$. The FSM of SNOW 3G consists of three 32-bit registers namely $\boldsymbol{R1}$, $\boldsymbol{R2}$ and $\boldsymbol{R3}$. For time instance t, the output of SNOW 3G is the 32-bit word $\boldsymbol{z}_t$ computed as Eq. (5).

$$\boldsymbol{z}_t = \boldsymbol{s}_t \oplus (\boldsymbol{s}_{t+15} \boxplus \boldsymbol{R1}_t) \oplus \boldsymbol{R2}_t \tag{5}$$

The FSM are updated afterwards as Eq. (6)

$$\begin{cases} \boldsymbol{R1}_{t+1} = \boldsymbol{R2}_t \boxplus (\boldsymbol{R3}_t \oplus \boldsymbol{s}_{t+5}) \\ \boldsymbol{R2}_{t+1} = \mathtt{S}_1(\boldsymbol{R1}_t) \\ \boldsymbol{R3}_{t+1} = \mathtt{S}_2(\boldsymbol{R2}_t) \end{cases} \tag{6}$$

$\mathtt{S}_1$ and $\mathtt{S}_2$ are two permutations over $\mathbb{F}_2^{32}$ ($\mathtt{S}_1$ is also used in SNOW 2.0) which can be represented as $\mathtt{S}_i(\cdot) = \mathtt{LL}_i \circ \mathtt{SBX}_i(\cdot)$ for $i = 1, 2$, where $\mathtt{SBX}_i$ is a nonlinear

[4]SNOW-Vi is exactly the same as SNOW-V, with the only difference in the LFSR update function and the tap $\boldsymbol{T2}$ moved to the higher half of LFSR-A.

operation using four 8-bit S-boxes in parallel and $\mathtt{LL}_i$ is the linear transformation similar to the $\mathtt{MC}$ in AES^R. The branch number of $\mathtt{LL}_i$ is also 5. Refer to [19] for more details of SNOW 3G.

2.2.3 SNOW 2.0

SNOW 2.0 [18] is quite similar to SNOW 3G. It also has a 512-bit LFSR consisting of 16 32-bit words. However, the FSM only contains two 32-bit registers $\boldsymbol{R1}$ and $\boldsymbol{R2}$. At the time instance t, the 32-bit output $\boldsymbol{z}_t$ is computed as

$$\boldsymbol{z}_t = \boldsymbol{s}_t \oplus (\boldsymbol{s}_{t+15} \boxplus \boldsymbol{R1}_t) \oplus \boldsymbol{R2}_t \tag{7}$$

and the updating function of FSM is Eq. (8)

$$\begin{cases} \boldsymbol{R1}_{t+1} = \boldsymbol{s}_{t+5} \boxplus \boldsymbol{R2}_t \\ \boldsymbol{R2}_{t+1} = \mathtt{S}_1(\boldsymbol{R1}_t) \end{cases} \tag{8}$$

The overall schematic of SNOW 2.0 is shown in Fig. A3 of Appendix A. Refer to [18] for more details of SNOW 2.0.

2.3 Linear Masks for Correlation Attack on Stream Ciphers

The correlation attack works especially for stream ciphers with LFSRs. The main idea for this type of attack is using linear approximations of the nonlinear operations in the cipher and to derive a linear relationship between the outputs and the LFSR states, then the correlation of such a linear relation is utilized to launch attacks. For stream ciphers, suppose some output bits $\boldsymbol{z}$ can be represented as a function of internal state bits as follows

$$\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r}), \tag{9}$$

where $\boldsymbol{\ell}$ represents the LFSR bits, and $\boldsymbol{r}$ represents the nonlinear bits. During the keystream generation process, the LFSR bits are updated according to a linear function g as $\boldsymbol{\ell} \leftarrow g(\boldsymbol{\ell})$.

In FCA, the linear approximation with linear masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{r}}, \Gamma_{\boldsymbol{z}})$ satisfying

$$\begin{cases} \Gamma_{\boldsymbol{r}} = \mathbf{0} \\ |\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})| := \left|\mathtt{Cor}\left[(\Gamma_{\boldsymbol{\ell}}, \mathbf{0}) \xrightarrow{F} \Gamma_{\boldsymbol{z}}\right]\right| > 0 \end{cases}$$

can be utilized to recover the LFSR (or the whole internal state). Generally, the FCA is modelled as a decoding problem, i.e., the keystream segment $\boldsymbol{z}$ can be seen as the transmission result of the LFSR sequence $\boldsymbol{u}$ through a Binary Symmetry Channel (BSC) with the error probability p, as shown in Fig. 1. The FCA is divided into the preprocessing and online processing phases. In the preprocessing phase, we first collect a number of samples involving only

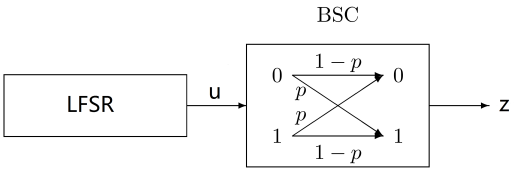

Fig. 1: Model for the fast correlation attack

output bits and LFSR initial state bits, and then try to reduce the involved LFSR initial state bits size at the expense of a folded noise level, and finally an $[N, k]$-linear code $\mathcal{C}$ is constructed. After this, we enter the processing phase to recover the target k bits of the LFSR by using the fast Walsh transfomr (FWT) as was done in [23], and further the whole LFSR bits $\boldsymbol{l}$.

Obviously, linear approximations with larger $|\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})|$ values enhance effectiveness of FCAs. Therefore, finding linear masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ yielding high correlations is requisite for the success of FCAs. However, $\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z}$ often contain many bits and the F function in Eq. (9) is quite complicated. To compute correlations, we may have to decompose F as

$$\boldsymbol{s}_0 = (\boldsymbol{\ell}, \boldsymbol{r}) \xrightarrow{p_0} \boldsymbol{s}_1 \rightarrow \cdots \rightarrow \boldsymbol{s}_{N-1} \xrightarrow{p_{N-1}} \boldsymbol{z} = \boldsymbol{s}_N.$$

where $p_0, \ldots, p_{N-1}$ are simple operations such as $\mathtt{MC}$, $\boxplus$, S-boxes etc., and $\boldsymbol{s}_i$'s $(i = 0, \ldots, N)$ are the intermediate states. Then, the masks of intermediate states compose a linear trail as follows

$$(\Gamma_{\boldsymbol{\ell}}, \mathbf{0}) \xrightarrow{p_0} \Gamma_{\boldsymbol{s}_1} \rightarrow \cdots \rightarrow \Gamma_{\boldsymbol{s}_{N-1}} \xrightarrow{p_{N-1}} \Gamma_{\boldsymbol{z}}. \tag{10}$$

According to the piling-up lemma [36], $\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ can be computed as the summation of the correlations of all linear trails as

$$\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}}) = \sum_{\forall(\Gamma_{\boldsymbol{s}_1}, \ldots, \Gamma_{\boldsymbol{s}_{N-1}})} \mathtt{Cor}\left[(\Gamma_{\boldsymbol{\ell}}, \mathbf{0}) \xrightarrow{p_0} \Gamma_{\boldsymbol{s}_1} \rightarrow \cdots \rightarrow \Gamma_{\boldsymbol{s}_{N-1}} \xrightarrow{p_{N-1}} \Gamma_{\boldsymbol{z}}\right].$$

Therefore, a highly qualified linear mask of F should meet the following criteria:

1. There exist many linear trails in Eq. (10) having non-zero correlations.
2. The absolute correlation values $\left|\mathtt{Cor}\left[(\Gamma_{\boldsymbol{\ell}}, \mathbf{0}) \xrightarrow{p_0} \Gamma_{\boldsymbol{s}_1} \rightarrow \cdots \rightarrow \Gamma_{\boldsymbol{s}_{N-1}} \xrightarrow{p_{N-1}} \Gamma_{\boldsymbol{z}}\right]\right|$'s of some linear trails are high.

Therefore, the *Candidate Search* described in Section 1 can now be redefined in a more specific manner as follows:
Candidate Search: Find many different linear trails as defined in Eq. (10) ($(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s are also different) that are likely to have high correlations.

2.4 MILP Modeling for Searching Linear Masks

The MILP modeling technique has long been used in the realm of cryptanalysis. It has good performance in fields such as finding differential/linear/integral characteristics of block ciphers, giving cube attacks on stream ciphers, and constructing all kinds of distinguishers on hash functions etc.

According to Section 2.2, the output and FSM updating functions of our targeted stream ciphers consists of basic operations of three categories namely linear, 8-bit S-box and modular addition. The corresponding linear propagation rules are studied thoroughly. There are also quite some efforts made to describe such rules with MILP models. We treat such three operations differently:

- **Linear:** Linear operations include $\oplus$, branching and all kinds of bitwise permutations. The corresponding linear propagation rules are quite straightforward and the MILP modeling techniques are also mature. For typical linear operations such as XOR and branching, the bitwise and truncated linear propagation rules along with the corresponding MILP modeling methods are detailed in Appendix B.
- **8-bit S-box:** The linear propagation rules of 8-bit S-boxes can be perfectly captured with a linear approximation table (LAT). However the MILP model describing such rules can be quite complicated: thousands of linear constraints are required making the model solving process computationally infeasible. Therefore, in our model, we simply require the bitwise input and output masks of S-boxes share the same truncated linear symbol and filter the feasible input-output pairs after the modeling solving process. Therefore, our MILP models are quite simple and can be solved efficiently. Such a S-box modeling technique is described as Algorithm 11 in Appendix B.
- **Modular Addition:** All of the current works simply consider the ordinary modular addition in Eq. (1). In our case, there are more complicated modular addition-based operation such as the consecutive modular addition such as $\boxplus_m^2$ in Eq. (2). We propose a bitwise breakdown framework in Section 3 followed by the corresponding MILP modeling technique in Section 4.1. We also propose a bytewise breakdown accordingly in Section 4.2 so as to deduce truncated linear masks.

In this way, we are able to construct MILP models for *Candidate Search*. Note that AES^R, $\mathtt{S}_1$ and $\mathtt{S}_2$ are all combinations of 8-bit S-box and linear operations. Their bitwise and truncated linear propagation rules can be described easily with MILP models. The model construction process are all defined as algorithms `aesModel`, `aesTruncModel` etc. in Appendix B as well.

2.5 Algebraic Bias Evaluation Technique for Correlation Computation

With the linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ and the function F s.t. $\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r})$ in Eq. (9), we may define the noise e as

$$e = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot \boldsymbol{z}) = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot F(\boldsymbol{\ell}, \boldsymbol{r})).$$

Computing the correlation $\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ is equivalent to evaluating the bias of e towards 0 according to the truth table of e. Nevertheless, such e is a polynomial of $\boldsymbol{\ell}$ and $\boldsymbol{r}$ bits with complicated algebraic normal form (ANF). So $\mathtt{Cor}(e)$ cannot be computed directly. Therefore, one has to group the basic operations $p_0, \ldots, p_{N-1}$ into operation groups $G_0, \ldots, G_{\kappa-1}$ whose inputs are independent as far as possible. The noises of different groups, denoted as $e_0, \ldots, e_{\kappa-1}$, should have simpler ANFs and their correlations $\mathtt{Cor}(e_i)$ $(i = 0, \ldots, \kappa - 1)$ can be accurately and efficiently computed. For groups with inputs in common, e.g., $\boldsymbol{u}_0, ..., \boldsymbol{u}_{m-1}$, we introduce the intermediate masks $\Gamma_{\boldsymbol{u}_0}, ..., \Gamma_{\boldsymbol{u}_{m-1}}$ and for each possible value of them, we compute a partial correlation over this value, which is a product of all the correlations $\mathtt{Cor}(e_i)$, namely $\prod_{i=0}^{\kappa-1} \mathtt{Cor}(e_i)$. Finally, the correlation of e can be computed as a sum of partial correlations over all intermediate masks as

$$\mathtt{Cor}(e) = \sum_{\forall \Gamma_{\boldsymbol{u}_0}, \ldots, \Gamma_{\boldsymbol{u}_{m-1}}} \prod_{i=0}^{\kappa-1} \mathtt{Cor}(e_i).$$

Apparently, dividing $p_0, \ldots, p_{N-1}$ into appropriate groups is quite technical depending highly on the definition of F in Eq. (9). The applications to SNOW family stream ciphers will be illustrated in the later Sections 6.1.2, 6.2.2 and 6.3.2.

3 Bitwise Breakdowns of Two Modular Addition-based Operations

In this part, we consider two nonlinear operations namely the ordinary modular 2^m addition in Eq. (1) and the consecutive modular 2^m addition defined in Eq. (2). We analyze the operations in a bitwise manner inherited from [26, 37], and deduce their effects on the linear correlations.

3.1 The Ordinary Modular Addition Operation

The ordinary modular addition operation in Eq. (1) can be regarded as a combination of two functions, namely the half adder $h_a : \mathbb{F}_2^2 \to \mathbb{F}_2^2$ and the full adder $f_a : \mathbb{F}_2^3 \to \mathbb{F}_2^2$.

A half adder h_a takes two bits $x, y \in \mathbb{F}_2$ as input and outputs two bits, namely the result bit z and a carry bit oc, denoted as $(z, oc) \leftarrow h_a(x, y)$ where:

$$2 * oc + z = x + y \Leftrightarrow \begin{cases} z = x \oplus y, \\ oc = xy. \end{cases}$$

A full adder f_a takes as input three bits namely $x, y, ic \in \mathbb{F}_2$ and also outputs (z, oc), denoted as $(z, oc) \leftarrow f_a(x, y, ic)$, where

$$2 * oc + z = x + y + ic \Leftrightarrow \begin{cases} z = x \oplus y \oplus ic, \\ oc = (x \cdot y) \vee (x \cdot ic) \vee (y \cdot ic). \end{cases}$$

Therefore, the ordinary modular addition in Eq. (1) is represented in a bitwise breakdown manner as Eq. (11), involving one half adder call and $w - 1$ full adder calls:

$$\begin{cases} (z_0, oc_0) \leftarrow h_a(x_0, y_0) \\ (z_1, oc_1) \leftarrow f_a(x_1, y_1, oc_0) \\ \quad \cdots \\ (z_{m-1}, oc_{m-1}) \leftarrow f_a(x_{m-1}, y_{m-1}, oc_{m-2}) \end{cases} \tag{11}$$

In order to capture the linear propagation rule of modular addition, we must handle the accurate linear propagation rules for h_a and f_a defined in Section 3.1 and Section 3.1 respectively.

For half adder h_a, we denote the two input linear mask bits as Γ_x, Γ_y and the two output mask bits as Γ_z, Γ_{oc} respectively. For all 16 possible values of the linear mask tuple $(\Gamma_x, \Gamma_y, \Gamma_z, \Gamma_{oc})$, we traverse all 2^2 (x, y) values and compute the correlation $\mathtt{Cor}$. We find that there are 10 out of 16 $(\Gamma_x, \Gamma_y, \Gamma_z, \Gamma_{oc})$ values having non-zero correlations. Such 10 available $(\Gamma_x, \Gamma_y, \Gamma_z, \Gamma_{oc})$ values are listed in Table 2 and their correlation satisfies $|\mathtt{Cor}| = 2^{-\Gamma_{oc}}$. In other words, the correlation of h_a can be determined simply with Γ_{oc}.

Table 2: h_a masks and their correlations.

No.	$(\Gamma_x, \Gamma_y, \Gamma_z, \Gamma_{oc})$	$\lvert\mathtt{Cor}\rvert$
1	(0,0,0,0)	1
2	(1,1,1,0)	1
3	(0,0,0,1)	2^{-1}
4	(1,0,0,1)	2^{-1}
5	(0,1,0,1)	2^{-1}
6	(1,1,0,1)	2^{-1}
7	(0,0,1,1)	2^{-1}
8	(1,0,1,1)	2^{-1}
9	(0,1,1,1)	2^{-1}
10	(1,1,1,1)	2^{-1}

Table 3: f_a masks and their correlations.

No.	$(\Gamma_x, \Gamma_y, \Gamma_{ic}, \Gamma_z, \Gamma_{oc})$	$\lvert\mathtt{Cor}\rvert$
1	(0,0,0,0,0)	1
2	(1,1,1,1,0)	1
3	(1,0,0,0,1)	2^{-1}
4	(0,1,0,0,1)	2^{-1}
5	(0,0,1,0,1)	2^{-1}
6	(1,1,1,0,1)	2^{-1}
7	(0,0,0,1,1)	2^{-1}
8	(1,1,0,1,1)	2^{-1}
9	(1,0,1,1,1)	2^{-1}
10	(0,1,1,1,1)	2^{-1}

For full-adder f_a, we denote the three input linear mask bits as $\Gamma_x, \Gamma_y, \Gamma_{ic}$ and the two output mask bits as Γ_z, Γ_{oc} respectively. There are 10 available

$(\Gamma_x, \Gamma_y, \Gamma_{ic}, \Gamma_z, \Gamma_{oc})$'s as shown in Table 3, and same with h_a, there is also $|\mathtt{Cor}| = 2^{-\Gamma_{oc}}$.

Therefore, for the ordinary modular addition in Eq. (1), in addition to the input-output masks $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{z}})$, we should further define $\Gamma_{\boldsymbol{oc}}$ in Eq. (12) representing the masks of carry bits.

$$\Gamma_{\boldsymbol{oc}} = (\Gamma_{oc_0}, \ldots, \Gamma_{oc_{m-1}}) \tag{12}$$

Then, we have the following Proposition 1.

Proposition 1 *For $\boxplus_m$ in Eq. (1) with input-output mask $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{z}})$, the propagation $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}) \xrightarrow{\boxplus_m} \Gamma_{\boldsymbol{z}}$ is available iff there is a linear mask of carry bits $\Gamma_{\boldsymbol{oc}} \in \mathbb{F}_2^m$ defined in Eq. (12) satisfying:*

- $\Gamma_{oc_{m-1}} = 0$.
- $(\Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{z_0}, \Gamma_{oc_0})$ *is in Table 2.*
- $(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{oc_{i-1}}, \Gamma_{z_i}, \Gamma_{oc_i})$ *is in Table 3 for* $i = 1, \ldots, m-1$.

Proof An available mask $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{z}})$ should have a non-zero correlation which, according to the piling-up lemma, can be computed as

$$\left|\mathtt{Cor}_{\boxplus_m}(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{z}})\right| = \left|\mathtt{Cor}(\Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{z_0}, \Gamma_{oc_0})\right| \cdot \prod_{i=1}^{m-1} \left|\mathtt{Cor}(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{oc_{i-1}}, \Gamma_{z_i}, \Gamma_{oc_i})\right|.$$

Since oc_{m-1} is not output, $\Gamma_{oc_{m-1}} = 0$. Further, since $\left|\mathtt{Cor}\left[(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}) \xrightarrow{\boxplus_m} \Gamma_{\boldsymbol{z}}\right]\right| > 0$, there must be

$$\begin{cases} \left|\mathtt{Cor}(\Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{z_0}, \Gamma_{oc_0})\right| > 0 \\ \left|\mathtt{Cor}(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{oc_{i-1}}, \Gamma_{z_i}, \Gamma_{oc_i})\right| > 0, \quad \text{where } i = 1, \ldots, m-1. \end{cases}$$

So there is $(\Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{z_0}, \Gamma_{oc_0})$ lying in Table 2 and $(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{oc_{i-1}}, \Gamma_{z_i}, \Gamma_{oc_i})$ in Table 3 simultaneously, which completes the proof. □

3.2 The Consecutive Modular Addition Operation

The consecutive modular addition $\boxplus_m^2$ can also be decomposed into bitwise additions from LSB to MSB: the addition at bit positions $0, \ldots, m-1$ can be one of the three function calls, namely $f_a : \mathbb{F}_2^3 \to \mathbb{F}_2^2$, $f_1 : \mathbb{F}_2^4 \to \mathbb{F}_2^3$ and $f_2 : \mathbb{F}_2^5 \to \mathbb{F}_2^3$. f_a has been defined in Section 3.1. For $f_1 : (x, y, w, ic) \to (z, oc, od)$, the input and output bits always satisfy

$$x + y + w + ic = z + 2 * oc + 4 * od.$$

There are 98 out of the 128 input-output mask $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$'s having non-zero correlations and we list them in Table C1 of Appendix C.

For $f_2 : (x, y, w, ic1, ic2) \to (z, oc1, oc2)$, the input and output bits always satisfy

$$x + y + w + ic + id = z + 2 * oc + 4 * od.$$

We list the 122 available input-output mask $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_{id}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$'s along with their correlations in Table C2 of Appendix C.

With f_a, f_1, f_2, the bitwise breakdown of the consecutive modular addition $\boxplus_m^2$ can be represented as

$$\begin{cases} (z_0, oc_0) \leftarrow f_a(x_0, y_0, w_0) \\ (z_1, oc_1, od_0) \leftarrow f_1(x_1, y_1, w_1, oc_0) \\ (z_2, oc_2, od_1) \leftarrow f_1(x_2, y_2, w_2, oc_1) \\ (z_3, oc_3, od_2) \leftarrow f_2(x_3, y_3, w_3, oc_2, od_0) \\ \cdots \\ (z_{m-1}, oc_{m-1}, od_{m-2}) \leftarrow f_2(x_{m-1}, y_{m-1}, w_{m-1}, oc_{m-1}, od_{m-3}) \end{cases}$$

As can be seen, each $\boxplus_m^2$ consists of one f_a, two f_1 and $(m-3)$ f_2 calls. The oc and od are referred as the 1st- and 2nd-order carry bits.

As to linear masks, in addition to $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{w}}, \Gamma_{\boldsymbol{z}})$, we further define carry bit mask vectors as $\Gamma_{\boldsymbol{oc}}, \Gamma_{\boldsymbol{od}}$ as

$$\begin{cases} \Gamma_{\boldsymbol{oc}} = (\Gamma_{oc_0}, \ldots, \Gamma_{oc_{m-1}}) \\ \Gamma_{\boldsymbol{od}} = (\Gamma_{od_0}, \ldots, \Gamma_{od_{m-2}}) \end{cases}$$

It can be proved that available $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{w}}, \Gamma_{\boldsymbol{z}})$'s of $\boxplus_m^2$ should satisfy the following Proposition 2.

Proposition 2 *For $\boxplus_m^2$ in Eq. (2) with input-output mask $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{w}}, \Gamma_{\boldsymbol{z}})$, the propagation $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{w}}) \xrightarrow{\boxplus_m} \Gamma_{\boldsymbol{z}}$ is available iff there are linear masks of carry bits $(\Gamma_{\boldsymbol{oc}}, \Gamma_{\boldsymbol{od}}) \in \mathbb{F}_2^m \times \mathbb{F}_2^{m-1}$ defined in Section 3.2 satisfying:*

- $\Gamma_{oc_{m-1}} = \Gamma_{od_{m-2}} = \Gamma_{od_{m-3}} = 0$.
- $(\Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{w_0}, \Gamma_{z_0}, \Gamma_{oc_0})$ *is in Table 3.*
- $(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{w_i}, \Gamma_{oc_{i-1}}, \Gamma_{z_i}, \Gamma_{oc_i}, \Gamma_{od_{i-1}})$ *is in Table C1 for* $i = 1, 2$.
- $(\Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{w_i}, \Gamma_{oc_{i-1}}, \Gamma_{od_{i-3}}, \Gamma_{z_i}, \Gamma_{oc_i}, \Gamma_{od_{i-1}})$ *is in Table C2 for* $i = 3, \ldots, m-1$.

The proof of Proposition 2 is exactly the same as that of Proposition 1.

4 MILP Models for Linear Propagation of Modular Additions

In order to find the particular linear mask with the highest correlation, a MILP model $\mathcal{M}$ is constructed where the linear mask bits are represented as binary

variables denoted as $\mathcal{M}.\texttt{var}$. The objective function of the model, denoted as $\mathcal{M}.\texttt{obj}$, is set to maximize the correlation. The constraints $\mathcal{M}.\texttt{con}$ are imposed so as to guarantee the availability of input-output linear masks. Additional variables may also be declared: such variables are to be used in constraints and the objective $\mathcal{M}.\texttt{obj}$ so as to track the magnitude of correlations.

4.1 MILP Models for Bitwise Linear Propagation of Modular Additions

According to Section 3, the correlation of h_a and f_a can be accurately determined with Γ_{oc}. Based on the bitwise breakdown in Eq. (11), we define the model construction process of the ordinary modular addition operation in Algorithm 1 as $(\mathcal{M}, \Gamma_{\boldsymbol{z}}, \Gamma_{\boldsymbol{oc}}) \leftarrow \texttt{ordModAdd}(\mathcal{M}, \Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}})$. The subroutine `haModel` and `faModel` are defined in Algorithm 15 and Algorithm 16 corresponding to the models describing h_a and f_a respectively.

Algorithm 1 Model Construction of Ordinary Modular Addition (`ordModAdd`)

Input: Initial model $\mathcal{M}$ and 2 binary variable vectors of length m: $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_{m-1}})$, $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{m-1}})$
Output: Updated model $\mathcal{M}$ and 2 binary variable vectors of length m: $\Gamma_{\boldsymbol{z}} = (\Gamma_{z_0}, \ldots, \Gamma_{z_{m-1}})$ and $\Gamma_{\boldsymbol{oc}} = (\Gamma_{oc_0}, \ldots, \Gamma_{oc_{m-1}})$
1: $(\mathcal{M}, \Gamma_{z_0}, \Gamma_{oc_0}) \leftarrow \texttt{haModel}(\mathcal{M}, \Gamma_{x_0}, \Gamma_{y_0})$
2: **for** $i = 1, \ldots, m-1$ **do**
3: $\quad (\mathcal{M}, \Gamma_{z_i}, \Gamma_{oc_i}) \leftarrow \texttt{faModel}(\mathcal{M}, \Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{oc_{i-1}})$
4: **end for**
5: $\mathcal{M}.\texttt{con} \leftarrow \Gamma_{oc_{m-1}} = 0$
6: Let $\Gamma_{\boldsymbol{z}} = (\Gamma_{z_0}, \ldots, \Gamma_{z_{m-1}})$ and $\Gamma_{\boldsymbol{oc}} = (\Gamma_{oc_0}, \ldots, \Gamma_{oc_{m-1}})$
7: Return $(\mathcal{M}, \Gamma_{\boldsymbol{z}}, \Gamma_{\boldsymbol{oc}})$

For consecutive modular addition, according to Table C1 and Table C2, the correlations of f_1 and f_2 cannot be represented with Γ_{oc} and Γ_{od}. Therefore, we define additional binary variables p, q. For each available input-output linear mask (Γ_i, Γ_o), the corresponding (p, q) takes two values namely $(\overline{p}, \overline{q})$ and $(\underline{p}, \underline{q})$ s.t.

$$2^{-(\underline{p}+2\underline{q})} \leq \texttt{Cor}(\Gamma_i \xrightarrow{f_j} \Gamma_o) \leq 2^{-(\overline{p}+2\overline{q})}, \qquad j = 1, 2.$$

We list the values of $\underline{p}$, $\underline{q}$, $\overline{p}$ and $\overline{q}$ in Table C1 and Table C2 as well. According to the bitwise breakdown in Section 3.2, we define the model construction process of the consecutive modular addition operation in Algorithm 2 as $(\mathcal{M}, \Gamma_{\boldsymbol{z}}, \boldsymbol{p}, \boldsymbol{q}) \leftarrow \texttt{conModAdd}(\mathcal{M}, \Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}}, \Gamma_{\boldsymbol{w}}, \lambda)$ where $\lambda \in \{0, 1\}$ determines whether to use $(\underline{p}, \underline{q})$ $(\lambda = 0)$ or $(\overline{p}, \overline{q})$ to evaluate the correlations. The subroutine `f1Model` and `f2Model` are defined in Algorithm 17 and Algorithm 18 in Appendix C corresponding to the models describing f_1 and f_2 respectively.

The combination of Algorithm 1 and Algorithm 2 is well enough to describe the bitwise linear propagation of SNOW family of stream ciphers. The optimal

Algorithm 2 Model Construction of Consecutive Modular Addition (`conModAdd`)

Input: Initial model $\mathcal{M}$; 3 binary variable vectors of length m: $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_{m-1}})$, $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{m-1}})$, $\Gamma_{\boldsymbol{w}} = (\Gamma_{w_0}, \ldots, \Gamma_{w_{m-1}})$; a binary flag $\lambda \in \{0, 1\}$

Output: Updated model $\mathcal{M}$ and 3 binary variable vectors: $\Gamma_{\boldsymbol{z}} = (\Gamma_{z_0}, \ldots, \Gamma_{z_{m-1}})$, $\boldsymbol{p} = (p_0, \ldots, p_{m-1})$ and $\boldsymbol{q} = (q_0, \ldots, q_{m-2})$

$(\mathcal{M}, \Gamma_{z_0}, \Gamma_{p_0}) \leftarrow$ `faModel`$(\mathcal{M}, \Gamma_{x_0}, \Gamma_{y_0}, \Gamma_{w_0})$
for $i = 1, 2$ **do**
  $(\mathcal{M}, \Gamma_{z_i}, \Gamma_{oc_i}, \Gamma_{od_{i-1}}, p_i, q_{i-1}) \leftarrow$ `f1Model`$(\mathcal{M}, \Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{w_i}, \Gamma_{oc_{i-1}}, \lambda)$
end for
for $i = 3, \ldots, m-1$ **do**
  $(\mathcal{M}, \Gamma_{z_i}, \Gamma_{oc_i}, \Gamma_{od_{i-1}}, p_i, q_{i-1}) \leftarrow$ `f2Model`$(\mathcal{M}, \Gamma_{x_i}, \Gamma_{y_i}, \Gamma_{w_i}, \Gamma_{oc_{i-1}}, \Gamma_{od_{i-3}}, \lambda)$
end for
$\mathcal{M}$.`con` $\leftarrow \Gamma_{oc_{m-1}} = 0$
$\mathcal{M}$.`con` $\leftarrow \Gamma_{od_j} = 0$ for $j = m-2, m-3$
Let $\boldsymbol{z} = (z_0, \ldots, z_{m-1})$, $\boldsymbol{p} = (p_0, \ldots, p_{m-1})$ and $\boldsymbol{q} = (q_0, \ldots, q_{m-2})$
Return $(\mathcal{M}, \Gamma_{\boldsymbol{z}}, \boldsymbol{p}, \boldsymbol{q})$

linear masks for SNOW 2.0 can be directly deduced. However, for SNOW-V/Vi and SNOW 3G, the bitwise model can be too complicated to be solved directly. Therefore, we should first deduce a good truncated linear mask as a hint. Such truncated linear mask hints are also deduced through MILP models as described in Section 4.2.

4.2 Truncated Linear Propagation of Modular Additions and Its MILP Description

In this part, we deduce the truncated linear propagation rules for ordinary modular addition and consecutive modular addition, and propose a MILP model capturing such propagation rules. We also propose a criteria of truncated linear characteristics based on their contributions to the linear correlations.

For $m = 8t$, there is a bytewise breakdown for $\boxplus_m$ as Eq. (13).

$$\begin{cases} (\boldsymbol{z}_0, oc_7) \leftarrow h_b(\boldsymbol{x}_0, \boldsymbol{y}_0) \\ (\boldsymbol{z}_1, oc_{15}) \leftarrow f_b(\boldsymbol{x}_1, \boldsymbol{y}_1, oc_7) \\ \cdots \\ (\boldsymbol{z}_{t-1}, oc_{8t+7}) \leftarrow f_b(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1}, oc_{8t-1}) \end{cases} \tag{13}$$

where $h_b : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \to \mathbb{F}_2^8 \times \mathbb{F}_2$ consists of a h_a call followed by seven f_a calls and $f_b : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2 \to \mathbb{F}_2^8 \times \mathbb{F}_2$ is eight f_a calls.

As to the truncated linear masks, in addition to $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}}, T_{\boldsymbol{z}})$, we further define the t truncated carry mask bits as:

$$T_{\boldsymbol{oc}} = (T_{oc_0}, T_{oc_1}, \ldots, T_{oc_{t-1}}) = (\Gamma_{oc_7}, \Gamma_{oc_{15}}, \ldots, \Gamma_{oc_{8t-1}}).$$

The truncated linear propagations of h_b and f_b can therefore be represented as follows:

$$(T_x, T_y) \xrightarrow{h_b} (T_z, T_{oc}), \quad (T_x, T_y, T_{ic}) \xrightarrow{f_b} (T_z, T_{oc}).$$

The available (T_x, T_y, T_z, T_{oc})'s for h_b and $(T_x, T_y, T_{ic}, T_z, T_{oc})$'s for f_b are listed in Table D3 and Table D4. The corresponding MILP model construction process of h_b and f_b are defined as Algorithm 21 and Algorithm 22. It is noticeable that `hbModel` and `fbModel` output not only the truncated linear masks but an additional $\tau \in \{0, 1\}$: $\tau = 1$ for non-zero input-output masks; otherwise $\tau = 0$. For h_b and f_b, the available input-output masks have $|\texttt{Cor}| \geq 2^{-8}$. The truncated linear propagation of the ordinary modular addition in Eq. (13) can therefore be defined as Algorithm 3. The $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$ output by Algorithm 3 can be used for evaluating the lowest correlation contribution $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}}) \to T_{\boldsymbol{z}}$ as $2^{-8\sum_{i=0}^{t-1}\tau_i}$.

