IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 2, pp. 71–103. DOI:10.46586/tosc.v2021.i2.71-103

# Comparing Large-unit and Bitwise Linear Approximations of SNOW 2.0 and SNOW 3G and Related Attacks

Xinxin Gong[1](✉) and Bin Zhang[2,1,3,4](✉)

[1] State Key Laboratory of Cryptology, P. O. Box 5159, Beijing, 100878, China,
xinxgong@126.com, martin_zhangbin@hotmail.com
[2] TCA Laboratory, SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China,
[3] University of Chinese Academy of Sciences, Beijing, 100049, China,
[4] Guizhou shujubao Network Technology Co., Ltd, Guizhou, China

**Abstract.** In this paper, we study and compare the byte-wise and bitwise linear approximations of SNOW 2.0 and SNOW 3G, and present a fast correlation attack on SNOW 3G by using our newly found bitwise linear approximations. On one side, we reconsider the relation between the large-unit linear approximation and the smaller-unit/bitwise ones derived from the large-unit one, showing that approximations on large-unit alphabets have advantages over all the smaller-unit/bitwise ones in linear attacks. But then on the other side, by comparing the byte-wise and bitwise linear approximations of SNOW 2.0 and SNOW 3G respectively, we have found many concrete examples of 8-bit linear approximations whose certain 1-dimensional/bitwise linear approximations have almost the same SEI (Squared Euclidean Imbalance) as that of the original 8-bit ones. That is, each of these byte-wise linear approximations is dominated by a single bitwise approximation, and thus the whole SEI is not essentially larger than the SEI of the dominating single bitwise approximation. Since correlation attacks can be more efficiently implemented using bitwise approximations rather than large-unit approximations, improvements over the large-unit linear approximation attacks are possible for SNOW 2.0 and SNOW 3G. For SNOW 3G, we make a careful search of the bitwise masks for the linear approximations of the FSM and obtain many mask tuples which yield high correlations. By using these bitwise linear approximations, we mount a fast correlation attack to recover the initial state of the LFSR with the time/memory/data/pre-computation complexities all upper bounded by $2^{174.16}$, improving slightly the previous best one which used an 8-bit (vectorized) linear approximation in a correlation attack with all the complexities upper bounded by $2^{176.56}$. Though not a significant improvement, our research results illustrate that we have an opportunity to achieve improvement over the large-unit attacks by using bitwise linear approximations in a linear approximation attack, and provide a new insight on the relation between large-unit and bitwise linear approximations.

**Keywords:** Stream ciphers · SNOW 3G · Bitwise linear approximation · Byte-wise linear approximation · Bitwise fast correlation attack

## 1 Introduction

A stream cipher ensures the privacy of the message transmitted over a communication channel. In such algorithms, the ciphertext is usually the XOR sum of the plaintext and the generated keystream, resembling the one-time pad primitive. Among these, the binary LFSR-based stream cipher is a classical class, while such designs have been the main target of correlation attacks [Sie84, Sie85]. With the development of modern computer


Received: 2020-11-23 Revised: 2021-03-01 Accepted: 2021-05-01 Published: 2021-06-11

science, many word-oriented stream ciphers are proposed, which are based on LFSR over the extension fields of the binary field $\mathbf{F}_2$, and a non-linear combiner, with or without memory, to generate the keystream from the underlying LFSR sequence. The typical examples include SOSEMANUK [BBC+08], SNOW 2.0 [EJ02], SNOW 3G [SAG] and SNOW-V [EJMY19].

SNOW 2.0 and SNOW 3G are both members of the SNOW family stream ciphers. SNOW 2.0 was proposed by Ekdahl and Johansson in 2002 as an improved version of SNOW 1.0 [EJ00], and selected as an ISO standard in 2005. It consists of two main components: a Linear Feedback Shift Register (LFSR) and a Finite State Machine (FSM), based on operations on 32-bit words, with high efficiency in both software and hardware environment. SNOW 3G was designed in 2006 by ETSI/SAGE, which differs from SNOW 2.0 by introducing a third 32-bit register to the FSM and a corresponding 32-bit nonlinear transformation for updating this register. SNOW 3G serves as the core of 3GPP Confidentiality and Integrity Algorithms UEA 2 & UIA2 for UMTS and LTE networks. It is currently in use in 3-4G mobile telephony systems. As a new member in the SNOW family, SNOW-V has kept most of the design from SNOW 3G in terms of the LFSR and the FSM, but both components are updated to better align with vectorized implementations.

Linear attacks have been widely used to analyze stream ciphers, and many research results have shown that SNOW ciphers are vulnerable to the class of linear approximation attacks, like distinguishing attacks and correlation attacks. The basic technique is to first approximate the nonlinear operations in the cipher and then derive a linear approximation relation involving the keystream symbols. If the linear approximation also involves symbols from the LFSR states, a correlation attack can be mounted by exploring some correlation between the keystream and the LFSR states. Fast correlation attack was first introduced by Meier and Staffelbach in 1989 by presenting two algorithms [MS89], and later evolved constantly and steadily [CT00, CJS00, CS91, CJM02, JJ99, JJ00], with wide applications to a lot of concrete constructions [LLP08, LV04, ZGM17]. However, the previous *bitwise* fast correlation attacks are not considered to work well for word-oriented stream ciphers, due to the complex form of the reduced LFSR recursion from the extension field to $\mathbf{F}_2$. As a big step, fast correlation attacks over extension fields are proposed in [ZXM15] based on linear approximations over larger alphabets, i.e., large-unit linear approximations, providing the best key recovery attack against SNOW 2.0 by using the byte-wise linear approximations. Later in [YJM19], inspired by the results of [ZXM15], a fast correlation attack on SNOW 3G is given using the method in [ZXM15] with the byte-wise linear approximations. In the design document [EJMY19] of SNOW-V, the designers present the linear approximation attacks by using the byte-wise linear approximations. All these works seem to show that the correlation attacks can be improved by using large-unit linear approximations.

**Related Work.** The resistance of SNOW 2.0 against linear approximation attacks have been widely studied. In [WBDC03], the bitwise linear approximations over two rounds of the FSM of SNOW 2.0 were constructed through linear masking method, and a distinguishing attack was given with the complexity $2^{225}$. At FSE 2006, Nyberg and Wallén [NW06] presented an improved distinguishing attack with the complexity $2^{174}$ by using the bitwise linear mask `0x00018001` for the two-round linear approximation of the FSM. Later in [LLP08], the same bitwise mask was applied to launch a correlation attack on SNOW 2.0 with the time complexity $2^{212.38}$ by using linear approximation relations between the keystream words and the LFSR states and combining the technique of fast Walsh transform (FWT). All these attacks in [WBDC03, NW06, LLP08] were launched by using the bitwise linear approximations. At CRYPTO 2015, Zhang et al. [ZXM15] introduced the terminology "large-unit" linear approximations, and mounted a fast correlation attack on SNOW 2.0 by building the two-round byte-wise (8-bit) linear approximations and adopting the $k$-tree algorithm [Wag02], giving the significantly reduced complexities all

below $2^{164.15}$. Recently, [GZ20] investigated the bitwise linear approximation of a certain type of composition function present in SNOW 2.0 and proposed a linear-time algorithm to compute the correlation for an arbitrary given linear mask. Based on this algorithm, they carried out a wider range of search for bitwise masks and found some strong linear approximations which enable them to slightly improve the data complexity of the previous fast correlation attacks by using multiple bitwise linear approximations.

Several linear attacks on SNOW 3G have been proposed. In [NW06], Nyberg and Wallén devoted one section to SNOW 3G, where the bitwise linear approximations over three rounds of the FSM were depicted, but only rough estimates of the upper bounds of their correlations were given. In [GZ20], a fast correlation attack was given with the time complexity $2^{222.33}$ by constructing bitwise linear approximations whose correlations were accurately computed. In [YJM19], inspired by the results of [ZXM15] where the large-unit approach was used to achieve improvements over the previous attacks on SNOW 2.0, Yang et al. constructed the three-round byte-wise (vectorized) linear approximations for the FSM of SNOW 3G and performed the searches for finding actual byte-wise masks that gave high SEI values for the approximations. The byte-wise linear approximations found in [YJM19] were also applied to launch a fast correlation attack against SNOW 3G by the method in [ZXM15] with all the complexities upper bounded by $2^{176.56}$.

For SNOW-V, the byte-wise linear approximation attacks and the bitwise ones are studied in [EJMY19] and [GZ21] respectively.

**Our Contributions.** In this paper, we study and compare the large-unit and bitwise linear approximations of SNOW 2.0 and SNOW 3G, and present a bitwise fast correlation attack on SNOW 3G by using our newly found bitwise linear approximations. On one hand, we first show that approximations on large-unit alphabets have advantages over all the smaller-unit/bitwise ones in linear approximation attacks, and meanwhile, the results on SNOW 2.0 in [ZXM15] gave the impression that large-unit approximations lead to larger SEI and also to better attacks. However, by studying and comparing the byte-wise and bitwise linear approximations of SNOW 2.0 and SNOW 3G, we have found many concrete examples of byte-wise linear approximations whose certain 1-dimensional/bitwise linear approximations have almost the same SEI as that of the original 8-bit ones. That is, each of these byte-wise approximations is dominated by a single bitwise approximation, and thus the whole SEI is not essentially larger than the SEI of the dominating single bitwise one. Since correlation attacks can be more efficiently implemented using bitwise approximations rather than large-unit approximations, improvements over the large-unit linear approximation attacks [ZXM15, YJM19] are possible for SNOW 2.0 and SNOW 3G. For SNOW 3G, we make a careful search of the bitwise masks for the linear approximations of the FSM and obtain many mask tuples which yield high correlations. By using these bitwise linear approximations, we mount a fast correlation attack to recover the initial state of the LFSR with the time/memory/data/pre-computation complexities all upper bounded by $2^{174.16}$, improving slightly the previous best one in [YJM19] which mounted a fast correlation attacks by using the 8-bit (vectorized) linear approximation with all the complexities upper bounded by $2^{176.56}$.

**Organization of the paper.** Some basic notations and definitions are presented in Section 2, together with the description of SNOW 2.0 and SNOW 3G. In Section 3 and Section 4, we study and compare the byte-wise and bitwise linear approximations of SNOW 2.0 and of SNOW 3G, respectively. In Section 5, we describe the detailed process on how to search for bitwise linear mask tuples for the linear approximation of SNOW 3G that yield high correlations. In Section 6, a bitwise fast correlation attack on SNOW 3G is given by using the bitwise masks. Finally, some conclusions are provided with the future work pointed out in Section 7.