Algorithm 3 Model Construction of the Ordinary Modular Addition (`ordTruncModAdd`)

Input: Initial model $\mathcal{M}$ and 2 binary variable vectors of length t: $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_{t-1}})$, $T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_{t-1}})$
Output: Updated model $\mathcal{M}$, 2 binary variable vectors of length t: $T_{\boldsymbol{z}} = (T_{z_0}, \ldots, T_{z_{t-1}})$ and $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$

$(\mathcal{M}, T_{z_0}, T_{oc_0}, \tau_0) \leftarrow \texttt{hbModel}(\mathcal{M}, T_{\boldsymbol{x}_0}, T_{\boldsymbol{y}_0})$
for $i = 1, \ldots, t-1$ **do**
    $(\mathcal{M}, T_{z_i}, T_{oc_i}, \tau_i) \leftarrow \texttt{fbModel}(\mathcal{M}, T_{\boldsymbol{x}_i}, T_{\boldsymbol{y}_i}, \tau_{i-1})$
end for
Add a constraint $\mathcal{M}.\texttt{con} \leftarrow T_{oc_{t-1}} = 0$
Let $T_{\boldsymbol{z}} = (T_{\boldsymbol{z}_0}, \ldots, T_{\boldsymbol{z}_{m-1}})$, $T_{\boldsymbol{oc}} = (T_{oc_0}, \ldots, T_{oc_{m-1}})$ and $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$
Return $(\mathcal{M}, T_{\boldsymbol{z}}, \boldsymbol{\tau})$

As to the $\boxplus_m^2$, its bytewise breakdown is of the form Eq. (14).

$$\begin{cases} (\boldsymbol{z}_0, oc_7, od_6, od_7) \leftarrow h_{b2}(\boldsymbol{x}_0, \boldsymbol{y}_0, \boldsymbol{w}_0) \\ (\boldsymbol{z}_1, oc_{15}, od_{14}, od_{15}) \leftarrow f_{b2}(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{w}_1, oc_7, od_6, od_7) \\ \cdots \\ (\boldsymbol{z}_{t-1}, oc_{8t+7}, od_{8t+6}, , od_{8t+7}) \leftarrow f_{b2}(\boldsymbol{x}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{w}_{t-1}, oc_{8t-1}, od_{8t-2}, od_{8t-1}) \end{cases} \tag{14}$$

$h_{b2} : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8 \to \mathbb{F}_2^8 \times \mathbb{F}_2^3$ in Eq. (14) consists of one f_a call, two f_1 calls and five f_2 calls. $f_{b2} : \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times \mathbb{F}_2^3 \to \mathbb{F}_2^8 \times \mathbb{F}_2^3$ consists of eight f_2 calls. The truncated input-output linear masks are $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}}, T_{\boldsymbol{w}}, T_{\boldsymbol{z}})$ and the truncated carry masks are defined as $T_{\boldsymbol{oc}}, T_{\boldsymbol{od}}$ in Section 4.2.

$$\begin{cases} T_{\boldsymbol{oc}} = (T_{oc_0}, \ldots, T_{oc_{t-1}}) = (\Gamma_{oc_7}, \ldots, \Gamma_{oc_{8t-1}}) \\ T_{\boldsymbol{od}} = (T_{od_0}, T_{od_1}, \ldots, T_{od_{2t-2}}, T_{od_{2t-1}}) = (\Gamma_{od_6}, \Gamma_{od_7}, \ldots, \Gamma_{od_{8t-2}}, \Gamma_{od_{8t-1}}). \end{cases}$$

According to Proposition 2, there is constantly $T_{oc_{t-1}} = 0$ and $T_{od_{2t-2}} = T_{od_{2t-1}} = 0$. Therefore, the truncated linear mask propagation of h_{b2} and f_{b2} can be defined as follows:

$$(T_x, T_y, T_w) \xrightarrow{h_{b2}} (T_z, T_{oc}, T_{od_0}, T_{od_1})$$
$$(T_x, T_y, T_w, T_{ic}, T_{id_0}, T_{id_1}) \xrightarrow{f_{b2}} (T_z, T_{oc}, , T_{od_0}, T_{od_1}).$$

The available $(T_x, T_y, T_w, T_z, T_{oc}, T_{od_0}, T_{od_1})$'s for h_{b2} and the available $(T_x, T_y, T_{ic}, T_{id_0}, T_{id_1}, T_z, T_{oc}, T_{od_0}, T_{od_1})$'s for f_{b2} are listed in Table D5 and Table D6. The construction of the MILP model for h_{b2} and f_{b2} are defined in Algorithm 19 and Algorithm 20 in Appendix D. The model describing the truncated linear propagation of the consecutive modular addition in Eq. (14) can therefore be constructed by calling Algorithm 4. Note that the magnitude of the correlations are evaluated by $\max\{\underline{p} + 2\underline{q}\}$ where $(\underline{p}, \underline{q})$ are defined in Table C1 and Table C2. According to their definitions, the larger $\max\{\underline{p}+2\underline{q}\}$ are, the lower the $\min |\texttt{Cor}|$ can be. It is noticeable that for h_{b2}, the non-zero $\max\{\underline{p} + 2\underline{q}\}$ values are within the range $[15, 22]$ while for f_{b2}, the range is $[17, 24]$. The average $\max\{\underline{p}+2\underline{q}\}$ values for h_{b2} and f_{b2} are therefore computed as 19 and 21 respectively. Specifically, for the f_{b2} call at the most significant byte, the non-zero $\max\{\underline{p} + 2\underline{q}\}$'s equal 20 according to Table D6. With the final $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$ output by Algorithm 4, we may evaluate the correlation of propagation $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}}, T_{\boldsymbol{w}}) \xrightarrow{\boxplus_m^2} T_{\boldsymbol{z}}$ as $2^{-(19\tau_0+20\tau_{t-1}+21\sum_{i=1}^{t-2}\tau_i)}$.

Algorithm 4 Truncated Linear Propagation Model Construction of the Consecutive Modular Addition (`conTruncModAdd`)

Input: Initial model $\mathcal{M}$ and 3 binary variable vectors of length t: $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_{t-1}})$, $T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_{t-1}})$ and $T_{\boldsymbol{w}} = (T_{\boldsymbol{w}_0}, \ldots, T_{\boldsymbol{w}_{t-1}})$
Output: Updated model $\mathcal{M}$, 2 binary variable vectors of length t: $T_{\boldsymbol{z}} = (T_{\boldsymbol{z}_0}, \ldots, T_{\boldsymbol{z}_{t-1}})$ and $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$

1: $(\mathcal{M}, T_{\boldsymbol{z}_0}, T_{oc_0}, T_{od_{0,0}}, T_{od_{1,0}}, \tau_0) \leftarrow$ `hb2Model`$(\mathcal{M}, T_{\boldsymbol{x}_0}, T_{\boldsymbol{y}_0}, T_{\boldsymbol{w}_0})$
2: **for** $i = 1, \ldots, t-1$ **do**
3: $\quad(\mathcal{M}, T_{z_i}, T_{oc_i}, T_{od_{0,i}, T_{od_{1,i}}}, \tau_i) \leftarrow$ `fb2Model`$(\mathcal{M}, T_{\boldsymbol{x}_i}, T_{\boldsymbol{y}_i}, T_{\boldsymbol{w}_i}, \tau_{i-1})$
4: **end for**
5: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow T_{oc_{t-1}} = 0$
6: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow T_{od_{0,t-1}} = 0$
7: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow T_{oc_{1,t-1}} = 0$
8: Let $T_{\boldsymbol{z}} = (T_{\boldsymbol{z}_0}, \ldots, T_{\boldsymbol{z}_{m-1}})$ and $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_{t-1})$
9: Return $(\mathcal{M}, T_{\boldsymbol{z}}, \boldsymbol{\tau})$

5 Algebraic Bias Evaluation of $\mathcal{F}$-Function

In this part, we detail the process for computing the linear approximations correlations of the $\mathcal{F}$-Function $\mathcal{F}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \sigma(\boldsymbol{x} \boxplus_{32} \boldsymbol{y}) \boxplus_{32} \boldsymbol{z}$ under any given linear masks using the algebraic bias evaluation technique.

For any 128-bit variable $\boldsymbol{X}$, we represent it in bytes as

$$\boldsymbol{X} = (\underbrace{\boldsymbol{x}_0^0 \| \boldsymbol{x}_1^0 \| \boldsymbol{x}_2^0 \| \boldsymbol{x}_3^0}_{\boldsymbol{X}_0} \| \underbrace{\boldsymbol{x}_0^1 \| \boldsymbol{x}_1^1 \| \boldsymbol{x}_2^1 \| \boldsymbol{x}_3^1}_{\boldsymbol{X}_1} \| \underbrace{\boldsymbol{x}_0^2 \| \boldsymbol{x}_1^2 \| \boldsymbol{x}_2^2 \| \boldsymbol{x}_3^2}_{\boldsymbol{X}_2} \| \underbrace{\boldsymbol{x}_0^3 \| \boldsymbol{x}_1^3 \| \boldsymbol{x}_2^3 \| \boldsymbol{x}_3^3}_{\boldsymbol{X}_3}).$$

We use temporarily the notation "$\boxplus$" to represent the operation "$\boxplus_8$", then the output of $\mathcal{F}$ can be represented as a 4×4 matrix as follows:

$$\begin{aligned}
&\mathcal{F}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \sigma(\boldsymbol{X} \boxplus_{32} \boldsymbol{Y}) \boxplus_{32} \boldsymbol{Z} \\
&= \begin{bmatrix}
(\boldsymbol{x}_0^0 \boxplus \boldsymbol{y}_0^0 \boxplus 0) \boxplus \boldsymbol{z}_0^0 \boxplus 0 & (\boldsymbol{x}_1^0 \boxplus \boldsymbol{y}_1^0 \boxplus c_0^0) \boxplus \boldsymbol{z}_0^1 \boxplus 0 & (\boldsymbol{x}_2^0 \boxplus \boldsymbol{y}_2^0 \boxplus c_1^0) \boxplus \boldsymbol{z}_0^2 \boxplus 0 & (\boldsymbol{x}_3^0 \boxplus \boldsymbol{y}_3^0 \boxplus c_2^0) \boxplus \boldsymbol{z}_0^3 \boxplus 0 \\
(\boldsymbol{x}_0^1 \boxplus \boldsymbol{y}_0^1 \boxplus 0) \boxplus \boldsymbol{z}_1^0 \boxplus d_0^0 & (\boldsymbol{x}_1^1 \boxplus \boldsymbol{y}_1^1 \boxplus c_0^1) \boxplus \boldsymbol{z}_1^1 \boxplus d_0^1 & (\boldsymbol{x}_2^1 \boxplus \boldsymbol{y}_2^1 \boxplus c_1^1) \boxplus \boldsymbol{z}_1^2 \boxplus d_0^2 & (\boldsymbol{x}_3^1 \boxplus \boldsymbol{y}_3^1 \boxplus c_2^1) \boxplus \boldsymbol{z}_1^3 \boxplus d_0^3 \\
(\boldsymbol{x}_0^2 \boxplus \boldsymbol{y}_0^2 \boxplus 0) \boxplus \boldsymbol{z}_2^0 \boxplus d_1^0 & (\boldsymbol{x}_1^2 \boxplus \boldsymbol{y}_1^2 \boxplus c_0^2) \boxplus \boldsymbol{z}_2^1 \boxplus d_1^1 & (\boldsymbol{x}_2^2 \boxplus \boldsymbol{y}_2^2 \boxplus c_1^2) \boxplus \boldsymbol{z}_2^2 \boxplus d_1^2 & (\boldsymbol{x}_3^2 \boxplus \boldsymbol{y}_3^2 \boxplus c_2^2) \boxplus \boldsymbol{z}_2^3 \boxplus d_1^3 \\
(\boldsymbol{x}_0^3 \boxplus \boldsymbol{y}_0^3 \boxplus 0) \boxplus \boldsymbol{z}_3^0 \boxplus d_2^0 & (\boldsymbol{x}_1^3 \boxplus \boldsymbol{y}_1^3 \boxplus c_0^3) \boxplus \boldsymbol{z}_3^1 \boxplus d_2^1 & (\boldsymbol{x}_2^3 \boxplus \boldsymbol{y}_2^3 \boxplus c_1^3) \boxplus \boldsymbol{z}_3^2 \boxplus d_2^2 & (\boldsymbol{x}_3^3 \boxplus \boldsymbol{y}_3^3 \boxplus c_2^3) \boxplus \boldsymbol{z}_3^3 \boxplus d_2^3
\end{bmatrix}
\end{aligned}$$

where $c_j^k, d_j^k \in \{0,1\}$ for $k = 0,1,2,3$, $j = 0,1,2,3$ are local carries introduced by the first and second $\boxplus_{32}$ formulated as

$$\begin{cases} c_{-1}^k = 0, \\ c_j^k = \lfloor (\boldsymbol{x}_j^k + \boldsymbol{y}_j^k + c_{j-1}^k)/2^8 \rfloor \\ \quad \triangleq \varphi(\cdot, c_{j-1}^k), \end{cases} \quad \begin{cases} d_{-1}^k = 0, \\ d_j^k = \left\lfloor [(\boldsymbol{x}_k^j \boxplus \boldsymbol{y}_k^j \boxplus c_{k-1}^j) + \boldsymbol{z}_j^k + d_{j-1}^k]/2^8 \right\rfloor \\ \quad \triangleq \psi(\cdot, c_{k-1}^j, d_{j-1}^k). \end{cases}$$

For any 128-bit mask tuple $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{A})$, we define

$$f^{(\boldsymbol{A},\boldsymbol{U},\boldsymbol{V},\boldsymbol{W})}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \boldsymbol{A} \cdot (\sigma(\boldsymbol{X} \boxplus_{32} \boldsymbol{Y}) \boxplus_{32} \boldsymbol{Z}) \oplus \boldsymbol{U} \cdot \boldsymbol{X} \oplus \boldsymbol{V} \cdot \boldsymbol{Y} \oplus \boldsymbol{W} \cdot \boldsymbol{Z}.$$

Then the correlation $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ can be computed for a uniformly distributed $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$ as:

$$\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A}) = \Pr\{f^{(\boldsymbol{A},\boldsymbol{U},\boldsymbol{V},\boldsymbol{W})} = 0\} - \Pr\{f^{(\boldsymbol{A},\boldsymbol{U},\boldsymbol{V},\boldsymbol{W})} = 1\}.$$

Computation of $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$. We represent the bytes in $(\boldsymbol{A}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W})$ as $(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)$ for $k, j \in \{0, \ldots, 3\}^2$ and further define the "coordinate" functions

$$\begin{aligned}
&\bar{f}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}(\boldsymbol{x}_k^j, \boldsymbol{y}_k^j, \boldsymbol{z}_j^k, c_{k-1}^j, d_{j-1}^k) \\
&= \boldsymbol{a}_j^k \cdot [(\boldsymbol{x}_k^j \boxplus \boldsymbol{y}_k^j \boxplus c_{k-1}^j) \boxplus \boldsymbol{z}_j^k \boxplus d_{j-1}^k] \oplus \boldsymbol{u}_k^j \cdot \boldsymbol{x}_k^j \oplus \boldsymbol{v}_k^j \cdot \boldsymbol{y}_k^j \oplus \boldsymbol{w}_j^k \cdot \boldsymbol{z}_j^k.
\end{aligned}$$

Then we obtain

$$f^{(\boldsymbol{A},\boldsymbol{U},\boldsymbol{V},\boldsymbol{W})}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \bigoplus_{k=0}^{3} \bigoplus_{j=0}^{3} \bar{f}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}(\boldsymbol{x}_k^j, \boldsymbol{y}_k^j, \boldsymbol{z}_j^k, c_{k-1}^j, d_{j-1}^k).$$

For each function $\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}$, we construct the 4×4 matrix $\boldsymbol{U}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}$ whose entry at position $(c_k^j|d_j^k, c_{k-1}^j|d_{j-1}^k)$[5] is defined as:

$$\begin{aligned}
&\boldsymbol{U}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k))}\left[c_k^j|d_j^k\right]\left[c_{k-1}^j|d_{j-1}^k\right]\\
&=\frac{1}{2^{24}}(|\{\boldsymbol{x}_k^j,\boldsymbol{y}_k^j,\boldsymbol{z}_j^k\in\mathbb{F}_{2^8}:\bar{f}(\cdot,c_{k-1}^j,d_{j-1}^k)=0,\,\varphi(\cdot,c_{k-1}^j)=c_k^j,\,\psi(\cdot,c_{k-1}^j,d_{j-1}^k)=d_j^k\}|\\
&\quad-|\{\boldsymbol{x}_k^j,\boldsymbol{y}_k^j,\boldsymbol{z}_j^k\in\mathbb{F}_{2^8}:\bar{f}(\cdot,c_{k-1}^j,d_{j-1}^k)=1,\,\varphi(\cdot,c_{k-1}^j)=c_k^j,\,\psi(\cdot,c_{k-1}^j,d_{j-1}^k)=d_j^k\}|),
\end{aligned}$$

where $c_{k-1}^j, d_{j-1}^k \in \{0,1\}$ are input carries of $\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}$, and $c_k^j, d_j^k \in \{0,1\}$ are output carries, which can be efficiently computed by adapting the bit-slicing technique proposed in [37], as shown in Corollary 1.

Algorithm 5 Construction of the matrices $\mathbf{u}^{(a,u,v,w)}$

Output: All the matrices $\mathbf{u}^{(a,u,v,w)}$ for $a,u,v,w\in\mathbb{F}_2$
1: Prepare a 4×4 matrix $\mathbf{N}$
2: **for** $a,u,v,w\in\mathbb{F}_2$ **do**
3: Create a matrix $\mathbf{u}^{(a,u,v,w)}$ of size 4×4
4: Initialize $\mathbf{N}$ with zeros
5: **for** $ic^0\in\{0,1\}$, $ic^1\in\{0,1\}$, $x,y,z\in\mathbb{F}_2$ **do**
6: Compute $r=a\cdot[(x\oplus y\oplus ic^0)\oplus z\oplus ic^1]\oplus u\cdot x\oplus v\cdot y\oplus w\cdot z$
7: Compute $oc^0=\left\lfloor(x+y+ic^0)/2\right\rfloor$
8: Compute $oc^1=\left\lfloor((x\oplus y\oplus ic^0)+z+ic^1)/2\right\rfloor$
9: **if** $r=0$ **then**
10: $\mathbf{N}[oc^0|oc^1][ic^0|ic^1]:=\mathbf{N}[oc^0|oc^1][ic^0|ic^1]+1$
11: **else if** $r=1$ **then**
12: $\mathbf{N}[oc^0|oc^1][ic^0|ic^1]:=\mathbf{N}[oc^0|oc^1][ic^0|ic^1]-1$
13: **end if**
14: **end for**
15: **for** $ic^0,ic^1\in\{0,1\}$, $oc^0,oc^1\in\{0,1\}$ **do**
16: $\mathbf{u}^{(a,u,v,w)}[oc^0|oc^1][ic^0|ic^1]:=\mathbf{N}[oc^0|oc^1][ic^0|ic^1]/2^3$
17: **end for**
18: **end for**

Corollary 1. *For any given 8-bit mask* $(\boldsymbol{a},\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})$*, we write them in bits, and let* $\mathbf{u}^{(a_j,u_j,v_j,w_j)}$ *be the corresponding* 4×4 *matrix pre-computed by Algorithm 5. Then the matrix* $\boldsymbol{U}^{(\boldsymbol{a},\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})}$ *can be computed as:*

$$\boldsymbol{U}^{(\boldsymbol{a},\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})}=\prod_{j=7}^{0}\mathbf{u}^{(a_j,u_j,v_j,w_j)}. \tag{15}$$

[5] We use the notation $c|d$ to represent the integer value $2c+d$, i.e., $c|d=2c+d$.

For convenience, we fill these carries into 8 sets as follows

$$Sc_0 = \{c_0^0, c_0^1, c_0^2, c_0^3\},\ Sd_0 = \{d_0^0, d_1^0, d_2^0, d_3^0\},\ Sc_1 = \{c_1^0, c_1^1, c_1^2, c_1^3\},\ Sd_1 = \{d_0^1, d_1^1, d_2^1, d_3^1\},$$
$$Sc_2 = \{c_2^0, c_2^1, c_2^2, c_2^3\},\ Sd_2 = \{d_0^2, d_1^2, d_2^2, d_3^2\},\ Sc_3 = \{c_3^0, c_3^1, c_3^2, c_3^3\},\ Sd_3 = \{d_0^3, d_1^3, d_2^3, d_3^3\}.$$

Based on the matrices $\boldsymbol{U}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}$, we give the definitions of functions ρ_0, ρ_1, ρ_2 and ρ_3 as

$$\rho_0(Sc_0, Sd_0) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^0, \boldsymbol{u}_0^j, \boldsymbol{v}_0^j, \boldsymbol{w}_j^0)}[c_0^j | d_j^0][0 | d_{j-1}^0],$$
$$\rho_1(Sc_0, Sc_1, Sd_1) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^1, \boldsymbol{u}_1^j, \boldsymbol{v}_1^j, \boldsymbol{w}_j^1)}[c_1^j | d_j^1][c_0^j | d_{j-1}^1],$$
$$\rho_2(Sc_1, Sc_2, Sd_2) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^2, \boldsymbol{u}_2^j, \boldsymbol{v}_2^j, \boldsymbol{w}_j^2)}[c_2^j | d_j^2][c_1^j | d_{j-1}^2],$$
$$\rho_3(Sc_2, Sc_3, Sd_3) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^3, \boldsymbol{u}_3^j, \boldsymbol{v}_3^j, \boldsymbol{w}_j^3)}[c_3^j | d_j^3][c_2^j | d_{j-1}^3],$$

and functions A, B, C and D as

$$A(Sc_0) = \sum_{Sd_0} \rho_0(Sc_0, Sd_0), \qquad B(Sc_1) = \sum_{Sd_1} \sum_{Sc_0} A(Sc_0) \cdot \rho_1(Sc_0, Sc_1, Sd_1),$$
$$C(Sc_2) = \sum_{Sd_2} \sum_{Sc_1} B(Sc_1) \cdot \rho_2(Sc_1, Sc_2, Sd_2),\ D(Sc_3) = \sum_{Sd_3} \sum_{Sc_2} C(Sc_2) \cdot \rho_3(Sc_2, Sc_3, Sd_3).$$

Algorithm 6 Computation of $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$

Input: the matrices $\boldsymbol{U}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}$ for $k = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$; the matrices $\boldsymbol{U}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}$ for $k = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$; 8 sets storing local carries, i.e., Sc_0, Sc_1,Sc_2,Sc_3,Sd_0, Sd_1,Sd_2,Sd_3; the functions ρ_0, ρ_1, ρ_2 and ρ_3; the functions A, B, C and D

Output: the accurate value of $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$

1: **Step 1**: Compute $A(Sc_0) = \sum_{Sd_0} \rho_0(Sc_0, Sd_0)$
2: **Step 2**: Compute $B(Sc_1) = \sum_{Sd_1} \sum_{Sc_0} A(Sc_0) \cdot \rho_1(Sc_0, Sc_1, Sd_1)$
3: **Step 3**: Compute $C(Sc_2) = \sum_{Sd_2} \sum_{Sc_1} B(Sc_1) \cdot \rho_2(Sc_1, Sc_2, Sd_2)$
4: **Step 4**: Compute $D(Sc_3) = \sum_{Sd_3} \sum_{Sc_2} C(Sc_2) \cdot \rho_3(Sc_2, Sc_3, Sd_3)$
5: **Step 5**: Compute $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A}) = \sum_{Sc_3} D(Sc_3)$

Then, we give Theorem 3 whose proof is in Appendix F.

Theorem 3 *For any given masks $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{A})$, the correlation of the bitwise linear mask of $\mathcal{F}$ is computed as $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A}) = \sum_{Sc_3} D(Sc_3)$.*

According to Theorem 3, for any given mask tuple $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{A})$, the correlation value $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ can be accurately computed by using the matrices $\boldsymbol{U}^{(\boldsymbol{a}_j^k, \boldsymbol{u}_k^j, \boldsymbol{v}_k^j, \boldsymbol{w}_j^k)}$, $k = 0, \ldots, 3$ and $j = 0, \ldots, 3$. We present Algorithm 6 as a high-level description of the computation, while the detailed process for carrying out **Step 1-5** is in Appendix G. For each mask tuple $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{A})$, the time complexity for Algorithm 6 is 2^{11} basic $+$ or $*$ operations: simpler than the method in [35] requiring 2^{11} (4×4)-matrix multiplications.

To sum up, for a given mask $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{A})$, the correlation $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ can be acquired by taking the following two phases:

- **Preprocessing:** pre-compute the matrices $\boldsymbol{U}^{(\boldsymbol{a}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})}$ for all 2^{32} possible values of $\boldsymbol{a}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}_{2^8}$ according to Eq. (15), which requires a total time and memory complexities of $\mathcal{O}(2^{32})$ and $\mathcal{O}(2^{36})$ for all $(\boldsymbol{a}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$.
- **Processing:** compute $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ following Step 1 to Step 5 in Appendix G, whose cost is 2^{11} basic $+$ or $*$ operations for each given mask tuple.

6 Linear Mask Search for SNOW-V/Vi, SNOW 3G and SNOW 2.0

With the aforementioned preparations, *Candidate Search* of Section 2.3 can be accomplished with MILP models. Such models are constructed with the techniques mentioned in Section 2.4. Besides, *Correlation Computation* can be accomplished with the classic algebraic bias evaluation technique as shown in Section 2.5, where each $\mathtt{Cor}(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ should be computed accurately and efficiently. In the following sections, we detail the general *Candidate Search* and *Correlation Computation* processes for SNOW-V/Vi, SNOW 3G and SNOW 2.0 respectively.

6.1 Linear Mask Search for SNOW-V/Vi

According to Section 2.3, we need to first define $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z})$ along with the function F in form of Eq. (9). The FSM part of SNOW-V/Vi has three 128-bit registers. To cancel out the contributions of the non-linear variables, we consider the 3-round linear approximations of the FSM, as depicted in Fig. 2. According to the keystream generation function and the FSM updating function in Eq. (3) and Eq. (4), we define $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z})$ as: $\boldsymbol{z} = (\boldsymbol{z}_{t-1}, \boldsymbol{z}_t, \boldsymbol{z}_{t+1})$, $\boldsymbol{\ell} = (\boldsymbol{T1}_{t-1}, \boldsymbol{T1}_t, \boldsymbol{T1}_{t+1}, \boldsymbol{T2}_t)$ and $\boldsymbol{r} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) = (\boldsymbol{R1}_{t-1}, \boldsymbol{R2}_{t-1}, \boldsymbol{R1}_t)$. Then from Fig. 2 we have $\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r})$ with the F function defined as follows

$$\begin{cases} \boldsymbol{z}_{t-1} = (\boldsymbol{T1}_{t-1} \boxplus_{32} \boldsymbol{u}) \oplus \boldsymbol{v}, \\ \boldsymbol{z}_t = (\boldsymbol{T1}_t \boxplus_{32} \boldsymbol{w}) \oplus AES^R(\boldsymbol{u}), \\ \boldsymbol{z}_{t+1} = (\boldsymbol{T1}_{t+1} \boxplus_{32} \sigma((\boldsymbol{T2}_t \oplus AES^R(\boldsymbol{v})) \boxplus_{32} AES^R(\boldsymbol{u}))) \oplus AES^R(\boldsymbol{w}). \end{cases} \tag{16}$$

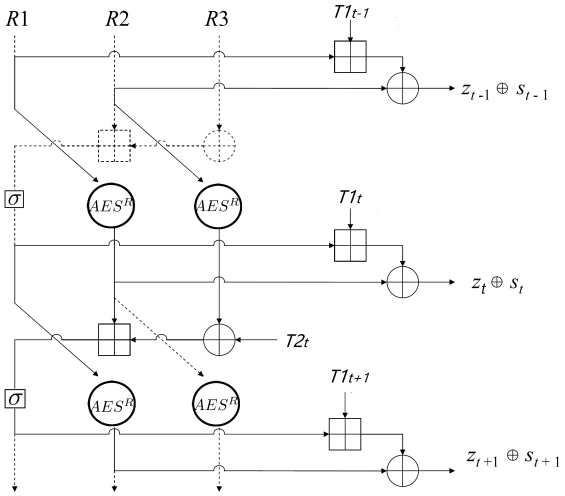

Fig. 2: The three-round linear approximations for the FSM of SNOW-V/Vi

For the linear variables $(\boldsymbol{\ell}, \boldsymbol{z})$, we define the targeted linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ as

$$\begin{cases} \Gamma_{\boldsymbol{\ell}} = (\Gamma_{\boldsymbol{T1}_0}, \Gamma_{\boldsymbol{T1}_1}, \Gamma_{\boldsymbol{T1}_2}, \Gamma_{\boldsymbol{T2}}) = (\Gamma_{\boldsymbol{T1}_{t-1}}, \Gamma_{\boldsymbol{T1}_t}, \Gamma_{\boldsymbol{T1}_{t+1}}, \Gamma_{\boldsymbol{T2}_t}) \\ \Gamma_{\boldsymbol{z}} = (\Gamma_{\boldsymbol{z}_0}, \Gamma_{\boldsymbol{z}_1}, \Gamma_{\boldsymbol{z}_2}) = (\Gamma_{\boldsymbol{z}_{t-1}}, \Gamma_{\boldsymbol{z}_t}, \Gamma_{\boldsymbol{z}_{t+1}}) \end{cases} \tag{17}$$

Then the noise e is $e = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot \boldsymbol{z}) = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot F(\boldsymbol{\ell}, \boldsymbol{r}))$, and the correlation $\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ is equal to that of e towards 0, i.e., $\mathtt{Cor}_F(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}}) = \Pr\{e = 0\} - \Pr\{e = 1\}$. Our target is to find linear masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ yielding high correlations.

The whole linear mask search is carried out following the *Candidate Search* and *Correlation Computation* processes as follows.

6.1.1 Candidate Search for SNOW-V/Vi

The definitions of $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z})$ and F guarantee that linear trails of the form Eq. (10) exist. To find particular $(\Gamma_{\boldsymbol{s}_0}, \Gamma_{\boldsymbol{s}_1}, \ldots, \Gamma_{s_N})$'s in Eq. (10), a MILP model $\mathcal{M}$ is constructed where all mask bits are represented with binary variables $\mathcal{M}.\mathtt{var}$ satisfying particular constraints $\mathcal{M}.\mathtt{con}$. The objective $\mathcal{M}.\mathtt{obj}$ should also be set properly so that the solution of $\mathcal{M}$ corresponds to the trails with high correlations. Finally, the MILP model $\mathcal{M}$ is solved and the solutions corresponding to highly qualified linear trails are output. According to Section 2.4, for each solution, the input-output masks of S-boxes should be checked with the LAT: if the masks are infeasible, the solution is aborted.

The bitwise model for SNOW-V/Vi is too complicated to be solved. Therefore, a truncated MILP model $\mathcal{M}_T$ is constructed so as to find the optimal $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$. Then, the bitwise model $\mathcal{M}$ for searching bitwise masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ satisfying the truncated masks $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$ is constructed and solved. Truncated and

bitwise linear masks of intermediate states satisfy Eq. (18): the constraints in $\mathcal{M}_T$ are listed on the left while those for $\mathcal{M}$ are on the right.

$$\begin{cases} T_{\boldsymbol{u}_0} = T_{\boldsymbol{u}_1}, T_{\boldsymbol{v}_0} = T_{\boldsymbol{v}_1}, T_{\boldsymbol{w}_0} = T_{\boldsymbol{w}_1} \\ T_{\boldsymbol{z}_{t-1}} = T_{\boldsymbol{v}_0} \\ (T_{\boldsymbol{T1}_{t-1}}, T_{\boldsymbol{u}_0}) \xrightarrow{\boxplus_{32}} T_{\boldsymbol{z}_{t-1}} \\ T_{\boldsymbol{u}_1} \xrightarrow{AES^R} T_{\boldsymbol{a}} \\ T_{\boldsymbol{a}} \xrightarrow{\texttt{branch}} (T_{\boldsymbol{a}_0}, T_{\boldsymbol{a}_1}) \\ (T_{\boldsymbol{T1}_t}, T_{\boldsymbol{w}_0}) \xrightarrow{\boxplus_{32}} T_{\boldsymbol{z}_t} \\ T_{\boldsymbol{a}_0} = T_{\boldsymbol{z}_t} \\ T_{\boldsymbol{v}_1} \xrightarrow{AES^R} T_{\boldsymbol{T2}_t} \\ (T_{\boldsymbol{T2}_t}, T_{\boldsymbol{a}_1}) \xrightarrow{\boxplus_{32}} T_\kappa \xrightarrow{\sigma} T_\eta \\ (T_{\boldsymbol{T1}_{t+1}}, T_\eta) \xrightarrow{\boxplus_{32}} T_{\boldsymbol{z}_{t+1}} \\ T_{\boldsymbol{w}_1} \xrightarrow{AES^R} T_{\boldsymbol{z}_{t+1}} \end{cases} \begin{cases} \Gamma_{\boldsymbol{u}_0} = \Gamma_{\boldsymbol{u}_1}, \Gamma_{\boldsymbol{v}_0} = \Gamma_{\boldsymbol{v}_1}, \Gamma_{\boldsymbol{w}_0} = \Gamma_{\boldsymbol{w}_1} \\ \Gamma_{\boldsymbol{z}_{t-1}} = \Gamma_{\boldsymbol{v}_0} \\ (\Gamma_{\boldsymbol{T1}_{t-1}}, \Gamma_{\boldsymbol{u}_0}) \xrightarrow{\boxplus_{32}} \Gamma_{\boldsymbol{z}_{t-1}} \\ \Gamma_{\boldsymbol{u}_1} \xrightarrow{AES^R} \Gamma_{\boldsymbol{a}} \\ \Gamma_{\boldsymbol{a}} \xrightarrow{\texttt{branch}} (\Gamma_{\boldsymbol{a}_0}, \Gamma_{\boldsymbol{a}_1}) \\ (\Gamma_{\boldsymbol{T1}_t}, \Gamma_{\boldsymbol{w}_0}) \xrightarrow{\boxplus_{32}} \Gamma_{\boldsymbol{z}_t} \\ \Gamma_{\boldsymbol{a}_0} = \Gamma_{\boldsymbol{z}_t} \\ \Gamma_{\boldsymbol{v}_1} \xrightarrow{AES^R} \Gamma_{\boldsymbol{T2}_t} \\ (\Gamma_{\boldsymbol{T2}_t}, \Gamma_{\boldsymbol{a}_1}) \xrightarrow{\boxplus_{32}} \Gamma_\kappa \xrightarrow{\sigma} \Gamma_\eta \\ (\Gamma_{\boldsymbol{T1}_{t+1}}, \Gamma_\eta) \xrightarrow{\boxplus_{32}} \Gamma_{\boldsymbol{z}_{t+1}} \\ \Gamma_{\boldsymbol{w}_1} \xrightarrow{AES^R} \Gamma_{\boldsymbol{z}_{t+1}} \end{cases} \tag{18}$$

Define Objective Functions for MILP Models. Empirically, the fewer active S-boxes the higher correlations can be. Therefore, our definition of objective functions aims at minimizing the number of active S-boxes. Such a "fewest active S-box preferred" strategy is applied to the objective function definitions through this paper. Since both bitwise and truncated linear propagation rules for $\boxplus$ and AES^R are captured by the MILP models, $\mathcal{M}_T$ and $\mathcal{M}$ can be constructed directly. In order to identify the $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$ and $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s with high correlations, the objective functions of $\mathcal{M}_T$ and $\mathcal{M}$ should be defined properly. For $\mathcal{M}_T$, each $\boxplus$ model construction call (Algorithm 3) returns a $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_3)$ vector. The number of active S-boxes equals to $|T_{\boldsymbol{u}_1}| + |T_{\boldsymbol{v}_1}| + |T_{\boldsymbol{w}_1}|$. Each active S-box should have a non-zero $|\texttt{Cor}| \geq 2^{-6}$. According to Section 4.2, we can define $t_{obj} = 8\sum_{\forall \boxplus}\sum_{i=0}^{3}\tau_i + 6(|T_{\boldsymbol{u}_1}| + |T_{\boldsymbol{v}_1}| + |T_{\boldsymbol{w}_1}|)$ and set $\mathcal{M}_T.\texttt{obj} \leftarrow \min t_{obj}$ for searching optimal $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$'s. As to bitwise model $\mathcal{M}$, for each $\boxplus$, the output of `ordModAdd` in Algorithm 1 returns a $\Gamma_{\boldsymbol{oc}}$. According to Section 4.1, its contribution to the correlations can be evaluated as $2^{-|\Gamma_{\boldsymbol{oc}}|}$. Therefore, for bitwise model $\mathcal{M}$, we can define the function $b_{obj} = \sum_{\forall \boxplus} |\Gamma_{\boldsymbol{oc}}|$ as and the objective of $\mathcal{M}$ as $\mathcal{M}.\texttt{obj} \leftarrow \min b_{obj}$.