# 2 Preliminaries

## 2.1 Notations and Definitions

The following notations and definitions are used throughout this paper.

- The modular addition is denoted by "$\boxplus$" and the bitwise exclusive-OR by "$\oplus$".
- The binary field is denoted by $\mathbf{F}_2$ and its $m$-dimensional extension field by $\mathbf{F}_{2^m}$. Besides, we denote by $\mathbf{F}_{2^m}^*$ the multiplicative group of nonzero elements of $\mathbf{F}_{2^m}$.
- Given two binary vectors $\mathbf{a} = (a_0, a_1, ..., a_{m-1}) \in \mathbf{F}_{2^m}$ and $\mathbf{b} = (b_0, b_1, ..., b_{m-1}) \in \mathbf{F}_{2^m}$, the standard inner product is defined as $\mathbf{a} \cdot \mathbf{b} = \bigoplus_{i=0}^{m-1} a_i b_i$, where the product $a_i b_i$ is taken in $\mathbf{F}_2$.
- Given two $m$-dimensional vectors $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_{m-1})$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{m-1})$ over $\mathbf{F}_{2^s}$ with the same defining polynomial, e.g., $p(z)$, the inner product of $\mathbf{x}$ and $\mathbf{y}$ over $\mathbf{F}_{2^s}$ is defined as $\mathbf{x} * \mathbf{y} = \bigoplus_{i=0}^{m-1} \mathbf{x}_i \mathbf{y}_i$, where $\mathbf{x}_i, \mathbf{y}_i \in \mathbf{F}_{2^s}$, and the multiplications $\mathbf{x}_i \mathbf{y}_i$ are taken over $\mathbf{F}_{2^s} = \mathbf{F}_2[z]/\langle p(z) \rangle$.
- Let $n$, $m$ be two positive integers such that $m$ divides $n$. For $\mathbf{x} \in \mathbf{F}_{2^n}$, it can be written as $\mathbf{x} = (\mathbf{x}_0 \parallel ... \parallel \mathbf{x}_{d-1})$, where $d = \frac{n}{m}$, $\mathbf{x}_i \in \mathbf{F}_{2^m}$ for $0 \leq i \leq d-1$, and $\mathbf{x}_0$ is the least significant part.
- For a set $S$, the number of elements in $S$ is denoted by $|S|$.
- An $n$-variable Boolean function $f(\mathbf{x})$ is a mapping from $\mathbf{F}_{2^n}$ to $\mathbf{F}_2$, i.e., $f : \mathbf{F}_{2^n} \to \mathbf{F}_2$.
- An $(n, m)$-function $F(\mathbf{x})$ is a mapping from $\mathbf{F}_{2^n}$ to $\mathbf{F}_{2^m}$, i.e., $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$ such that $\mathbf{x} \mapsto (f_0, ..., f_{m-1})$, where $f_i$s are $n$-variable Boolean functions, called the coordinate functions of $F$. $F$ is also called an $m$-dimensional vectorial Boolean function.
- Given an $(n, m)$-function $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$, and a nonzero vector $\mathbf{v} = (v_0, ..., v_{m-1}) \in \mathbf{F}_{2^m}$, the $n$-variable Boolean function $\mathbf{v} \cdot F$ defined as $\mathbf{v} \cdot F(\mathbf{x}) = v_0 f_0(\mathbf{x}) \oplus \cdots \oplus v_{m-1} f_{m-1}(\mathbf{x})$ is called a (non-zero) component of $F$. In our analysis, the Boolean function $\mathbf{v} \cdot F$ is denoted by $F^{\mathbf{v}}$ for $\mathbf{v} \in \mathbf{F}_{2^m}$ and $\mathbf{v} \neq \mathbf{0}$.

**Definition 1.** Let $X$ be a binary random variable, the correlation between $X$ and zero is defined as $\epsilon(X) = \Pr\{X = 0\} - \Pr\{X = 1\}$. Given a Boolean function $f : \mathbf{F}_{2^n} \to \mathbf{F}_2$, the correlation of $f$ to zero is defined as

$$\begin{aligned} \epsilon(f) &= 2^{-n}(|\{\mathbf{x} \in \mathbf{F}_{2^n} : f(\mathbf{x}) = 0\}| - |\{\mathbf{x} \in \mathbf{F}_{2^n} : f(\mathbf{x}) = 1\}|) \\ &= \Pr\{f(X) = 0\} - \Pr\{f(X) = 1\}, \end{aligned}$$

where $X$ is a uniformly distributed random variable in $\mathbf{F}_{2^n}$.

Note that "correlation" is often used to evaluate the efficiency of *bitwise* linear approximations in a linear approximation attack, where the data complexity is proportional to $1/\epsilon^2(f)$.

**Definition 2.** The correlation of an $(n, m)$-function $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$ with a linear output mask $\mathbf{\Gamma} \in \mathbf{F}_{2^m}$ and a linear input mask $\mathbf{\Lambda} \in \mathbf{F}_{2^n}$ is defined as

$$\epsilon_F(\mathbf{\Gamma}; \mathbf{\Lambda}) = \Pr\{\mathbf{\Gamma} \cdot F(X) = \mathbf{\Lambda} \cdot X\} - \Pr\{\mathbf{\Gamma} \cdot F(X) \neq \mathbf{\Lambda} \cdot X\},$$

where $X$ is a uniformly distributed random variable in $\mathbf{F}_{2^n}$.

At CRYPTO 2015, Zhang et al. [ZXM15] introduced the terminology "large-unit" linear approximations, and achieved improvements over the previous attacks on SNOW 2.0 by providing a fast correlation attack over $\mathbf{F}_{2^8}$ using the byte-wise (8-bit) linear approximations. Given an $(n, m)$-function $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$, the probability distribution $D_F$ of $F$ is

$$D_F(\mathbf{a}) = \frac{(|\{\mathbf{x} \in \mathbf{F}_{2^n} : F(\mathbf{x}) = \mathbf{a}\}|)}{2^n}, \text{ for all } \mathbf{a} \in \mathbf{F}_{2^m}$$

Here comes the definition of *Squared Euclidean Imbalance* (SEI), which is usually used to evaluate the efficiency of a large-unit linear approximation in a linear approximation attack.

**Definition 3.** The *Squared Euclidean Imbalance (SEI)* of a distribution $D_F$ is defined as

$$\Delta(D_F) = 2^m \sum_{\mathbf{a} \in \mathbf{F}_{2^m}} \left( D_F(\mathbf{a}) - \frac{1}{2^m} \right)^2,$$

which measures the distance between the target distribution and the uniform distribution. Especially for $m = 1$, $\Delta(D_F)$ is closely related to the correlation of $F$ by $\Delta(D_F) = \epsilon^2(F)$.

Note that the "SEI" of a distribution $D_F$ over a general alphabet is used to evaluate the efficiency of *large-unit* linear approximations in a linear approximation attack, where the data complexity is proportional to the value of $1/\Delta(D_F)$. Besides, there is a fundamental fact about the SEI of a distribution [BJV04, NH07] shown as follows.

**Lemma 1.** *For an $(n, m)$-function $F$ with the probability distribution vector $D_F$, we have*

$$\Delta(D_F) = \sum_{\mathbf{v} \in \mathbf{F}_{2^m}^*} \epsilon^2(F^{\mathbf{v}}) = \sum_{\mathbf{v} \in \mathbf{F}_{2^m}^*} \Delta(D_{F^{\mathbf{v}}}).$$

For brevity, we adopt the simplified notation $\Delta(F)$ to denote $\Delta(D_F)$ hereafter. Then we have $\Delta(F) = \epsilon^2(F)$ when $m = 1$, and $\Delta(F) = \sum_{\mathbf{v} \in \boldsymbol{F}_{2^m}^*} \epsilon^2(F^{\mathbf{v}}) = \sum_{\mathbf{v} \in \boldsymbol{F}_{2^m}^*} \Delta(F^{\mathbf{v}})$. With the notations we just introduced, we now study the relation between a large-unit linear approximation and some certain smaller-unit/bitwise linear approximations derived from the large-unit one.

Let $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$ be a large-unit linear approximation relation such that $\mathbf{x} \mapsto (f_0, ..., f_{m-1})$. Let $\mathtt{low}_l(\mathbf{x})$ be the $l$ least significant bits of $\mathbf{x}$. For $m > 1$ and $1 \leq m' < m$, we define another linear approximation relation $F^{(m')}$ according to $F$ as[1]

$$\begin{aligned} F^{(m')} : \mathbf{F}_{2^n} &\to \mathbf{F}_{2^{m'}} \\ \mathbf{x} &\mapsto \mathtt{low}_{m'}(F(\mathbf{x})) \end{aligned}$$

Then we call $F^{(m')}$ a smaller-unit linear approximation function, which is actually the low-dimensional projective function derived from $F$. According to Lemma 1, we get that the SEI of the distribution $D_F$ of $F$ is equal to the sum of the SEIs of the distributions $D_{F^{\mathbf{v}}}$, where $\mathbf{v}$ runs through all non-zero vectors in $\mathbf{F}_{2^m}$. When $1 \leq m' < m$, the SEI of the distribution $D_{F^{(m')}}$ is a partial sum consisting of those SEIs of the distributions $D_{F^{\mathbf{v}}}$, where $\mathbf{v}$ has non-zero terms only in the $m'$ least significant bits. The following conclusion then follows immediately.

**Property 1.** Let $F$ be an $m$-bit ($m > 1$) large-unit linear approximation, and $F^{(m')}$ be an $m'$-bit ($1 \leq m' < m$) linear approximation derived from $F$ as above. Then for any integer $m' : 1 \leq m' < m$, we have $\Delta(F^{(m')}) \leq \Delta(F)$.

[1] Note that for a given $m'$, the choice of the least significant bits is not crucial but is introduced here to simplify the presentation.

Property 1 shows a theoretical relation between different size linear approximations. Since the data complexity in a linear approximation attack is proportional to the value of $1/\Delta(F)$, Property 1 seems to suggest that the larger the unit, the better complexity result we can get in a linear approximation attack.

For the large-unit linear approximation relation $F : \mathbf{F}_{2^n} \to \mathbf{F}_{2^m}$, we can get that $F^{\mathbf{v}} : \mathbf{F}_{2^n} \to \mathbf{F}_2$ is a bitwise linear approximation for any nonzero vector $\mathbf{v} \in \mathbf{F}_{2^m}$. The following conclusion follows directly from Lemma 1, which depicts the relation between the distributions of any bitwise linear approximations and that of the original large-unit one.

**Property 2.** Let $F$ be an $m$-bit ($m > 1$) large-unit linear approximation, and $F^{\mathbf{v}}$ be the bitwise linear approximation derived from $F$ as above. Then for any nonzero vector $\mathbf{v} \in \mathbf{F}_{2^m}$, we have $\Delta(F^{\mathbf{v}}) \leq \Delta(F)$.

Property 2 seems to suggest that approximations on large-unit alphabets have advantages over all the bitwise ones in linear approximation attacks. However, as shown in Section 3.3 and Section 4.3 for the linear approximations of SNOW 2.0 and SNOW 3G, we have found that there are many concrete examples of byte-wise linear approximations whose certain 1-dimensional/bitwise linear approximations have almost the same SEI as that of the original large-unit ones. Since correlation attacks can be more efficiently implemented using bitwise approximations rather than large-unit approximations, improvements over large-unit linear approximation attacks [ZXM15, YJM19] are possible for SNOW 2.0 and SNOW 3G.

## 2.2 Description of SNOW 2.0

Both SNOW 2.0 and SNOW 3G are word-oriented stream ciphers and contain two main components: a Linear Feedback Shift Register (LFSR) and a Finite State Machine (FSM). SNOW 3G differs from SNOW 2.0 by introducing a third 32-bit register to the FSM and a corresponding 32-bit nonlinear transformation for updating this register. The keystream generation phase of SNOW 2.0 is depicted in Fig.1. For more details on the design, please refer to the original design documents [EJ02].

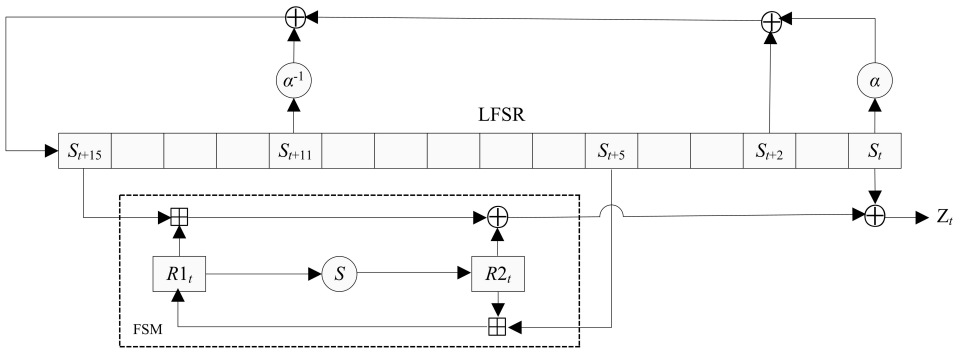


**Figure 1:** The keystream generation phase of SNOW 2.0

The LFSR part of SNOW 2.0 consists of 16 words of length 32 bits each, with the feedback polynomial

$$\alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1 \in \mathbf{F}_{2^{32}}[x],$$

where $\alpha \in \mathbf{F}_{2^{32}}$ is a root of the primitive polynomial $y^4 + \beta^{23} y^3 + \beta^{245} y^2 + \beta^{48} y + \beta^{239} \in \mathbf{F}_{2^8}[y]$, and $\beta$ is a root of the polynomial $z^8 + z^7 + z^5 + z^3 + 1 \in \mathbf{F}_2[z]$ (field constant `0xA9`).

Let $(\mathbf{s}_{t+15}, \mathbf{s}_{t+14}, ..., \mathbf{s}_t)$, $\mathbf{s}_{t+i} \in \mathbf{F}_{2^{32}}$, denote the LFSR state at time $t$. For the FSM part, it has two 32-bit registers $R1$ and $R2$. The LFSR state feeds into the FSM with

the input $(\mathbf{s}_{t+15}, \mathbf{s}_{t+5})$, and the output of the FSM is $\mathbf{F}_t = (\mathbf{s}_{t+15} \boxplus \mathbf{R1}_t) \oplus \mathbf{R2}_t$. The keystream $\mathbf{z}_t$ is generated by $\mathbf{z}_t = \mathbf{F}_t \oplus \mathbf{s}_t$. The registers $R1$ and $R2$ are updated according to

$$\begin{aligned} \mathbf{R1}_{t+1} &= \mathbf{s}_{t+5} \boxplus \mathbf{R2}_t, \\ \mathbf{R2}_{t+1} &= \mathrm{S}(\mathbf{R1}_t), \end{aligned}$$

where $\mathrm{S}(\cdot)$ is a 32-bit to 32-bit mapping composed of four parallel AES S-boxes (denoted by $S_R$) followed by the AES MixColumn operation (denoted by $\mathbf{M}_1$). Here $\mathbf{M}_1$ is defined over $\mathbf{F}_{2^8}$ with the polynomial $z_1^8 + z_1^4 + z_1^3 + z_1 + 1 \in \mathbf{F}_2[z_1]$ (field constant `0x1B`) for field multiplication. Precisely, let $\mathbf{w} = (\mathbf{w}_0 \parallel \mathbf{w}_1 \parallel \mathbf{w}_2 \parallel \mathbf{w}_3)$, $\mathbf{w}_i \in \mathbf{F}_{2^8}$, be the 32-bit input to $\mathrm{S}(\cdot)$, then $\mathrm{S}(\mathbf{w})$ is computed as

$$\mathrm{S}(\mathbf{w}) = \mathbf{M}_1 * \begin{pmatrix} S_R(\mathbf{w}_0) \\ S_R(\mathbf{w}_1) \\ S_R(\mathbf{w}_2) \\ S_R(\mathbf{w}_3) \end{pmatrix} = \begin{pmatrix} z_1 & z_1+1 & 1 & 1 \\ 1 & z_1 & z_1+1 & 1 \\ 1 & 1 & z_1 & z_1+1 \\ z_1+1 & 1 & 1 & z_1 \end{pmatrix} * \begin{pmatrix} S_R(\mathbf{w}_0) \\ S_R(\mathbf{w}_1) \\ S_R(\mathbf{w}_2) \\ S_R(\mathbf{w}_3) \end{pmatrix},$$

where the operation "$*$" is taken over $\mathcal{G}_1 = \mathbf{F}_2[z_1]/\langle z_1^8 + z_1^4 + z_1^3 + z_1 + 1 \rangle$.

## 2.3 Description of SNOW 3G

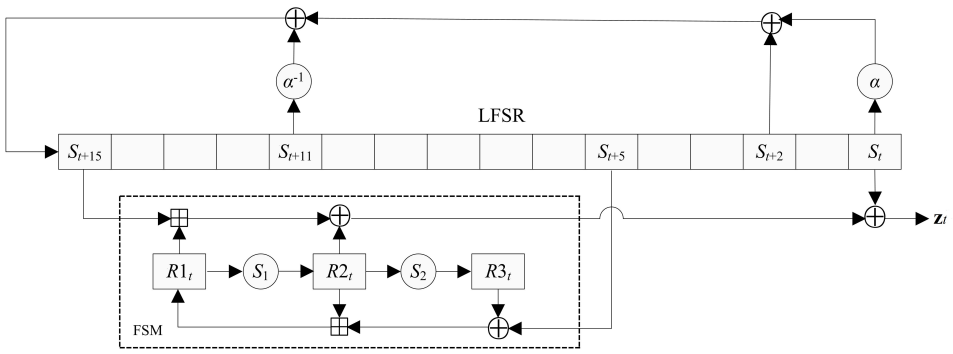


**Figure 2:** The keystream generation phase of SNOW 3G

The keystream generation phase of SNOW 3G is depicted in Fig.2. For more details on the design, please refer to the original design documents [SAG]. SNOW 3G preserves all features of SNOW 2.0, but adds a third register $R3$ to the FSM and a transformation $\mathrm{S}_2$. The FSM part has three 32-bit registers $R1$, $R2$ and $R3$. The output of the FSM is $\mathbf{F}_t = (\mathbf{s}_{t+15} \boxplus \mathbf{R1}_t) \oplus \mathbf{R2}_t$. The keystream $\mathbf{z}_t$ is generated by $\mathbf{z}_t = \mathbf{F}_t \oplus \mathbf{s}_t$. The registers $R1$, $R2$ and $R3$ are updated according to

$$\begin{aligned} \mathbf{R1}_{t+1} &= (\mathbf{s}_{t+5} \oplus \mathbf{R3}_t) \boxplus \mathbf{R2}_t, \\ \mathbf{R2}_{t+1} &= \mathrm{S}_1(\mathbf{R1}_t), \\ \mathbf{R3}_{t+1} &= \mathrm{S}_2(\mathbf{R2}_t), \end{aligned}$$

where $\mathrm{S}_1(\cdot) = \mathrm{S}(\cdot)$, and $\mathrm{S}_2(\cdot)$ is another 32-bit to 32-bit mapping composed of four parallel 8-bit to 8-bit substitutions (denoted by $S_Q$) followed by the AES MixColumn operation (denoted by $\mathbf{M}_2$). Note that $\mathbf{M}_2$ is defined over $\mathbf{F}_{2^8}$ with the polynomial $z_2^8 + z_2^6 + z_2^5 + z_2^3 + 1 \in \mathbf{F}_2[z_2]$ (field constant `0x69`) for field multiplication, which is different from that used in the transformation $\mathrm{S}_1$ (= S), i.e., for $\mathbf{w} = (\mathbf{w}_0 \parallel \mathbf{w}_1 \parallel \mathbf{w}_2 \parallel \mathbf{w}_3)$,

$\mathbf{w}_i \in \mathbf{F}_{2^8}$, $\mathrm{S}_2(\mathbf{w})$ is computed as

$$\mathrm{S}_2(\mathbf{w}) = \mathbf{M}_2 * \begin{pmatrix} S_Q(\mathbf{w}_0) \\ S_Q(\mathbf{w}_1) \\ S_Q(\mathbf{w}_2) \\ S_Q(\mathbf{w}_3) \end{pmatrix} = \begin{pmatrix} z_2 & z_2+1 & 1 & 1 \\ 1 & z_2 & z_2+1 & 1 \\ 1 & 1 & z_2 & z_2+1 \\ z_2+1 & 1 & 1 & z_2 \end{pmatrix} * \begin{pmatrix} S_Q(\mathbf{w}_0) \\ S_Q(\mathbf{w}_1) \\ S_Q(\mathbf{w}_2) \\ S_Q(\mathbf{w}_3) \end{pmatrix},$$

where the operation "$*$" is taken over $\mathcal{G}_2 = \mathbf{F}_2[z_2]/\langle z_2^8 + z_2^6 + z_2^5 + z_2^3 + 1\rangle$.