Note that b_{obj} for SNOW-V/Vi do not consider the effect of S-boxes because the number of active S-boxes $|T_{\boldsymbol{u}_1}| + |T_{\boldsymbol{v}_1}| + |T_{\boldsymbol{w}_1}|$ value has already been determined with the solution of $\mathcal{M}_T$.

Framework for Candidate Search. Following the analysis above, *Candidate Search* for SNOW-V/Vi can be accomplished following the framework in Algorithm 7: for an arbitrary positive integer N, we are able to find a set $\mathcal{B}$ containing N targeted linear masks $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s having low b_{obj} values. For SNOW-V/Vi, the optimal $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$ we found is of the form Eq. (19) which is

in accordance with that used in [34, 35]

$$\begin{cases} T_{\boldsymbol{T1}_0} = T_{\boldsymbol{T1}_1} = T_{\boldsymbol{z}_0} = T_{\boldsymbol{z}_1} = \texttt{0x1000}, \\ T_{\boldsymbol{T1}_2} = T_{\boldsymbol{T2}} = T_{\boldsymbol{z}_2} = \texttt{0xf000}. \end{cases} \tag{19}$$

In fact, by solving $\mathcal{M}_T$ for SNOW-V/Vi we may find many $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$'s sharing the same optimal t_{obj} value, but the b_{obj}'s of the corresponding $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ have significant differences: $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ satisfying Eq. (19) has a $\min b_{obj} = 37$ while that satisfying Eq. (20) has $\min b_{obj} = 42$.

$$\begin{cases} T_{\boldsymbol{T1}_0} = T_{\boldsymbol{z}_0} = \texttt{0x40} \\ T_{\boldsymbol{T1}_1} = T_{\boldsymbol{z}_1} = \texttt{0x1000} \\ T_{\boldsymbol{T1}_2} = T_{\boldsymbol{z}_2} = \texttt{0xf000}, T_{\boldsymbol{T2}} = \texttt{0xe000} \end{cases} \tag{20}$$

For other $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$'s, $\min b_{obj}$ values are even higher. Thus, the best truncated mask in Eq. (19) is determined easily. Therefore, *Candidate Search* for SNOW-V/Vi is accomplished by calling Algorithm 7 with large N settings: after hours' running, all 34463 $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s satisfying $b_{obj} \leq 41$ are found and there are also millions with $b_{obj} \geq 42$. More details and tricks for accelerating Algorithm 7 of such can be found in Appendix E.

Algorithm 7 Find N optimum linear masks (`task1Frame`)

Input: Integer N as the tageted number of linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s
Output: A set $\mathcal{B}$ containing $\leq N$ linear masks

1: Define an empty set $\mathcal{B} = \phi$
2: Construct MILP model $\mathcal{M}_T$ and set $\mathcal{M}_T.\texttt{obj} \leftarrow \min t_{obj}$
3: Solve $\mathcal{M}_T$ and acquire truncated linear mask $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$ with lowest t_{obj} value
4: Construct MILP model $\mathcal{M}$ and set $\mathcal{M}.\texttt{obj} \leftarrow \min b_{obj}$
5: Add constraints to $\mathcal{M}$ s.t. $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ follows $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$
6: Solve model $\mathcal{M}$ with MILP model solver
7: **while** an optimum solution of $\mathcal{M}$ is found **do**
8: Define $\eta = 1$
9: **for** all S-boxes **do**
10: Identify the input-output mask (Γ_i, Γ_o)
11: Find the correlation $|\texttt{Cor}(\Gamma_i, \Gamma_o)|$ by refering to the S-box LAT
12: If $|\texttt{Cor}(\Gamma_i, \Gamma_0)| = 0$, set $\eta = 0$
13: **end for**
14: If $\eta = 1$, extract $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ and add it to $\mathcal{B}$
15: If $|\mathcal{B}| \geq N$, **break**
16: **end while**
17: Return $\mathcal{B}$

6.1.2 Correlation Computation for SNOW-V/Vi

According to the function $\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r})$ of SNOW-V/Vi in Eq. (16), the noise $e = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot F(\boldsymbol{\ell}, \boldsymbol{r}))$ is a polynomial of $\boldsymbol{\ell}, \boldsymbol{z}$ and $\boldsymbol{r}$ bits with complicated ANF. To cancel out the non-linear contributions from $\boldsymbol{r}$, we decompose the

whole expression of e wisely and analyze the algebraic property of each noise in detail.

For convenience, we define some 128-bit intermediate states as follows: $\boldsymbol{U} = \mathtt{SB}(\boldsymbol{u})$, $\boldsymbol{V} = \mathtt{SB}(\boldsymbol{v})$, $\boldsymbol{W} = \mathtt{SB}(\boldsymbol{w})$, $\boldsymbol{p}_t = \boldsymbol{T2}_t \oplus AES^R(\boldsymbol{v})$ and $\boldsymbol{q}_t = AES^R(\boldsymbol{u})$. By introducing a 128-bit intermediate mask Θ, we decompose the noise e into four sub-noises e_1, e_2, e_3, e_4 s.t. $e = e_1 \oplus e_2 \oplus e_3 \oplus e_4$ as follows:

$$\begin{cases} e_1 = \Gamma_{\boldsymbol{z}_0} \cdot (\boldsymbol{T1}_{t-1} \boxplus_{32} \mathtt{SB}^{-1}(\boldsymbol{U})) \oplus \Gamma_{\boldsymbol{T1}_0} \cdot \boldsymbol{T1}_{t-1} \oplus \Theta' \cdot \boldsymbol{U}, \\ e_2 = \Gamma_{\boldsymbol{z}_1} \cdot (\boldsymbol{T1}_t \boxplus_{32} \mathtt{SB}^{-1}(\boldsymbol{W})) \oplus \Gamma_{\boldsymbol{T1}_1} \cdot \boldsymbol{T1}_t \oplus \Gamma'_{\boldsymbol{z}_2} \cdot \boldsymbol{W}, \\ e_3 = \Gamma_{\boldsymbol{z}_2} \cdot (\boldsymbol{T1}_{t+1} \boxplus_{32} \sigma(\boldsymbol{p}_t \boxplus_{32} \boldsymbol{q}_t)) \oplus \Gamma_{\boldsymbol{T1}_2} \cdot \boldsymbol{T1}_{t+1} \oplus \Gamma_{\boldsymbol{T2}} \cdot \boldsymbol{p}_t \oplus (\Theta \oplus \Gamma_{\boldsymbol{z}_1}) \cdot \boldsymbol{q}_t, \\ e_4 = \Gamma_{\boldsymbol{z}_0} \cdot \mathtt{SB}^{-1}(\boldsymbol{V}) \oplus \Gamma'_{\boldsymbol{T2}} \cdot \boldsymbol{V}. \end{cases}$$

where the linear masks Θ', $\Gamma'_{\boldsymbol{z}_2}$ and $\Gamma'_{\boldsymbol{T2}}$ are defined s.t. $\Theta' \cdot \boldsymbol{U} = \Theta \cdot \mathtt{MC}(\mathtt{SR}(\boldsymbol{U}))$ for all $\boldsymbol{U}$, $\Gamma'_{\boldsymbol{z}_2} \cdot \boldsymbol{W} = \Gamma_{\boldsymbol{z}_2} \cdot \mathtt{MC}(\mathtt{SR}(\boldsymbol{W}))$ for all $\boldsymbol{W}$, and $\Gamma'_{\boldsymbol{T2}} \cdot \boldsymbol{V} = \Gamma_{\boldsymbol{T2}} \cdot \mathtt{MC}(\mathtt{SR}(\boldsymbol{V}))$ for all $\boldsymbol{V}$.

From the above expressions, we define the basic operation groups $\mathcal{G}(\boldsymbol{X}, \boldsymbol{Y}) = \boldsymbol{X} \boxplus_{32} \mathtt{SB}^{-1}(\boldsymbol{Y})$, $\mathcal{S}(\boldsymbol{X}) = \mathtt{SB}^{-1}(\boldsymbol{X})$, and $\mathcal{F}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \sigma(\boldsymbol{X} \boxplus_{32} \boldsymbol{Y}) \boxplus_{32} \boldsymbol{Z}$, then the correlations of sub-noises are simply

$$\mathtt{Cor}(e_1) = \mathtt{Cor}((\Gamma_{\boldsymbol{T1}_0}, \Theta') \xrightarrow{\mathcal{G}} \Gamma_{\boldsymbol{z}_0}),\ \mathtt{Cor}(e_3) = \mathtt{Cor}((\Gamma_{\boldsymbol{T2}}, \Theta \oplus \Gamma_{\boldsymbol{z}_1}, \Gamma_{\boldsymbol{T1}_2}) \xrightarrow{\mathcal{F}} \Gamma_{\boldsymbol{z}_2}),$$
$$\mathtt{Cor}(e_2) = \mathtt{Cor}((\Gamma_{\boldsymbol{T1}_1}, \Gamma'_{\boldsymbol{z}_2}) \xrightarrow{\mathcal{G}} \Gamma_{\boldsymbol{z}_1}), \mathtt{Cor}(e_4) = \mathtt{Cor}(\Gamma'_{\boldsymbol{T2}} \xrightarrow{\mathtt{SB}^{-1}} \Gamma_{\boldsymbol{z}_0}).$$

The correlations of $\mathcal{G}$ and $\mathcal{S}$ can be computed accurately with the constant-time complexity algorithms in [31]. For computing the correlations of the $\mathcal{F}$-function, we have detailed its algebraic bias evaluation in Section 5. By applying the results about correlations over composition functions in [38] and the piling-up lemma, the correlation of the linear approximation for the FSM of SNOW-V/Vi can be computed as $\mathtt{Cor}(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}}) = \mathtt{Cor}(e_2)\mathtt{Cor}(e_4) \sum_{\Theta} \mathtt{Cor}(e_1)\mathtt{Cor}(e_3)$.

The Selective Linear Masks. With the algebraic bias evaluation above, we are able to compute the accurate $\mathtt{Cor}(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ for all linear mask candidates given in Section 6.1 and identify the ones most suitable for FCAs. For the optimal truncated mask pattern of the form Eq. (19), from those with $b_{obj} \leq 41$, we find 8 candidate $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s having correlation $|\mathtt{Cor}| > 2^{-48}$, as shown in Table 4[6], and another 127 within the range $2^{-49} \leq |\mathtt{Cor}| < 2^{-48}$, which are in accordance with that in [35]. We tried millions of other candidates with $b_{obj} \geq 42$ and cannot find better correlations.

We also evaluate the candidates with truncated pattern in Eq. (20) and find that the absolute correlations are higher than $2^{-50.816}$ and the best mask

[6] We use $\mathsf{msw}(\cdot)$ to denote the most significant 32-bit word (MSW) of a 128-bit mask.

Table 4: The best masks satisfying Eq. (19) and $\mathsf{msw}(\Gamma_{\boldsymbol{z}_2}) = \mathsf{msw}(\Gamma_{\boldsymbol{T2}}) =$ 0x81ec5a80 such that $|\mathtt{Cor}| > 2^{-48}$.

$\mathsf{msw}(\Gamma_{\boldsymbol{z}_0})$	$\mathsf{msw}(\Gamma_{\boldsymbol{z}_1})$	$\mathsf{msw}(\Gamma_{\boldsymbol{T1}_0})$	$\mathsf{msw}(\Gamma_{\boldsymbol{T1}_1})$	$\mathsf{msw}(\Gamma_{\boldsymbol{T1}_2})$	$\log\|\mathtt{Cor}\|$	Ref.
0x0000000c	0x00000040	0x00000008	0x00000040	0x81ec5a00	-47.567	[35]
0x0000000c	0x00000080	0x00000008	0x00000080	0x81ec5a00	-47.579	[35]
0x00000020	0x00000040	0x00000030	0x00000040	0x81ec5a00	-47.660	[35]
0x00000020	0x00000080	0x00000030	0x00000080	0x81ec5a00	-47.672	[35]
0x0000000d	0x00000040	0x0000000d	0x00000040	0x81ec5a00	-47.760	[34, 35]
0x0000000d	0x00000080	0x0000000d	0x00000080	0x81ec5a00	-47.772	[35]
0x00000078	0x00000040	0x00000078	0x00000040	0x81ec5a00	-47.839	[35]
0x00000078	0x00000080	0x00000078	0x00000080	0x81ec5a00	-47.851	[35]

is as follows:

$$\Gamma_{\boldsymbol{z}_0} = \Gamma_{\boldsymbol{T1}_0} = (\mathbf{0} \| \mathtt{0x00300000} \| \mathbf{0} \| \mathbf{0}), \qquad \Gamma_{\boldsymbol{z}_1} = (\mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathtt{0x00000008}),$$
$$\Gamma_{\boldsymbol{z}_2} = (\mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathtt{0x81ec5a80}), \qquad \Gamma_{\boldsymbol{T1}_1} = (\mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathtt{0x0000000c}),$$
$$\Gamma_{\boldsymbol{T1}_2} = (\mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathtt{0x81ec5a00}), \qquad \Gamma_{\boldsymbol{T2}} = (\mathbf{0} \| \mathbf{0} \| \mathbf{0} \| \mathtt{0xc17a8fa8}).$$

This indicates that the masks in Table 4 are highly likely to be optimal.

6.2 Linear Mask Search for SNOW 3G

6.2.1 Candidate Search for SNOW 3G

Similarly, the FSM part of SNOW 3G has 32-bit three registers, thus 3-round linear approximations of the FSM are considered, as depicted in Fig.3. According to the keystream generation function and the FSM

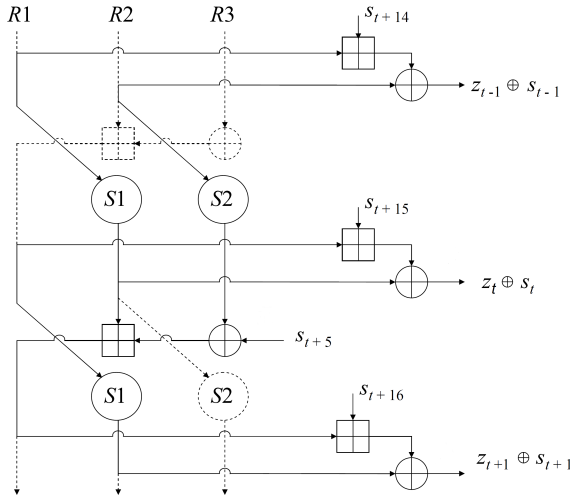

Fig. 3: The three-round linear approximations for the FSM of SNOW 3G

updating function in Eq. (5) and Eq. (6), we define $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z})$ as: $\boldsymbol{z} =$

$(\boldsymbol{z}_{t-1}, \boldsymbol{z}_t, \boldsymbol{z}_{t+1})$, $\boldsymbol{\ell} = (\boldsymbol{s}_{t-1}, \boldsymbol{s}_t, \boldsymbol{s}_{t+1}, \boldsymbol{s}_{t+5}, \boldsymbol{s}_{t+14}, \boldsymbol{s}_{t+15}, \boldsymbol{s}_{t+16})$ and $\boldsymbol{r} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) = (\boldsymbol{R1}_{t-1}, \boldsymbol{R2}_{t-1}, \boldsymbol{R1}_t)$, Then from Fig. 3 we have $\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r})$ with the F function defined as follows

$$\begin{cases} \boldsymbol{z}_{t-1} = (\boldsymbol{s}_{t+14} \boxplus \boldsymbol{u}) \oplus \boldsymbol{v} \oplus \boldsymbol{s}_{t-1} \\ \boldsymbol{z}_t = (\boldsymbol{s}_{t+15} \boxplus \boldsymbol{w}_t) \oplus \mathtt{S}_1(\boldsymbol{u}) \oplus \boldsymbol{s}_t \\ \boldsymbol{z}_{t+1} = (\boldsymbol{s}_{t+16} \boxplus (\boldsymbol{s}_{t+5} \oplus \mathtt{S}_2(\boldsymbol{v})) \boxplus \mathtt{S}_1(\boldsymbol{u})) \oplus \mathtt{S}_1(\boldsymbol{w}) \oplus \boldsymbol{s}_{t+1} \end{cases} \tag{21}$$

We let Γ_i be the linear mask of $\boldsymbol{s}_{t+i}$. The targeted linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ can therefore be defined as Eq. (22)

$$\begin{cases} \Gamma_{\boldsymbol{\ell}} = (\Gamma_{-1}, \Gamma_0, \Gamma_1, \Gamma_5, \Gamma_{14}, \Gamma_{15}, \Gamma_{16}) \\ \Gamma_{\boldsymbol{z}} = (\Gamma_{\boldsymbol{z}_0}, \Gamma_{\boldsymbol{z}_1}, \Gamma_{\boldsymbol{z}_2}) = (\Gamma_{\boldsymbol{z}_{t-1}}, \Gamma_{\boldsymbol{z}_t}, \Gamma_{\boldsymbol{z}_{t+1}}) \end{cases} \tag{22}$$

The bitwise model $\mathcal{M}$ for SNOW 3G is also hard to solve directly, so the task *Candidate Search* of SNOW 3G is also accomplished with Algorithm 7 enabling us to acquire large number of highly qualified linear masks. The constraints used in $\mathcal{M}_T$ and $\mathcal{M}$ as well as other details are given in Section E.1.

6.2.2 Correlation Computation for SNOW 3G

According to Eq. (21), we define 32-bit intermediate states $\boldsymbol{x} = \mathtt{SBX}_1(\boldsymbol{u})$, $\boldsymbol{y} = \mathtt{SBX}_1(\boldsymbol{w})$, $\xi_t = \boldsymbol{s}_{t+5} \oplus \mathtt{S}_2(\boldsymbol{v})$ and $\eta_t = \mathtt{S}_1(\boldsymbol{u})$. With the linear mask in $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ in Eq. (22), we decompose the noise $e = (\Gamma_{\boldsymbol{\ell}} \cdot \boldsymbol{\ell}) \oplus (\Gamma_{\boldsymbol{z}} \cdot F(\boldsymbol{\ell}, \boldsymbol{r}))$ into four sub-noises as

$$\begin{cases} e_1 = \Gamma_{\boldsymbol{z}_0} \cdot (\boldsymbol{s}_{t+14} \boxplus \mathtt{SBX}_1{}^{-1}(\boldsymbol{x})) \oplus \Gamma_{14} \cdot \boldsymbol{s}_{t+14} \oplus \Theta' \cdot \boldsymbol{x}, \\ e_2 = \Gamma_{\boldsymbol{z}_1} \cdot (\boldsymbol{s}_{t+15} \boxplus \mathtt{SBX}_1{}^{-1}(\boldsymbol{y})) \oplus \Gamma_{15} \cdot \boldsymbol{s}_{t+15} \oplus \Gamma'_{\boldsymbol{z}_2} \cdot \boldsymbol{y}, \\ e_3 = \Gamma_{\boldsymbol{z}_2} \cdot (\boldsymbol{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \Gamma_{16} \cdot \boldsymbol{s}_{t+16} \oplus \Gamma_5 \cdot \xi_t \oplus (\Theta \oplus \Gamma_{\boldsymbol{z}_1}) \cdot \eta_t, \\ e_4 = \Gamma''_5 \cdot \mathtt{SBX}_2(\boldsymbol{v}) \oplus \Gamma_{\boldsymbol{z}_0} \cdot \boldsymbol{v}, \end{cases}$$

where Θ', $\Gamma'_{\boldsymbol{z}_2}$ and Γ''_5 are defined s.t. $\Theta' \cdot \boldsymbol{x} = \Theta \cdot \mathtt{LL}_1(\boldsymbol{x})$, $\Gamma'_{\boldsymbol{z}_2} \cdot \boldsymbol{x} = \mathtt{LL}_1(\boldsymbol{x})$, and $\Gamma''_5 \cdot \boldsymbol{x} = \Gamma_5 \cdot \mathtt{LL}_2(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{F}_2^{32}$. Defining the basic operation groups G and F as $G(\boldsymbol{X}, \boldsymbol{Y}) = \boldsymbol{X} \boxplus_{32} \mathtt{SBX}_1{}^{-1}(\boldsymbol{Y})$ and $F(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = \boldsymbol{X} \boxplus_{32} \boldsymbol{Y} \boxplus_{32} \boldsymbol{Z}$, we are able to define the sub-noise correlations as

$$\mathtt{Cor}(e_1) = \mathtt{Cor}((\Gamma_{14}, \Theta') \xrightarrow{G} \Gamma_{\boldsymbol{z}_0}), \quad \mathtt{Cor}(e_3) = \mathtt{Cor}((\Gamma_{16}, \Gamma_5, \Theta \oplus \Gamma_{\boldsymbol{z}_1}) \xrightarrow{F} \Gamma_{\boldsymbol{z}_2}),$$
$$\mathtt{Cor}(e_2) = \mathtt{Cor}((\Gamma_{15}, \Gamma'_{\boldsymbol{z}_2}) \xrightarrow{G} \Gamma_{\boldsymbol{z}_1}), \quad \mathtt{Cor}(e_4) = \mathtt{Cor}(\Gamma_{\boldsymbol{z}_0} \xrightarrow{\mathtt{SBX}_2} \Gamma''_5).$$

We use the constant-time algorithms in [28] and [26] respectively to accurately computing $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}) \xrightarrow{G} \boldsymbol{A})$ and $\mathtt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{F} \boldsymbol{A})$, while $\mathtt{Cor}(\boldsymbol{B} \xrightarrow{\mathtt{SBX}_2} \boldsymbol{A})$ can be obtained through four LAT lookups. Then, same as SNOW-V/Vi, the targeted $\mathtt{Cor}$ for SNOW 3G can also be computed efficiently

as $\mathtt{Cor}(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}}) = \mathtt{Cor}(e_2)\mathtt{Cor}(e_4) \sum_{\Theta} \mathtt{Cor}(e_1)\mathtt{Cor}(e_3)$.

The Selective Linear Masks. From the millions of linear mask candidates given in Section 6.2.1, using the algebraic bias evaluation technique above, we are able to obtain 18 mask tuples having $|\mathtt{Cor}| > 2^{-21}$, and the best is $|\mathtt{Cor}| = 2^{-20.386}$ breaking the previous record [30] as well. We compare the results of linear masks in Table E7 of Section E.1. Besides, we obtain another 175 tuples s.t. $2^{-22} < |\mathtt{Cor}| \leq 2^{-21}$. It is noticeable that there is only one optimal solution for $\mathcal{M}_T$ and all the best masks in Table E7 are found from the candidates with $b_{obj} \leq 37$. For those with $b_{obj} \geq 38$, the best correlation does not increase, indicating that the masks in Table E7 are likely to be optimal.

6.3 Linear Mask Search for SNOW 2.0

6.3.1 Candidate Search for SNOW 2.0

The FSM part of SNOW 2.0 has two 32-bit registers, thus 2-round linear approximations are considered to cancel out the non-linear contributions, as depicted in Fig.4. According to the keystream generation function and the

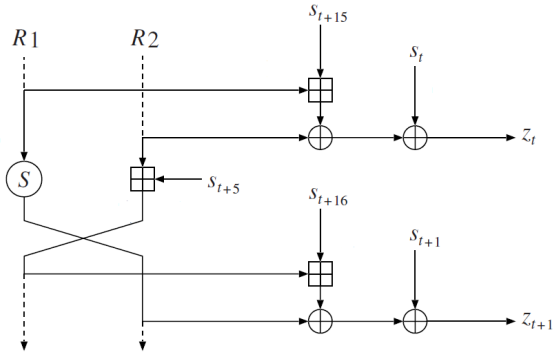

Fig. 4: The two-round linear approximations for the FSM of SNOW 2.0

FSM updating function in Eq. (7) and Eq. (8), we define $(\boldsymbol{\ell}, \boldsymbol{r}, \boldsymbol{z})$ as: $\boldsymbol{z} = (\boldsymbol{z}_t, \boldsymbol{z}_{t+1})$, $\boldsymbol{\ell} = (\boldsymbol{s}_t, \boldsymbol{s}_{t+1}, \boldsymbol{s}_{t+5}, \boldsymbol{s}_{t+15}, \boldsymbol{s}_{t+16})$ and $\boldsymbol{r} = (\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{R1}_t, \boldsymbol{R2}_t)$. Then $\boldsymbol{z} = F(\boldsymbol{\ell}, \boldsymbol{r})$ where the F function is defined as follows

$$\begin{cases} \boldsymbol{z}_t = (\boldsymbol{s}_{t+15} \boxplus \boldsymbol{u}) \oplus \boldsymbol{v} \oplus \boldsymbol{s}_t \\ \boldsymbol{z}_{t+1} = (\boldsymbol{s}_{t+16} \boxplus \boldsymbol{s}_{t+5} \boxplus \boldsymbol{v}) \oplus \mathtt{S}_1(\boldsymbol{u}) \oplus \boldsymbol{s}_{t+1} \end{cases} \tag{23}$$

We let Γ_i be the linear mask of $\boldsymbol{s}_{t+i}$. The targeted linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ can therefore be defined as:

$$\begin{cases} \Gamma_{\boldsymbol{\ell}} = (\Gamma_0, \Gamma_1, \Gamma_5, \Gamma_{15}, \Gamma_{16}) \\ \Gamma_{\boldsymbol{z}} = (\Gamma_{\boldsymbol{z}_0}, \Gamma_{\boldsymbol{z}_1}) = (\Gamma_{\boldsymbol{z}_t}, \Gamma_{\boldsymbol{z}_{t+1}}) \end{cases} \tag{24}$$

Unlike SNOW-V/Vi and SNOW 3G, the bitwise model $\mathcal{M}$ for SNOW 2.0 can be solved directly. Therefore, *Candidate Search* for SNOW 2.0 is accomplished simply with a bitwise model $\mathcal{M}$. Details are given in Section E.2.

6.3.2 Correlation Computation for SNOW 2.0

With linear mask $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ in Eq. (24), there is $e = e_1 \oplus e_2$ with e_1, e_2 defined as follows:

$$\begin{cases} e_1 = \Gamma_{\boldsymbol{z}_0} \cdot (\boldsymbol{s}_{t+15} \boxplus \mathtt{SBX}_1{}^{-1}(\boldsymbol{x})) \oplus \Gamma_{15} \cdot \boldsymbol{s}_{t+15} \oplus \Gamma'_{\boldsymbol{z}_1} \cdot \boldsymbol{x} \\ e_2 = \Gamma_{\boldsymbol{z}_1} \cdot (\boldsymbol{s}_{t+16} \boxplus \boldsymbol{s}_{t+5} \boxplus \boldsymbol{v}) \oplus \Gamma_{16} \cdot \boldsymbol{s}_{t+16} \oplus \Gamma_5 \cdot \boldsymbol{s}_{t+5} \oplus \Gamma_{\boldsymbol{z}_0} \cdot \boldsymbol{v} \end{cases}$$

where $\Gamma'_{\boldsymbol{z}_1}$ satisfies $\Gamma'_{\boldsymbol{z}_1} \cdot \boldsymbol{x} = \Gamma_{\boldsymbol{z}_1} \cdot \mathtt{LL}_1(\boldsymbol{x})$ for $\forall \boldsymbol{x} \in \mathbb{F}_2^{32}$. The targeted correlation is computed as $\mathtt{Cor}(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}}) = \mathtt{Cor}(e_1)\mathtt{Cor}(e_2)$, where $\mathtt{Cor}(e_1) = \mathtt{Cor}((\Gamma_{15}, \Gamma'_{\boldsymbol{z}_1}) \xrightarrow{G} \Gamma_{\boldsymbol{z}_0})$ and $\mathtt{Cor}(e_2) = \mathtt{Cor}((\Gamma_{16}, \Gamma_5, \Gamma_{\boldsymbol{z}_0}) \xrightarrow{F} \Gamma_{\boldsymbol{z}_1})$.

The Selective Linear Masks. For SNOW 2.0, we obtain 26 mask tuples with $|\mathtt{Cor}| > 2^{-15}$, while only 3 were found in [28]. Among these masks, 4 tuples yield the same highest correlation $\pm 2^{-14.411}$. We compare the results of linear masks in Table E8 of Section E.2. Besides, we obtain another 372 ones with $2^{-16} < |\mathtt{Cor}| \leq 2^{-15}$.

7 Using Linear Masks in Fast Correlation Attacks

Correlation Attack on Full SNOW-V/Vi. We launch our attacks following strictly the preprocessing phase and processing phase. We use all the $8 + 127 = 135$ mask tuples which yield correlations $|\mathtt{Cor}| > 2^{-49}$ to build approximation relations. The average correlation is computed as $\alpha \triangleq 2^{-48.58}$.

We first collect N (to be determined) samples involving only the keystream outputs and the l-bit LFSR initial state bits, and then try to reduce the number of the involved LFSR initial state bits to $l'(< l)$ bits by searching for pairs of columns of the generator matrix which add to 0 on the most significant $l - l'$ bits, at the expense of the increased noise level with the correlation α^2. Regarding the column vectors of the generator matrix as random vectors, there are about $M \triangleq C_N^2 2^{-(l-l')}$ such pairs, corresponding to M approximation relations with correlation α^2 involving only the first l' bits of the LFSR initial state, which can be found by the sort-and-merge procedure with the time complexity $\mathcal{O}(N \log_2 N)$ and the memory complexity $\mathcal{O}(N)$.

To recover the value of the target l' bits, we use the FWT to speed up the evaluation of the M linear approximation relations, which needs a time complexity $\mathcal{O}(M + l'2^{l'})$ and a memory complexity $\mathcal{O}(2^{l'})$. To guarantee a high success probability, M is usually chosen as $M = 2l' \ln 2/(\alpha^2)^2$, the parameter N is thus determined to be $N \approx \sqrt{M2^{l-l'+1}}$, and the required number of keystream outputs is $D = N/135$.

Complexity Analysis. For SNOW-V/Vi, we follow the above procedure with the parameters $l = 512$, $l' = 238$. In the preprocessing phase, we need to prepare $M = 2l' \ln 2/(\alpha^2)^2 = 2^{202.69}$ approximation relations with correlation α^2 involving the first 238 bits of the LFSR initial state. Thus the number of samples required is $N = \sqrt{M2^{l-l'+1}} = 2^{238.84}$ and we need to know $D = N/135 = 2^{231.76}$ keystream outputs. The required time and memory complexity for preparing M approximation relations is $2^{246.74}$ and $2^{238.84}$, respectively. In the processing phase, the FWT is utilized to determine the first 238 bits of the LFSR initial state, which needs a time complexity $2^{245.89}$ and a memory complexity 2^{238}. In summary, the total time and memory complexities are $2^{247.38}$ and $2^{239.48}$ respectively, and the keystream length needed is $2^{231.76}$. Once the first 238 bits are recovered, the other bits and the FSM state can be recovered by using a similar method and a small-scale exhaustive search with a much lower complexity.

Comparison. Table 1 presents a comparison of our attack with the previous ones in [34, 35], from which we know the data complexity is reduced.

Correlation Attack on SNOW 3G. Different from the correlation attack on SNOW-V, where pairs of column vectors of the generator matrix vanishing on some bits are searched, we look for $k(= 4)$-tuples of columns of the generator matrix for SNOW 3G.

For SNOW 3G, we have $l = 512$. Suppose our top n best masks in Table E7 of Appendix E.1 with an average absolute correlation α are used for building the approximations. In the preprocessing phase, we first collect N (to be determined) samples involving only the keystream outputs and l LFSR initial state bits, and then try to reduce the involved LFSR initial state bits size to l' bits, by searching for a number of $k(= 4)$-tuples of columns of the generator matrix which add to 0 on the most significant $l - l'$ bits, at the expense of the increased noise level with the correlation α^4. After this, we enter the processing phase to recover the target l' bits by using the FWT for calculating and evaluating the parity check equations.

Complexity Analysis. We discuss the attack by setting $l' = 166$ from the following two cases:

Case 1: We use our top 6 best masks in Table E7 for approximations, whose average absolute correlation is $\alpha \triangleq 2^{-20.468}$. In the preprocessing phase, we need to prepare $M = 2l' \ln 2/(\alpha^4)^2 = 2^{171.59}$ approximation relations with correlation α^4 involving the first 166 bits of the LFSR initial state, i.e., the number of 4-tuples found from N samples should be at least $2^{171.59}$. This can be solved by the method described in Appendix H. By choosing $l_1 = 173$ and

$l_2 = 173$, we have $m_1 = 2^{172.80}$ and $N = 2^{173.40}$. Thus it requires the keystream outputs of length $D = N/6 = 2^{170.81}$, and the time/memory complexity for preparing M approximation relations is $\mathcal{O}(N + m_1)$, i.e., $2^{174.13}$. In the processing phase, the FWT is utilized to determine the first 166 bits of the LFSR initial state, which costs a time complexity $2^{173.74}$ and a memory complexity 2^{166}. In summary, the total time and memory complexities are $2^{174.95}$ and $2^{174.13}$ respectively, and the keystream length needed is $2^{170.81}$.

Case 2: We use all the 18 masks in Table E7 for approximations, whose average absolute correlation is $\alpha \triangleq 2^{-20.738}$. In the preprocessing phase, we prepare $M = 2l' \ln 2/(\alpha^4)^2 = 2^{173.75}$ approximation relations with correlation α^4 involving the first 166 bits of the LFSR initial state. By choosing $l_1 = 172$ and $l_2 = 174$, we have $m_1 = 2^{174.38}$ and $N = 2^{173.69}$. Thus it requires the keystream outputs of length $D = N/18 = 2^{169.52}$, and the time/memory complexity for preparing M approximation relations is $\mathcal{O}(N + m_1)$, i.e., $2^{175.07}$. In the processing phase, the FWT is utilized to determine the first 166 bits of the LFSR initial state, which costs a time complexity $2^{174.58}$ and a memory complexity 2^{166}. In summary, the total time and memory complexities are $2^{175.85}$ and $2^{175.07}$ respectively, and the keystream length needed is $2^{169.52}$.

Comparison. Table 1 presents a comparison of our attack with the previous ones in [29, 30]. We want to emphasize that the advantage of our method is it can find many masks with high correlations efficiently. Making good use of this, we target attacks with data complexity as low as possible. Such a low-data result reflects the advantage of our method.

Correlation Attack on SNOW 2.0. We launch our attacks following the two stages of correlation attacks in Section 4.3 of [28]. Different from [28] where only 3 mask tuples yielding an average correlation $2^{-14.51}$ were used for linear approximations, we here use all the 26 mask tuples in Table E8 to build approximation relations, whose average absolute correlation is $\alpha \triangleq 2^{-14.76}$.