# 3 Comparing Large-unit and Bitwise Linear Approximations of SNOW 2.0

In this section, we frist make a recap of the previous bitwise linear approximations of SNOW 2.0 used in [WBDC03, NW06, LLP08, GZ20, FTIM18], then review on the byte-wise linear approximations in [ZXM15], and finally illustrate the relationship between the bitwise and byte-wise linear approximations by some concrete examples.

## 3.1 Recap on the Bitwise Linear Approximations of the FSM

In [WBDC03], the linear masking method was applied to SNOW 2.0, and the bitwise linear approximations over two rounds of the FSM were constructed, as depicted in Fig.3. In [WBDC03], it is always assumed that all masks $\mathbf{\Gamma}_i \in \mathbf{F}_2^{32}$ used in the linear approximations in Fig.3 have the same value. Denote all the masks by $\mathbf{\Gamma}$, the bitwise linear approximations of the FSM take the following form:

$$\mathbf{\Gamma} \cdot \mathbf{z}_t \oplus \mathbf{\Gamma} \cdot \mathbf{z}_{t+1} = \mathbf{\Gamma} \cdot \mathbf{s}_t \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+1} \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+5} \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+15} \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+16} \oplus n^{(t)},$$

where $n^{(t)}$ is the binary noise.

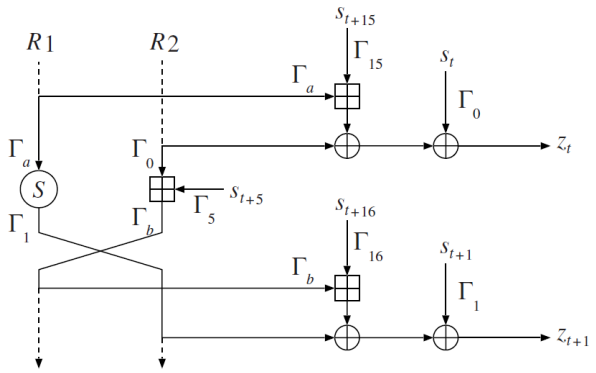


**Figure 3:** The two-round bitwise linear approximations for the FSM of SNOW 2.0

At FSE 2006, Nyberg and Wallén [NW06] considered the case when the output masks at time $t$ and $t+1$ are different, and used the following bitwise linear approximation relation (1) four times according to the feedback polynomial of the LFSR to build the linear distinguisher: two times with the mask tuple $(\mathbf{\Gamma}, \mathbf{\Lambda})$ at time $t+2$ and $t+16$, then once with the mask tuple $(\mathbf{\Gamma}\alpha, \mathbf{\Lambda}\alpha)$ at time $t$, and finally once with the mask tuple $(\mathbf{\Gamma}\alpha^{-1}, \mathbf{\Lambda}\alpha^{-1})$ at time $t+11$.

$$\mathbf{\Gamma} \cdot \mathbf{z}_t \oplus \mathbf{\Lambda} \cdot \mathbf{z}_{t+1} = \mathbf{\Gamma} \cdot \mathbf{s}_t \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+1} \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+5} \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+15} \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+16} \oplus n^{(t)}. \quad (1)$$

Let $\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}, \mathbf{\Lambda})$ denote the correlation of the linear approximation relation (1) under the bitwise mask tuple $(\mathbf{\Gamma}, \mathbf{\Lambda})$. The total correlation for the linear distinguisher is calculated by combining these four approximations as $\epsilon^2_{\mathrm{FSM}}(\mathbf{\Gamma}, \mathbf{\Lambda}) \cdot \epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}\alpha, \mathbf{\Lambda}\alpha) \cdot \epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}\alpha^{-1}, \mathbf{\Lambda}\alpha^{-1})$ according to the Piling-up Lemma. By using the bitwise masks $\mathbf{\Gamma} = \mathbf{\Lambda}$=0x00018001 for approximating the FSM, they constructed their best linear distinguisher and presented a distinguishing attack accordingly. When $\mathbf{\Gamma} = \mathbf{\Lambda}$=0x00018001, they obtained $\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}, \mathbf{\Lambda}) = 2^{-14.496}$, $\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}\alpha, \mathbf{\Lambda}\alpha) = 2^{-26.676}$ and $\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}\alpha^{-1}, \mathbf{\Lambda}\alpha^{-1}) = 2^{-30.221}$, and thus the correlation of their best linear distinguisher is $2^{-85.89}$. Using this mask tuple, a distinguishing attack on SNOW 2.0 was given with time complexity $2^{174}$, given $2^{174}$ keystream words. Later in [LLP08], the same bitwise masks $\mathbf{\Gamma} = \mathbf{\Lambda}$ =0x00018001 for the linear approximation (1) was applied to launch a correlation attack on SNOW 2.0 with the time complexity $2^{212.38}$, given $2^{193.77}$ keystream words.

**Table 1:** The best bitwise mask tuples $(\mathbf{\Gamma}, \mathbf{\Lambda})$ for the linear approximation (1) in [GZ20]

| No. | Ref. | $\mathbf{\Gamma} = \mathbf{\Lambda}$ | $\log(\lvert\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma}, \mathbf{\Lambda})\rvert)$ |
|---|---|---|---|
| (1) | [FTIM18, GZ20] | 0x01800001 | $-14.411$ |
| (2) | [NW06, GZ20] | 0x00018001 | $-14.496$ |
| (3) | [GZ20] | 0x00010081 | $-14.635$ |

Recently, [GZ20] investigated the bitwise linear approximation of a certain type of composition function present in SNOW 2.0 and proposed a linear-time algorithm for computing the correlation. Based on this algorithm, they found some strong bitwise mask tuples for the linear approximation (1) yielding high correlations, of which the best three ones were listed in Table 1. Note that the mask tuple $\mathbf{\Gamma} = \mathbf{\Lambda}$ =0x01800001 numbered (1) in Table 1 is the same as the one in [FTIM18], which is found by using the MILP-aided automatic search algorithm, and $\mathbf{\Gamma} = \mathbf{\Lambda}$ = 0x00018001 numbered (2) is the same as the one found in [NW06]. They also applied these bitwise masks to launch a fast correlation attack with the total time complexity $2^{162.86}$ and data complexity $2^{159.62}$, improving the previous fast correlation attacks in [ZXM15, FTIM18].

## 3.2 Recap on the Byte-wise Linear Approximations of the FSM

In [ZXM15], the large-unit approach was used to achieve improvements over the previous attacks on SNOW 2.0, where two-round byte-wise linear approximations for the FSM of SNOW 2.0 were constructed, as depicted in Fig.4.

Let $\mathbf{T} = (\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3)$ and $\mathbf{N} = (\mathbf{N}_0, \mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3)$ be the *4-byte* linear masks defined over the AES MixColumn field $\mathbf{F}_{2^8}$, i.e., $\mathcal{G}_1 \triangleq \mathbf{F}_2[z_1]/\langle z_1^8 + z_1^4 + z_1^3 + z_1 + 1\rangle$, where $\mathbf{T}_0$ and $\mathbf{N}_0$ are the least significant bytes. First, $\mathbf{T}$ and $\mathbf{N}$ are applied to $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ respectively as follows:

$$\mathbf{T} * (\mathbf{z}_t \oplus \mathbf{s}_t) = \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathbf{R1}_t) \oplus \mathbf{T} * \mathbf{R2}_t,$$
$$\mathbf{N} * (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1}) = \mathbf{N} * (\mathbf{s}_{t+16} \boxplus \mathbf{s}_{t+5} \boxplus \mathbf{R2}_t) \oplus \mathbf{N} * \mathrm{S}(\mathbf{R1}_t),$$

where "$*$" is operated in $\mathcal{G}_1$. Then, the following two byte-wise linear approximations are used,

$$\mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathbf{R1}_t) = \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N} * \mathrm{S}(\mathbf{R1}_t) \oplus \mathbf{n}_1^{(t)},$$
$$\mathbf{N} * (\mathbf{s}_{t+16} \boxplus \mathbf{s}_{t+5} \boxplus \mathbf{R2}_t) = \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{N} * \mathbf{s}_{t+5} \oplus \mathbf{T} * \mathbf{R2}_t \oplus \mathbf{n}_2^{(t)},$$

where $\mathbf{n}_1^{(t)}$ and $\mathbf{n}_2^{(t)}$ are the 8-bit noises introduced by these approximations. Let $\mathbf{W}_t =$

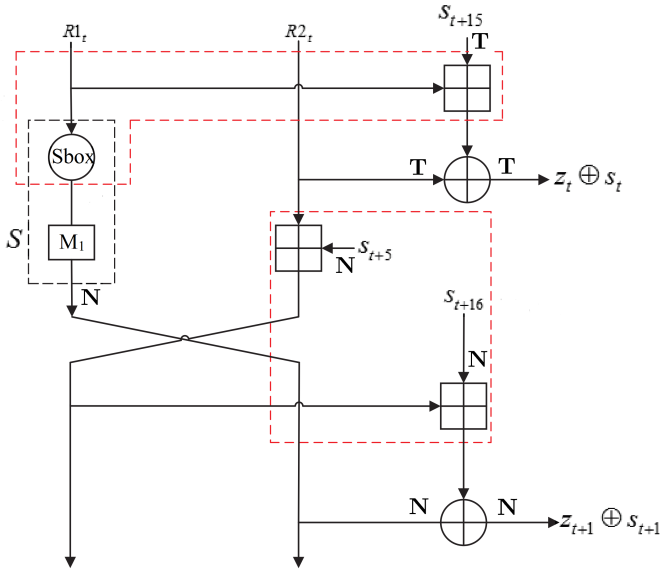


**Figure 4:** The two-round byte-wise linear approximations for the FSM of SNOW 2.0

$\text{Sbox}(\mathbf{R1}_t)$, then $\mathbf{R1}_t = \text{Sbox}^{-1}(\mathbf{W}_t)$ and $\text{S}(\mathbf{R1}_t) = \mathbf{M}_1 * \mathbf{W}_t$. Let $\mathbf{N}' = \mathbf{N} * \mathbf{M}_1$, then

$$\mathbf{n}_1^{(t)} = \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \text{Sbox}^{-1}(\mathbf{W}_t)) \oplus \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N}' * \mathbf{W}_t,$$
$$\mathbf{n}_2^{(t)} = \mathbf{N} * (\mathbf{s}_{t+16} \boxplus \mathbf{s}_{t+5} \boxplus \mathbf{R2}_t) \oplus \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{N} * \mathbf{s}_{t+5} \oplus \mathbf{T} * \mathbf{R2}_t.$$

Let $\mathbf{n}^{(t)} = \mathbf{n}_1^{(t)} \oplus \mathbf{n}_2^{(t)}$ be the folded noise introduced by the above two linear approximations, the following *byte-wise* linear approximations for the FSM of SNOW 2.0 are obtained

$$\mathbf{T} * \mathbf{z}_t \oplus \mathbf{N} * \mathbf{z}_{t+1} = \mathbf{T} * \mathbf{s}_t \oplus \mathbf{N} * \mathbf{s}_{t+1} \oplus \mathbf{N} * \mathbf{s}_{t+5} \oplus \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{n}^{(t)}. \quad (2)$$

It should be noted that the operation "$*$" in the linear approximation (2) and also the byte-wise masks $\mathbf{T}$, $\mathbf{N}$ are defined in the AES MixColumn field which is denoted by $\mathcal{G}_1$, while the state elements $\mathbf{s}_{t+i}$ are generated by the LFSR defined by another polynomial $z^8 + z^7 + z^5 + z^3 + 1 \in \mathbf{F}_2[z]$. Thus it is necessary to unify the two fields for an efficient decoding. In [ZXM15], a general routine is described to solve this problem, where they try to find an equivalent representation of the LFSR part theoretically so that it is defined over the new $\mathbf{F}_{2^{32}}$ field. In this paper, as illustrated in Section 4.2 for SNOW 3G case, we describe in more details on how to unify the fields defined by different polynomials from a different perspective.

In [ZXM15], two algorithms are provided to compute the distributions[2] of $\mathbf{n}_1$ and $\mathbf{n}_2$ over the AES MixColumn field $\mathbf{F}_{2^8}$ with the complexities $2^{26.58}$ and $2^{33.58}$ respectively for each given byte-wise mask tuple. In the following parts, we will provide two slightly improved algorithms to compute these distributions, whose complexities are $2^{20.25}$ and $2^{27.33}$ respectively.

[2]Since the distributions of $\mathbf{n}_1^{(t)}$, $\mathbf{n}_2^{(t)}$ and $\mathbf{n}^{(t)}$ are independent of the time instance $t$, we simplify them by writing $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}$ respectively.

### 3.2.1 Improving the Computation of the Distribution of $\mathbf{n_1}$

For the given 4-byte masks $\mathbf{T}$ and $\mathbf{N}$, let $\mathbf{N}' = \mathbf{N} * \mathbf{M}_1$, the noise variable $\mathbf{n}_1$ can be expressed as

$$\mathbf{n}_1 = \mathbf{T} * (\mathbf{X} \boxplus \mathrm{Sbox}^{-1}(\mathbf{Y})) \oplus \mathbf{T} * \mathbf{X} \oplus \mathbf{N}' * \mathbf{Y},$$

where $\mathbf{X}$ and $\mathbf{Y}$ are the uniformly distributed random variables in $\mathbf{F}_{2^{32}}$, which can be written into 4 bytes as $\mathbf{X} = (\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)$, $\mathbf{Y} = (\mathbf{Y}_0, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_3)$. Following the basic idea of Algorithm 3 in [ZXM15], we first split the expression of $\mathbf{n}_1$ into 4 sub-expressions $\mathbf{n}_1^j$ $(j = 0, 1, 2, 3)$ as follows:

$$\begin{aligned}
\mathbf{n}_1^0 =& \mathbf{0} \oplus \mathbf{T}_0 * (\mathbf{X}_0 \boxplus_8 S_R^{-1}(\mathbf{Y}_0) \boxplus_8 0) \oplus \mathbf{T}_0 * \mathbf{X}_0 \oplus \mathbf{N}'_0 * \mathbf{Y}_0,\\
\mathbf{n}_1^1 =& \mathbf{n}_1^0 \oplus \mathbf{T}_1 * (\mathbf{X}_1 \boxplus_8 S_R^{-1}(\mathbf{Y}_1) \boxplus_8 cr_0) \oplus \mathbf{T}_1 * \mathbf{X}_1 \oplus \mathbf{N}'_1 * \mathbf{Y}_1,\\
\mathbf{n}_1^2 =& \mathbf{n}_1^1 \oplus \mathbf{T}_2 * (\mathbf{X}_2 \boxplus_8 S_R^{-1}(\mathbf{Y}_2) \boxplus_8 cr_1) \oplus \mathbf{T}_2 * \mathbf{X}_2 \oplus \mathbf{N}'_2 * \mathbf{Y}_2,\\
\mathbf{n}_1^3 =& \mathbf{n}_1^2 \oplus \mathbf{T}_3 * (\mathbf{X}_3 \boxplus_8 S_R^{-1}(\mathbf{Y}_3) \boxplus_8 cr_2) \oplus \mathbf{T}_3 * \mathbf{X}_3 \oplus \mathbf{N}'_3 * \mathbf{Y}_3 = \mathbf{n}_1,
\end{aligned}$$

where "$\boxplus_8$" represents the addition modulo $2^8$, and $cr_j \in \{0, 1\}$ are local carries introduced by the addition modulo $2^{32}$ such that $cr_j = \lfloor (\mathbf{X}_j + S_R^{-1}(\mathbf{Y}_j) + cr_{j-1})/2^8 \rfloor$ for $j = 0, 1, 2, 3$ ($cr_{-1} = 0$ by default) with "$\lfloor\ \rfloor$" being the floor function of integers. The sub-expressions are connected with each other by the one direction information propagation from the least significant $\mathbf{n}_1^0$ to the most significant $\mathbf{n}_1^3$, caused by the local carries introduced by the addition modulo $2^{32}$ and the output of $\mathbf{n}_1^j$.

Based on the above, we describe our algorithm for computing the distribution of $\mathbf{n}_1$ as follows. We first describe a sub-algorithm called *ComputeMatrix*($cr_{j-1}$, $\mathbf{T}_j$, $\mathbf{N}'_j$), which computes the values of $\mathbf{r} = \mathbf{T}_j * (\mathbf{X}_j \boxplus_8 S_R^{-1}(\mathbf{Y}_j) \boxplus_8 cr_{j-1}) \oplus \mathbf{T}_j * \mathbf{X}_j \oplus \mathbf{N}'_j * \mathbf{Y}_j$ and the carries $cr_j = \lfloor (\mathbf{X}_j + S_R^{-1}(\mathbf{Y}_j) + cr_{j-1})/2^8 \rfloor$ for all $\mathbf{X}_j \in \mathbf{F}_{2^8}$ and $\mathbf{Y}_j \in \mathbf{F}_{2^8}$, and stores all the output and carry information in a $256 \times 2$ matrix **Mat**. Then we present Algorithm 1 to compute the distribution of $\mathbf{n}_1$ by using the sub-algorithm *ComputeMatrix*.

*ComputeMatrix*($cr_{j-1}$, $\mathbf{T}_j$, $\mathbf{N}'_j$)

**Parameters**: the partial masks $\mathbf{T}_j$, $\mathbf{N}'_j$, and the local carry $cr_{j-1} \in \{0, 1\}$

1: Create a $256 \times 2$ matrix **Mat** initialized with zeros;
2: **for** $\mathbf{X}_j = 0, 1, ..., 255$ and $\mathbf{Y}_j = 0, 1, ..., 255$ **do**
3: Compute $\mathbf{r} = \mathbf{T}_j * (\mathbf{X}_j \boxplus_8 S_R^{-1}(\mathbf{Y}_j) \boxplus_8 cr_{j-1}) \oplus \mathbf{T}_j * \mathbf{X}_j \oplus \mathbf{N}'_j * \mathbf{Y}_j$;
4: Compute $cr_j = \lfloor (\mathbf{X}_j + S_R^{-1}(\mathbf{Y}_j) + cr_{j-1})/2^8 \rfloor$;
5: $\mathbf{Mat}[\mathbf{r}][cr_j] + +$;
6: **end for**
7: $\mathbf{Mat} \leftarrow \mathbf{Mat}/(2^8)^2$;

**Output**: the $256 \times 2$ matrix **Mat**.

**Algorithm 1** Compute the distribution of $\mathbf{n}_1$ over the AES MixColumn field $\mathbf{F}_{2^8}$

**Parameters**: the masks $\mathbf{T}$ and $\mathbf{N}'$, where $\mathbf{N}' = \mathbf{N} * \mathbf{M}_1$

1: Prepare three $256 \times 2$ matrices $\mathbf{Mat}$, $\mathbf{Mat}_1$ and $\mathbf{Mat}_2$;
2: $\mathbf{Mat}_1 \leftarrow$ ComputeMatrix(0, $\mathbf{T}_0$, $\mathbf{N}'_0$);
3: **for** $j = 1, 2, 3$ **do**
4: Initialize $\mathbf{Mat}_2$ with zeros;
5: **for** $cr_{j-1} = 0, 1$ **do**
6: $\mathbf{Mat} \leftarrow$ ComputeMatrix($cr_{j-1}$, $\mathbf{T}_j$, $\mathbf{N}'_j$);
7: **for** $\mathbf{n}_1^{j-1} = 0, 1, ..., 255$, $\mathbf{n}_1^j = 0, 1, ..., 255$, and $cr_j = 0, 1$ **do**
8: $\mathbf{Mat}_2[\mathbf{n}_1^j][cr_j] = \mathbf{Mat}_2[\mathbf{n}_1^j][cr_j] + \mathbf{Mat}[\mathbf{n}_1^{j-1} \oplus \mathbf{n}_1^j][cr_j] \times \mathbf{Mat}_1[\mathbf{n}_1^{j-1}][cr_{j-1}]$;
9: **end for**
10: **end for**
11: $\mathbf{Mat}_1 \leftarrow \mathbf{Mat}_2$;
12: **end for**

**Output**: $\Pr\{\mathbf{n}_1 = \mathbf{a}\} = \mathbf{Mat}_2[\mathbf{a}][0] + \mathbf{Mat}_2[\mathbf{a}][1]$, for $\mathbf{a} = 0, 1, ..., 255$.