Similar with the attack on SNOW 3G, we also look for 4-tuples of columns of the generator matrix for SNOW 2.0 such that the values of XOR are 0 on the most significant $l - l'$ bits. Let $l = 512$ and $l' = 154$. In the preprocessing phase, we need to prepare $M = 2l' \ln 2/(\alpha^4)^2 = 2^{125.82}$ approximation relations with correlation α^4 involving the first 154 bits of the LFSR initial state, i.e., the number of 4-tuples found from N samples should be at least $2^{125.82}$. This can be solved by the method described in Appendix H. By choosing $l_1 = 159$ and $l_2 = 199$, we have $m_1 = 2^{162.91}$ and $N = 2^{161.45}$. Thus it requires the keystream outputs of length $D = N/26 = 2^{156.75}$, and the time/memory complexity for preparing M approximation relations is $\mathcal{O}(N+m_1)$, i.e., $2^{163.36}$. In the processing phase, the FWT is utilized to determine the first 154 bits of the LFSR initial state, which costs a time complexity $2^{161.27}$ and a memory complexity 2^{154}. In summary, the total time and memory complexities are $2^{163.66}$ and $2^{163.36}$ respectively, and the keystream length needed is $2^{156.82}$.

Comparison. Table 1 presents a comparison of our attack with previously the best ones in [28]. Similarly, we reduce the data complexity with the time and memory complexities slightly increased.

8 Conclusion

This paper proposed a new bitwise breakdown strategy for describing the linear propagation rules of the modular addition-based operations. When applied to the SNOW family stream ciphers, a large number of highly qualified linear trails with the help of MILP tools are found. Besides, we use an algebraic bias evaluation technique to analyze the strength of linear approximations, and present an efficient algorithm for accurately computing the correlation values of linear approximations of a certain type of composition function. Combining the two techniques, we find many linear masks for SNOW family yielding high correlations and some of them are likely to be optimal. Based on newly found linear masks, we give improved correlation attacks on the SNOW family. Our results further confirm their resistance to correlation attacks.

The uniform MILP-aided framework proposed for finding linear masks is not only easy to understand but also flexible for describing the linear propagation of consecutive modular addition-based operations and deducing optimal truncated linear masks. The main focus of this paper is SNOW family, however, this uniform framework can be directly used to analyze the linear property of LFSR-FSM structures composed of modular addition and S-boxes as non-linear components, for instance SOSEMANUK and ZUC, in a similar way.

Information on Supplementary. We provide the verification codes for SNOW 2.0, SNOW 3G and SNOW-V/Vi, practically evaluating linear masks we found. Our code can be compiled on Linux or on Windows with Visual Studio 2019.

Declarations

The data sets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

References

[1] Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer, Berlin, Heidelberg (2011)

[2] Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBLOCK, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, pp. 158–178. Springer, Berlin, Heidelberg (2014)

[3] Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: MILP-based automatic search algorithms for differential and linear trails for Speck. In: Peyrin, T. (ed.)

FSE 2016. LNCS, vol. 9783, pp. 268–288. Springer, Berlin, Heidelberg (2016)

[4] Cui, T., Chen, S., Fu, K., Wang, M., Jia, K.: New automatic tool for finding impossible differentials and zero-correlation linear approximations. Sci. China Inf. Sci. **64**(2) (2021)

[5] Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 648–678 (2016)

[6] Sun, L., Wang, W., Liu, R., Wang, M.: MILP-aided bit-based division property for ARX ciphers. Sci. China Inf. Sci. **61**(11), 118102–11181023 (2018)

[7] Huang, S., Wang, X., Xu, G., Wang, M., Zhao, J.: Conditional cube attack on reduced-round Keccak sponge function. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 259–288 (2017)

[8] Todo, Y., Isobe, T., Hao, Y., Meier, W.: Cube attacks on non-blackbox polynomials based on division property. IEEE Trans. Computers **67**(12), 1720–1736 (2018)

[9] Wang, Q., Hao, Y., Todo, Y., Li, C., Isobe, T., Meier, W.: Improved division property based cube attacks exploiting algebraic properties of superpoly. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 275–305. Springer, Berlin, Heidelberg (2018)

[10] Funabiki, Y., Todo, Y., Isobe, T., Morii, M.: Several MILP-aided attacks against SNOW 2.0. In: Camenisch, J., Papadimitratos, P. (eds.) CANS 2018. LNCS, vol. 11124, pp. 394–413. Springer, Berlin, Heidelberg (2018)

[11] Hao, Y., Leander, G., Meier, W., Todo, Y., Wang, Q.: Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 466–495. Springer, Berlin, Heidelberg (2020)

[12] Hu, K., Sun, S., Todo, Y., Wang, M., Wang, Q.: Massive superpoly recovery with nested monomial predictions. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 392–421. Springer, Berlin, Heidelberg (2021)

[13] Todo, Y., Isobe, T., Meier, W., Aoki, K., Zhang, B.: Fast correlation attack revisited - cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS,

vol. 10992, pp. 129–159. Springer, Berlin, Heidelberg (2018)

[14] Gurobi optimizer reference manual. https://www.gurobi.com

[15] Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) S-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017)

[16] Udovenko, A.: MILP modeling of Boolean functions by minimum number of inequalities. Cryptology ePrint Archive, Report 2021/1099 (2021)

[17] Sun, Y.: Towards the Least Inequalities for Describing a Subset in Z_2^n. Cryptology ePrint Archive, Report 2021/1084 (2021)

[18] Ekdahl, P., Johansson, T.: A new version of the stream cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer

[19] SAGE, E.: Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2, document 2: SNOW 3G specification, v1.1, 2006

[20] Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new SNOW stream cipher called SNOW-V. IACR Trans. Symmetric Cryptol. **2019**(3), 1–42 (2019)

[21] Ekdahl, P., Maximov, A., Johansson, T., Yang, J.: SNOW-Vi : An extreme performance variant of SNOW-V for lower grade cpus. In: WiSec 2021, pp. 261–272. (ACM)

[22] Chepyzhov, V.V., Johansson, T., Smeets, B.J.M.: A simple algorithm for fast correlation attacks on stream ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Berlin, Heidelberg (2000)

[23] Chose, P., Joux, A., Mitton, M.: Fast correlation attacks: An algorithmic point of view. In: Knudsen, L.R. (ed.) EUROCRYPT 2002, pp. 209–221. Springer, Berlin, Heidelberg (2002)

[24] Coppersmith, D., Halevi, S., Jutla, C.: Cryptanalysis of stream ciphers with linear masking. In: Yung, M. (ed.) CRYPTO 2002, pp. 515–532. Springer, Berlin, Heidelberg (2002)

[25] Watanabe, D., Biryukov, A., Cannière, C.D.: A distinguishing attack of SNOW 2.0 with linear masking method. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 222–233. Springer, Berlin, Heidelberg (2003)

[26] Nyberg, K., Wallén, J.: Improved linear distinguishers for SNOW 2.0. In:

Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 144–162. Springer, Berlin, Heidelberg (2006)

[27] Zhang, B., Xu, C., Meier, W.: Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 643–662. Springer, Berlin, Heidelberg (2015)

[28] Gong, X., Zhang, B.: Fast computation of linear approximation over certain composition functions and applications to SNOW 2.0 and SNOW 3G. Designs, Codes and Cryptography **88**(11), 2407–2431 (2020)

[29] Yang, J., Johansson, T., Maximov, A.: Vectorized linear approximations for attacks on SNOW 3G. IACR Trans. Symmetric Cryptol. **2019**(4), 249–271 (2019)

[30] Gong, X., Zhang, B.: Comparing large-unit and bitwise linear approximations of SNOW 2.0 and SNOW 3G and related attacks. IACR Trans. Symmetric Cryptol. **2021**(2), 71–103 (2021)

[31] Gong, X., Zhang, B.: Resistance of SNOW-V against fast correlation attacks. IACR Trans. Symmetric Cryptol. **2021**(1), 378–410 (2021)

[32] Yang, J., Johansson, T., Maximov, A.: Improved guess-and-determine and distinguishing attacks on SNOW-V. IACR Trans. Symmetric Cryptol. **2021**(3), 54–83 (2021)

[33] Shi, Z., Jin, C., Zhang, J., Cui, T., Ding, L.: A Correlation Attack on Full SNOW-V and SNOW-Vi. Cryptology ePrint Archive, Report 2021/1047 (2021)

[34] Shi, Z., Jin, C., Jin, Y.: Improved Linear Approximations of SNOW-V and SNOW-Vi. Cryptology ePrint Archive, Report 2021/1105 (2021)

[35] Zhou, Z., Feng, D., Zhang, B.: Efficient and extensive search for precise linear approximations with high correlations of full SNOW-V. Designs, Codes and Cryptography **90**(10), 2449–2479 (2022). https://doi.org/10.1007/s10623-022-01090-8

[36] Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT'93. LNCS, vol. 765, pp. 386–397. Springer, Berlin, Heidelberg

[37] Maximov, A., Johansson, T.: Fast computation of large distributions and its cryptographic applications. In: Roy, B. (ed.) Advances in Cryptology - ASIACRYPT 2005, pp. 313–332. Springer, Berlin, Heidelberg (2005)

[38] Correlation theorems in cryptanalysis. Discrete Applied Mathematics **111**(1), 177–188 (2001). https://doi.org/10.1016/S0166-218X(00)00351-6. Coding and Cryptology

[39] Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) Advances in Cryptology — CRYPTO 2002, pp. 288–304. Springer, Berlin, Heidelberg (2002)

Appendix A Overall Schematic of SNOW-V/Vi, SNOW 3G and SNOW 2.0

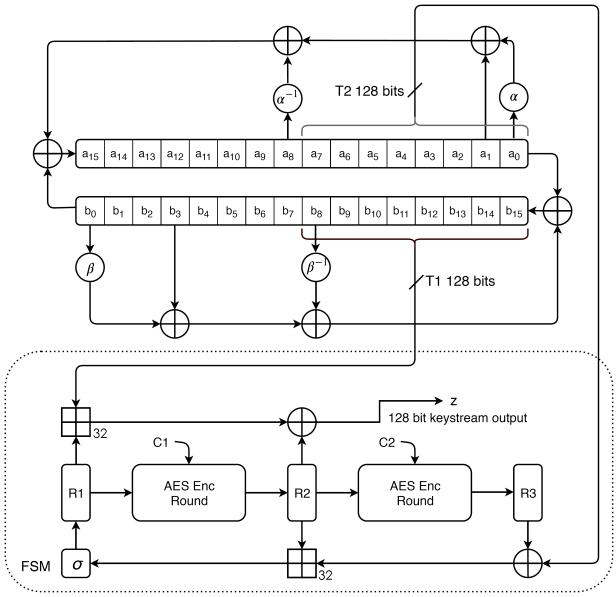

Fig. A1: The keystream generation phase of the SNOW-V stream cipher

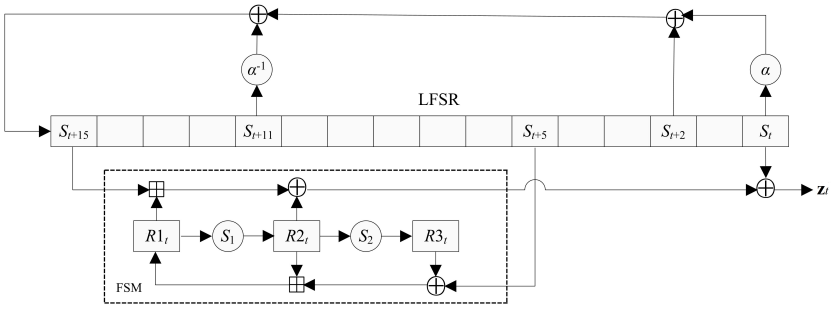

Fig. A2: The keystream generation phase of SNOW 3G

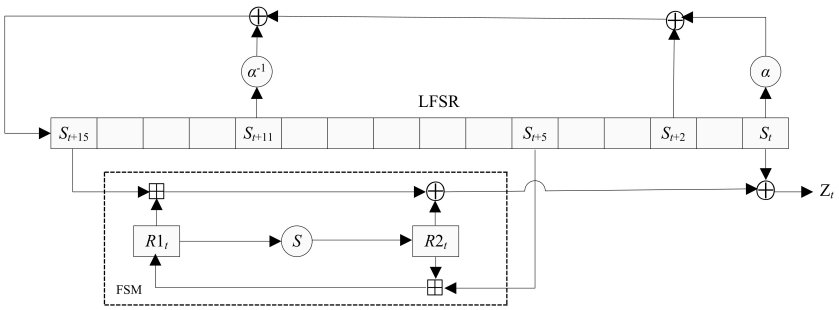

Fig. A3: The keystream generation phase of SNOW 2.0

Appendix B MILP Model Construction of Common Operations

According to Section 2.4, basic linear operations can be perfectly described with MILP model constraints. For $x_0, x_1, y \in \mathbb{F}_2$, an XOR operation $(x_0, x_1) \to y = x_0 \oplus x_1$ can be represented as linear constraints in MILP models as Eq. (B1)

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} x_0 + x_1 - y \geq 0 \\ x_0 - x_1 + y \geq 0 \\ -x_0 + x_1 + y \geq 0 \\ x_0 + x_1 + y \leq 2 \end{cases} \tag{B1}$$

and we may simplify the representation of such linear constraints as Eq. (B2).

$$\mathcal{M}.\texttt{con} \leftarrow y = x_0 \oplus x_1 \tag{B2}$$

For 0-1 matrix $M \in \mathbb{F}_2^{m \times n}$, the (column) vectors $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}_2^n \times \mathbb{F}_2^m$ satisfying $\boldsymbol{y} = M\boldsymbol{x}$ can be regarded as composition of XORs which is described with MILP model $\mathcal{M}$ in the form of Eq. (B3).

$$\begin{cases} \mathcal{M}.\texttt{var} \leftarrow \boldsymbol{x}, \boldsymbol{y} \text{ as binaries} \\ \mathcal{M}.\texttt{con} \leftarrow \boldsymbol{y} = M|_{\oplus}\boldsymbol{x} \end{cases} \tag{B3}$$

B.1 Bitwise Linear Propagation Rules

For the XOR operation $\boldsymbol{y} = \boldsymbol{x}_0 \oplus \boldsymbol{x}_1$, the available linear masks satisfy $\Gamma_{\boldsymbol{y}} = \Gamma_{\boldsymbol{x}_0} = \Gamma_{\boldsymbol{x}_1}$ which is a straightforward linear constraint in MILP model as well. For branch operation $\boldsymbol{x} \xrightarrow{\texttt{branch}} (\boldsymbol{y}_0, \boldsymbol{y}_1) = (\boldsymbol{x}, \boldsymbol{x})$, the corresponding linear mask $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}_0}, \Gamma_{\boldsymbol{y}_1})$ satisfies Eq. (B4). According to Eq. (B1) and Eq. (B2), Eq. (B4) is also a straightforward linear constraint in MILP model.

$$\Gamma_{\boldsymbol{x}} = \Gamma_{\boldsymbol{y}_0} \oplus \Gamma_{\boldsymbol{y}_1} \tag{B4}$$

For linear $\mathbb{F}_2^n \to \mathbb{F}_2^m$ transformation $\boldsymbol{y} = M\boldsymbol{x}$, the available linear masks $(\Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}})$ satisfy $\Gamma_{\boldsymbol{x}} = M^T\Gamma_{\boldsymbol{y}}$ which, according to Eq. (B3), can be captured by the MILP model constraint

$$\begin{cases} \mathcal{M}.\texttt{var} \leftarrow \Gamma_{\boldsymbol{x}}, \Gamma_{\boldsymbol{y}} \text{ as binaries} \\ \mathcal{M}.\texttt{con} \leftarrow \Gamma_{\boldsymbol{x}} = M^T|_{\oplus}\Gamma_{\boldsymbol{y}} \end{cases}$$

For 8-bit S-boxes, according to Section 2.4, the input-output masks (Γ_i, Γ_o) simply share the same truncated linear masks: $T_i = T_o$. We define Algorithm 11 as the MILP description of the input-output masks of S-boxes.

$\texttt{S}_1$ and $\texttt{S}_2$ used in SNOW 3G (also SNOW 2.0) FSM updating functions can be regarded as an S-box layer (using four 8-bit S-boxes in parallel) followed by a linear diffusion using 0-1 matrices $M_1, M_2 \in \mathbb{F}_2^{32\times 32}$. With M_1 and M_2 defined in Eq. (B5) and Eq. (B6), we can construct the MILP model for $\texttt{S}_i$ $(i = 1, 2)$ by calling `siModel` as in Algorithm 9.

$$M_1 = \begin{pmatrix}
0&0&0&0&0&0&0&0&1&1&1&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0\\
0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0\\
0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0\\
0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1\\
1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0\\
0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1\\
0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1\\
0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1
\end{pmatrix} \tag{B5}$$

$$M_2 = \begin{pmatrix}
0&0&0&0&0&0&0&0&1&1&1&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0\\
0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0\\
0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0\\
0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0&1\\
1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1&0\\
0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0&1\\
0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1\\
0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0&0\\
0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&1&0&1&1
\end{pmatrix} \tag{B6}$$

$AES^R(\cdot) = \texttt{MC} \circ \texttt{SR} \circ \texttt{SB}(\cdot)$ used in SNOW-V/Vi FSM updating functions is a permutation on $\mathbb{F}_2^{128}$. It can also be regarded as an S-box layer $\texttt{SB}$ followed by the linear layer $\texttt{MC} \circ \texttt{SR}$: the nonlinear layer $\texttt{SB}$ uses 16 8-bit S-boxes in parallel;

the `MC` can be regarded as the M_1 diffusion carried out in parallel. Therefore, the MILP model construction for AES^R can be provided as Algorithm 8.

Algorithm 8 Model construction for AES^R (`aesModel`)

Input: Initial model $\mathcal{M}$, a vector of 128 binary variables $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_{127}})$
Output: Updated model $\mathcal{M}$, a vector of 128 binary variables $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{127}})$ and 2 vector of 16 binary variables $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_{15}}), T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_{15}})$ as the input-output truncated linear masks

for $j = 0, \ldots, 15$ **do**
  Define vectors $\Gamma_{\boldsymbol{x}_j} = (\Gamma_{x_{8j}}, \ldots, \Gamma_{x_{8j+7}})$
  $(\mathcal{M}, \Gamma_{\boldsymbol{w}_j}, T_{\boldsymbol{x}_j}) \leftarrow \texttt{sbox}(\mathcal{M}, \Gamma_{\boldsymbol{x}_j})$
end for
Define permutation over integers

$$\sigma : (0, \ldots 15) \rightarrow (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11)$$

Define vector $\Gamma_{\boldsymbol{w}} = (\Gamma_{w_0}, \ldots, \Gamma_{w_{127}}) = (\Gamma_{\boldsymbol{w}_{\sigma(0)}}, \ldots, \Gamma_{\boldsymbol{w}_{\sigma(15)}})$
Declare variables $\mathcal{M}.\texttt{var} \leftarrow \Gamma_{y_0}, \ldots, \Gamma_{y_{127}}$ as binaries
Define vector $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{127}})$
for $j = 0, \ldots, 3$ **do**
  Define vector $\Gamma_{\boldsymbol{p}_j} = (\Gamma_{y_{32j}}, \ldots, \Gamma_{y_{32j+31}})$
  Define vector $\Gamma_{\boldsymbol{q}_j} = (\Gamma_{w_{32j}}, \ldots, \Gamma_{w_{32j+31}})$
  Add constraints $\mathcal{M}.\texttt{con} \leftarrow \Gamma_{\boldsymbol{q}_j} = M_1^T|_{\oplus}\Gamma_{\boldsymbol{p}_j}$
end for
for $j = 0, \ldots, 15$ **do**
  Define vectors $\Gamma_{\boldsymbol{y}_j} = (\Gamma_{y_{8j}}, \ldots, \Gamma_{y_{8j+7}})$
  $(\mathcal{M}, T_{\boldsymbol{y}_j}) \leftarrow \texttt{actSym}(\mathcal{M}, \Gamma_{\boldsymbol{y}_j})$
end for
Define vectors $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_{15}})$ and $T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_{15}})$
Return $(\mathcal{M}, \Gamma_{\boldsymbol{y}}, T_{\boldsymbol{x}}, T_{\boldsymbol{y}})$

B.2 Truncated Linear Propagation Rules

It is obvious that $\Gamma_{\boldsymbol{x}} = \Gamma_{\boldsymbol{y}} \Rightarrow T_{\boldsymbol{x}} = T_{\boldsymbol{y}}$. Therefore, for $\boldsymbol{y} = \boldsymbol{x}_0 \oplus \boldsymbol{x}_1$, the corresponding truncated linear mask $(T_{\boldsymbol{x}_0}, T_{\boldsymbol{x}_1}, T_{\boldsymbol{y}})$ always satisfy $T_{\boldsymbol{x}_0} = T_{\boldsymbol{x}_1} = T_{\boldsymbol{y}}$.

Let $\boldsymbol{x}$ be an 8-bit word. The available input-output truncated linear mask of the branch operation $\boldsymbol{x} \xrightarrow{\texttt{branch}} (\boldsymbol{y}_0, \boldsymbol{y}_1) = (\boldsymbol{x}, \boldsymbol{x})$ contains 3 bits namely $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}_0}, T_{\boldsymbol{y}_1})$. Defining the set $\mathcal{S} = \{(0,0,0), (1,0,1), (0,1,0), (1,1,0), (1,1,1)\} \subseteq \mathbb{F}_2^3$, we have $(T_{\boldsymbol{x}}, T_{\boldsymbol{y}_0}, T_{\boldsymbol{y}_1}) \in \mathcal{S}$ which can be described as MILP model constraints as Eq. (B7).

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} T_{\boldsymbol{x}} + T_{\boldsymbol{y}_0} - T_{\boldsymbol{y}_1} \geq 0 \\ T_{\boldsymbol{x}} - T_{\boldsymbol{y}_0} + T_{\boldsymbol{y}_1} \geq 0 \\ -T_{\boldsymbol{x}} + T_{\boldsymbol{y}_0} + T_{\boldsymbol{y}_1} \geq 0 \end{cases} \tag{B7}$$

Algorithm 9 Model construction for $\mathtt{S}_i$ (`siModel`)

Input: Initial model $\mathcal{M}$, a vector of 32 binary variables $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_{31}})$ and $i \in \{1, 2\}$
Output: Updated model $\mathcal{M}$, a vector of 32 binary variables $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{31}})$ and 2 vector of 4 binary variables $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_3}), T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_3})$ as the input-output truncated linear masks

for $j = 0, \ldots, 3$ **do**
  Define vectors $\Gamma_{\boldsymbol{x}_j} = (\Gamma_{x_{8j}}, \ldots, \Gamma_{x_{8j+7}})$
  $(\mathcal{M}, \Gamma_{\boldsymbol{w}_j}, T_{\boldsymbol{x}_j}) \leftarrow \mathtt{sbox}(\mathcal{M}, \Gamma_{\boldsymbol{x}_j})$
end for
Define vector $\Gamma_{\boldsymbol{w}} = (\Gamma_{w_0}, \ldots, \Gamma_{w_{31}}) = (\Gamma_{\boldsymbol{w}_0}, \ldots, \Gamma_{\boldsymbol{w}_3})$
Declare variables $\mathcal{M}.\mathtt{var} \leftarrow \Gamma_{y_0}, \ldots, \Gamma_{y_{31}}$ as binaries
Define vector $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_{31}})$
Add constraints $\mathcal{M}.\mathtt{con} \leftarrow \Gamma_{\boldsymbol{w}} = M_i^T|_{\oplus}\Gamma_{\boldsymbol{y}}$
for $j = 0, \ldots, 3$ **do**
  Define vectors $\Gamma_{\boldsymbol{y}_j} = (\Gamma_{y_{8j}}, \ldots, \Gamma_{y_{8j+7}})$
  $(\mathcal{M}, T_{\boldsymbol{y}_j}) \leftarrow \mathtt{actSym}(\mathcal{M}, \Gamma_{\boldsymbol{y}_j})$
end for
Define vectors $T_{\boldsymbol{x}} = (T_{\boldsymbol{x}_0}, \ldots, T_{\boldsymbol{x}_3})$ and $T_{\boldsymbol{y}} = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_3})$
Return $(\mathcal{M}, \Gamma_{\boldsymbol{y}}, T_{\boldsymbol{x}}, T_{\boldsymbol{y}})$

Algorithm 10 Model construction of truncated linear symbol `actSym`

Input: Initial model $\mathcal{M}$ and a vector of m binary variables $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_{m-1}})$
Output: Updated model $\mathcal{M}$, a and a binary variable τ as the truncated linear symbol

Declare a variable $\mathcal{M}.\mathtt{con} \leftarrow \tau$ as binary
for $i = 0, \ldots, m-1$ **do**
  $\mathcal{M}.\mathtt{con} \leftarrow \tau \geq \Gamma_{x_i}$
end for
$\mathcal{M}.\mathtt{con} \leftarrow \tau \leq \sum_{i=0}^{m-1} \Gamma_{x_i}$
Return $(\mathcal{M}, \tau)$

Algorithm 11 Model construction for 8-bit S-box (`sbox`)

Input: Initial model $\mathcal{M}$ and a vector of 8 binary variables $\Gamma_{\boldsymbol{x}} = (\Gamma_{x_0}, \ldots, \Gamma_{x_7})$
Output: Updated model $\mathcal{M}$, a vector of 8 binary variables $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_7})$ and a binary variable τ as the truncated linear

Declare 8 variables $\mathcal{M}.\mathtt{var} \leftarrow \Gamma_{y_0}, \ldots, \Gamma_{y_7}$ as binaries
Define vector $\Gamma_{\boldsymbol{y}} = (\Gamma_{y_0}, \ldots, \Gamma_{y_7})$
$(\mathcal{M}, \tau) \leftarrow \mathtt{actSym}(\mathcal{M}, \Gamma_{\boldsymbol{y}})$
$(\mathcal{M}, \tau') \leftarrow \mathtt{actSym}(\mathcal{M}, \Gamma_{\boldsymbol{x}})$
$\mathcal{M}.\mathtt{con} \leftarrow \tau = \tau'$
Return $(\mathcal{M}, \Gamma_{\boldsymbol{y}}, \tau)$

We simplify Eq. (B7) as Eq. (B8)

$$\mathcal{M}.\mathtt{con} \leftarrow T_{\boldsymbol{x}} = T_{\boldsymbol{y}_0} \oplus_T T_{\boldsymbol{y}_1} \tag{B8}$$

Therefore, for $\boldsymbol{x} \xrightarrow{\mathtt{branch}} (\boldsymbol{y}_0, \boldsymbol{y}_1)$ of an arbitrary length, the corresponding truncated linear MILP model can be constructed as Algorithm 12.

Algorithm 12 Truncated linear model of the branch operation (`branchTrunc`)

Input: Initial model $\mathcal{M}$ and a vector of m binary variables $T_{\boldsymbol{x}} = (T_{x_0}, \ldots, T_{x_{m-1}})$
Output: Updated model $\mathcal{M}$, 2 vectors of m binary variables $T_{\boldsymbol{y}_0} = (T_{0,0}, \ldots, T_{0,m-1})$ and $T_{\boldsymbol{y}_1} = (T_{1,0}, \ldots, T_{1,m-1})$
1: **for** $j = 0, \ldots, m-1$ **do**
2: Declare 2 variables $\mathcal{M}.\texttt{var} \leftarrow T_{0,j}, T_{1,j}$ as binaries
3: $\mathcal{M}.\texttt{con} \leftarrow T_{x_j} = T_{0,j} \oplus_T T_{1,j}$
4: **end for**
5: Define vectors $T_{\boldsymbol{y}_0} = (T_{0,0}, \ldots, T_{0,m-1})$ and $T_{\boldsymbol{y}_1} = (T_{1,0}, \ldots, T_{1,m-1})$
6: Return $(\mathcal{M}, T_{\boldsymbol{y}_0}, T_{\boldsymbol{y}_1})$

For $\mathtt{S}_1$, $\mathtt{S}_2$ and AES^R, the S-box layer does not affect the truncated linear masks and the effects of linear layer can be modeled simply with its branch number. Since both M_1 and M_2 have branch number 5, the truncated linear models for $\mathtt{S}_i$ $(i = 0, 1)$ and AES^R can be described with Algorithm 13 and Algorithm 14 respectively.

Algorithm 13 Truncated linear model of $\mathtt{S}_i$ $(i = 0, 1)$ (`siTruncModel`)

Input: Initial model $\mathcal{M}$ and a vector of 4 binary variables $T_{\boldsymbol{x}} = (T_{x_0}, \ldots, T_{x_3})$
Output: Updated model $\mathcal{M}$, a vector of 4 binary variables $T_{\boldsymbol{y}} = (T_{y_0}, \ldots, T_{y_3})$
1: Declare 4 variables $\mathcal{M}.\texttt{var} \leftarrow T_{y_0}, \ldots, T_{y_3}$ as binaries
2: Define vector $T_{\boldsymbol{y}} = (T_{y_0}, \ldots, T_{y_3})$
3: Declare 1 variable $\mathcal{M}.\texttt{var} \leftarrow \tau$ as binary
4: $\mathcal{M}.\texttt{con} \leftarrow \sum_{i=0}^{3}(T_{x_i} + T_{y_i}) \leq 8\tau$
5: $\mathcal{M}.\texttt{con} \leftarrow \sum_{i=0}^{3}(T_{x_i} + T_{y_i}) \geq 5\tau$
6: Return $(\mathcal{M}, T_{\boldsymbol{y}})$

Algorithm 14 Truncated linear model of AES^R (`aesTruncModel`)

Input: Initial model $\mathcal{M}$ and a vector of 16 binary variables $T_{\boldsymbol{x}} = (T_{x_0}, \ldots, T_{x_{15}})$
Output: Updated model $\mathcal{M}$, a vector of 16 binary variables $T_{\boldsymbol{y}} = (T_{y_0}, \ldots, T_{y_{15}})$
1: Define permutation over integers: $\sigma : (0, \ldots 15) \rightarrow (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11)$
2: Define vector $T_{\boldsymbol{w}} = (T_{w_0}, \ldots, T_{w_{15}}) = (T_{x_{\sigma(0)}}, \ldots, T_{x_{\sigma(15)}})$
3: **for** $j = 0, \ldots 3$ **do**
4: Define vector $T_{\boldsymbol{p}_j} = (T_{w_{4j}}, \ldots, T_{w_{4j+3}})$
5: $(\mathcal{M}, T_{\boldsymbol{y}_j}) \leftarrow \texttt{siTruncModel}(\mathcal{M}, T_{\boldsymbol{p}_j})$
6: **end for**
7: Define vector $T_{\boldsymbol{y}} = (T_{y_0}, \ldots, T_{y_{15}}) = (T_{\boldsymbol{y}_0}, \ldots, T_{\boldsymbol{y}_3})$
8: Return $(\mathcal{M}, T_{\boldsymbol{y}})$

Appendix C MILP Model Construction of Bitwise Breakdown Functions

This part describes the model construction process of h_a, f_a, f_1 and f_2 functions. All MILP model constraints are deduced with the H-representation method in [2] according to the available input-output linear masks.

The available input-output linear masks for h_a, f_a have been given in Table 2 and Table 3. The MILP models are constructed with Algorithm 15 and Algorithm 16.

There are 98 available linear mask $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$'s (Table C1) for the f_1 and the MILP model is constructed as Algorithm 17. For f_2, the 122 available $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_{id}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$'s and the corresponding MILP models are given in Table C2 and Algorithm 18 respectively.