We emphasize that the basic idea of our method for computing the distribution of $\mathbf{n}_1$ is the same as that in [ZXM15], i.e., we use the connection matrix to characterize the one direction information propagation caused by the modular addition. But we carry out the process in a slightly different way.

**Complexity.** The complexity of the sub-algorithm *ComputeMatrix* is $2^{16}$. Thus Algorithm 1 has the complexity around $2^{16} + 3 \times 2 \times (2^{16} + 2^{16} \times 2) = 2^{20.25}$, slightly improving that in [ZXM15] whose time complexity is $2^{26.58}$.

### 3.2.2 Improving the Computation of the Distribution of $\mathbf{n_2}$

For the given 4-byte masks $\mathbf{T}$ and $\mathbf{N}$, the noise variable $\mathbf{n}_2$ can be expressed as

$$\mathbf{n}_2 = \mathbf{N} * (\mathbf{X} \boxplus \mathbf{Y} \boxplus \mathbf{Z}) \oplus \mathbf{N} * \mathbf{X} \oplus \mathbf{N} * \mathbf{Y} \oplus \mathbf{T} * \mathbf{Z},$$

where $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ are uniformly distributed random variables in $\mathbf{F}_{2^{32}}$. As was done in Section 3.2.1, we write the 32-bit values in $\mathbf{n}_2$ into 4 bytes, and split it into 4 sub-expressions as follows:

$$\begin{aligned}
\mathbf{n}_2^0 &= \mathbf{0} \oplus \mathbf{N}_0 * (\mathbf{X}_0 \boxplus_8 \mathbf{Y}_0 \boxplus_8 \mathbf{Z}_0 \boxplus_8 0) \oplus \mathbf{N}_0 * \mathbf{X}_0 \oplus \mathbf{N}_0 * \mathbf{Y}_0 \oplus \mathbf{T}_0 * \mathbf{Z}_0, \\
\mathbf{n}_2^1 &= \mathbf{n}_2^0 \oplus \mathbf{N}_1 * (\mathbf{X}_1 \boxplus_8 \mathbf{Y}_1 \boxplus_8 \mathbf{Z}_1 \boxplus_8 cr_0) \oplus \mathbf{N}_1 * \mathbf{X}_1 \oplus \mathbf{N}_1 * \mathbf{Y}_1 \oplus \mathbf{T}_1 * \mathbf{Z}_1, \\
\mathbf{n}_2^2 &= \mathbf{n}_2^1 \oplus \mathbf{N}_2 * (\mathbf{X}_2 \boxplus_8 \mathbf{Y}_2 \boxplus_8 \mathbf{Z}_2 \boxplus_8 cr_1) \oplus \mathbf{N}_2 * \mathbf{X}_2 \oplus \mathbf{N}_2 * \mathbf{Y}_2 \oplus \mathbf{T}_2 * \mathbf{Z}_2, \\
\mathbf{n}_2^3 &= \mathbf{n}_2^2 \oplus \mathbf{N}_3 * (\mathbf{X}_3 \boxplus_8 \mathbf{Y}_3 \boxplus_8 \mathbf{Z}_3 \boxplus_8 cr_2) \oplus \mathbf{N}_3 * \mathbf{X}_3 \oplus \mathbf{N}_3 * \mathbf{Y}_3 \oplus \mathbf{T}_3 * \mathbf{Z}_3 = \mathbf{n}_2,
\end{aligned}$$

where $cr_j \in \{0, 1, 2\}$ are local carries introduced by the addition modulo $2^{32}$ such that $cr_j = \lfloor (\mathbf{X}_j + \mathbf{Y}_j + \mathbf{Z}_j + cr_{j-1})/2^8 \rfloor$ for $j = 0, 1, 2, 3$ ($cr_{-1} = 0$ by default).

*ComputeMatrix*($cr_{j-1}$, $\mathbf{T}_j$, $\mathbf{N}_j$)

**Parameters**: the partial masks $\mathbf{T}_j$, $\mathbf{N}_j$, and the local carry $cr_{j-1} \in \{0, 1, 2\}$

1: Create a $256 \times 3$ matrix $\mathbf{Mat}$ initialized with zeros;
2: **for** $\mathbf{X}_j = 0, 1, ..., 255$, $\mathbf{Y}_j = 0, 1, ..., 255$ and $\mathbf{Z}_j = 0, 1, ..., 255$ **do**
3: Compute $\mathbf{r} = \mathbf{N}_j * (\mathbf{X}_j \boxplus_8 \mathbf{Y}_j \boxplus_8 \mathbf{Z}_j \boxplus_8 cr_{j-1}) \oplus \mathbf{N}_j * \mathbf{X}_j \oplus \mathbf{N}_j * \mathbf{Y}_j \oplus \mathbf{T}_j * \mathbf{Z}_j$;
4: Compute $cr_j = \lfloor (\mathbf{X}_j + \mathbf{Y}_j + \mathbf{Z}_j + cr_{j-1})/2^8 \rfloor$;
5: $\mathbf{Mat}[\mathbf{r}][cr_j] + +$;
6: **end for**
7: $\mathbf{Mat} \leftarrow \mathbf{Mat}/(2^8)^3$;

**Output**: the $256 \times 3$ matrix $\mathbf{Mat}$.

**Algorithm 2** Compute the distribution of $\mathbf{n}_2$ over the AES MixColumn field $\mathbf{F}_{2^8}$

**Parameters**: the masks $\mathbf{T}$ and $\mathbf{N}$

Prepare three $256 \times 3$ matrices $\mathbf{Mat}$, $\mathbf{Mat}_1$ and $\mathbf{Mat}_2$;
$\mathbf{Mat}_1 \leftarrow$ ComputeMatrix(0, $\mathbf{T}_0$, $\mathbf{N}_0$);
**for** $j = 1, 2, 3$ **do**
  Initialize $\mathbf{Mat}_2$ with zeros;
  **for** $cr_{j-1} = 0, 1, 2$ **do**
    $\mathbf{Mat} \leftarrow$ ComputeMatrix($cr_{j-1}$, $\mathbf{T}_j$, $\mathbf{N}_j$);
    **for** $\mathbf{n}_2^{j-1} = 0, 1, ..., 255$, $\mathbf{n}_2^j = 0, 1, ..., 255$, and $cr_j = 0, 1, 2$ **do**
      $\mathbf{Mat}_2[\mathbf{n}_2^j][cr_j] = \mathbf{Mat}_2[\mathbf{n}_2^j][cr_j] + \mathbf{Mat}[\mathbf{n}_2^{j-1} \oplus \mathbf{n}_2^j][cr_j] \times \mathbf{Mat}_1[\mathbf{n}_2^{j-1}][cr_{j-1}]$;
    **end for**
  **end for**
  $\mathbf{Mat}_1 \leftarrow \mathbf{Mat}_2$;
**end for**

**Output**: $\Pr\{\mathbf{n}_2 = \mathbf{a}\} = \mathbf{Mat}_2[\mathbf{a}][0] + \mathbf{Mat}_2[\mathbf{a}][1] + \mathbf{Mat}_2[\mathbf{a}][2]$, for $\mathbf{a} = 0, 1, ..., 255$.

**Complexity.** Algorithm 2 shown above is used to compute the distribution of $\mathbf{n}_2$, slightly different from that in [ZXM15]. Our improved algorithm has the complexity around $2^{24} + 3 \times 3 \times (2^{24} + 2^{16} \times 3) = 2^{27.33}$, while it is $2^{33.58}$ in [ZXM15].

#### 3.2.3 New Results of the Byte-wise Linear Approximations

As illustrated in the above sections, the distributions of the noise variables $\mathbf{n}_1$ and $\mathbf{n}_2$ can be accurately computed by Algorithm 1 and Algorithm 2. Then the distribution of the folded noise variable $\mathbf{n} = \mathbf{n}_1 \oplus \mathbf{n}_2$ can be derived by the convolution of the above two noise distributions. In our experiments, we have carried out a wide range of search for good byte-wise masks $(\mathbf{T}, \mathbf{N})$ for SNOW 2.0. One important observation from our experiments is that the best byte-wise mask tuple $(\mathbf{T}, \mathbf{N})$ given in [ZXM15], i.e., $\mathbf{T} = \mathbf{N} = (0x03, 0x00, 0x01, 0x00)$ is not optimal. In our experiments, we have found two more independent byte-wise masks which give larger SEIs values, as shown in Table 2.

**Table 2:** The best byte-wise masks $(\mathbf{T}, \mathbf{N})$ for the linear approximation (2)

| No. | Ref. | $\mathbf{T} = \mathbf{N}$ | $\log(\Delta(\mathbf{n}))$ |
|---|---|---|---|
| (1) | New | $(0x01, 0x00, 0x02, 0x01)$ | $-28.82$ |
| (2) | New | $(0x01, 0x02, 0x01, 0x00)$ | $-28.99$ |
| (3) | [ZXM15] | $(0x03, 0x00, 0x01, 0x00)$ | $-29.27$ |

### 3.3 Examples of Relations Between Large-unit and Bitwise Linear Approximations

According to Property 1, the SEI of a smaller-unit linear approximation is not larger than that of the original large-unit one. Besides, Property 2 indicates that approximations on large-unit alphabets have advantages over all the bitwise ones derived from the large-unit one. For SNOW 2.0, we let $F_{(\mathbf{T},\mathbf{N})}$ denote the byte-wise linear approximation with the 4-byte mask tuple $(\mathbf{T}, \mathbf{N})$, and $f_{(\mathbf{\Gamma},\mathbf{\Lambda})}$ denote the bitwise linear approximation with the 32-bit mask tuple $(\mathbf{\Gamma}, \mathbf{\Lambda})$, such that

$$F_{(\mathbf{T},\mathbf{N})}(\cdot) = \mathbf{T} * (\mathbf{z}_t \oplus \mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{N} * (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16}),$$
$$f_{(\mathbf{\Gamma},\mathbf{\Lambda})}(\cdot) = \mathbf{\Gamma} \cdot (\mathbf{z}_t \oplus \mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{\Lambda} \cdot (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16}).$$

Note that $F^{\mathbf{v}}_{(\mathbf{T},\mathbf{N})} = \mathbf{v} \cdot F_{(\mathbf{T},\mathbf{N})}$ is a bitwise linear approximation for any nonzero vector $\mathbf{v} \in \mathbf{F}_{2^m}$, and we always have $\Delta(F^{\mathbf{v}}_{(\mathbf{T},\mathbf{N})}) \leq \Delta(F_{(\mathbf{T},\mathbf{N})})$. For SNOW 2.0, we have $m = 8$.