Algorithm 15 Model construction for h_a (`haModel`)

Input: Initial model $\mathcal{M}$ and 2 binary variables (Γ_x, Γ_y)
Output: Updated model $\mathcal{M}$ and 2 binary variables Γ_z, Γ_{oc}
1: Declare two binary variables $\mathcal{M}.\texttt{var} \leftarrow \Gamma_z, \Gamma_{oc}$ as binaries
2: Update $\mathcal{M}$ by adding the constraints in Eq. (C9):

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} -\Gamma_x + \Gamma_z + \Gamma_{oc} \geq 0 \\ \Gamma_y - \Gamma_z + \Gamma_{oc} \geq 0 \\ \Gamma_x - \Gamma_y + \Gamma_{oc} \geq 0 \end{cases} \tag{C9}$$

3: Return $(\mathcal{M}, \Gamma_z, \Gamma_{oc})$

Algorithm 16 Model construction for f_a (`faModel`)

Input: Initial model $\mathcal{M}$ and 3 binary variables $(\Gamma_x, \Gamma_y, \Gamma_{ic})$
Input: Updated model $\mathcal{M}$ and 2 binary variables Γ_z, Γ_{oc}
1: Declare 2 binary variables $\mathcal{M}.\texttt{var} \leftarrow \Gamma_z, \Gamma_{oc}$ as binaries
2: Update $\mathcal{M}$ by adding the constraints in Eq. (C10)

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} -\Gamma_x + \Gamma_y - \Gamma_{ic} + \Gamma_z + \Gamma_{oc} \geq 0 \\ \Gamma_x - \Gamma_y + \Gamma_{ic} - \Gamma_z + \Gamma_{oc} \geq 0 \\ -\Gamma_x - \Gamma_y + \Gamma_{ic} + \Gamma_z + \Gamma_{oc} \geq 0 \\ -\Gamma_x + \Gamma_y + \Gamma_{ic} - \Gamma_z + \Gamma_{oc} \geq 0 \\ \Gamma_x - \Gamma_y - \Gamma_{ic} + \Gamma_z + \Gamma_{oc} \geq 0 \\ \Gamma_x + \Gamma_y - \Gamma_{ic} - \Gamma_z + \Gamma_{oc} \geq 0 \\ 4 - \Gamma_x - \Gamma_y - \Gamma_{ic} - \Gamma_z - \Gamma_{oc} \geq 0 \\ \Gamma_x + \Gamma_y + \Gamma_{ic} + \Gamma_z - \Gamma_{oc} \geq 0 \end{cases} \tag{C10}$$

3: Return $(\mathcal{M}, \Gamma_z, \Gamma_{oc})$

Algorithm 17 Model construction for f_1 (`f1Model`)

Input: Initial model $\mathcal{M}$, 4 binary variables $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic})$ and a binary flag $\lambda \in \{0, 1\}$.
Output: Updated model $\mathcal{M}$ and 5 binary variables $\Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q$

1: Declare 5 binary variables $\mathcal{M}.\texttt{var} \leftarrow \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q$ as binaries
2: Define a vector of variables as $\boldsymbol{v} = (\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q)$
3: Update $\mathcal{M}$ by adding the constraints in Eq. (C11):

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \underline{A}\boldsymbol{v} + \underline{\boldsymbol{\alpha}} \geq \boldsymbol{0}, & \text{if } \lambda = 0 \\ \overline{A}\boldsymbol{v} + \overline{\boldsymbol{\alpha}} \geq \boldsymbol{0}, & \text{if } \lambda = 1 \end{cases} \tag{C11}$$

where $(\underline{A}, \underline{\boldsymbol{\alpha}})$ and $(\overline{A}, \overline{\boldsymbol{\alpha}})$ are defined in Eq. (C13) and Eq. (C14) respectively.
4: Return $(\mathcal{M}, \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q)$

Algorithm 18 Model construction for f_2 (`f2Model`)

Input: Initial model $\mathcal{M}$ and 5 binary variables $(\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_{id})$
Output: Updated model $\mathcal{M}$ and 5 binary variables $\Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q$

1: Declare 5 variables $\mathcal{M}.\texttt{var} \leftarrow \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q$ as binaries
2: Define a vector of variables as $\boldsymbol{v} = (\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_{id}, \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q)$
3: Update $\mathcal{M}$ by adding the constraints in Eq. (C12)

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} \underline{B}\boldsymbol{v} + \underline{\boldsymbol{\beta}} \geq \boldsymbol{0}, & \text{if } \lambda = 0 \\ \overline{B}\boldsymbol{v} + \overline{\boldsymbol{\beta}} \geq \boldsymbol{0}, & \text{if } \lambda = 1 \end{cases} \tag{C12}$$

where $(\underline{B}, \underline{\boldsymbol{\beta}})$ and $(\overline{B}, \overline{\boldsymbol{\beta}})$ are defined in Eq. (C15) and Eq. (C16) respectively.
4: Return $(\mathcal{M}, \Gamma_z, \Gamma_{oc}, \Gamma_{od}, p, q)$

Table C1: The available linear masks of f_1. We define $\boldsymbol{\Gamma} = (\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$

No.	$\boldsymbol{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$	No.	$\boldsymbol{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$
1	(0,0,0,0,0,0,0)	1	0	0	0	0	50	(1,1,1,1,0,0,1)	0.125	1	1	1	1
2	(1,1,1,1,1,0,0)	1	0	0	0	0	51	(0,0,0,0,1,0,1)	0.125	1	1	1	1
3	(0,0,0,0,0,1,0)	0.25	0	1	0	1	52	(1,0,0,0,1,0,1)	0.125	1	1	1	1
4	(1,0,0,0,0,1,0)	0.25	0	1	0	1	53	(0,1,0,0,1,0,1)	0.125	1	1	1	1
5	(0,1,0,0,0,1,0)	0.25	0	1	0	1	54	(1,1,0,0,1,0,1)	0.125	1	1	1	1
6	(1,1,0,0,0,1,0)	0.25	0	1	0	1	55	(0,0,1,0,1,0,1)	0.125	1	1	1	1
7	(0,0,1,0,0,1,0)	0.25	0	1	0	1	56	(1,0,1,0,1,0,1)	0.125	1	1	1	1
8	(1,0,1,0,0,1,0)	0.25	0	1	0	1	57	(0,1,1,0,1,0,1)	0.125	1	1	1	1
9	(0,1,1,0,0,1,0)	0.25	0	1	0	1	58	(1,1,1,0,1,0,1)	0.125	1	1	1	1
10	(1,1,1,0,0,1,0)	0.25	0	1	0	1	59	(0,0,0,1,1,0,1)	0.125	1	1	1	1
11	(0,0,0,1,0,1,0)	0.25	0	1	0	1	60	(1,0,0,1,1,0,1)	0.125	1	1	1	1
12	(1,0,0,1,0,1,0)	0.25	0	1	0	1	61	(0,1,0,1,1,0,1)	0.125	1	1	1	1
13	(0,1,0,1,0,1,0)	0.25	0	1	0	1	62	(1,1,0,1,1,0,1)	0.125	1	1	1	1
14	(1,1,0,1,0,1,0)	0.25	0	1	0	1	63	(0,0,1,1,1,0,1)	0.125	1	1	1	1
15	(0,0,1,1,0,1,0)	0.25	0	1	0	1	64	(1,0,1,1,1,0,1)	0.125	1	1	1	1
16	(1,0,1,1,0,1,0)	0.25	0	1	0	1	65	(0,1,1,1,1,0,1)	0.125	1	1	1	1
17	(0,1,1,1,0,1,0)	0.25	0	1	0	1	66	(1,1,1,1,1,0,1)	0.875	0	0	0	0
18	(1,1,1,1,0,1,0)	0.25	0	1	0	1	67	(0,0,0,0,0,1,1)	0.375	1	0	1	0
19	(0,0,0,0,1,1,0)	0.25	0	1	0	1	68	(1,0,0,0,0,1,1)	0.375	1	0	1	0
20	(1,0,0,0,1,1,0)	0.25	0	1	0	1	69	(0,1,0,0,0,1,1)	0.375	1	0	1	0
21	(0,1,0,0,1,1,0)	0.25	0	1	0	1	70	(1,1,0,0,0,1,1)	0.125	1	1	1	1
22	(1,1,0,0,1,1,0)	0.25	0	1	0	1	71	(0,0,1,0,0,1,1)	0.375	1	0	1	0
23	(0,0,1,0,1,1,0)	0.25	0	1	0	1	72	(1,0,1,0,0,1,1)	0.125	1	1	1	1
24	(1,0,1,0,1,1,0)	0.25	0	1	0	1	73	(0,1,1,0,0,1,1)	0.125	1	1	1	1
25	(0,1,1,0,1,1,0)	0.25	0	1	0	1	74	(1,1,1,0,0,1,1)	0.125	1	1	1	1
26	(1,1,1,0,1,1,0)	0.25	0	1	0	1	75	(0,0,0,1,0,1,1)	0.375	1	0	1	0
27	(0,0,0,1,1,1,0)	0.25	0	1	0	1	76	(1,0,0,1,0,1,1)	0.125	1	1	1	1
28	(1,0,0,1,1,1,0)	0.25	0	1	0	1	77	(0,1,0,1,0,1,1)	0.125	1	1	1	1
29	(0,1,0,1,1,1,0)	0.25	0	1	0	1	78	(1,1,0,1,0,1,1)	0.125	1	1	1	1
30	(1,1,0,1,1,1,0)	0.25	0	1	0	1	79	(0,0,1,1,0,1,1)	0.125	1	1	1	1

Continued

No.	$\mathbf{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$	No.	$\mathbf{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$
31	(0,0,1,1,1,1,0)	0.25	0	1	0	1	80	(1,0,1,1,0,1,1)	0.125	1	1	1	1
32	(1,0,1,1,1,1,0)	0.25	0	1	0	1	81	(0,1,1,1,0,1,1)	0.125	1	1	1	1
33	(0,1,1,1,1,1,0)	0.25	0	1	0	1	82	(1,1,1,1,0,1,1)	0.375	1	0	1	0
34	(1,1,1,1,1,1,0)	0.25	0	1	0	1	83	(0,0,0,0,1,1,1)	0.375	1	0	1	0
35	(0,0,0,0,0,0,1)	0.875	0	0	0	0	84	(1,0,0,0,1,1,1)	0.125	1	1	1	1
36	(1,0,0,0,0,0,1)	0.125	1	1	1	1	85	(0,1,0,0,1,1,1)	0.125	1	1	1	1
37	(0,1,0,0,0,0,1)	0.125	1	1	1	1	86	(1,1,0,0,1,1,1)	0.125	1	1	1	1
38	(1,1,0,0,0,0,1)	0.125	1	1	1	1	87	(0,0,1,0,1,1,1)	0.125	1	1	1	1
39	(0,0,1,0,0,0,1)	0.125	1	1	1	1	88	(1,0,1,0,1,1,1)	0.125	1	1	1	1
40	(1,0,1,0,0,0,1)	0.125	1	1	1	1	89	(0,1,1,0,1,1,1)	0.125	1	1	1	1
41	(0,1,1,0,0,0,1)	0.125	1	1	1	1	90	(1,1,1,0,1,1,1)	0.375	1	0	1	0
42	(1,1,1,0,0,0,1)	0.125	1	1	1	1	91	(0,0,0,1,1,1,1)	0.125	1	1	1	1
43	(0,0,0,1,0,0,1)	0.125	1	1	1	1	92	(1,0,0,1,1,1,1)	0.125	1	1	1	1
44	(1,0,0,1,0,0,1)	0.125	1	1	1	1	93	(0,1,0,1,1,1,1)	0.125	1	1	1	1
45	(0,1,0,1,0,0,1)	0.125	1	1	1	1	94	(1,1,0,1,1,1,1)	0.375	1	0	1	0
46	(1,1,0,1,0,0,1)	0.125	1	1	1	1	95	(0,0,1,1,1,1,1)	0.125	1	1	1	1
47	(0,0,1,1,0,0,1)	0.125	1	1	1	1	96	(1,0,1,1,1,1,1)	0.375	1	0	1	0
48	(1,0,1,1,0,0,1)	0.125	1	1	1	1	97	(0,1,1,1,1,1,1)	0.375	1	0	1	0
49	(0,1,1,1,0,0,1)	0.125	1	1	1	1	98	(1,1,1,1,1,1,1)	0.375	1	0	1	0

Table C2: The available linear masks of f_2. We define $\mathbf{\Gamma} = (\Gamma_x, \Gamma_y, \Gamma_w, \Gamma_{ic}, \Gamma_{id}, \Gamma_z, \Gamma_{oc}, \Gamma_{od})$

No.	$\mathbf{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$	No.	$\mathbf{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$
1	(0,0,0,0,0,0,0,0)	1	0	0	0	0	62	(0,0,0,1,0,1,0,1)	0.125	1	1	1	1
2	(1,1,1,1,1,1,0,0)	1	0	0	0	0	63	(1,1,0,1,0,1,0,1)	0.125	1	1	1	1
3	(0,0,0,0,0,0,1,0)	0.25	0	1	0	1	64	(1,0,1,1,0,1,0,1)	0.125	1	1	1	1
4	(1,1,0,0,0,0,1,0)	0.25	0	1	0	1	65	(0,1,1,1,0,1,0,1)	0.125	1	1	1	1
5	(1,0,1,0,0,0,1,0)	0.25	0	1	0	1	66	(1,1,1,1,0,1,0,1)	0.25	0	1	0	1
6	(0,1,1,0,0,0,1,0)	0.25	0	1	0	1	67	(0,0,0,0,1,1,0,1)	0.125	1	1	1	1
7	(1,0,0,1,0,0,1,0)	0.25	0	1	0	1	68	(1,1,0,0,1,1,0,1)	0.125	1	1	1	1
8	(0,1,0,1,0,0,1,0)	0.25	0	1	0	1	69	(1,0,1,0,1,1,0,1)	0.125	1	1	1	1
9	(0,0,1,1,0,0,1,0)	0.25	0	1	0	1	70	(0,1,1,0,1,1,0,1)	0.125	1	1	1	1
10	(1,1,1,1,0,0,1,0)	0.25	0	1	0	1	71	(1,1,1,0,1,1,0,1)	0.25	0	1	0	1
11	(1,0,0,0,1,0,1,0)	0.25	0	1	0	1	72	(1,0,0,1,1,1,0,1)	0.125	1	1	1	1
12	(0,1,0,0,1,0,1,0)	0.25	0	1	0	1	73	(0,1,0,1,1,1,0,1)	0.125	1	1	1	1
13	(0,0,1,0,1,0,1,0)	0.25	0	1	0	1	74	(1,1,0,1,1,1,0,1)	0.25	0	1	0	1
14	(1,1,1,0,1,0,1,0)	0.25	0	1	0	1	75	(0,0,1,1,1,1,0,1)	0.125	1	1	1	1
15	(0,0,0,1,1,0,1,0)	0.25	0	1	0	1	76	(1,0,1,1,1,1,0,1)	0.25	0	1	0	1
16	(1,1,0,1,1,0,1,0)	0.25	0	1	0	1	77	(0,1,1,1,1,1,0,1)	0.25	0	1	0	1
17	(1,0,1,1,1,0,1,0)	0.25	0	1	0	1	78	(1,1,1,1,1,1,0,1)	0.625	0	0	1	0
18	(0,1,1,1,1,0,1,0)	0.25	0	1	0	1	79	(0,0,0,0,0,0,1,1)	0.625	0	0	1	0
19	(1,0,0,0,0,1,1,0)	0.25	0	1	0	1	80	(1,0,0,0,0,0,1,1)	0.25	0	1	0	1
20	(0,1,0,0,0,1,1,0)	0.25	0	1	0	1	81	(0,1,0,0,0,0,1,1)	0.25	0	1	0	1
21	(0,0,1,0,0,1,1,0)	0.25	0	1	0	1	82	(1,1,0,0,0,0,1,1)	0.125	1	1	1	1
22	(1,1,1,0,0,1,1,0)	0.25	0	1	0	1	83	(0,0,1,0,0,0,1,1)	0.25	0	1	0	1
23	(0,0,0,1,0,1,1,0)	0.25	0	1	0	1	84	(1,0,1,0,0,0,1,1)	0.125	1	1	1	1
24	(1,1,0,1,0,1,1,0)	0.25	0	1	0	1	85	(0,1,1,0,0,0,1,1)	0.125	1	1	1	1
25	(1,0,1,1,0,1,1,0)	0.25	0	1	0	1	86	(0,0,0,1,0,0,1,1)	0.25	0	1	0	1
26	(0,1,1,1,0,1,1,0)	0.25	0	1	0	1	87	(1,0,0,1,0,0,1,1)	0.125	1	1	1	1
27	(0,0,0,0,1,1,1,0)	0.25	0	1	0	1	88	(0,1,0,1,0,0,1,1)	0.125	1	1	1	1
28	(1,1,0,0,1,1,1,0)	0.25	0	1	0	1	89	(0,0,1,1,0,0,1,1)	0.125	1	1	1	1
29	(1,0,1,0,1,1,1,0)	0.25	0	1	0	1	90	(1,1,1,1,0,0,1,1)	0.125	1	1	1	1
30	(0,1,1,0,1,1,1,0)	0.25	0	1	0	1	91	(0,0,0,0,1,0,1,1)	0.25	0	1	0	1
31	(1,0,0,1,1,1,1,0)	0.25	0	1	0	1	92	(1,0,0,0,1,0,1,1)	0.125	1	1	1	1
32	(0,1,0,1,1,1,1,0)	0.25	0	1	0	1	93	(0,1,0,0,1,0,1,1)	0.125	1	1	1	1
33	(0,0,1,1,1,1,1,0)	0.25	0	1	0	1	94	(0,0,1,0,1,0,1,1)	0.125	1	1	1	1
34	(1,1,1,1,1,1,1,0)	0.25	0	1	0	1	95	(1,1,1,0,1,0,1,1)	0.125	1	1	1	1
35	(0,0,0,0,0,0,0,1)	0.625	0	0	1	0	96	(0,0,0,1,1,0,1,1)	0.125	1	1	1	1
36	(1,0,0,0,0,0,0,1)	0.25	0	1	0	1	97	(1,1,0,1,1,0,1,1)	0.125	1	1	1	1
37	(0,1,0,0,0,0,0,1)	0.25	0	1	0	1	98	(1,0,1,1,1,0,1,1)	0.125	1	1	1	1
38	(1,1,0,0,0,0,0,1)	0.125	1	1	1	1	99	(0,1,1,1,1,0,1,1)	0.125	1	1	1	1
39	(0,0,1,0,0,0,0,1)	0.25	0	1	0	1	100	(1,1,1,1,1,0,1,1)	0.25	0	1	0	1
40	(1,0,1,0,0,0,0,1)	0.125	1	1	1	1	101	(0,0,0,0,0,1,1,1)	0.25	0	1	0	1
41	(0,1,1,0,0,0,0,1)	0.125	1	1	1	1	102	(1,0,0,0,0,1,1,1)	0.125	1	1	1	1
42	(0,0,0,1,0,0,0,1)	0.25	0	1	0	1	103	(0,1,0,0,0,1,1,1)	0.125	1	1	1	1
43	(1,0,0,1,0,0,0,1)	0.125	1	1	1	1	104	(0,0,1,0,0,1,1,1)	0.125	1	1	1	1
44	(0,1,0,1,0,0,0,1)	0.125	1	1	1	1	105	(1,1,1,0,0,1,1,1)	0.125	1	1	1	1
45	(0,0,1,1,0,0,0,1)	0.125	1	1	1	1	106	(0,0,0,1,0,1,1,1)	0.125	1	1	1	1
46	(1,1,1,1,0,0,0,1)	0.125	1	1	1	1	107	(1,1,0,1,0,1,1,1)	0.125	1	1	1	1
47	(0,0,0,0,1,0,0,1)	0.25	0	1	0	1	108	(1,0,1,1,0,1,1,1)	0.125	1	1	1	1
48	(1,0,0,0,1,0,0,1)	0.125	1	1	1	1	109	(0,1,1,1,0,1,1,1)	0.125	1	1	1	1
49	(0,1,0,0,1,0,0,1)	0.125	1	1	1	1	110	(1,1,1,1,0,1,1,1)	0.25	0	1	0	1
50	(0,0,1,0,1,0,0,1)	0.125	1	1	1	1	111	(0,0,0,0,1,1,1,1)	0.125	1	1	1	1
51	(1,1,1,0,1,0,0,1)	0.125	1	1	1	1	112	(1,1,0,0,1,1,1,1)	0.125	1	1	1	1
52	(0,0,0,1,1,0,0,1)	0.125	1	1	1	1	113	(1,0,1,0,1,1,1,1)	0.125	1	1	1	1
53	(1,1,0,1,1,0,0,1)	0.125	1	1	1	1	114	(0,1,1,0,1,1,1,1)	0.125	1	1	1	1
54	(1,0,1,1,1,0,0,1)	0.125	1	1	1	1	115	(1,1,1,0,1,1,1,1)	0.25	0	1	0	1
55	(0,1,1,1,1,0,0,1)	0.125	1	1	1	1	116	(1,0,0,1,1,1,1,1)	0.125	1	1	1	1
56	(1,1,1,1,1,0,0,1)	0.25	0	1	0	1	117	(0,1,0,1,1,1,1,1)	0.125	1	1	1	1
57	(0,0,0,0,0,1,0,1)	0.25	0	1	0	1	118	(1,1,0,1,1,1,1,1)	0.25	0	1	0	1
58	(1,0,0,0,0,1,0,1)	0.125	1	1	1	1	119	(0,0,1,1,1,1,1,1)	0.125	1	1	1	1
59	(0,1,0,0,0,1,0,1)	0.125	1	1	1	1	120	(1,0,1,1,1,1,1,1)	0.25	0	1	0	1

Continued

No.	$\boldsymbol{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$	No.	$\boldsymbol{\Gamma}$	|Cor|	$\overline{p}$	$\overline{q}$	$\underline{p}$	$\underline{q}$
60	(0,0,1,0,0,1,0,1)	0.125	1	1	1	1	121	(0,1,1,1,1,1,1,1)	0.25	0	1	0	1
61	(1,1,1,0,0,1,0,1)	0.125	1	1	1	1	122	(1,1,1,1,1,1,1,1)	0.625	0	0	1	0

$$\underline{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \\ 1 & 1 & -4 & 1 & 1 & -1 & 0 & -1 & 4 \\ -1 & -1 & 4 & -1 & -1 & -1 & 0 & -1 & 4 \\ 2 & -1 & 2 & -1 & -1 & 0 & -2 & 2 & 3 \\ -2 & 1 & -2 & 1 & 1 & 0 & -2 & 2 & 3 \\ -1 & -1 & -1 & -1 & -1 & -1 & 0 & -2 & -1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 0 & -2 & -1 \\ 2 & 2 & -1 & -1 & -1 & 0 & -2 & 2 & 3 \\ -1 & -1 & 2 & 2 & -1 & 0 & -2 & 2 & 3 \\ 1 & -2 & -2 & 1 & 1 & 0 & -2 & 2 & 3 \\ -2 & 1 & 1 & -2 & 1 & 0 & -2 & 2 & 3 \end{pmatrix}, \quad \underline{\boldsymbol{\alpha}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 7 \\ 2 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{C13}$$

$$\overline{A} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ -1 & -1 & -1 & -1 & 4 & -1 & -6 & 5 & -1 \\ 1 & 1 & 1 & 1 & -4 & -1 & -6 & 5 & -1 \\ -2 & 2 & 0 & 2 & -2 & 1 & 0 & 1 & 3 \\ 0 & -2 & 2 & -2 & 2 & 1 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -1 \\ 2 & 0 & -2 & -2 & 2 & 1 & 0 & 1 & 3 \\ 2 & -2 & -2 & 2 & 0 & 1 & 0 & 1 & 3 \\ 2 & 2 & 2 & -4 & -4 & 1 & 0 & 1 & 5 \\ -1 & -1 & -1 & -1 & -1 & -1 & -3 & 1 & -2 \\ 1 & 1 & 1 & 1 & 1 & -1 & -3 & 1 & -2 \\ -2 & -2 & 1 & 1 & 1 & 0 & -1 & 1 & 2 \\ -1 & 2 & -1 & -1 & 2 & 0 & -1 & 1 & 2 \\ 4 & -2 & -2 & 4 & -2 & 1 & 0 & 1 & 5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \end{pmatrix}, \quad \overline{\boldsymbol{\alpha}} = \begin{pmatrix} 0 \\ 6 \\ 6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 8 \\ 3 \\ 2 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{C14}$$

$$\underline{B} = \begin{pmatrix} 5 & -1 & -1 & -1 & -1 & -1 & 0 & 1 & -1 & 4 \\ -5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & -1 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ -1 & -1 & 3 & -1 & 3 & -1 & 0 & -3 & 3 & 4 \\ 1 & 1 & -3 & 1 & -3 & 1 & 0 & -3 & 3 & 4 \\ -1 & -1 & 3 & 3 & -1 & -1 & 0 & -3 & 3 & 4 \\ -1 & -1 & -1 & 3 & 3 & -1 & 0 & -3 & 3 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & -1 & -1 & -2 \\ 1 & 1 & 1 & -3 & -3 & 1 & 0 & -3 & 3 & 4 \\ 1 & 1 & -3 & -3 & 1 & 1 & 0 & -3 & 3 & 4 \\ -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & -2 \\ 1 & -1 & 1 & -1 & 1 & -1 & 0 & -1 & 1 & 2 \\ -1 & 1 & 1 & -1 & 1 & -1 & 0 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & 1 & -1 & 0 & -1 & 1 & 2 \\ -1 & 1 & -1 & 1 & 1 & -1 & 0 & -1 & 1 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 & 0 & -1 & 1 & 2 \\ 1 & -1 & -1 & 1 & -1 & 1 & 0 & -1 & 1 & 2 \\ 1 & 1 & -1 & -1 & -1 & 1 & 0 & -1 & 1 & 2 \\ -1 & -1 & 1 & 1 & -1 & 1 & 0 & -1 & 1 & 2 \\ -1 & 1 & 1 & -1 & -1 & 1 & 0 & -1 & 1 & 2 \\ 1 & -1 & 1 & -1 & -1 & 1 & 0 & -1 & 1 & 2 \\ 1 & 1 & -1 & -1 & 1 & -1 & 0 & -1 & 1 & 2 \\ 1 & 1 & -1 & 1 & -1 & -1 & 0 & -1 & 1 & 2 \\ -1 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 2 \\ 1 & -1 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 2 \\ -1 & -1 & 1 & -1 & 1 & 1 & 0 & -1 & 1 & 2 \\ -1 & 1 & 1 & 1 & -1 & -1 & 0 & -1 & 1 & 2 \\ -1 & -1 & 1 & 1 & 1 & -1 & 0 & -1 & 1 & 2 \\ -1 & -1 & -1 & 1 & 1 & 1 & 0 & -1 & 1 & 2 \\ 1 & -1 & 1 & 1 & -1 & -1 & 0 & -1 & 1 & 2 \\ 1 & 1 & 1 & -1 & -1 & -1 & 0 & -1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & 1 & 0 & 1 & -1 & 0 \\ 1 & -1 & 1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 \\ 1 & 1 & 1 & -1 & 1 & 1 & 0 & 1 & -1 & 0 \\ 1 & 1 & 1 & 1 & 1 & -1 & 0 & 1 & -1 & 0 \\ 1 & 1 & -1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 \\ -1 & -1 & -1 & -1 & 5 & -1 & 0 & 1 & -1 & 4 \\ -1 & 5 & -1 & -1 & -1 & -1 & 0 & 1 & -1 & 4 \\ -1 & -1 & -1 & -1 & -1 & 5 & 0 & 1 & -1 & 4 \\ -1 & -1 & 1 & -1 & -1 & -1 & 0 & 1 & -1 & 0 \\ -1 & -1 & -1 & 5 & -1 & -1 & 0 & 1 & -1 & 4 \end{pmatrix}, \quad \underline{\boldsymbol{\beta}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4 \\ 0 \end{pmatrix} \tag{C15}$$

$$\overline{B} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
-1 & 5 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & 5 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 \\
1 & -5 & 1 & 1 & 1 & 1 & 0 & 0 & -1 & 5 \\
-1 & -1 & 3 & -1 & 3 & -1 & 0 & -3 & 3 & 1 \\
1 & -3 & -3 & 1 & 1 & 1 & 0 & -3 & 3 & 1 \\
3 & 3 & -1 & -1 & -1 & -1 & 0 & -3 & 3 & 1 \\
-3 & 1 & 1 & 1 & -3 & 1 & 0 & -3 & 3 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 \\
1 & -1 & 1 & 1 & -1 & -1 & 0 & -1 & 1 & 1 \\
-1 & 1 & -1 & 1 & 1 & -1 & 0 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & 0 & -1 & 1 & 1 \\
-1 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 1 \\
1 & 1 & -1 & -1 & 1 & -1 & 0 & 0 & 0 & 2 \\
1 & 1 & -1 & 1 & -1 & -1 & 0 & 0 & 0 & 2 \\
-1 & -1 & 1 & -1 & 1 & 1 & 0 & 0 & 0 & 2 \\
1 & 1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 2 \\
-1 & -1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 2 \\
1 & -1 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 2 \\
-1 & 1 & 1 & 1 & -1 & -1 & 0 & -1 & 1 & 1 \\
-1 & -1 & 1 & 1 & 1 & -1 & 0 & -1 & 1 & 1 \\
1 & -1 & -1 & 1 & 1 & -1 & 0 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 0 & -1 & 1 & 1 \\
-1 & 1 & -1 & 1 & -1 & 1 & 0 & -1 & 1 & 1 \\
-1 & 1 & 1 & -1 & 1 & -1 & 0 & -1 & 1 & 1 \\
-1 & 1 & -1 & 1 & -1 & 1 & 0 & -1 & 1 & 1 \\
1 & -1 & -1 & 1 & -1 & -1 & 0 & -1 & 1 & 1 \\
1 & 1 & -1 & -1 & -1 & 1 & 0 & -1 & 1 & 1 \\
-1 & -1 & -1 & 1 & 1 & 1 & 0 & -1 & 1 & 1 \\
-1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & 1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
-1 & -1 & -1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 & 1 \\
-1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & -1 & 0 & 1 & -1 & 0 \\
-1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 \\
1 & -1 & 1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 \\
1 & 1 & 1 & 1 & -1 & 1 & 0 & 1 & -1 & 0 \\
1 & 1 & -1 & 1 & 1 & 1 & 0 & 1 & -1 & 0 \\
1 & 1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & 1
\end{pmatrix}, \quad \overline{\boldsymbol{\beta}} = \begin{pmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 5 \\ 3 \\ 5 \\ 1 \\ 7 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 4 \\ 4 \\ 4 \\ 4 \\ 4 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix} \tag{C16}$$

Appendix D Truncated Linear MILP Model Construction of Bytewise Breakdown Functions

This part describes the truncated linear propagation model construction process of h_b, f_b, h_{b2} and f_{b2} functions.

The available input-output truncated linear masks for h_b and f_b are given in Table D3 and Table D4. The MILP model are constructed accordingly as Algorithm 21 and Algorithm 22. Note that the entries in Table 2 for h_a are identical to those in Table D3 for h_b. Therefore, we can directly use Algorithm 15 as an subroutine of Algorithm 21.

For h_{b2}, all 114 available $(T_x, T_y, T_w, T_z, T_{oc}, T_{od_0}, T_{od_1})$'s are listed in Table D5 and the MILP model can be constructed with Algorithm 19. As to f_{b2}, the 905 $(T_x, T_y, T_{ic}, T_{id_0}, T_{id_1}, T_z, T_{oc}, T_{od_0}, T_{od_1})$'s in Table D6 can be modeled with Algorithm 19.

Table D3: The available input-output truncated linear masks for h_b.

No.	(T_x, T_y, T_z, T_{oc})
1	(0,0,0,0)
2	(1,1,1,0)
3	(0,0,0,1)
4	(1,0,0,1)
5	(0,1,0,1)
6	(1,1,0,1)
7	(0,0,1,1)
8	(1,0,1,1)
9	(0,1,1,1)
10	(1,1,1,1)

Table D4: The available input-output truncated linear masks for f_b

No.	$(T_x, T_y, T_{ic}, T_z, T_{oc})$
1	(0,0,0,0,0)
2	(1,1,0,1,0)
3	(1,1,1,1,0)
4	(1,0,0,0,1)
5	(0,1,0,0,1)
6	(1,1,0,0,1)
7	(0,0,1,0,1)
8	(1,1,1,0,1)
9	(0,0,0,1,1)
10	(1,0,0,1,1)
11	(0,1,0,1,1)
12	(1,1,0,1,1)
13	(1,0,1,1,1)
14	(0,1,1,1,1)
15	(1,1,1,1,1)

Algorithm 19 Truncated Linear Model construction for h_{b2} (`hb2Model`)

Input: Initial model $\mathcal{M}$ and 3 binary variables (T_x, T_y, T_w)
Output: Updated model $\mathcal{M}$ and 5 binary variables $(T_z, T_{oc}, T_{od_0}, T_{od_1}, \tau)$

Declare 4 variables $\mathcal{M}.\texttt{con} \leftarrow T_z, T_{oc}, T_{od_0}, T_{od_1}$ as binaries
Define a vector of variables as $\boldsymbol{v} = (T_x, T_y, T_w, T_z, T_{oc}, T_{od_0}, T_{od_1})$
Update $\mathcal{M}$ by adding the following constraint Eq. (D17)

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} T_z + T_{oc} + -T_{od_0} + T_{od_1} \geq 0 \\ T_w - T_z + T_{oc} + T_{od_1} \geq 0 \\ -T_y + T_w + T_{oc} + T_{od_1} \geq 0 \\ T_x - T_w + T_{oc} + T_{od_1} \geq 0 \\ -T_x + T_z + T_{oc} + T_{od_1} \geq 0 \end{cases} \tag{D17}$$

Declare 1 binary variable $\mathcal{M}.\texttt{var} \leftarrow \tau$
Define set $\mathcal{S} = \{T_x, T_y, T_w, T_z, T_{oc}, T_{od_0}, T_{od_1}\}$
for $s \in \mathcal{S}$ **do**
    Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \geq s$
end for
Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \leq \sum_{s \in \mathcal{S}} s$
Return $(\mathcal{M}, T_z, T_{oc}, \tau)$

Table D5: The available input-output truncated linear masks for h_{b2}. We define $\boldsymbol{T} = (T_x, T_y, T_w, T_z, T_{oc}, T_{od_0}, T_{od_1})$ and $|\log \underline{\texttt{Cor}}| = \max\{\underline{p} + 2\underline{q}\}$

No.	$\boldsymbol{T}$	$\lvert\log \underline{\texttt{Cor}}\rvert$	No.	$\boldsymbol{T}$	$\lvert\log \underline{\texttt{Cor}}\rvert$	No.	$\boldsymbol{T}$	$\lvert\log \underline{\texttt{Cor}}\rvert$
1	(0,0,0,0,0,0,0)	0	39	(0,0,1,0,1,1,0)	21	77	(0,1,0,1,1,0,1)	21
2	(1,1,1,1,0,0,0)	18	40	(1,0,1,0,1,1,0)	21	78	(1,1,0,1,1,0,1)	21
3	(0,0,0,0,1,0,0)	17	41	(0,1,1,0,1,1,0)	21	79	(0,0,1,1,1,0,1)	21
4	(1,0,0,0,1,0,0)	20	42	(1,1,1,0,1,1,0)	21	80	(1,0,1,1,1,0,1)	21
5	(0,1,0,0,1,0,0)	20	43	(0,0,0,1,1,1,0)	21	81	(0,1,1,1,1,0,1)	21

Continued

No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$	No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$	No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$
6	(1,1,0,0,1,0,0)	20	44	(1,0,0,1,1,1,0)	21	82	(1,1,1,1,1,0,1)	21
7	(0,0,1,0,1,0,0)	20	45	(0,1,0,1,1,1,0)	21	83	(0,0,0,0,0,1,1)	19
8	(1,0,1,0,1,0,0)	20	46	(1,1,0,1,1,1,0)	21	84	(1,0,0,0,0,1,1)	22
9	(0,1,1,0,1,0,0)	20	47	(0,0,1,1,1,1,0)	21	85	(0,1,0,0,0,1,1)	22
10	(1,1,1,0,1,0,0)	20	48	(1,0,1,1,1,1,0)	21	86	(1,1,0,0,0,1,1)	22
11	(0,0,0,1,1,0,0)	20	49	(0,1,1,1,1,1,0)	21	87	(0,0,1,0,0,1,1)	22
12	(1,0,0,1,1,0,0)	20	50	(1,1,1,1,1,1,0)	21	88	(1,0,1,0,0,1,1)	22
13	(0,1,0,1,1,0,0)	20	51	(0,0,0,0,0,0,1)	18	89	(0,1,1,0,0,1,1)	22
14	(1,1,0,1,1,0,0)	20	52	(1,0,0,0,0,0,1)	21	90	(1,1,1,0,0,1,1)	22
15	(0,0,1,1,1,0,0)	20	53	(0,1,0,0,0,0,1)	21	91	(0,0,0,1,0,1,1)	22
16	(1,0,1,1,1,0,0)	20	54	(1,1,0,0,0,0,1)	21	92	(1,0,0,1,0,1,1)	22
17	(0,1,1,1,1,0,0)	20	55	(0,0,1,0,0,0,1)	21	93	(0,1,0,1,0,1,1)	22
18	(1,1,1,1,1,0,0)	20	56	(1,0,1,0,0,0,1)	21	94	(1,1,0,1,0,1,1)	22
19	(0,0,0,0,0,1,0)	15	57	(0,1,1,0,0,0,1)	21	95	(0,0,1,1,0,1,1)	22
20	(1,0,0,0,0,1,0)	18	58	(1,1,1,0,0,0,1)	21	96	(1,0,1,1,0,1,1)	22
21	(0,1,0,0,0,1,0)	18	59	(0,0,0,1,0,0,1)	21	97	(0,1,1,1,0,1,1)	22
22	(1,1,0,0,0,1,0)	18	60	(1,0,0,1,0,0,1)	21	98	(1,1,1,1,0,1,1)	22
23	(0,0,1,0,0,1,0)	18	61	(0,1,0,1,0,0,1)	21	99	(0,0,0,0,1,1,1)	19
24	(1,0,1,0,0,1,0)	18	62	(1,1,0,1,0,0,1)	21	100	(1,0,0,0,1,1,1)	22
25	(0,1,1,0,0,1,0)	18	63	(0,0,1,1,0,0,1)	21	101	(0,1,0,0,1,1,1)	22
26	(1,1,1,0,0,1,0)	18	64	(1,0,1,1,0,0,1)	21	102	(1,1,0,0,1,1,1)	22
27	(0,0,0,1,0,1,0)	18	65	(0,1,1,1,0,0,1)	21	103	(0,0,1,0,1,1,1)	22
28	(1,0,0,1,0,1,0)	18	66	(1,1,1,1,0,0,1)	21	104	(1,0,1,0,1,1,1)	22
29	(0,1,0,1,0,1,0)	18	67	(0,0,0,0,1,0,1)	18	105	(0,1,1,0,1,1,1)	22
30	(1,1,0,1,0,1,0)	18	68	(1,0,0,0,1,0,1)	21	106	(1,1,1,0,1,1,1)	22
31	(0,0,1,1,0,1,0)	18	69	(0,1,0,0,1,0,1)	21	107	(0,0,0,1,1,1,1)	22
32	(1,0,1,1,0,1,0)	18	70	(1,1,0,0,1,0,1)	21	108	(1,0,0,1,1,1,1)	22
33	(0,1,1,1,0,1,0)	18	71	(0,0,1,0,1,0,1)	21	109	(0,1,0,1,1,1,1)	22
34	(1,1,1,1,0,1,0)	19	72	(1,0,1,0,1,0,1)	21	110	(1,1,0,1,1,1,1)	22
35	(0,0,0,0,1,1,0)	18	73	(0,1,1,0,1,0,1)	21	111	(0,0,1,1,1,1,1)	22
36	(1,0,0,0,1,1,0)	21	74	(1,1,1,0,1,0,1)	21	112	(1,0,1,1,1,1,1)	22
37	(0,1,0,0,1,1,0)	21	75	(0,0,0,1,1,0,1)	21	113	(0,1,1,1,1,1,1)	22
38	(1,1,0,0,1,1,0)	21	76	(1,0,0,1,1,0,1)	21	114	(1,1,1,1,1,1,1)	22

Table D6: The available input-output truncated linear masks for f_{b2} and the logarithm of the correlation lower bounds. The lines in **bold** satisfies the $(T_{oc}, T_{od_0}, T_{od_1}) = (0,0,0)$: this is required by Proposition 2 and should be satisfied by the final f_{b2} call in Eq. (14). We define $\boldsymbol{T} = (T_x, T_y, T_{ic}, T_{id_0}, T_{id_1}, T_z, T_{oc}, T_{od_0}, T_{od_1})$ and $\lvert \log \underline{\mathtt{Cor}} \rvert = \max\{\underline{p} + 2\underline{q}\}$

No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$	No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$	No.	$\boldsymbol{T}$	$\lvert \log \underline{\mathtt{Cor}} \rvert$
1	**(0,0,0,0,0,0,0,0,0,0)**	**0**	303	(1,0,1,0,0,1,0,1,1,0)	23	605	(1,1,0,0,1,0,1,1,0,1)	23
2	**(1,1,1,0,0,0,1,0,0,0)**	**20**	304	(0,1,1,0,0,1,0,1,1,0)	23	606	(0,0,1,0,1,0,1,1,0,1)	23
3	**(1,1,1,1,0,0,1,0,0,0)**	**20**	305	(1,1,1,0,0,1,0,1,1,0)	23	607	(1,0,1,0,1,0,1,1,0,1)	23
4	**(1,1,1,0,1,0,1,0,0,0)**	**20**	306	(0,0,0,1,0,1,0,1,1,0)	22	608	(0,1,1,0,1,0,1,1,0,1)	23
5	**(1,1,1,1,1,0,1,0,0,0)**	**20**	307	(1,0,0,1,0,1,0,1,1,0)	23	609	(1,1,1,0,1,0,1,1,0,1)	23
6	**(1,1,1,0,0,1,1,0,0,0)**	**20**	308	(0,1,0,1,0,1,0,1,1,0)	23	610	(0,0,0,1,1,0,1,1,0,1)	23
7	**(1,1,1,1,0,1,1,0,0,0)**	**20**	309	(1,1,0,1,0,1,0,1,1,0)	23	611	(1,0,0,1,1,0,1,1,0,1)	23
8	**(1,1,1,0,1,1,1,0,0,0)**	**20**	310	(0,0,1,1,0,1,0,1,1,0)	23	612	(0,1,0,1,1,0,1,1,0,1)	23
9	**(1,1,1,1,1,1,1,0,0,0)**	**20**	311	(1,0,1,1,0,1,0,1,1,0)	23	613	(1,1,0,1,1,0,1,1,0,1)	23
10	(0,0,0,0,0,0,0,1,0,0)	19	312	(0,1,1,1,0,1,0,1,1,0)	23	614	(0,0,1,1,1,0,1,1,0,1)	23
11	(1,0,0,0,0,0,0,1,0,0)	21	313	(1,1,1,1,0,1,0,1,1,0)	23	615	(1,0,1,1,1,0,1,1,0,1)	23
12	(0,1,0,0,0,0,0,1,0,0)	21	314	(0,0,0,0,1,1,0,1,1,0)	22	616	(0,1,1,1,1,0,1,1,0,1)	23
13	(1,1,0,0,0,0,0,1,0,0)	22	315	(1,0,0,0,1,1,0,1,1,0)	23	617	(1,1,1,1,1,0,1,1,0,1)	23
14	(0,0,1,0,0,0,0,1,0,0)	21	316	(0,1,0,0,1,1,0,1,1,0)	23	618	(0,0,0,0,0,1,1,1,0,1)	22
15	(1,0,1,0,0,0,0,1,0,0)	22	317	(1,1,0,0,1,1,0,1,1,0)	23	619	(1,0,0,0,0,1,1,1,0,1)	23
16	(0,1,1,0,0,0,0,1,0,0)	22	318	(0,0,1,0,1,1,0,1,1,0)	23	620	(0,1,0,0,0,1,1,1,0,1)	23
17	(1,1,1,0,0,0,0,1,0,0)	22	319	(1,0,1,0,1,1,0,1,1,0)	23	621	(1,1,0,0,0,1,1,1,0,1)	23
18	(0,0,0,1,0,0,0,1,0,0)	20	320	(0,1,1,0,1,1,0,1,1,0)	23	622	(0,0,1,0,0,1,1,1,0,1)	23
19	(1,0,0,1,0,0,0,1,0,0)	22	321	(1,1,1,0,1,1,0,1,1,0)	23	623	(1,0,1,0,0,1,1,1,0,1)	23
20	(0,1,0,1,0,0,0,1,0,0)	22	322	(0,0,0,1,1,1,0,1,1,0)	23	624	(0,1,1,0,0,1,1,1,0,1)	23
21	(1,1,0,1,0,0,0,1,0,0)	22	323	(1,0,0,1,1,1,0,1,1,0)	23	625	(1,1,1,0,0,1,1,1,0,1)	23
22	(0,0,1,1,0,0,0,1,0,0)	22	324	(0,1,0,1,1,1,0,1,1,0)	23	626	(0,0,0,1,0,1,1,1,0,1)	23
23	(1,0,1,1,0,0,0,1,0,0)	22	325	(1,1,0,1,1,1,0,1,1,0)	23	627	(1,0,0,1,0,1,1,1,0,1)	23
24	(0,1,1,1,0,0,0,1,0,0)	22	326	(0,0,1,1,1,1,0,1,1,0)	23	628	(0,1,0,1,0,1,1,1,0,1)	23
25	(1,1,1,1,0,0,0,1,0,0)	22	327	(1,0,1,1,1,1,0,1,1,0)	23	629	(1,1,0,1,0,1,1,1,0,1)	23
26	(0,0,0,0,1,0,0,1,0,0)	20	328	(0,1,1,1,1,1,0,1,1,0)	23	630	(0,0,1,1,0,1,1,1,0,1)	23
27	(1,0,0,0,1,0,0,1,0,0)	22	329	(1,1,1,1,1,1,0,1,1,0)	23	631	(1,0,1,1,0,1,1,1,0,1)	23
28	(0,1,0,0,1,0,0,1,0,0)	22	330	(0,0,0,0,0,0,1,1,1,0)	22	632	(0,1,1,1,0,1,1,1,0,1)	23
29	(1,1,0,0,1,0,0,1,0,0)	22	331	(1,0,0,0,0,0,1,1,1,0)	23	633	(1,1,1,1,0,1,1,1,0,1)	23
30	(0,0,1,0,1,0,0,1,0,0)	22	332	(0,1,0,0,0,0,1,1,1,0)	23	634	(0,0,0,0,1,1,1,1,0,1)	23
31	(1,0,1,0,1,0,0,1,0,0)	22	333	(1,1,0,0,0,0,1,1,1,0)	23	635	(1,0,0,0,1,1,1,1,0,1)	23
32	(0,1,1,0,1,0,0,1,0,0)	22	334	(0,0,1,0,0,0,1,1,1,0)	23	636	(0,1,0,0,1,1,1,1,0,1)	23
33	(1,1,1,0,1,0,0,1,0,0)	22	335	(1,0,1,0,0,0,1,1,1,0)	23	637	(1,1,0,0,1,1,1,1,0,1)	23
34	(0,0,0,1,1,0,0,1,0,0)	21	336	(0,1,1,0,0,0,1,1,1,0)	23	638	(0,0,1,0,1,1,1,1,0,1)	23
35	(1,0,0,1,1,0,0,1,0,0)	22	337	(1,1,1,0,0,0,1,1,1,0)	23	639	(1,0,1,0,1,1,1,1,0,1)	23
36	(0,1,0,1,1,0,0,1,0,0)	22	338	(0,0,0,1,0,0,1,1,1,0)	23	640	(0,1,1,0,1,1,1,1,0,1)	23
37	(1,1,0,1,1,0,0,1,0,0)	22	339	(1,0,0,1,0,0,1,1,1,0)	23	641	(1,1,1,0,1,1,1,1,0,1)	23
38	(0,0,1,1,1,0,0,1,0,0)	22	340	(0,1,0,1,0,0,1,1,1,0)	23	642	(0,0,0,1,1,1,1,1,0,1)	23
39	(1,0,1,1,1,0,0,1,0,0)	22	341	(1,1,0,1,0,0,1,1,1,0)	23	643	(1,0,0,1,1,1,1,1,0,1)	23

Continued

No.	T	log Cor	No.	T	log Cor	No.	T	log Cor
40	(0,1,1,1,1,0,0,1,0,0)	22	342	(0,0,1,1,0,0,1,1,1,0)	23	644	(0,1,0,1,1,1,1,1,0,1)	23
41	(1,1,1,1,1,0,0,1,0,0)	22	343	(1,0,1,1,0,0,1,1,1,0)	23	645	(1,1,0,1,1,1,1,1,0,1)	23
42	(0,0,0,0,0,1,0,1,0,0)	20	344	(0,1,1,1,0,0,1,1,1,0)	23	646	(0,0,1,1,1,1,1,1,0,1)	23
43	(1,0,0,0,0,1,0,1,0,0)	21	345	(1,1,1,1,0,0,1,1,1,0)	23	647	(1,0,1,1,1,1,1,1,0,1)	23
44	(0,1,0,0,0,1,0,1,0,0)	21	346	(0,0,0,0,1,0,1,1,1,0)	23	648	(0,1,1,1,1,1,1,1,0,1)	23
45	(1,1,0,0,0,1,0,1,0,0)	22	347	(1,0,0,0,1,0,1,1,1,0)	23	649	(1,1,1,1,1,1,1,1,0,1)	23
46	(0,0,1,0,0,1,0,1,0,0)	21	348	(0,1,0,0,1,0,1,1,1,0)	23	650	(0,0,0,0,0,0,0,0,1,1)	21
47	(1,0,1,0,0,1,0,1,0,0)	22	349	(1,1,0,0,1,0,1,1,1,0)	23	651	(1,0,0,0,0,0,0,0,1,1)	23
48	(0,1,1,0,0,1,0,1,0,0)	22	350	(0,0,1,0,1,0,1,1,1,0)	23	652	(0,1,0,0,0,0,0,0,1,1)	23
49	(1,1,1,0,0,1,0,1,0,0)	22	351	(1,0,1,0,1,0,1,1,1,0)	23	653	(1,1,0,0,0,0,0,0,1,1)	24
50	(0,0,0,1,0,1,0,1,0,0)	21	352	(0,1,1,0,1,0,1,1,1,0)	23	654	(0,0,1,0,0,0,0,0,1,1)	23
51	(1,0,0,1,0,1,0,1,0,0)	22	353	(1,1,1,0,1,0,1,1,1,0)	23	655	(1,0,1,0,0,0,0,0,1,1)	24
52	(0,1,0,1,0,1,0,1,0,0)	22	354	(0,0,0,1,1,0,1,1,1,0)	23	656	(0,1,1,0,0,0,0,0,1,1)	24
53	(1,1,0,1,0,1,0,1,0,0)	22	355	(1,0,0,1,1,0,1,1,1,0)	23	657	(1,1,1,0,0,0,0,0,1,1)	24
54	(0,0,1,1,0,1,0,1,0,0)	22	356	(0,1,0,1,1,0,1,1,1,0)	23	658	(0,0,0,1,0,0,0,0,1,1)	22
55	(1,0,1,1,0,1,0,1,0,0)	22	357	(1,1,0,1,1,0,1,1,1,0)	23	659	(1,0,0,1,0,0,0,0,1,1)	24
56	(0,1,1,1,0,1,0,1,0,0)	22	358	(0,0,1,1,1,0,1,1,1,0)	23	660	(0,1,0,1,0,0,0,0,1,1)	24
57	(1,1,1,1,0,1,0,1,0,0)	22	359	(1,0,1,1,1,0,1,1,1,0)	23	661	(1,1,0,1,0,0,0,0,1,1)	24
58	(0,0,0,0,1,1,0,1,0,0)	21	360	(0,1,1,1,1,0,1,1,1,0)	23	662	(0,0,1,1,0,0,0,0,1,1)	24
59	(1,0,0,0,1,1,0,1,0,0)	22	361	(1,1,1,1,1,0,1,1,1,0)	23	663	(1,0,1,1,0,0,0,0,1,1)	24
60	(0,1,0,0,1,1,0,1,0,0)	22	362	(0,0,0,0,0,1,1,1,1,0)	22	664	(0,1,1,1,0,0,0,0,1,1)	24
61	(1,1,0,0,1,1,0,1,0,0)	22	363	(1,0,0,0,0,1,1,1,1,0)	23	665	(1,1,1,1,0,0,0,0,1,1)	24
62	(0,0,1,0,1,1,0,1,0,0)	22	364	(0,1,0,0,0,1,1,1,1,0)	23	666	(0,0,0,0,1,0,0,0,1,1)	22
63	(1,0,1,0,1,1,0,1,0,0)	22	365	(1,1,0,0,0,1,1,1,1,0)	23	667	(1,0,0,0,1,0,0,0,1,1)	24
64	(0,1,1,0,1,1,0,1,0,0)	22	366	(0,0,1,0,0,1,1,1,1,0)	23	668	(0,1,0,0,1,0,0,0,1,1)	24
65	(1,1,1,0,1,1,0,1,0,0)	22	367	(1,0,1,0,0,1,1,1,1,0)	23	669	(1,1,0,0,1,0,0,0,1,1)	24
66	(0,0,0,1,1,1,0,1,0,0)	22	368	(0,1,1,0,0,1,1,1,1,0)	23	670	(0,0,1,0,1,0,0,0,1,1)	24
67	(1,0,0,1,1,1,0,1,0,0)	22	369	(1,1,1,0,0,1,1,1,1,0)	23	671	(1,0,1,0,1,0,0,0,1,1)	24
68	(0,1,0,1,1,1,0,1,0,0)	22	370	(0,0,0,1,0,1,1,1,1,0)	23	672	(0,1,1,0,1,0,0,0,1,1)	24
69	(1,1,0,1,1,1,0,1,0,0)	22	371	(1,0,0,1,0,1,1,1,1,0)	23	673	(1,1,1,0,1,0,0,0,1,1)	24
70	(0,0,1,1,1,1,0,1,0,0)	22	372	(0,1,0,1,0,1,1,1,1,0)	23	674	(0,0,0,1,1,0,0,0,1,1)	23
71	(1,0,1,1,1,1,0,1,0,0)	22	373	(1,1,0,1,0,1,1,1,1,0)	23	675	(1,0,0,1,1,0,0,0,1,1)	24
72	(0,1,1,1,1,1,0,1,0,0)	22	374	(0,0,1,1,0,1,1,1,1,0)	23	676	(0,1,0,1,1,0,0,0,1,1)	24
73	(1,1,1,1,1,1,0,1,0,0)	22	375	(1,0,1,1,0,1,1,1,1,0)	23	677	(1,1,0,1,1,0,0,0,1,1)	24
74	(0,0,0,0,0,0,1,1,0,0)	21	376	(0,1,1,1,0,1,1,1,1,0)	23	678	(0,0,1,1,1,0,0,0,1,1)	24
75	(1,0,0,0,0,0,1,1,0,0)	22	377	(1,1,1,1,0,1,1,1,1,0)	23	679	(1,0,1,1,1,0,0,0,1,1)	24
76	(0,1,0,0,0,0,1,1,0,0)	22	378	(0,0,0,0,1,1,1,1,1,0)	23	680	(0,1,1,1,1,0,0,0,1,1)	24
77	(1,1,0,0,0,0,1,1,0,0)	22	379	(1,0,0,0,1,1,1,1,1,0)	23	681	(1,1,1,1,1,0,0,0,1,1)	24
78	(0,0,1,0,0,0,1,1,0,0)	22	380	(0,1,0,0,1,1,1,1,1,0)	23	682	(0,0,0,0,0,1,0,0,1,1)	22
79	(1,0,1,0,0,0,1,1,0,0)	22	381	(1,1,0,0,1,1,1,1,1,0)	23	683	(1,0,0,0,0,1,0,0,1,1)	23
80	(0,1,1,0,0,0,1,1,0,0)	22	382	(0,0,1,0,1,1,1,1,1,0)	23	684	(0,1,0,0,0,1,0,0,1,1)	23
81	(1,1,1,0,0,0,1,1,0,0)	22	383	(1,0,1,0,1,1,1,1,1,0)	23	685	(1,1,0,0,0,1,0,0,1,1)	24
82	(0,0,0,1,0,0,1,1,0,0)	22	384	(0,1,1,0,1,1,1,1,1,0)	23	686	(0,0,1,0,0,1,0,0,1,1)	23
83	(1,0,0,1,0,0,1,1,0,0)	22	385	(1,1,1,0,1,1,1,1,1,0)	23	687	(1,0,1,0,0,1,0,0,1,1)	24
84	(0,1,0,1,0,0,1,1,0,0)	22	386	(0,0,0,1,1,1,1,1,1,0)	23	688	(0,1,1,0,0,1,0,0,1,1)	24
85	(1,1,0,1,0,0,1,1,0,0)	22	387	(1,0,0,1,1,1,1,1,1,0)	23	689	(1,1,1,0,0,1,0,0,1,1)	24
86	(0,0,1,1,0,0,1,1,0,0)	22	388	(0,1,0,1,1,1,1,1,1,0)	23	690	(0,0,0,1,0,1,0,0,1,1)	23
87	(1,0,1,1,0,0,1,1,0,0)	22	389	(1,1,0,1,1,1,1,1,1,0)	23	691	(1,0,0,1,0,1,0,0,1,1)	24
88	(0,1,1,1,0,0,1,1,0,0)	22	390	(0,0,1,1,1,1,1,1,1,0)	23	692	(0,1,0,1,0,1,0,0,1,1)	24
89	(1,1,1,1,0,0,1,1,0,0)	22	391	(1,0,1,1,1,1,1,1,1,0)	23	693	(1,1,0,1,0,1,0,0,1,1)	24
90	(0,0,0,0,1,0,1,1,0,0)	22	392	(0,1,1,1,1,1,1,1,1,0)	23	694	(0,0,1,1,0,1,0,0,1,1)	24
91	(1,0,0,0,1,0,1,1,0,0)	22	393	(1,1,1,1,1,1,1,1,1,0)	23	695	(1,0,1,1,0,1,0,0,1,1)	24
92	(0,1,0,0,1,0,1,1,0,0)	22	394	(0,0,0,0,0,0,0,0,0,1)	20	696	(0,1,1,1,0,1,0,0,1,1)	24
93	(1,1,0,0,1,0,1,1,0,0)	22	395	(1,0,0,0,0,0,0,0,0,1)	22	697	(1,1,1,1,0,1,0,0,1,1)	24
94	(0,0,1,0,1,0,1,1,0,0)	22	396	(0,1,0,0,0,0,0,0,0,1)	22	698	(0,0,0,0,1,1,0,0,1,1)	23
95	(1,0,1,0,1,0,1,1,0,0)	22	397	(1,1,0,0,0,0,0,0,0,1)	23	699	(1,0,0,0,1,1,0,0,1,1)	24
96	(0,1,1,0,1,0,1,1,0,0)	22	398	(0,0,1,0,0,0,0,0,0,1)	22	700	(0,1,0,0,1,1,0,0,1,1)	24
97	(1,1,1,0,1,0,1,1,0,0)	22	399	(1,0,1,0,0,0,0,0,0,1)	23	701	(1,1,0,0,1,1,0,0,1,1)	24
98	(0,0,0,1,1,0,1,1,0,0)	22	400	(0,1,1,0,0,0,0,0,0,1)	23	702	(0,0,1,0,1,1,0,0,1,1)	24
99	(1,0,0,1,1,0,1,1,0,0)	22	401	(1,1,1,0,0,0,0,0,0,1)	23	703	(1,0,1,0,1,1,0,0,1,1)	24
100	(0,1,0,1,1,0,1,1,0,0)	22	402	(0,0,0,1,0,0,0,0,0,1)	21	704	(0,1,1,0,1,1,0,0,1,1)	24
101	(1,1,0,1,1,0,1,1,0,0)	22	403	(1,0,0,1,0,0,0,0,0,1)	23	705	(1,1,1,0,1,1,0,0,1,1)	24
102	(0,0,1,1,1,0,1,1,0,0)	22	404	(0,1,0,1,0,0,0,0,0,1)	23	706	(0,0,0,1,1,1,0,0,1,1)	24
103	(1,0,1,1,1,0,1,1,0,0)	22	405	(1,1,0,1,0,0,0,0,0,1)	23	707	(1,0,0,1,1,1,0,0,1,1)	24
104	(0,1,1,1,1,0,1,1,0,0)	22	406	(0,0,1,1,0,0,0,0,0,1)	23	708	(0,1,0,1,1,1,0,0,1,1)	24
105	(1,1,1,1,1,0,1,1,0,0)	22	407	(1,0,1,1,0,0,0,0,0,1)	23	709	(1,1,0,1,1,1,0,0,1,1)	24
106	(0,0,0,0,0,1,1,1,0,0)	21	408	(0,1,1,1,0,0,0,0,0,1)	23	710	(0,0,1,1,1,1,0,0,1,1)	24
107	(1,0,0,0,0,1,1,1,0,0)	22	409	(1,1,1,1,0,0,0,0,0,1)	23	711	(1,0,1,1,1,1,0,0,1,1)	24
108	(0,1,0,0,0,1,1,1,0,0)	22	410	(0,0,0,0,1,0,0,0,0,1)	21	712	(0,1,1,1,1,1,0,0,1,1)	24
109	(1,1,0,0,0,1,1,1,0,0)	22	411	(1,0,0,0,1,0,0,0,0,1)	23	713	(1,1,1,1,1,1,0,0,1,1)	24
110	(0,0,1,0,0,1,1,1,0,0)	22	412	(0,1,0,0,1,0,0,0,0,1)	23	714	(0,0,0,0,0,0,1,0,1,1)	23
111	(1,0,1,0,0,1,1,1,0,0)	22	413	(1,1,0,0,1,0,0,0,0,1)	23	715	(1,0,0,0,0,0,1,0,1,1)	24
112	(0,1,1,0,0,1,1,1,0,0)	22	414	(0,0,1,0,1,0,0,0,0,1)	23	716	(0,1,0,0,0,0,1,0,1,1)	24
113	(1,1,1,0,0,1,1,1,0,0)	22	415	(1,0,1,0,1,0,0,0,0,1)	23	717	(1,1,0,0,0,0,1,0,1,1)	24
114	(0,0,0,1,0,1,1,1,0,0)	22	416	(0,1,1,0,1,0,0,0,0,1)	23	718	(0,0,1,0,0,0,1,0,1,1)	24
115	(1,0,0,1,0,1,1,1,0,0)	22	417	(1,1,1,0,1,0,0,0,0,1)	23	719	(1,0,1,0,0,0,1,0,1,1)	24
116	(0,1,0,1,0,1,1,1,0,0)	22	418	(0,0,0,1,1,0,0,0,0,1)	22	720	(0,1,1,0,0,0,1,0,1,1)	24
117	(1,1,0,1,0,1,1,1,0,0)	22	419	(1,0,0,1,1,0,0,0,0,1)	23	721	(1,1,1,0,0,0,1,0,1,1)	24
118	(0,0,1,1,0,1,1,1,0,0)	22	420	(0,1,0,1,1,0,0,0,0,1)	23	722	(0,0,0,1,0,0,1,0,1,1)	24
119	(1,0,1,1,0,1,1,1,0,0)	22	421	(1,1,0,1,1,0,0,0,0,1)	23	723	(1,0,0,1,0,0,1,0,1,1)	24
120	(0,1,1,1,0,1,1,1,0,0)	22	422	(0,0,1,1,1,0,0,0,0,1)	23	724	(0,1,0,1,0,0,1,0,1,1)	24
121	(1,1,1,1,0,1,1,1,0,0)	22	423	(1,0,1,1,1,0,0,0,0,1)	23	725	(1,1,0,1,0,0,1,0,1,1)	24
122	(0,0,0,0,1,1,1,1,0,0)	22	424	(0,1,1,1,1,0,0,0,0,1)	23	726	(0,0,1,1,0,0,1,0,1,1)	24
123	(1,0,0,0,1,1,1,1,0,0)	22	425	(1,1,1,1,1,0,0,0,0,1)	23	727	(1,0,1,1,0,0,1,0,1,1)	24
124	(0,1,0,0,1,1,1,1,0,0)	22	426	(0,0,0,0,0,1,0,0,0,1)	21	728	(0,1,1,1,0,0,1,0,1,1)	24
125	(1,1,0,0,1,1,1,1,0,0)	22	427	(1,0,0,0,0,1,0,0,0,1)	22	729	(1,1,1,1,0,0,1,0,1,1)	24
126	(0,0,1,0,1,1,1,1,0,0)	22	428	(0,1,0,0,0,1,0,0,0,1)	22	730	(0,0,0,0,1,0,1,0,1,1)	24
127	(1,0,1,0,1,1,1,1,0,0)	22	429	(1,1,0,0,0,1,0,0,0,1)	23	731	(1,0,0,0,1,0,1,0,1,1)	24
128	(0,1,1,0,1,1,1,1,0,0)	22	430	(0,0,1,0,0,1,0,0,0,1)	22	732	(0,1,0,0,1,0,1,0,1,1)	24

Continued

No.	T	$\log \underline{\mathrm{Cor}}$	No.	T	$\log \underline{\mathrm{Cor}}$	No.	T	$\log \underline{\mathrm{Cor}}$
129	(1,1,1,0,1,1,1,1,0,0)	22	431	(1,0,1,0,0,1,0,0,0,1)	23	733	(1,1,0,0,1,0,1,0,1,1)	24
130	(0,0,0,1,1,1,1,1,0,0)	22	432	(0,1,1,0,0,1,0,0,0,1)	23	734	(0,0,1,0,1,0,1,0,1,1)	24
131	(1,0,0,1,1,1,1,1,0,0)	22	433	(1,1,1,0,0,1,0,0,0,1)	23	735	(1,0,1,0,1,0,1,0,1,1)	24
132	(0,1,0,1,1,1,1,1,0,0)	22	434	(0,0,0,1,0,1,0,0,0,1)	22	736	(0,1,1,0,1,0,1,0,1,1)	24
133	(1,1,0,1,1,1,1,1,0,0)	22	435	(1,0,0,1,0,1,0,0,0,1)	23	737	(1,1,1,0,1,0,1,0,1,1)	24
134	(0,0,1,1,1,1,1,1,0,0)	22	436	(0,1,0,1,0,1,0,0,0,1)	23	738	(0,0,0,1,1,0,1,0,1,1)	24
135	(1,0,1,1,1,1,1,1,0,0)	22	437	(1,1,0,1,0,1,0,0,0,1)	23	739	(1,0,0,1,1,0,1,0,1,1)	24
136	(0,1,1,1,1,1,1,1,0,0)	22	438	(0,0,1,1,0,1,0,0,0,1)	23	740	(0,1,0,1,1,0,1,0,1,1)	24
137	(1,1,1,1,1,1,1,1,0,0)	22	439	(1,0,1,1,0,1,0,0,0,1)	23	741	(1,1,0,1,1,0,1,0,1,1)	24
138	(0,0,0,0,0,0,0,0,1,0)	17	440	(0,1,1,1,0,1,0,0,0,1)	23	742	(0,0,1,1,1,0,1,0,1,1)	24
139	(1,0,0,0,0,0,0,0,1,0)	19	441	(1,1,1,1,0,1,0,0,0,1)	23	743	(1,0,1,1,1,0,1,0,1,1)	24
140	(0,1,0,0,0,0,0,0,1,0)	19	442	(0,0,0,0,1,1,0,0,0,1)	22	744	(0,1,1,1,1,0,1,0,1,1)	24
141	(1,1,0,0,0,0,0,0,1,0)	20	443	(1,0,0,0,1,1,0,0,0,1)	23	745	(1,1,1,1,1,0,1,0,1,1)	24
142	(0,0,1,0,0,0,0,0,1,0)	19	444	(0,1,0,0,1,1,0,0,0,1)	23	746	(0,0,0,0,0,1,1,0,1,1)	23
143	(1,0,1,0,0,0,0,0,1,0)	20	445	(1,1,0,0,1,1,0,0,0,1)	23	747	(1,0,0,0,0,1,1,0,1,1)	24
144	(0,1,1,0,0,0,0,0,1,0)	20	446	(0,0,1,0,1,1,0,0,0,1)	23	748	(0,1,0,0,0,1,1,0,1,1)	24
145	(1,1,1,0,0,0,0,0,1,0)	20	447	(1,0,1,0,1,1,0,0,0,1)	23	749	(1,1,0,0,0,1,1,0,1,1)	24
146	(0,0,0,1,0,0,0,0,1,0)	18	448	(0,1,1,0,1,1,0,0,0,1)	23	750	(0,0,1,0,0,1,1,0,1,1)	24
147	(1,0,0,1,0,0,0,0,1,0)	20	449	(1,1,1,0,1,1,0,0,0,1)	23	751	(1,0,1,0,0,1,1,0,1,1)	24
148	(0,1,0,1,0,0,0,0,1,0)	20	450	(0,0,0,1,1,1,0,0,0,1)	23	752	(0,1,1,0,0,1,1,0,1,1)	24
149	(1,1,0,1,0,0,0,0,1,0)	20	451	(1,0,0,1,1,1,0,0,0,1)	23	753	(1,1,1,0,0,1,1,0,1,1)	24
150	(0,0,1,1,0,0,0,0,1,0)	20	452	(0,1,0,1,1,1,0,0,0,1)	23	754	(0,0,0,1,0,1,1,0,1,1)	24
151	(1,0,1,1,0,0,0,0,1,0)	20	453	(1,1,0,1,1,1,0,0,0,1)	23	755	(1,0,0,1,0,1,1,0,1,1)	24
152	(0,1,1,1,0,0,0,0,1,0)	20	454	(0,0,1,1,1,1,0,0,0,1)	23	756	(0,1,0,1,0,1,1,0,1,1)	24
153	(1,1,1,1,0,0,0,0,1,0)	20	455	(1,0,1,1,1,1,0,0,0,1)	23	757	(1,1,0,1,0,1,1,0,1,1)	24
154	(0,0,0,0,1,0,0,0,1,0)	18	456	(0,1,1,1,1,1,0,0,0,1)	23	758	(0,0,1,1,0,1,1,0,1,1)	24
155	(1,0,0,0,1,0,0,0,1,0)	20	457	(1,1,1,1,1,1,0,0,0,1)	23	759	(1,0,1,1,0,1,1,0,1,1)	24
156	(0,1,0,0,1,0,0,0,1,0)	20	458	(0,0,0,0,0,0,1,0,0,1)	22	760	(0,1,1,1,0,1,1,0,1,1)	24
157	(1,1,0,0,1,0,0,0,1,0)	20	459	(1,0,0,0,0,0,1,0,0,1)	23	761	(1,1,1,1,0,1,1,0,1,1)	24
158	(0,0,1,0,1,0,0,0,1,0)	20	460	(0,1,0,0,0,0,1,0,0,1)	23	762	(0,0,0,0,1,1,1,0,1,1)	24
159	(1,0,1,0,1,0,0,0,1,0)	20	461	(1,1,0,0,0,0,1,0,0,1)	23	763	(1,0,0,0,1,1,1,0,1,1)	24
160	(0,1,1,0,1,0,0,0,1,0)	20	462	(0,0,1,0,0,0,1,0,0,1)	23	764	(0,1,0,0,1,1,1,0,1,1)	24
161	(1,1,1,0,1,0,0,0,1,0)	20	463	(1,0,1,0,0,0,1,0,0,1)	23	765	(1,1,0,0,1,1,1,0,1,1)	24
162	(0,0,0,1,1,0,0,0,1,0)	19	464	(0,1,1,0,0,0,1,0,0,1)	23	766	(0,0,1,0,1,1,1,0,1,1)	24
163	(1,0,0,1,1,0,0,0,1,0)	20	465	(1,1,1,0,0,0,1,0,0,1)	23	767	(1,0,1,0,1,1,1,0,1,1)	24
164	(0,1,0,1,1,0,0,0,1,0)	20	466	(0,0,0,1,0,0,1,0,0,1)	23	768	(0,1,1,0,1,1,1,0,1,1)	24
165	(1,1,0,1,1,0,0,0,1,0)	20	467	(1,0,0,1,0,0,1,0,0,1)	23	769	(1,1,1,0,1,1,1,0,1,1)	24
166	(0,0,1,1,1,0,0,0,1,0)	20	468	(0,1,0,1,0,0,1,0,0,1)	23	770	(0,0,0,1,1,1,1,0,1,1)	24
167	(1,0,1,1,1,0,0,0,1,0)	20	469	(1,1,0,1,0,0,1,0,0,1)	23	771	(1,0,0,1,1,1,1,0,1,1)	24
168	(0,1,1,1,1,0,0,0,1,0)	20	470	(0,0,1,1,0,0,1,0,0,1)	23	772	(0,1,0,1,1,1,1,0,1,1)	24
169	(1,1,1,1,1,0,0,0,1,0)	20	471	(1,0,1,1,0,0,1,0,0,1)	23	773	(1,1,0,1,1,1,1,0,1,1)	24
170	(0,0,0,0,0,1,0,0,1,0)	18	472	(0,1,1,1,0,0,1,0,0,1)	23	774	(0,0,1,1,1,1,1,0,1,1)	24
171	(1,0,0,0,0,1,0,0,1,0)	19	473	(1,1,1,1,0,0,1,0,0,1)	23	775	(1,0,1,1,1,1,1,0,1,1)	24
172	(0,1,0,0,0,1,0,0,1,0)	19	474	(0,0,0,0,1,1,0,0,0,1)	23	776	(0,1,1,1,1,1,1,0,1,1)	24
173	(1,1,0,0,0,1,0,0,1,0)	20	475	(1,0,0,0,1,0,1,0,0,1)	23	777	(1,1,1,1,1,1,1,0,1,1)	24
174	(0,0,1,0,0,1,0,0,1,0)	19	476	(0,1,0,0,1,0,1,0,0,1)	23	778	(0,0,0,0,0,0,0,1,1,1)	21
175	(1,0,1,0,0,1,0,0,1,0)	20	477	(1,1,0,0,1,0,1,0,0,1)	23	779	(1,0,0,0,0,0,0,1,1,1)	23
176	(0,1,1,0,0,1,0,0,1,0)	20	478	(0,0,1,0,1,0,1,0,0,1)	23	780	(0,1,0,0,0,0,0,1,1,1)	23
177	(1,1,1,0,0,1,0,0,1,0)	20	479	(1,0,1,0,1,0,1,0,0,1)	23	781	(1,1,0,0,0,0,0,1,1,1)	24
178	(0,0,0,1,0,1,0,0,1,0)	19	480	(0,1,1,0,1,0,1,0,0,1)	23	782	(0,0,1,0,0,0,0,1,1,1)	23
179	(1,0,0,1,0,1,0,0,1,0)	20	481	(1,1,1,0,1,0,1,0,0,1)	23	783	(1,0,1,0,0,0,0,1,1,1)	24
180	(0,1,0,1,0,1,0,0,1,0)	20	482	(0,0,0,1,1,0,1,0,0,1)	23	784	(0,1,1,0,0,0,0,1,1,1)	24
181	(1,1,0,1,0,1,0,0,1,0)	20	483	(1,0,0,1,1,0,1,0,0,1)	23	785	(1,1,1,0,0,0,0,1,1,1)	24
182	(0,0,1,1,0,1,0,0,1,0)	20	484	(0,1,0,1,1,0,1,0,0,1)	23	786	(0,0,0,1,0,0,0,1,1,1)	22
183	(1,0,1,1,0,1,0,0,1,0)	20	485	(1,1,0,1,1,0,1,0,0,1)	23	787	(1,0,0,1,0,0,0,1,1,1)	24
184	(0,1,1,1,0,1,0,0,1,0)	20	486	(0,0,1,1,1,0,1,0,0,1)	23	788	(0,1,0,1,0,0,0,1,1,1)	24
185	(1,1,1,1,0,1,0,0,1,0)	20	487	(1,0,1,1,1,0,1,0,0,1)	23	789	(1,1,0,1,0,0,0,1,1,1)	24
186	(0,0,0,0,1,1,0,0,1,0)	19	488	(0,1,1,1,1,0,1,0,0,1)	23	790	(0,0,1,1,0,0,0,1,1,1)	24
187	(1,0,0,0,1,1,0,0,1,0)	20	489	(1,1,1,1,1,0,1,0,0,1)	23	791	(1,0,1,1,0,0,0,1,1,1)	24
188	(0,1,0,0,1,1,0,0,1,0)	20	490	(0,0,0,0,0,1,1,0,0,1)	22	792	(0,1,1,1,0,0,0,1,1,1)	24
189	(1,1,0,0,1,1,0,0,1,0)	20	491	(1,0,0,0,0,1,1,0,0,1)	23	793	(1,1,1,1,0,0,0,1,1,1)	24
190	(0,0,1,0,1,1,0,0,1,0)	20	492	(0,1,0,0,0,1,1,0,0,1)	23	794	(0,0,0,0,1,0,0,1,1,1)	22
191	(1,0,1,0,1,1,0,0,1,0)	20	493	(1,1,0,0,0,1,1,0,0,1)	23	795	(1,0,0,0,1,0,0,1,1,1)	24
192	(0,1,1,0,1,1,0,0,1,0)	20	494	(0,0,1,0,0,1,1,0,0,1)	23	796	(0,1,0,0,1,0,0,1,1,1)	24
193	(1,1,1,0,1,1,0,0,1,0)	20	495	(1,0,1,0,0,1,1,0,0,1)	23	797	(1,1,0,0,1,0,0,1,1,1)	24
194	(0,0,0,1,1,1,0,0,1,0)	20	496	(0,1,1,0,0,1,1,0,0,1)	23	798	(0,0,1,0,1,0,0,1,1,1)	24
195	(1,0,0,1,1,1,0,0,1,0)	20	497	(1,1,1,0,0,1,1,0,0,1)	23	799	(1,0,1,0,1,0,0,1,1,1)	24
196	(0,1,0,1,1,1,0,0,1,0)	20	498	(0,0,0,1,0,1,1,0,0,1)	23	800	(0,1,1,0,1,0,0,1,1,1)	24
197	(1,1,0,1,1,1,0,0,1,0)	20	499	(1,0,0,1,0,1,1,0,0,1)	23	801	(1,1,1,0,1,0,0,1,1,1)	24
198	(0,0,1,1,1,1,0,0,1,0)	20	500	(0,1,0,1,0,1,1,0,0,1)	23	802	(0,0,0,1,1,0,0,1,1,1)	23
199	(1,0,1,1,1,1,0,0,1,0)	20	501	(1,1,0,1,0,1,1,0,0,1)	23	803	(1,0,0,1,1,0,0,1,1,1)	24
200	(0,1,1,1,1,1,0,0,1,0)	20	502	(0,0,1,1,0,1,1,0,0,1)	23	804	(0,1,0,1,1,0,0,1,1,1)	24
201	(1,1,1,1,1,1,0,0,1,0)	20	503	(1,0,1,1,0,1,1,0,0,1)	23	805	(1,1,0,1,1,0,0,1,1,1)	24
202	(0,0,0,0,0,0,1,0,1,0)	19	504	(0,1,1,1,0,1,1,0,0,1)	23	806	(0,0,1,1,1,0,0,1,1,1)	24
203	(1,0,0,0,0,0,1,0,1,0)	20	505	(1,1,1,1,0,1,1,0,0,1)	23	807	(1,0,1,1,1,0,0,1,1,1)	24
204	(0,1,0,0,0,0,1,0,1,0)	20	506	(0,0,0,0,1,1,1,0,0,1)	23	808	(0,1,1,1,1,0,0,1,1,1)	24
205	(1,1,0,0,0,0,1,0,1,0)	20	507	(1,0,0,0,1,1,1,0,0,1)	23	809	(1,1,1,1,1,0,0,1,1,1)	24
206	(0,0,1,0,0,0,1,0,1,0)	20	508	(0,1,0,0,1,1,1,0,0,1)	23	810	(0,0,0,0,0,1,0,1,1,1)	22
207	(1,0,1,0,0,0,1,0,1,0)	20	509	(1,1,0,0,1,1,1,0,0,1)	23	811	(1,0,0,0,0,1,0,1,1,1)	23
208	(0,1,1,0,0,0,1,0,1,0)	20	510	(0,0,1,0,1,1,1,0,0,1)	23	812	(0,1,0,0,0,1,0,1,1,1)	23
209	(1,1,1,0,0,0,1,0,1,0)	21	511	(1,0,1,0,1,1,1,0,0,1)	23	813	(1,1,0,0,0,1,0,1,1,1)	24
210	(0,0,0,1,0,0,1,0,1,0)	20	512	(0,1,1,0,1,1,1,0,0,1)	23	814	(0,0,1,0,0,1,0,1,1,1)	23
211	(1,0,0,1,0,0,1,0,1,0)	20	513	(1,1,1,0,1,1,1,0,0,1)	23	815	(1,0,1,0,0,1,0,1,1,1)	24
212	(0,1,0,1,0,0,1,0,1,0)	20	514	(0,0,0,1,1,1,1,0,0,1)	23	816	(0,1,1,0,0,1,0,1,1,1)	24
213	(1,1,0,1,0,0,1,0,1,0)	20	515	(1,0,0,1,1,1,1,0,0,1)	23	817	(1,1,1,0,0,1,0,1,1,1)	24
214	(0,0,1,1,0,0,1,0,1,0)	20	516	(0,1,0,1,1,1,1,0,0,1)	23	818	(0,0,0,1,0,1,0,1,1,1)	23
215	(1,0,1,1,0,0,1,0,1,0)	20	517	(1,1,0,1,1,1,1,0,0,1)	23	819	(1,0,0,1,0,1,0,1,1,1)	24
216	(0,1,1,1,0,0,1,0,1,0)	20	518	(0,0,1,1,1,1,1,0,0,1)	23	820	(0,1,0,1,0,1,0,1,1,1)	24
217	(1,1,1,1,0,0,1,0,1,0)	21	519	(1,0,1,1,1,1,1,0,0,1)	23	821	(1,1,0,1,0,1,0,1,1,1)	24

Continued

No.	T	log Cor	No.	T	log Cor	No.	T	log Cor
218	(0,0,0,0,1,0,1,0,1,0)	20	520	(0,1,1,1,1,1,1,0,0,1)	23	822	(0,0,1,1,0,1,0,1,1,1)	24
219	(1,0,0,0,1,0,1,0,1,0)	20	521	(1,1,1,1,1,1,1,0,0,1)	23	823	(1,0,1,1,0,1,0,1,1,1)	24
220	(0,1,0,0,1,0,1,0,1,0)	20	522	(0,0,0,0,0,0,0,1,0,1)	20	824	(0,1,1,1,0,1,0,1,1,1)	24
221	(1,1,0,0,1,0,1,0,1,0)	20	523	(1,0,0,0,0,0,0,1,0,1)	22	825	(1,1,1,1,0,1,0,1,1,1)	24
222	(0,0,1,0,1,0,1,0,1,0)	20	524	(0,1,0,0,0,0,0,1,0,1)	22	826	(0,0,0,0,1,1,0,1,1,1)	23
223	(1,0,1,0,1,0,1,0,1,0)	20	525	(1,1,0,0,0,0,0,1,0,1)	23	827	(1,0,0,0,1,1,0,1,1,1)	24
224	(0,1,1,0,1,0,1,0,1,0)	20	526	(0,0,1,0,0,0,0,1,0,1)	22	828	(0,1,0,0,1,1,0,1,1,1)	24
225	(1,1,1,0,1,0,1,0,1,0)	21	527	(1,0,1,0,0,0,0,1,0,1)	23	829	(1,1,0,0,1,1,0,1,1,1)	24
226	(0,0,0,1,1,0,1,0,1,0)	20	528	(0,1,1,0,0,0,0,1,0,1)	23	830	(0,0,1,0,1,1,0,1,1,1)	24
227	(1,0,0,1,1,0,1,0,1,0)	20	529	(1,1,1,0,0,0,0,1,0,1)	23	831	(1,0,1,0,1,1,0,1,1,1)	24
228	(0,1,0,1,1,0,1,0,1,0)	20	530	(0,0,0,1,0,0,0,1,0,1)	21	832	(0,1,1,0,1,1,0,1,1,1)	24
229	(1,1,0,1,1,0,1,0,1,0)	20	531	(1,0,0,1,0,0,0,1,0,1)	23	833	(1,1,1,0,1,1,0,1,1,1)	24
230	(0,0,1,1,1,0,1,0,1,0)	20	532	(0,1,0,1,0,0,0,1,0,1)	23	834	(0,0,0,1,1,1,0,1,1,1)	24
231	(1,0,1,1,1,0,1,0,1,0)	20	533	(1,1,0,1,0,0,0,1,0,1)	23	835	(1,0,0,1,1,1,0,1,1,1)	24
232	(0,1,1,1,1,0,1,0,1,0)	20	534	(0,0,1,1,0,0,0,1,0,1)	23	836	(0,1,0,1,1,1,0,1,1,1)	24
233	(1,1,1,1,1,0,1,0,1,0)	21	535	(1,0,1,1,0,0,0,1,0,1)	23	837	(1,1,0,1,1,1,0,1,1,1)	24
234	(0,0,0,0,0,1,1,0,1,0)	19	536	(0,1,1,1,0,0,0,1,0,1)	23	838	(0,0,1,1,1,1,0,1,1,1)	24
235	(1,0,0,0,0,1,1,0,1,0)	20	537	(1,1,1,1,0,0,0,1,0,1)	23	839	(1,0,1,1,1,1,0,1,1,1)	24
236	(0,1,0,0,0,1,1,0,1,0)	20	538	(0,0,0,0,1,0,0,1,0,1)	21	840	(0,1,1,1,1,1,0,1,1,1)	24
237	(1,1,0,0,0,1,1,0,1,0)	20	539	(1,0,0,0,1,0,0,1,0,1)	23	841	(1,1,1,1,1,1,0,1,1,1)	24
238	(0,0,1,0,0,1,1,0,1,0)	20	540	(0,1,0,0,1,0,0,1,0,1)	23	842	(0,0,0,0,0,0,1,1,1,1)	23
239	(1,0,1,0,0,1,1,0,1,0)	20	541	(1,1,0,0,1,0,0,1,0,1)	23	843	(1,0,0,0,0,0,1,1,1,1)	24
240	(0,1,1,0,0,1,1,0,1,0)	20	542	(0,0,1,0,1,0,0,1,0,1)	23	844	(0,1,0,0,0,0,1,1,1,1)	24
241	(1,1,1,0,0,1,1,0,1,0)	21	543	(1,0,1,0,1,0,0,1,0,1)	23	845	(1,1,0,0,0,0,1,1,1,1)	24
242	(0,0,0,1,0,1,1,0,1,0)	20	544	(0,1,1,0,1,0,0,1,0,1)	23	846	(0,0,1,0,0,0,1,1,1,1)	24
243	(1,0,0,1,0,1,1,0,1,0)	20	545	(1,1,1,0,1,0,0,1,0,1)	23	847	(1,0,1,0,0,0,1,1,1,1)	24
244	(0,1,0,1,0,1,1,0,1,0)	20	546	(0,0,0,1,1,0,0,1,0,1)	22	848	(0,1,1,0,0,0,1,1,1,1)	24
245	(1,1,0,1,0,1,1,0,1,0)	20	547	(1,0,0,1,1,0,0,1,0,1)	23	849	(1,1,1,0,0,0,1,1,1,1)	24
246	(0,0,1,1,0,1,1,0,1,0)	20	548	(0,1,0,1,1,0,0,1,0,1)	23	850	(0,0,0,1,0,0,1,1,1,1)	24
247	(1,0,1,1,0,1,1,0,1,0)	20	549	(1,1,0,1,1,0,0,1,0,1)	23	851	(1,0,0,1,0,0,1,1,1,1)	24
248	(0,1,1,1,0,1,1,0,1,0)	20	550	(0,0,1,1,1,0,0,1,0,1)	23	852	(0,1,0,1,0,0,1,1,1,1)	24
249	(1,1,1,1,0,1,1,0,1,0)	21	551	(1,0,1,1,1,0,0,1,0,1)	23	853	(1,1,0,1,0,0,1,1,1,1)	24
250	(0,0,0,0,1,1,1,0,1,0)	20	552	(0,1,1,1,1,0,0,1,0,1)	23	854	(0,0,1,1,0,0,1,1,1,1)	24
251	(1,0,0,0,1,1,1,0,1,0)	20	553	(1,1,1,1,1,0,0,1,0,1)	23	855	(1,0,1,1,0,0,1,1,1,1)	24
252	(0,1,0,0,1,1,1,0,1,0)	20	554	(0,0,0,0,0,1,0,1,0,1)	21	856	(0,1,1,1,0,0,1,1,1,1)	24
253	(1,1,0,0,1,1,1,0,1,0)	20	555	(1,0,0,0,0,1,0,1,0,1)	22	857	(1,1,1,1,0,0,1,1,1,1)	24
254	(0,0,1,0,1,1,1,0,1,0)	20	556	(0,1,0,0,0,1,0,1,0,1)	22	858	(0,0,0,0,1,0,1,1,1,1)	24
255	(1,0,1,0,1,1,1,0,1,0)	20	557	(1,1,0,0,0,1,0,1,0,1)	23	859	(1,0,0,0,1,0,1,1,1,1)	24
256	(0,1,1,0,1,1,1,0,1,0)	20	558	(0,0,1,0,0,1,0,1,0,1)	22	860	(0,1,0,0,1,0,1,1,1,1)	24
257	(1,1,1,0,1,1,1,0,1,0)	21	559	(1,0,1,0,0,1,0,1,0,1)	23	861	(1,1,0,0,1,0,1,1,1,1)	24
258	(0,0,0,1,1,1,1,0,1,0)	20	560	(0,1,1,0,0,1,0,1,0,1)	23	862	(0,0,1,0,1,0,1,1,1,1)	24
259	(1,0,0,1,1,1,1,0,1,0)	20	561	(1,1,1,0,0,1,0,1,0,1)	23	863	(1,0,1,0,1,0,1,1,1,1)	24
260	(0,1,0,1,1,1,1,0,1,0)	20	562	(0,0,0,1,0,1,0,1,0,1)	22	864	(0,1,1,0,1,0,1,1,1,1)	24
261	(1,1,0,1,1,1,1,0,1,0)	20	563	(1,0,0,1,0,1,0,1,0,1)	23	865	(1,1,1,0,1,0,1,1,1,1)	24
262	(0,0,1,1,1,1,1,0,1,0)	20	564	(0,1,0,1,0,1,0,1,0,1)	23	866	(0,0,0,1,1,0,1,1,1,1)	24
263	(1,0,1,1,1,1,1,0,1,0)	20	565	(1,1,0,1,0,1,0,1,0,1)	23	867	(1,0,0,1,1,0,1,1,1,1)	24
264	(0,1,1,1,1,1,1,0,1,0)	20	566	(0,0,1,1,0,1,0,1,0,1)	23	868	(0,1,0,1,1,0,1,1,1,1)	24
265	(1,1,1,1,1,1,1,0,1,0)	21	567	(1,0,1,1,0,1,0,1,0,1)	23	869	(1,1,0,1,1,0,1,1,1,1)	24
266	(0,0,0,0,0,0,0,1,1,0)	20	568	(0,1,1,1,0,1,0,1,0,1)	23	870	(0,0,1,1,1,0,1,1,1,1)	24
267	(1,0,0,0,0,0,0,1,1,0)	22	569	(1,1,1,1,0,1,0,1,0,1)	23	871	(1,0,1,1,1,0,1,1,1,1)	24
268	(0,1,0,0,0,0,0,1,1,0)	22	570	(0,0,0,0,1,1,0,1,0,1)	22	872	(0,1,1,1,1,0,1,1,1,1)	24
269	(1,1,0,0,0,0,0,1,1,0)	23	571	(1,0,0,0,1,1,0,1,0,1)	23	873	(1,1,1,1,1,0,1,1,1,1)	24
270	(0,0,1,0,0,0,0,1,1,0)	22	572	(0,1,0,0,1,1,0,1,0,1)	23	874	(0,0,0,0,0,1,1,1,1,1)	23
271	(1,0,1,0,0,0,0,1,1,0)	23	573	(1,1,0,0,1,1,0,1,0,1)	23	875	(1,0,0,0,0,1,1,1,1,1)	24
272	(0,1,1,0,0,0,0,1,1,0)	23	574	(0,0,1,0,1,1,0,1,0,1)	23	876	(0,1,0,0,0,1,1,1,1,1)	24
273	(1,1,1,0,0,0,0,1,1,0)	23	575	(1,0,1,0,1,1,0,1,0,1)	23	877	(1,1,0,0,0,1,1,1,1,1)	24
274	(0,0,0,1,0,0,0,1,1,0)	21	576	(0,1,1,0,1,1,0,1,0,1)	23	878	(0,0,1,0,0,1,1,1,1,1)	24
275	(1,0,0,1,0,0,0,1,1,0)	23	577	(1,1,1,0,1,1,0,1,0,1)	23	879	(1,0,1,0,0,1,1,1,1,1)	24
276	(0,1,0,1,0,0,0,1,1,0)	23	578	(0,0,0,1,1,1,0,1,0,1)	23	880	(0,1,1,0,0,1,1,1,1,1)	24
277	(1,1,0,1,0,0,0,1,1,0)	23	579	(1,0,0,1,1,1,0,1,0,1)	23	881	(1,1,1,0,0,1,1,1,1,1)	24
278	(0,0,1,1,0,0,0,1,1,0)	23	580	(0,1,0,1,1,1,0,1,0,1)	23	882	(0,0,0,1,0,1,1,1,1,1)	24
279	(1,0,1,1,0,0,0,1,1,0)	23	581	(1,1,0,1,1,1,0,1,0,1)	23	883	(1,0,0,1,0,1,1,1,1,1)	24
280	(0,1,1,1,0,0,0,1,1,0)	23	582	(0,0,1,1,1,1,0,1,0,1)	23	884	(0,1,0,1,0,1,1,1,1,1)	24
281	(1,1,1,1,0,0,0,1,1,0)	23	583	(1,0,1,1,1,1,0,1,0,1)	23	885	(1,1,0,1,0,1,1,1,1,1)	24
282	(0,0,0,0,1,0,0,1,1,0)	21	584	(0,1,1,1,1,1,0,1,0,1)	23	886	(0,0,1,1,0,1,1,1,1,1)	24
283	(1,0,0,0,1,0,0,1,1,0)	23	585	(1,1,1,1,1,1,0,1,0,1)	23	887	(1,0,1,1,0,1,1,1,1,1)	24
284	(0,1,0,0,1,0,0,1,1,0)	23	586	(0,0,0,0,0,0,1,1,0,1)	22	888	(0,1,1,1,0,1,1,1,1,1)	24
285	(1,1,0,0,1,0,0,1,1,0)	23	587	(1,0,0,0,0,0,1,1,0,1)	23	889	(1,1,1,1,0,1,1,1,1,1)	24
286	(0,0,1,0,1,0,0,1,1,0)	23	588	(0,1,0,0,0,0,1,1,0,1)	23	890	(0,0,0,0,1,1,1,1,1,1)	24
287	(1,0,1,0,1,0,0,1,1,0)	23	589	(1,1,0,0,0,0,1,1,0,1)	23	891	(1,0,0,0,1,1,1,1,1,1)	24
288	(0,1,1,0,1,0,0,1,1,0)	23	590	(0,0,1,0,0,0,1,1,0,1)	23	892	(0,1,0,0,1,1,1,1,1,1)	24
289	(1,1,1,0,1,0,0,1,1,0)	23	591	(1,0,1,0,0,0,1,1,0,1)	23	893	(1,1,0,0,1,1,1,1,1,1)	24
290	(0,0,0,1,1,0,0,1,1,0)	22	592	(0,1,1,0,0,0,1,1,0,1)	23	894	(0,0,1,0,1,1,1,1,1,1)	24
291	(1,0,0,1,1,0,0,1,1,0)	23	593	(1,1,1,0,0,0,1,1,0,1)	23	895	(1,0,1,0,1,1,1,1,1,1)	24
292	(0,1,0,1,1,0,0,1,1,0)	23	594	(0,0,0,1,0,0,1,1,0,1)	23	896	(0,1,1,0,1,1,1,1,1,1)	24
293	(1,1,0,1,1,0,0,1,1,0)	23	595	(1,0,0,1,0,0,1,1,0,1)	23	897	(1,1,1,0,1,1,1,1,1,1)	24
294	(0,0,1,1,1,0,0,1,1,0)	23	596	(0,1,0,1,0,0,1,1,0,1)	23	898	(0,0,0,1,1,1,1,1,1,1)	24
295	(1,0,1,1,1,0,0,1,1,0)	23	597	(1,1,0,1,0,0,1,1,0,1)	23	899	(1,0,0,1,1,1,1,1,1,1)	24
296	(0,1,1,1,1,0,0,1,1,0)	23	598	(0,0,1,1,0,0,1,1,0,1)	23	900	(0,1,0,1,1,1,1,1,1,1)	24
297	(1,1,1,1,1,0,0,1,1,0)	23	599	(1,0,1,1,0,0,1,1,0,1)	23	901	(1,1,0,1,1,1,1,1,1,1)	24
298	(0,0,0,0,0,1,0,1,1,0)	21	600	(0,1,1,1,0,0,1,1,0,1)	23	902	(0,0,1,1,1,1,1,1,1,1)	24
299	(1,0,0,0,0,1,0,1,1,0)	22	601	(1,1,1,1,0,0,1,1,0,1)	23	903	(1,0,1,1,1,1,1,1,1,1)	24
300	(0,1,0,0,0,1,0,1,1,0)	22	602	(0,0,0,0,1,0,1,1,0,1)	23	904	(0,1,1,1,1,1,1,1,1,1)	24
301	(1,1,0,0,0,1,0,1,1,0)	23	603	(1,0,0,0,1,0,1,1,0,1)	23	905	(1,1,1,1,1,1,1,1,1,1)	24
302	(0,0,1,0,0,1,0,1,1,0)	22	604	(0,1,0,0,1,0,1,1,0,1)	23			

Algorithm 20 Truncated Linear Model construction for f_{b2} (`fb2Model`)

Input: Initial model $\mathcal{M}$ and 6 binary variables $(T_x, T_y, T_{ic}, T_{id_0}, T_{id_1})$
Output: Updated model $\mathcal{M}$ and 3 binary variables $T_z, T_{oc}, T_{od_0}, T_{od_1}, \tau$

1: Declare 4 variables $\mathcal{M}.\texttt{var} \leftarrow T_z, T_{oc}, T_{od_0}, T_{od_1}$ as binaries
2: Update $\mathcal{M}$ by adding the following constraint Eq. (D18):

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} -T_{id_0} + T_z + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ T_x - T_z, +T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ -T_x + T_w + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ -T_w + T_z + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ T_y - T_w + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ T_w - T_{id_1} + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ -T_y + T_w + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \\ T_x - T_{ic} + T_{oc} + T_{od_0} + T_{od_1} \geq 0 \end{cases} \tag{D18}$$

3: Declare 1 binary variable $\mathcal{M}.\texttt{var} \leftarrow \tau$
4: Define set $\mathcal{S} = \{T_x, T_y, T_{ic}, T_{id_0}, T_{od_1}, T_z, T_{oc}, T_{od_0}, T_{od_1}\}$
5: **for** $s \in \mathcal{S}$ **do**
6: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \geq s$
7: **end for**
8: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \leq \sum_{s \in \mathcal{S}} s$
9: Return $(\mathcal{M}, T_z, T_{oc}, T_{od_0}, T_{od_1}, \tau)$

Algorithm 21 Truncated Linear Model construction for h_b (`hbModel`)

Input: Initial model $\mathcal{M}$ and 2 binary variables (T_x, T_y)
Output: Updated model $\mathcal{M}$ and 3 binary variables (T_z, T_{oc}, τ)

1: $(\mathcal{M}, T_z, T_{oc}) \leftarrow \texttt{haModel}(\mathcal{M}, T_x, T_y)$
2: Declare 1 binary variable $\mathcal{M}.\texttt{var} \leftarrow \tau$
3: Define set $\mathcal{S} = (T_x, T_y, T_z, T_{oc})$
4: **for** $s \in \mathcal{S}$ **do**
5: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \geq s$
6: **end for**
7: Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \leq \sum_{s \in \mathcal{S}} s$
8: Return $(\mathcal{M}, T_z, T_{oc}, \tau)$

Appendix E Details for Accomplishing Candidate Search

When constructing bitwise model $\mathcal{M}$'s, we may define parameters $(\underline{P}, \overline{P})$ as upper and lower bounds of b_{obj}. By adding MILP model constraint

$$\mathcal{M}.\texttt{con} \leftarrow \underline{P} \leq b_{obj} \leq \overline{P}$$

we are able to acquire quickly the $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ candidates with $b_{obj} \in \left[\underline{P}, \overline{P}\right]$. The values of $(\underline{P}, \overline{P})$ are set based on the global optimal value of $\min b_{obj}$, denoted as $P_{\min}$, and the power of the MILP solver: for too large $\overline{P}$, there might be too many solutions so that the solver cannot terminate in feasible time.

Algorithm 22 Truncated Linear Model construction for f_b (`fbModel`)

Input: Initial model $\mathcal{M}$ and 3 binary variables (T_x, T_y, T_{ic})
Output: Updated model $\mathcal{M}$ and 3 binary variables (T_z, T_{oc}, τ)
Declare 2 binary variables $\mathcal{M}.\texttt{var} \leftarrow T_z, T_{oc}$
Update $\mathcal{M}$ by adding the constraints in Eq. (D19)

$$\mathcal{M}.\texttt{con} \leftarrow \begin{cases} T_x + T_y - T_{ic} - T_z + T_{oc} \geq 0 \\ -T_x + T_y - T_{ic} + T_z + T_{oc} \geq 0 \\ T_x - T_y - T_{ic} + T_z + T_{oc} \geq 0 \\ T_x + T_y + T_{ic} + T_z - T_{oc} \geq 0 \\ T_x - T_z + T_{oc} \geq 0 \\ T_y - T_z + T_{oc} \geq 0 \\ -T_x + T_z + T_{oc} \geq 0 \end{cases} \tag{D19}$$

Declare 1 binary variable $\mathcal{M}.\texttt{var} \leftarrow \tau$
Define set $\mathcal{S} = \{T_x, T_y, T_{ic}, T_z, T_{oc}\}$
for $s \in \mathcal{S}$ **do**
  Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \geq s$
end for
Add a constraint $\mathcal{M}.\texttt{con} \leftarrow \tau \leq \sum_{s \in \mathcal{S}} s$
Return $(\mathcal{M}, T_z, T_{oc}, \tau)$

The parameter λ in Algorithm 2 is set to $\lambda = 0$ by default.

E.1 Details for SNOW 3G

Truncated and bitwise linear masks of intermediate states satisfy Eq. (E20): the constraints in $\mathcal{M}_T$ are listed on the left while those for $\mathcal{M}$ are on the right.

$$\begin{cases} T_{\boldsymbol{u}_0} = T_{\boldsymbol{u}_1},\ T_{\boldsymbol{v}_0} = T_{\boldsymbol{v}_1},\ T_{\boldsymbol{w}_0} = T_{\boldsymbol{w}_1} \\ T_{\boldsymbol{u}_1} \xrightarrow{\mathrm{S}_1} T_{\boldsymbol{a}},\ T_{\boldsymbol{a}} \xrightarrow{\text{branch}} (T_{\boldsymbol{a}_0}, T_{\boldsymbol{a}_1}) \\ T_{\boldsymbol{z}_{t-1}} = T_{\boldsymbol{v}_0},\ T_{\boldsymbol{z}_{t-1}} = T_{-1}, \\ (T_{14}, T_{\boldsymbol{u}_0}) \xrightarrow{\boxplus} T_{\boldsymbol{z}_{t-1}} \\ T_{\boldsymbol{z}_t} = T_0,\ T_{\boldsymbol{z}_t} = T_{\boldsymbol{a}_0},\ (T_{15}, T_{\boldsymbol{w}_0}) \xrightarrow{\boxplus} T_{\boldsymbol{z}_t} \\ T_{\boldsymbol{z}_{t+1}} = T_1,\ T_{\boldsymbol{w}_1} \xrightarrow{\mathrm{S}_1} T_{\boldsymbol{z}_{t+1}},\ T_{\boldsymbol{v}_1} \xrightarrow{\mathrm{S}_2} T_5 \\ (T_{16}, T_5, T_{\boldsymbol{a}_1}) \xrightarrow{\boxplus^2} T_{\boldsymbol{z}_{t+1}} \end{cases} \quad \begin{cases} \Gamma_{\boldsymbol{u}_0} = \Gamma_{\boldsymbol{u}_1}, \Gamma_{\boldsymbol{v}_0} = \Gamma_{\boldsymbol{v}_1}, \Gamma_{\boldsymbol{w}_0} = \Gamma_{\boldsymbol{w}_1} \\ \Gamma_{\boldsymbol{u}_1} \xrightarrow{\mathrm{S}_1} \Gamma_{\boldsymbol{a}}, \Gamma_{\boldsymbol{a}} \xrightarrow{\text{branch}} (\Gamma_{\boldsymbol{a}_0}, \Gamma_{\boldsymbol{a}_1}) \\ \Gamma_{\boldsymbol{z}_{t-1}} = \Gamma_{\boldsymbol{v}_0}, \Gamma_{\boldsymbol{z}_{t-1}} = \Gamma_{-1}, \\ (\Gamma_{14}, \Gamma_{\boldsymbol{u}_0}) \xrightarrow{\boxplus} \Gamma_{\boldsymbol{z}_{t-1}} \\ \Gamma_{\boldsymbol{z}_t} = \Gamma_0,\ \Gamma_{\boldsymbol{z}_t} = \Gamma_{\boldsymbol{a}_0},\ (\Gamma_{15}, \Gamma_{\boldsymbol{w}_0}) \xrightarrow{\boxplus} \Gamma_{\boldsymbol{z}_t} \\ \Gamma_{\boldsymbol{z}_{t+1}} = \Gamma_1, \Gamma_{\boldsymbol{w}_1} \xrightarrow{\mathrm{S}_1} \Gamma_{\boldsymbol{z}_{t+1}}, \Gamma_{\boldsymbol{v}_1} \xrightarrow{\mathrm{S}_2} \Gamma_5 \\ (\Gamma_{16}, \Gamma_5, \Gamma_{\boldsymbol{a}_1}) \xrightarrow{\boxplus^2} \Gamma_{\boldsymbol{z}_{t+1}} \end{cases} \tag{E20}$$

For each $\boxplus^2$, the output of `conModAdd` in Algorithm 2 returns $(\boldsymbol{p}, \boldsymbol{q})$. According to Section 4.1, its contribution to the correlations can be evaluated as $2^{-(|\boldsymbol{p}|+2|\boldsymbol{q}|)}$. We use b_{obj} in Eq. (E21) to define $\mathcal{M}.\texttt{obj}$.

$$b_{obj} = \sum_{\forall \boxplus} |\Gamma_{\boldsymbol{oc}}| + \sum_{\forall \boxplus^2} (|\boldsymbol{p}| + 2|\boldsymbol{q}|) + 6(|T_{\boldsymbol{u}_1}| + |T_{\boldsymbol{v}_1}| + |T_{\boldsymbol{w}_1}|) \tag{E21}$$

For $\mathcal{M}_T$, each $\boxplus$ and $\boxplus^2$ model construction call (Algorithm 3 and Algorithm 4) returns a $\boldsymbol{\tau} = (\tau_0, \ldots, \tau_3)$ vector. According to Section 4.2, we can

define t_{obj} as Eq. (E22) and set $\mathcal{M}_T.\texttt{obj} \leftarrow \min t_{obj}$ for searching optimal $(T_{\boldsymbol{\ell}}, T_{\boldsymbol{z}})$'s.

$$t_{obj} = 8\sum_{\forall \boxplus}\sum_{i=0}^{3}\tau_i + \sum_{\forall \boxplus^2}(19\tau_0 + 21\tau_1 + 21\tau_2 + 20\tau_3) + 6(|T_{\boldsymbol{u}_1}| + |T_{\boldsymbol{v}_1}| + |T_{\boldsymbol{w}_1}|) \tag{E22}$$

There is only 1 optimal truncated mask solution for $\mathcal{M}_T$ with $t_{obj} = 53$. With b_{obj} in Eq. (E22), There is $P_{\min} = 35$ and the Gurobi solver can exhaust all solutions with $\overline{P} \leq 37$. We can further set $\underline{P}$ to values ≥ 38 so as to acquire more candidates. The best masks in Table E7 can be covered with $\overline{P} = 38$.

Table E7: The linear masks for SNOW 3G when $\Gamma_{\mathbf{z}_2} = \Gamma_5 = \texttt{0x1014190f}$ such that $|\texttt{Cor}| \geq 2^{-21}$

$\Gamma_{\mathbf{z}_0}$	$\Gamma_{\mathbf{z}_1}$	Γ_{14}	Γ_{15}	Γ_{16}	$\log(\lvert\texttt{Cor}\rvert)$	Ref.
0x00000002	0x00000001	0x00000003	0x00000001	0x1014190f	-20.386	new
0x00000078	0x00000001	0x00000058	0x00000001	0x1014190f	-20.409	new
0x00000058	0x00000001	0x00000078	0x00000001	0x1014190f	-20.409	new
0x00000030	0x00000001	0x00000030	0x00000001	0x1014190f	-20.479	[30]
0x00000020	0x00000001	0x00000020	0x00000001	0x1014190f	-20.479	[30]
0x000000d1	0x00000001	0x000000d9	0x00000001	0x1014190f	-20.665	new
0x00000001	0x00000001	0x00000001	0x00000001	0x1014190f	-20.801	[30]
0x00000002	0x00000005	0x00000003	0x00000007	0x1014190b	-20.804	new
0x00000078	0x00000005	0x00000058	0x00000007	0x1014190b	-20.827	new
0x00000058	0x00000005	0x00000078	0x00000007	0x1014190b	-20.827	new
0x0000002d	0x00000001	0x0000003d	0x00000001	0x1014190f	-20.894	new
0x00000020	0x00000005	0x00000020	0x00000007	0x1014190b	-20.898	new
0x00000030	0x00000005	0x00000030	0x00000007	0x1014190b	-20.898	new
0x000000a0	0x00000001	0x000000f0	0x00000001	0x1014190f	-20.902	new
0x000000c2	0x00000001	0x00000082	0x00000001	0x1014190f	-20.943	new
0x00000002	0x00000004	0x00000003	0x00000004	0x1014190a	-20.974	new
0x00000078	0x00000004	0x00000058	0x00000004	0x1014190a	-20.997	new
0x00000058	0x00000004	0x00000078	0x00000004	0x1014190a	-20.997	new

E.2 Details for SNOW 2.0

For SNOW 2.0, we can directly deduce the bitwise MILP model $\mathcal{M}$ and acquire $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$ directly. $\mathcal{M}$ can be deduced from the intermediate state linear masks satisfying Eq. (E23).

$$\begin{cases} \Gamma_{\boldsymbol{u}_0} = \Gamma_{\boldsymbol{u}_1}, \Gamma_{\boldsymbol{v}_0} = \Gamma_{\boldsymbol{v}_1} \\ \Gamma_{\boldsymbol{z}_t} = \Gamma_{\boldsymbol{v}_0}, \Gamma_{\boldsymbol{z}_t} = \Gamma_0, (\Gamma_{15}, \Gamma_{\boldsymbol{u}_0}) \xrightarrow{\boxplus} \Gamma_{\boldsymbol{z}_t} \\ \Gamma_{\boldsymbol{z}_{t+1}} = \Gamma_1, \Gamma_{\boldsymbol{u}_1} \xrightarrow{\mathtt{S}_1} \Gamma_{\boldsymbol{z}_{t+1}}, (\Gamma_{16}, \Gamma_5, \Gamma_{\boldsymbol{v}_1}) \xrightarrow{\boxplus^2} \Gamma_{\boldsymbol{z}_{t+1}} \end{cases} \tag{E23}$$

Similar to SNOW 3G, $\mathcal{M}$ for SNOW 2.0 has objective $\mathcal{M}.\texttt{obj} \leftarrow \min b_{obj}$ with the b_{obj} defined as Eq. (E24)

$$b_{obj} = \sum_{\forall \boxplus} |\Gamma_{\boldsymbol{oc}}| + \sum_{\forall \boxplus^2} (|\boldsymbol{p}| + 2|\boldsymbol{q}|) + 6|T_{\boldsymbol{u}_1}| \tag{E24}$$

Therefore, *Candidate Search* for SNOW 2.0 is accomplished with a simplified Algorithm 7 skipping all $\mathcal{M}_T$-related steps (Step 2,3,5). According to the solutions of $\mathcal{M}$, we have $P_{\min} = 21$ and the solver enables us to acquire all solutions with $b_{obj} \leq 24$. We also get another millions of by setting $\underline{P} = 25$ and $\overline{P} \geq 26$. In fact, the best $(\Gamma_{\boldsymbol{\ell}}, \Gamma_{\boldsymbol{z}})$'s in Table E8 are acquired with $\overline{P} \leq 26$.

Table E8: The linear masks for SNOW 2.0 such that $|\texttt{Cor}| \geq 2^{-15}$

$\Gamma_{\mathbf{z}_0}$	$\Gamma_{\mathbf{z}_1}$	Γ_{15}	Γ_{16}	Γ_5	$\log(\lvert\texttt{Cor}\rvert)$	Ref.
0x01800001	0x01800001	0x01800001	0x01800001	0x01800001	-14.411	[28]
0x01000001	0x01800001	0x01000001	0x01000001	0x01800001	-14.411	new
0x01000001	0x01800001	0x01000001	0x01800001	0x01000001	-14.411	new
0x01800001	0x01800001	0x01800001	0x01000001	0x01000001	-14.411	new
0x00018001	0x00018001	0x00018001	0x00018001	0x00018001	-14.496	[28]
0x00010001	0x00018001	0x00010001	0x00018001	0x00010001	-14.496	new
0x00018001	0x00018001	0x00018001	0x00010001	0x00010001	-14.496	new
0x00010001	0x00018001	0x00010001	0x00010001	0x00018001	-14.496	new
0x00010081	0x00010081	0x00010081	0x00010081	0x00010081	-14.635	[28]
0x00010081	0x00010081	0x00010081	0x000100c1	0x000100c1	-14.635	new
0x000180c1	0x000140c1	0x000100c1	0x000180c1	0x000180c1	-14.963	new
0x000100c1	0x000140c1	0x000180c1	0x000180c1	0x000100c1	-14.963	new
0x000180c1	0x000140c1	0x000100c1	0x000100c1	0x000100c1	-14.963	new
0x00018081	0x000140c1	0x00010081	0x000180c1	0x00018081	-14.963	new
0x000100c1	0x000140c1	0x000180c1	0x000100c1	0x000180c1	-14.963	new
0x00018081	0x000140c1	0x00010081	0x00018081	0x000180c1	-14.963	new
0x000180c1	0x000140c1	0x000100c1	0x00018081	0x00018081	-14.963	new
0x00018081	0x000140c1	0x00010081	0x000100c1	0x00010081	-14.963	new
0x000180c1	0x000140c1	0x000100c1	0x00010081	0x00010081	-14.963	new
0x00018081	0x000140c1	0x00010081	0x00010081	0x000100c1	-14.963	new
0x000100c1	0x000140c1	0x000180c1	0x00018081	0x00010081	-14.963	new
0x000100c1	0x000140c1	0x000180c1	0x00010081	0x00018081	-14.963	new
0x00010081	0x000140c1	0x00018081	0x00018081	0x000100c1	-14.963	new
0x00010081	0x000140c1	0x00018081	0x000180c1	0x00010081	-14.963	new
0x00010081	0x000140c1	0x00018081	0x000100c1	0x00018081	-14.963	new
0x00010081	0x000140c1	0x00018081	0x00010081	0x000180c1	-14.963	new

Appendix F Proof of Theorem 3

Proof The function A is defined as

$$A(Sc_0) = \sum_{Sd_0} \rho_0(Sc_0, Sd_0),$$

where

$$\rho_0(Sc_0, Sd_0) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^0, \boldsymbol{u}_0^j, \boldsymbol{v}_0^j, \boldsymbol{w}_j^0)}[c_0^j | d_j^0][0 | d_{j-1}^0].$$

In terms of the definition of the matrices $\boldsymbol{U}^{(\boldsymbol{a}_j^0,\boldsymbol{u}_0^j,\boldsymbol{v}_0^j,\boldsymbol{w}_j^0)}$ for $j=0,1,2,3$, we deduce that for any fixed $(\sigma_0^0,\sigma_0^1,\sigma_0^2,\sigma_0^3)\in\{0,1\}^4$, we have

$$\begin{aligned}A(\sigma_0^0,\sigma_0^1,\sigma_0^2,\sigma_0^3)=&\Pr\left\{\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^0,\boldsymbol{u}_0^j,\boldsymbol{v}_0^j,\boldsymbol{w}_j^0)}(\cdot)=0,c_0^0=\sigma_0^0,c_0^1=\sigma_0^1,c_0^2=\sigma_0^2,c_0^3=\sigma_0^3\right\}\\&-\Pr\left\{\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^0,\boldsymbol{u}_0^j,\boldsymbol{v}_0^j,\boldsymbol{w}_j^0)}(\cdot)=1,c_0^0=\sigma_0^0,c_0^1=\sigma_0^1,c_0^2=\sigma_0^2,c_0^3=\sigma_0^3\right\}.\end{aligned}$$

The function B is defined as

$$B(Sc_1)=\sum_{Sd_1}\sum_{Sc_0}A(Sc_0)\cdot\rho_1(Sc_0,Sc_1,Sd_1),$$

where

$$\rho_1(Sc_0,Sc_1,Sd_1)=\prod\nolimits_{j=0}^{3}\boldsymbol{U}^{(\boldsymbol{a}_j^1,\boldsymbol{u}_1^j,\boldsymbol{v}_1^j,\boldsymbol{w}_j^1)}[c_1^j|d_j^1][c_0^j|d_{j-1}^1].$$

Similarly we deduce that for any fixed $(\sigma_1^0,\sigma_1^1,\sigma_1^2,\sigma_1^3)\in\{0,1\}^4$,

$$\begin{aligned}B(\sigma_1^0,\sigma_1^1,\sigma_1^2,\sigma_1^3)=&\Pr\left\{\bigoplus_{k=0}^{1}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=0,c_1^0=\sigma_1^0,c_1^1=\sigma_1^1,c_1^2=\sigma_1^2,c_1^3=\sigma_1^3\right\}\\&-\Pr\left\{\bigoplus_{k=0}^{1}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=1,c_1^0=\sigma_1^0,c_1^1=\sigma_1^1,c_1^2=\sigma_1^2,c_1^3=\sigma_1^3\right\}\end{aligned}$$

Similarly, from the definitions of the functions C and D respectively, we can deduce that

$$\begin{aligned}C(\sigma_2^0,\sigma_2^1,\sigma_2^2,\sigma_2^3)=&\Pr\left\{\bigoplus_{k=0}^{2}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=0,c_2^0=\sigma_2^0,c_2^1=\sigma_2^1,c_2^2=\sigma_2^2,c_2^3=\sigma_2^3\right\}\\&-\Pr\left\{\bigoplus_{k=0}^{2}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=1,c_2^0=\sigma_2^0,c_2^1=\sigma_2^1,c_2^2=\sigma_2^2,c_2^3=\sigma_2^3\right\},\end{aligned}$$

for any fixed $(\sigma_2^0,\sigma_2^1,\sigma_2^2,\sigma_2^3)\in\{0,1\}^4$, and

$$\begin{aligned}D(\sigma_3^0,\sigma_3^1,\sigma_3^2,\sigma_3^3)=&\Pr\left\{\bigoplus_{k=0}^{3}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=0,c_3^0=\sigma_3^0,c_3^1=\sigma_3^1,c_3^2=\sigma_3^2,c_3^3=\sigma_3^3\right\}\\&-\Pr\left\{\bigoplus_{k=0}^{3}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=1,c_3^0=\sigma_3^0,c_3^1=\sigma_3^1,c_3^2=\sigma_3^2,c_3^3=\sigma_3^3\right\},\end{aligned}$$

for any fixed $(\sigma_3^0,\sigma_3^1,\sigma_3^2,\sigma_3^3)\in\{0,1\}^4$.

Finally, we derive

$$\begin{aligned}\sum_{Sc_3}D(Sc_3)=&\sum_{\sigma_3^0\in\{0,1\}}\sum_{\sigma_3^1\in\{0,1\}}\sum_{\sigma_3^2\in\{0,1\}}\sum_{\sigma_3^3\in\{0,1\}}D(\sigma_3^0,\sigma_3^1,\sigma_3^2,\sigma_3^3)\\=&\Pr\left\{\bigoplus_{k=0}^{3}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=0\right\}-\Pr\left\{\bigoplus_{k=0}^{3}\bigoplus_{j=0}^{3}\bar{f}^{(\boldsymbol{a}_j^k,\boldsymbol{u}_k^j,\boldsymbol{v}_k^j,\boldsymbol{w}_j^k)}(\cdot)=1\right\}\\=&\mathtt{Cor}((\boldsymbol{U},\boldsymbol{V},\boldsymbol{W})\xrightarrow{\mathcal{F}}\boldsymbol{A}).\end{aligned}$$

Thus we complete the proof. □

Appendix G Detailed Process for Carrying out Step 1 to Step 5

In the following part, we will describe our strategy for carrying out **Step 1** to **Step 4** in turn.

- **Step 1:** Let us recall that the expression for $A(Sc_0)$ is

$$A(Sc_0) = \sum_{Sd_0} \rho_0(Sc_0, Sd_0),$$

where

$$\rho_0(Sc_0, Sd_0) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^0, \boldsymbol{u}_0^j, \boldsymbol{v}_0^j, \boldsymbol{w}_j^0)}[c_0^j | d_j^0][0 | d_{j-1}^0].$$

To compute $A(Sc_0)$ for all the 2^4 combinations of involved local carries in Sc_0, we do the following:

1.1 Compute $A_0(c_0^0, c_0^1, d_1^0) \triangleq \sum_{d_0^0} \boldsymbol{U}^{(\boldsymbol{a}_0^0, \boldsymbol{u}_0^0, \boldsymbol{v}_0^0, \boldsymbol{w}_0^0)}[c_0^0 | d_0^0][0|0] \cdot \boldsymbol{U}^{(\boldsymbol{a}_1^0, \boldsymbol{u}_0^1, \boldsymbol{v}_0^1, \boldsymbol{w}_1^0)}[c_0^1 | d_1^0][0 | d_0^0]$ for all the 2^3 choices of (c_0^0, c_0^1, d_1^0). The time complexity is $\mathcal{O}(2^4)$ and the memory complexity is $\mathcal{O}(2^3)$.

1.2 Compute $A_1(c_0^0, c_0^1, c_0^2, d_2^0) \triangleq \sum_{d_1^0} A_0(c_0^0, c_0^1, d_1^0) \cdot \boldsymbol{U}^{(\boldsymbol{a}_2^0, \boldsymbol{u}_0^2, \boldsymbol{v}_0^2, \boldsymbol{w}_2^0)}[c_0^2 | d_2^0][0 | d_1^0]$ for all the 2^4 choices of $(c_0^0, c_0^1, c_0^2, d_2^0)$. The time complexity is $\mathcal{O}(2^5)$ and the memory complexity is $\mathcal{O}(2^4)$.

1.3 Compute $A_2(c_0^0, c_0^1, c_0^2, c_0^3, d_3^0) \triangleq \sum_{d_2^0} A_1(c_0^0, c_0^1, c_0^2, d_2^0) \cdot \boldsymbol{U}^{(\boldsymbol{a}_3^0, \boldsymbol{u}_0^3, \boldsymbol{v}_0^3, \boldsymbol{w}_3^0)}[c_0^3 | d_3^0][0 | d_2^0]$ for all the 2^5 choices of $(c_0^0, c_0^1, c_0^2, c_0^3, d_3^0)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

1.4 Compute $A(Sc_0) = \sum_{d_3^0} A_2(c_0^0, c_0^1, c_0^2, c_0^3, d_3^0)$ for all the 2^4 choices in (Sc_0), i.e., $(c_0^0, c_0^1, c_0^2, c_0^3)$. The time complexity is $\mathcal{O}(2^5)$ and the memory complexity is $\mathcal{O}(2^4)$.

Complexity of Step 1. The total time complexity of Step 1 is around $\mathcal{O}(2^{7.17})$.

- **Step 2:** The expression for $B(Sc_1)$ is

$$B(Sc_1) = \sum_{Sd_1} \sum_{Sc_0} A(Sc_0) \cdot \rho_1(Sc_0, Sc_1, Sd_1),$$

where

$$\rho_1(Sc_0, Sc_1, Sd_1) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^1, \boldsymbol{u}_1^j, \boldsymbol{v}_1^j, \boldsymbol{w}_j^1)}[c_1^j | d_j^1][c_0^j | d_{j-1}^1].$$

We describe the process for computing $B(Sc_1)$ for all 2^4 combinations of involved local carries in Sc_1 as follows.

2.1 Compute $B_0(c_0^1, c_0^2, c_0^3, c_1^0, d_0^1) \triangleq \sum_{c_0^0} A(c_0^0, c_0^1, c_0^2, c_0^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_0^1, \boldsymbol{u}_1^0, \boldsymbol{v}_1^0, \boldsymbol{w}_0^1)}[c_1^0|d_0^1][c_0^0|0]$ for all the 2^5 choices of $(c_0^1, c_0^2, c_0^3, c_1^0, d_0^1)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

2.2 Compute $B_1(c_0^1, c_0^2, c_0^3, c_1^0, c_1^1, d_1^1) \triangleq \sum_{d_0^1} B_0(c_0^1, c_0^2, c_0^3, c_1^0, d_0^1) \cdot \boldsymbol{U}^{(\boldsymbol{a}_1^1, \boldsymbol{u}_1^1, \boldsymbol{v}_1^1, \boldsymbol{w}_1^1)}[c_1^1|d_1^1][c_0^1|d_0^1]$ for all the 2^6 choices of $(c_0^1, c_0^2, c_0^3, c_1^0, c_1^1, d_1^1)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

2.3 Compute $B_2(c_0^2, c_0^3, c_1^0, c_1^1, d_1^1) \triangleq \sum_{c_0^1} B_1(c_0^1, c_0^2, c_0^3, c_1^0, c_1^1, d_1^1)$ for all the 2^5 choices of $(c_0^2, c_0^3, c_1^0, c_1^1, d_1^1)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

2.4 Compute $B_3(c_0^2, c_0^3, c_1^0, c_1^1, c_1^2, d_2^1) \triangleq \sum_{d_1^1} B_2(c_0^2, c_0^3, c_1^0, c_1^1, d_1^1) \cdot \boldsymbol{U}^{(\boldsymbol{a}_2^1, \boldsymbol{u}_1^2, \boldsymbol{v}_1^2, \boldsymbol{w}_2^1)}[c_1^2|d_2^1][c_0^2|d_1^1]$ for all the 2^6 choices of $(c_0^2, c_0^3, c_1^0, c_1^1, c_1^2, d_2^1)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

2.5 Compute $B_4(c_0^3, c_1^0, c_1^1, c_1^2, d_2^1) \triangleq \sum_{c_0^2} B_3(c_0^2, c_0^3, c_1^0, c_1^1, c_1^2, d_2^1)$ for all the 2^5 choices of $(c_0^3, c_1^0, c_1^1, c_1^2, d_2^1)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

2.6 Compute $B_5(c_0^3, c_1^0, c_1^1, c_1^2, c_1^3, d_3^1) \triangleq \sum_{d_2^1} B_4(c_0^3, c_1^0, c_1^1, c_1^2, d_2^1) \cdot \boldsymbol{U}^{(\boldsymbol{a}_3^1, \boldsymbol{u}_1^3, \boldsymbol{v}_1^3, \boldsymbol{w}_3^1)}[c_1^3|d_3^1][c_0^3|d_2^1]$ for all the 2^6 choices of $(c_0^3, c_1^0, c_1^1, c_1^2, c_1^3, d_3^1)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

2.7 Compute $B_6(c_1^0, c_1^1, c_1^2, c_1^3, d_3^1) \triangleq \sum_{c_0^3} B_5(c_0^3, c_1^0, c_1^1, c_1^2, c_1^3, d_3^1)$ for all the 2^5 choices of $(c_1^0, c_1^1, c_1^2, c_1^3, d_3^1)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

2.8 Compute $B(Sc_1) = \sum_{d_3^1} B_6(c_1^0, c_1^1, c_1^2, c_1^3, d_3^1)$ for all the 2^4 choices in (Sc_1), i.e., $(c_1^0, c_1^1, c_1^2, c_1^3)$. The time complexity is $\mathcal{O}(2^5)$ and the memory complexity is $\mathcal{O}(2^4)$.

Complexity of Step 2. The total time complexity of Step 2 is around $\mathcal{O}(2^{9.39})$.

- **Step 3:** The expression for $C(Sc_2)$ is

$$C(Sc_2) = \sum_{Sd_2} \sum_{Sc_1} B(Sc_1) \cdot \rho_2(Sc_1, Sc_2, Sd_2),$$

where

$$\rho_2(Sc_1, Sc_2, Sd_2) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^2, \boldsymbol{u}_2^j, \boldsymbol{v}_2^j, \boldsymbol{w}_j^2)}[c_2^j|d_j^2][c_1^j|d_{j-1}^2].$$

The computation of $C(Sc_2)$ for all 2^4 combinations of involved local carries in Sc_2 is carried out according to the following steps.

3.1 Compute $C_0(c_1^1, c_1^2, c_1^3, c_2^0, d_0^2) \triangleq \sum_{c_1^0} B(c_1^0, c_1^1, c_1^2, c_1^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_0^2, \boldsymbol{u}_2^0, \boldsymbol{v}_2^0, \boldsymbol{w}_0^2)}[c_2^0|d_0^2][c_1^0|0]$ for all the 2^5 choices of $(c_1^1, c_1^2, c_1^3, c_2^0, d_0^2)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

3.2 Compute $C_1(c_1^1, c_1^2, c_1^3, c_2^0, c_2^1, d_1^2) \triangleq \sum_{d_0^2} C_0(c_1^1, c_1^2, c_1^3, c_2^0, d_0^2) \cdot \boldsymbol{U}^{(\boldsymbol{a}_1^2, \boldsymbol{u}_2^1, \boldsymbol{v}_2^1, \boldsymbol{w}_1^2)}[c_2^1|d_1^2][c_1^1|d_0^2]$ for all the 2^6 choices of $(c_1^1, c_1^2, c_1^3, c_2^0, c_2^1, d_1^2)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

3.3 Compute $C_2(c_1^2, c_1^3, c_2^0, c_2^1, d_1^2) \triangleq \sum_{c_1^1} C_1(c_1^1, c_1^2, c_1^3, c_2^0, c_2^1, d_1^2)$ for all the 2^5 choices of $(c_1^2, c_1^3, c_2^0, c_2^1, d_1^2)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

3.4 Compute $C_3(c_1^2, c_1^3, c_2^0, c_2^1, c_2^2, d_2^2) \triangleq \sum_{d_1^2} C_2(c_1^2, c_1^3, c_2^0, c_2^1, d_1^2) \cdot \boldsymbol{U}^{(\boldsymbol{a}_2^2, \boldsymbol{u}_2^2, \boldsymbol{v}_2^2, \boldsymbol{w}_2^2)}[c_2^2|d_2^2][c_1^2|d_1^2]$ for all the 2^6 choices of $(c_1^2, c_1^3, c_2^0, c_2^1, c_2^2, d_2^2)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

3.5 Compute $C_4(c_1^3, c_2^0, c_2^1, c_2^2, d_2^2) \triangleq \sum_{c_1^2} C_3(c_1^2, c_1^3, c_2^0, c_2^1, c_2^2, d_2^2)$ for all the 2^5 choices of $(c_1^3, c_2^0, c_2^1, c_2^2, d_2^2)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

3.6 Compute $C_5(c_1^3, c_2^0, c_2^1, c_2^2, c_2^3, d_3^2) \triangleq \sum_{d_2^2} C_4(c_1^3, c_2^0, c_2^1, c_2^2, d_2^2) \cdot \boldsymbol{U}^{(\boldsymbol{a}_3^2, \boldsymbol{u}_2^3, \boldsymbol{v}_2^3, \boldsymbol{w}_3^2)}[c_2^3|d_3^2][c_1^3|d_2^2]$ for all the 2^6 choices of $(c_1^3, c_2^0, c_2^1, c_2^2, c_2^3, d_3^2)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

3.7 Compute $C_6(c_2^0, c_2^1, c_2^2, c_2^3, d_3^2) \triangleq \sum_{c_1^3} C_5(c_1^3, c_2^0, c_2^1, c_2^2, c_2^3, d_3^2)$ for all the 2^5 choices of $(c_2^0, c_2^1, c_2^2, c_2^3, d_3^2)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

3.8 Compute $C(Sc_2) = \sum_{d_3^2} C_6(c_2^0, c_2^1, c_2^2, c_2^3, d_3^2)$ for all the 2^4 choices in (Sc_2), i.e., $(c_2^0, c_2^1, c_2^2, c_2^3)$. The time complexity is $\mathcal{O}(2^5)$ and the memory complexity is $\mathcal{O}(2^4)$.

Complexity of Step 3. The total time complexity of Step 3 is around $\mathcal{O}(2^{9.39})$.

- **Step 4:** The expression for $D(Sc_3)$ is

$$D(Sc_3) = \sum_{Sd_3} \sum_{Sc_2} C(Sc_2) \cdot \rho_3(Sc_2, Sc_3, Sd_3),$$

where

$$\rho_3(Sc_2, Sc_3, Sd_3) = \prod_{j=0}^{3} \boldsymbol{U}^{(\boldsymbol{a}_j^3, \boldsymbol{u}_3^j, \boldsymbol{v}_3^j, \boldsymbol{w}_j^3)}[c_3^j|d_j^3][c_2^j|d_{j-1}^3].$$

The computation of $D(Sc_3)$ for all 2^4 combinations of involved local carries in Sc_3 is carried out according to the following steps.

4.1 Compute $D_0(c_2^1, c_2^2, c_2^3, c_3^0, d_0^3) \triangleq \sum_{c_2^0} C(c_2^0, c_2^1, c_2^2, c_2^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_0^3, \boldsymbol{u}_3^0, \boldsymbol{v}_3^0, \boldsymbol{w}_0^3)}[c_3^0|d_0^3][c_2^0|0]$ for all the 2^5 choices of $(c_2^1, c_2^2, c_2^3, c_3^0, d_0^3)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

4.2 Compute $D_1(c_2^1, c_2^2, c_2^3, c_3^0, c_3^1, d_1^3) \triangleq \sum_{d_0^3} D_0(c_2^1, c_2^2, c_2^3, c_3^0, d_0^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_1^3, \boldsymbol{u}_3^1, \boldsymbol{v}_3^1, \boldsymbol{w}_1^3)}[c_3^1|d_1^3][c_2^1|d_0^3]$ for all the 2^6 choices of $(c_2^1, c_2^2, c_2^3, c_3^0, c_3^1, d_1^3)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

4.3 Compute $D_2(c_2^2, c_2^3, c_3^0, c_3^1, d_1^3) \triangleq \sum_{c_2^1} D_1(c_2^1, c_2^2, c_2^3, c_3^0, c_3^1, d_1^3)$ for all the 2^5 choices of $(c_2^2, c_2^3, c_3^0, c_3^1, d_1^3)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

4.4 Compute $D_3(c_2^2, c_2^3, c_3^0, c_3^1, c_3^2, d_2^3) \triangleq \sum_{d_1^3} D_2(c_2^2, c_2^3, c_3^0, c_3^1, d_1^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_2^3, \boldsymbol{u}_3^2, \boldsymbol{v}_3^2, \boldsymbol{w}_2^3)}[c_3^2|d_2^3][c_2^2|d_1^3]$ for all the 2^6 choices of $(c_2^2, c_2^3, c_3^0, c_3^1, c_3^2, d_2^3)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

4.5 Compute $D_4(c_2^3, c_3^0, c_3^1, c_3^2, d_2^3) \triangleq \sum_{c_2^2} D_3(c_2^2, c_2^3, c_3^0, c_3^1, c_3^2, d_2^3)$ for all the 2^5 choices of $(c_2^3, c_3^0, c_3^1, c_3^2, d_2^3)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

4.6 Compute $D_5(c_2^3, c_3^0, c_3^1, c_3^2, c_3^3, d_3^3) \triangleq \sum_{d_2^3} D_4(c_2^3, c_3^0, c_3^1, c_3^2, d_2^3) \cdot \boldsymbol{U}^{(\boldsymbol{a}_3^3, \boldsymbol{u}_3^3, \boldsymbol{v}_3^3, \boldsymbol{w}_3^3)}[c_3^3|d_3^3][c_2^3|d_2^3]$ for all the 2^6 choices of $(c_2^3, c_3^0, c_3^1, c_3^2, c_3^3, d_3^3)$. The time complexity is $\mathcal{O}(2^7)$ and the memory complexity is $\mathcal{O}(2^6)$.

4.7 Compute $D_6(c_3^0, c_3^1, c_3^2, c_3^3, d_3^3) \triangleq \sum_{c_2^3} D_5(c_2^3, c_3^0, c_3^1, c_3^2, c_3^3, d_3^3)$ for all the 2^5 choices of $(c_3^0, c_3^1, c_3^2, c_3^3, d_3^3)$. The time complexity is $\mathcal{O}(2^6)$ and the memory complexity is $\mathcal{O}(2^5)$.

4.8 Compute $D(Sc_3) = \sum_{d_3^3} D_6(c_3^0, c_3^1, c_3^2, c_3^3, d_3^3)$ for all the 2^4 choices in (Sc_3), i.e., $(c_3^0, c_3^1, c_3^2, c_3^3)$. The time complexity is $\mathcal{O}(2^5)$ and the memory complexity is $\mathcal{O}(2^4)$.

Complexity of Step 4. The total time complexity of Step 4 is around $\mathcal{O}(2^{9.39})$.

After the values of $D(Sc_3)$ for all the combinations of involved local carries in Sc_3 are obtained, the accurate value of $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ can be derived according to Theorem 3 as $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A}) = \sum_{Sc_3} D(Sc_3)$ with a time complexity of $\mathcal{O}(2^4)$. To sum up, the total time complexity for computing $\texttt{Cor}((\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}) \xrightarrow{\mathcal{F}} \boldsymbol{A})$ according to Step 1 to Step 5 is around $2^{7.17} + 2^{9.39} + 2^{9.39} + 2^{9.39} + 2^4 = \mathcal{O}(2^{11})$.

Appendix H Searching for 4-tuples of Vectors

Here we present the method in [28, 30] for searching 4-tuples of column vectors of the generator matrix $\boldsymbol{G}$ which add to 0 on some bits.

Rewriting the matrix $\boldsymbol{G}$ in column vectors as $\boldsymbol{G} = (\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_N)$, we try to find the XORs of the l-bit column vectors that vanish on some $l - l'$ bits. Specifically for SNOW 3G and SNOW 2.0, we look for a number of 4-tuples from $\boldsymbol{G}$ which add to 0 on their most significant $l-l'$ bits. As stated in [28, 30], this can be solved using Wagner's k-tree algorithm [39] by combining a small technique. Below we illustrate this process.

Let l_1 and l_2 be two positive integers such that $l_1 + l_2 = l - l'$, and $\mathtt{high}_n(\mathbf{a})$ be the value of the vector $\mathbf{a}$ on the most significant n bits. Collecting the N column vectors of $\boldsymbol{G}$ in one single list $\boldsymbol{L}$, we carry out the following two steps:

- Create a new list $\boldsymbol{L}_1$ from the original list $\boldsymbol{L}$ composed of all the XORs of $\mathbf{g}_{j_1}$ and $\mathbf{g}_{j_2}$ with $\mathbf{g}_{j_1} \neq \mathbf{g}_{j_2}$, $\mathbf{g}_{j_1}, \mathbf{g}_{j_2} \in L$ such that $\mathtt{high}_{l_1}(\mathbf{g}_{j_1} \oplus \mathbf{g}_{j_2}) = 0$. We say that l_1 bits are eliminated. For $j = 1, 2, ..., N$, we will regard the column vectors $\mathbf{g}_j$ as random vectors, thus $\boldsymbol{L}_1$ has an expected size of $m_1 \triangleq \binom{N}{2} 2^{-l_1} \approx N^2 2^{-(l_1+1)}$. This step is fulfilled by a sort-and-merge procedure as follows: First, sort the N vectors into 2^{l_1} equivalence classes according to their values on the most significant l_1 bits, thus any two vectors in the same equivalence class have the same value on these bits. Then, look at each pair of vectors $(\mathbf{g}_{j_1}, \mathbf{g}_{j_2})$ in each equivalence class to create $\boldsymbol{L}_1$.
- Create a new list $\boldsymbol{L}_2$ from $\boldsymbol{L}_1$ by further eliminating l_2 bits using the same sort-and-merge procedure as that in Step 1. That is, first sort the m_1 vectors in $\boldsymbol{L}_1$ into 2^{l_2} equivalence classes according to their values on the next most significant l_2 bits, and then look at each pair of vectors in each equivalence class to create $\boldsymbol{L}_2$. Similarly, the expected number of elements in $\boldsymbol{L}_2$ is $m_2 \triangleq \binom{m_1}{2} 2^{-l_2} \approx m_1^2 2^{-(l_2+1)}$.

Following the above steps, we make an estimation that, we obtain about m_2 4-tuples[7] $(\mathbf{g}_{j_1}, \mathbf{g}_{j_2}, \mathbf{g}_{j_3}, \mathbf{g}_{j_4})$ such that $\mathtt{high}_{l-l'}(\mathbf{g}_{j_1} \oplus \mathbf{g}_{j_2} \oplus \mathbf{g}_{j_3} \oplus \mathbf{g}_{j_4}) = 0$, which correspond to m_2 parity checks with the correlation α^4 involving only $x_0, x_1, ..., x_{l'-1}$. The running time and memory complexities of the above procedure are essentially proportional to the size of the lists that have been processed, which can be estimated as $\mathcal{O}(N + m_1)$.

[7] As illustrated in [28], there may exist some repeated tuples, whose number is comparatively quite small to the usual cases with non-repeated elements. Note that these repeated samples will not affect the processing phase of the LFSR initial state recovery, since the absolute values of the correlation of folded approximation relations in such cases is instead larger than the normal cases.