Let $\mathbf{I} = (1,0,0,0,0,0,0,0)$, then $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) \leq \Delta(F_{(\mathbf{T},\mathbf{N})})$ for any given 4-byte mask tuple $(\mathbf{T},\mathbf{N})$. Below we compare the bitwise masks in Table 1 and the byte-wise masks in Table 2, and give some concrete examples to show the relationship between the 8-bit linear approximation $F_{(\mathbf{T},\mathbf{N})}$ and the bitwise linear approximation $F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}$.

- For the 4-byte masks $\mathbf{T}=\mathbf{N}=(0x01,0x00,0x02,0x01)$ numbered (1) in Table 2, and the 32-bit masks $\mathbf{\Gamma}=\mathbf{\Lambda}=$`0x01800001` numbered (1) in Table 1, we have verified that $F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})} = f_{(\mathbf{\Gamma},\mathbf{\Lambda})}$. In such a case we have $|\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})| = 2^{-14.411}$ from Table 1, and then $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = \epsilon^2(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = \epsilon^2(f_{(\mathbf{\Gamma},\mathbf{\Lambda})}) = |\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})|^2 = 2^{-28.82}$. Note that we also have $\Delta(F_{(\mathbf{T},\mathbf{N})}) = 2^{-28.82}$ from Table 2. Thus $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = \Delta(F_{(\mathbf{T},\mathbf{N})})$.

- Similarly, for $\mathbf{T} = \mathbf{N} = (0x01,0x02,0x01,0x00)$ numbered (2) in Table 2, and $\mathbf{\Gamma} = \mathbf{\Lambda} =$`0x00018001` numbered (2) in Table 1, we have verified that $F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})} = f_{(\mathbf{\Gamma},\mathbf{\Lambda})}$. From Table 1, $|\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})| = 2^{-14.496}$, then $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = |\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})|^2 = 2^{-28.99}$. From Table 2 we also have $\Delta(F_{(\mathbf{T},\mathbf{N})}) = 2^{-28.99}$. Thus $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = \Delta(F_{(\mathbf{T},\mathbf{N})})$.

- Similarly, for $\mathbf{T} = \mathbf{N} = (0x03,0x00,0x01,0x00)$ numbered (3) in Table 2, and $\mathbf{\Gamma} = \mathbf{\Lambda} =$`0x00010081` numbered (3) in Table 1, we have verified that $F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})} = f_{(\mathbf{\Gamma},\mathbf{\Lambda})}$. Note that $|\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})| = 2^{-14.635}$, and $\Delta(F_{(\mathbf{T},\mathbf{N})}) = 2^{-29.27} = |\epsilon_{\mathrm{FSM}}(\mathbf{\Gamma},\mathbf{\Lambda})|^2$. Thus $\Delta(F^{\mathbf{I}}_{(\mathbf{T},\mathbf{N})}) = \Delta(F_{(\mathbf{T},\mathbf{N})})$.

The results on SNOW 2.0 in [ZXM15] gave the impression that large-unit approximations lead to larger SEI and also to better attacks. In our experiments, however, we have found many concrete examples of 8-bit large-unit linear approximations for SNOW 2.0 whose certain 1-dimensional bitwise linear approximations have almost the same SEI as that of the original large-unit ones. That is, each of these byte-wise linear approximations is dominated by a single bitwise approximation, and thus the whole SEI is not essentially larger than the SEI (squared correlation) of the dominating single bitwise approximation. Since correlation attacks can be more efficiently implemented using bitwise approximations rather than byte-wise approximations, this provides an opportunity to achieve improvement over the large-unit linear approximation attack on SNOW 2.0 in [ZXM15]. Actually, a bitwise fast correlation attack on SNOW 2.0 has been mounted in [GZ20] by using multiple bitwise masks as listed in Table 1 for linear approximation, with the total time complexity $2^{162.86}$ and data complexity $2^{159.62}$, slightly improving the large-unit correlation attack of [ZXM15] whose total time complexity is $2^{164.15}$ and data complexity is $2^{163.59}$.

# 4 Comparing Large-unit and Bitwise Linear Approximations of SNOW 3G

In this section, we frist study the bitwise and the byte-wise linear approximations of the FSM of SNOW 3G respectively, and then illustrate the relationship between the bitwise and byte-wise linear approximations by some concrete examples.

## 4.1 Bitwise Linear Approximations of the FSM

In [NW06], the bitwise linear approximations over three rounds of the FSM were depicted, as shown in Fig. 5, but only rough estimates of the upper bounds of their correlations were given. We consider a similar approach to analyze SNOW 3G against linear approximation attacks. That is, we try to approximate the FSM part through linear masking and then to cancel out the contributions of the registers $R1$, $R2$ and $R3$ by combining expressions for several keystream words.

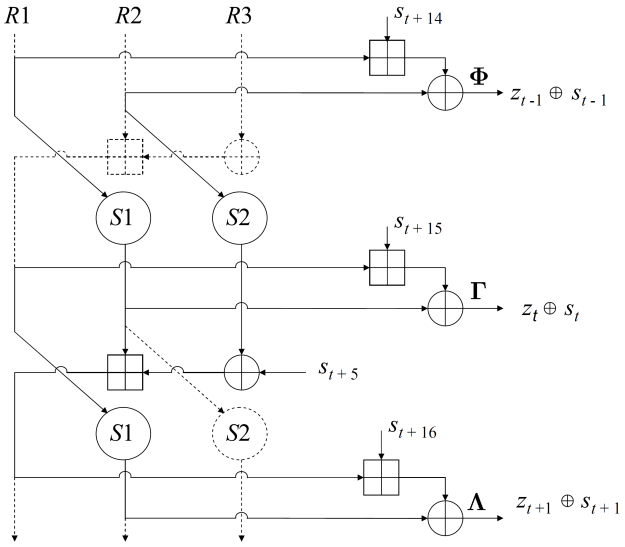


**Figure 5:** The three-round bitwise linear approximations for the FSM of SNOW 3G

Generally, to build the bitwise linear approximation of the FSM of SNOW 3G, we consider to apply the *32-bit* linear masks $\mathbf{\Phi}$, $\mathbf{\Gamma}$ and $\mathbf{\Lambda}$ to $\mathbf{z}_{t-1}$, $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ respectively through the linear masking as follows:

$$\begin{aligned}\mathbf{\Phi}\cdot(\mathbf{z}_{t-1}\oplus\mathbf{s}_{t-1}) &= \mathbf{\Phi}\cdot(\mathbf{s}_{t+14}\boxplus\mathbf{R1}_{t-1})\oplus\mathbf{\Phi}\cdot\mathbf{R2}_{t-1}\\ \mathbf{\Gamma}\cdot(\mathbf{z}_{t}\oplus\mathbf{s}_{t}) &= \mathbf{\Gamma}\cdot(\mathbf{s}_{t+15}\boxplus\mathbf{R1}_{t})\oplus\mathbf{\Gamma}\cdot\mathbf{R2}_{t}\\ \mathbf{\Lambda}\cdot(\mathbf{z}_{t+1}\oplus\mathbf{s}_{t+1}) &= \mathbf{\Lambda}\cdot(\mathbf{s}_{t+16}\boxplus\mathbf{R1}_{t+1})\oplus\mathbf{\Lambda}\cdot\mathbf{R2}_{t+1}\end{aligned}$$

Let $\mathbf{u}_t = \mathbf{R1}_{t-1}$, $\mathbf{v}_t = \mathbf{R2}_{t-1}$ and $\mathbf{w}_t = \mathbf{R1}_t$. According to the update expressions for the registers of the FSM, the first register $R1$ is updated according to $\mathbf{R1}_{t+1} = (\mathbf{s}_{t+5}\oplus\mathbf{R3}_t)\boxplus\mathbf{R2}_t$. We then have $\mathbf{R1}_{t+1} = (\mathbf{s}_{t+5}\oplus \mathrm{S}_2(\mathbf{v}_t))\boxplus \mathrm{S}_1(\mathbf{u}_t)$, $\mathbf{R2}_t = \mathrm{S}_1(\mathbf{u}_t)$ and $\mathbf{R2}_{t+1} = \mathrm{S}_1(\mathbf{w}_t)$, and thus

$$\begin{aligned}\mathbf{\Phi}\cdot(\mathbf{z}_{t-1}\oplus\mathbf{s}_{t-1}) &= \mathbf{\Phi}\cdot(\mathbf{s}_{t+14}\boxplus\mathbf{u}_t)\oplus\mathbf{\Phi}\cdot\mathbf{v}_t\\ \mathbf{\Gamma}\cdot(\mathbf{z}_{t}\oplus\mathbf{s}_{t}) &= \mathbf{\Gamma}\cdot(\mathbf{s}_{t+15}\boxplus\mathbf{w}_t)\oplus\mathbf{\Gamma}\cdot\mathrm{S}_1(\mathbf{u}_t)\\ \mathbf{\Lambda}\cdot(\mathbf{z}_{t+1}\oplus\mathbf{s}_{t+1}) &= \mathbf{\Lambda}\cdot(\mathbf{s}_{t+16}\boxplus(\mathbf{s}_{t+5}\oplus\mathrm{S}_2(\mathbf{v}_t))\boxplus\mathrm{S}_1(\mathbf{u}_t))\oplus\mathbf{\Lambda}\cdot\mathrm{S}_1(\mathbf{w}_t)\end{aligned}$$

Regarding to the internal states and keystream words, we consider the following four associated linear approximations by introducing a *32-bit* intermediate linear mask $\mathbf{\Theta}$, and write them as follows:

(1) $\mathbf{\Phi}\cdot(\mathbf{s}_{t+14}\boxplus\mathbf{u}_t) = \mathbf{\Phi}\cdot\mathbf{s}_{t+14}\oplus\mathbf{\Theta}\cdot\mathrm{S}_1(\mathbf{u}_t)\oplus e_1^{(t)}$,

(2) $\mathbf{\Gamma}\cdot(\mathbf{s}_{t+15}\boxplus\mathbf{w}_t) = \mathbf{\Gamma}\cdot\mathbf{s}_{t+15}\oplus\mathbf{\Lambda}\cdot\mathrm{S}_1(\mathbf{w}_t)\oplus e_2^{(t)}$,

(3) $\mathbf{\Lambda}\cdot(\mathbf{s}_{t+16}\boxplus(\mathbf{s}_{t+5}\oplus\mathrm{S}_2(\mathbf{v}_t))\boxplus\mathrm{S}_1(\mathbf{u}_t)) = \mathbf{\Lambda}\cdot\mathbf{s}_{t+16}\oplus\mathbf{\Lambda}\cdot(\mathbf{s}_{t+5}\oplus\mathrm{S}_2(\mathbf{v}_t))\oplus(\mathbf{\Theta}\oplus\mathbf{\Gamma})\cdot\mathrm{S}_1(\mathbf{u}_t)\oplus e_3^{(t)}$,

(4) $\mathbf{\Lambda}\cdot\mathrm{S}_2(\mathbf{v}_t) = \mathbf{\Phi}\cdot\mathbf{v}_t\oplus e_4^{(t)}$,

where $e_j^{(t)}$ for $j = 1, 2, 3, 4$ are binary noises introduced by these bitwise linear approximations. Let $e^{(t)} = e_1^{(t)}\oplus e_2^{(t)}\oplus e_3^{(t)}\oplus e_4^{(t)}$. With the above relations, the *bitwise* linear approximations of the FSM of SNOW 3G have the following form,

$$\begin{aligned}&\mathbf{\Phi}\cdot\mathbf{z}_{t-1}\oplus\mathbf{\Gamma}\cdot\mathbf{z}_t\oplus\mathbf{\Lambda}\cdot\mathbf{z}_{t+1}\\ &= \mathbf{\Phi}\cdot(\mathbf{s}_{t-1}\oplus\mathbf{s}_{t+14})\oplus\mathbf{\Gamma}\cdot(\mathbf{s}_t\oplus\mathbf{s}_{t+15})\oplus\mathbf{\Lambda}\cdot(\mathbf{s}_{t+1}\oplus\mathbf{s}_{t+5}\oplus\mathbf{s}_{t+16})\oplus e^{(t)}.\end{aligned} \tag{3}$$

Basically, we want to find mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ for (3) which yield highly biased linear approximations, and then employ them in a bitwise fast correlation attack. Before that, we present the following illustrations for the above four linear approximation relations.

1. For the linear approximation relation (1), we write $\mathbf{u}_t = (\mathbf{u}_{t,0} \parallel \mathbf{u}_{t,1} \parallel \mathbf{u}_{t,2} \parallel \mathbf{u}_{t,3})$, where $\mathbf{u}_{t,j} \in \mathbf{F}_{2^8}$ for $j = 0, 1, 2, 3$. Let $\mathbf{x}_t = (\mathbf{x}_{t,0} \parallel \mathbf{x}_{t,1} \parallel \mathbf{x}_{t,2} \parallel \mathbf{x}_{t,3})$ be the output of four parallel AES S-boxes $S_R$, i.e.,

$$\mathbf{x}_t = \mathrm{Sbox}(\mathbf{u}_t) = (S_R(\mathbf{u}_{t,0}) \parallel S_R(\mathbf{u}_{t,1}) \parallel S_R(\mathbf{u}_{t,2}) \parallel S_R(\mathbf{u}_{t,3})),$$

   then we have $\mathbf{u}_t = \mathrm{Sbox}^{-1}(\mathbf{x}_t)$, and $\mathrm{S}_1(\mathbf{u}_t) = \mathbf{M}_1 \cdot \mathbf{x}_t$, where $\mathbf{M}_1$ is expressed as the $32 \times 32$ binary matrix and $\mathbf{x}_t$ is viewed as a 32-bit variable. Given the 32-bit linear mask $\mathbf{\Theta}$, we let $\mathbf{\Theta}'$ be the mask such that $\mathbf{\Theta}' \cdot \mathbf{x}_t = \mathbf{\Theta} \cdot (\mathbf{M}_1 \cdot \mathbf{x}_t)$. (We refer to Appendix A for the computation of $\mathbf{\Theta}'$ from the given mask $\mathbf{\Theta}$). Based on these expressions, we have

$$e_1^{(t)} = \mathbf{\Phi} \cdot (\mathbf{s}_{t+14} \boxplus \mathrm{Sbox}^{-1}(\mathbf{x}_t)) \oplus \mathbf{\Phi} \cdot \mathbf{s}_{t+14} \oplus \mathbf{\Theta}' \cdot \mathbf{x}_t.$$

2. Similarly, for the linear approximation relation (2), we let $\mathbf{y}_t = \mathrm{Sbox}(\mathbf{w}_t)$, and $\mathbf{\Lambda}'$ be the mask such that $\mathbf{\Lambda}' \cdot \mathbf{y}_t = \mathbf{\Lambda} \cdot (\mathbf{M}_1 \cdot \mathbf{y}_t)$. Then the noise $e_2^{(t)}$ can be expressed as

$$e_2^{(t)} = \mathbf{\Gamma} \cdot (\mathbf{s}_{t+15} \boxplus \mathrm{Sbox}^{-1}(\mathbf{y}_t)) \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+15} \oplus \mathbf{\Lambda}' \cdot \mathbf{y}_t.$$

3. For the linear approximation relation (3), let $\xi_t = \mathbf{s}_{t+5} \oplus \mathrm{S}_2(\mathbf{v}_t)$ and $\eta_t = \mathrm{S}_1(\mathbf{u}_t)$, then

$$e_3^{(t)} = \mathbf{\Lambda} \cdot (\mathbf{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+16} \oplus \mathbf{\Lambda} \cdot \xi_t \oplus (\mathbf{\Theta} \oplus \mathbf{\Gamma}) \cdot \eta_t.$$

4. For the linear approximation relation (4), note that the transformation $\mathrm{S}_2$ of SNOW 3G is composed of four parallel 8-bit to 8-bit substitutions $S_Q$, followed by the AES MixColumn transform $\mathbf{M}_2$. We will use $\mathrm{Sbox}'(\cdot)$ to denote the output of four parallel substitutions $S_Q$. Given the mask $\mathbf{\Lambda}$, let $\mathbf{\Lambda}''$ be the mask such that $\mathbf{\Lambda}'' \cdot \mathbf{x} = \mathbf{\Lambda} \cdot (\mathbf{M}_2 \cdot \mathbf{x})$ for all 32-bit $\mathbf{x}$, where $\mathbf{M}_2$ is expressed as the $32 \times 32$ binary matrix. (See Appendix B for the computation of $\mathbf{\Lambda}''$ from the given mask $\mathbf{\Lambda}$). Then we have

$$e_4^{(t)} = \mathbf{\Lambda}'' \cdot \mathrm{Sbox}'(\mathbf{v}_t) \oplus \mathbf{\Phi} \cdot \mathbf{v}_t.$$

To sum up, the four linear approximation relations can be rewritten as follows:

$$\begin{aligned}
e_1^{(t)} &= \mathbf{\Phi} \cdot (\mathbf{s}_{t+14} \boxplus \mathrm{Sbox}^{-1}(\mathbf{x}_t)) \oplus \mathbf{\Phi} \cdot \mathbf{s}_{t+14} \oplus \mathbf{\Theta}' \cdot \mathbf{x}_t, \\
e_2^{(t)} &= \mathbf{\Gamma} \cdot (\mathbf{s}_{t+15} \boxplus \mathrm{Sbox}^{-1}(\mathbf{y}_t)) \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+15} \oplus \mathbf{\Lambda}' \cdot \mathbf{y}_t, \\
e_3^{(t)} &= \mathbf{\Lambda} \cdot (\mathbf{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+16} \oplus \mathbf{\Lambda} \cdot \xi_t \oplus (\mathbf{\Theta} \oplus \mathbf{\Gamma}) \cdot \eta_t, \\
e_4^{(t)} &= \mathbf{\Lambda}'' \cdot \mathrm{Sbox}'(\mathbf{v}_t) \oplus \mathbf{\Phi} \cdot \mathbf{v}_t.
\end{aligned}$$

Since the distributions of $e_j^{(t)}$ are independent of the time instance $t$, we will simplify them by writing $e_j$, and denote $\epsilon(e_j)$ the corresponding correlations to zero, i.e., $\epsilon(e_j) = \Pr\{e_j = 0\} - \Pr\{e_j = 1\}$ for $j = 0, 1, 2, 3$. Let $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ denote the correlation of the linear approximation relation (3), corresponding to the linear mask tuple $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$. By applying the theorem about correlations over composed functions in [Nyb01], together with the Piling-up lemma [Mat93], we get that $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ can be computed as a sum of partial correlations over all *32-bit* intermediate linear masks $\mathbf{\Theta}$ as follows:

$$\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda}) = \epsilon(e_2)\epsilon(e_4) \sum_{\mathbf{\Theta}} \epsilon(e_1)\epsilon(e_3).$$

**Results.** In Section 5, we will present the detailed process on how to search for linear mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $|\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})|$ are as large as possible. The results we obtained are as follows:

- Let $\mathbf{\Lambda} =$ `0x1014190f`, there exist three linear mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $|\epsilon_{\mathrm{FSM}}| \geq 2^{-21}$, the linear masks $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ are listed in Table 3;
- Let $\mathbf{\Lambda} =$ `0x1014190f`, there exist 16 linear mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $2^{-22} < |\epsilon_{\mathrm{FSM}}| < 2^{-21}$, the linear masks $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ are listed in Table 4.

**Table 3:** The best bitwise masks $(\mathbf{\Phi}, \mathbf{\Gamma})$ for linear approximation (3) when $\mathbf{\Lambda} =$ `0x1014190f`

| No. | $\mathbf{\Phi}$ | $\mathbf{\Gamma}$ | $\log(\lvert \sum_{\Theta} \epsilon(e_1)\epsilon(e_3)\rvert)$ | $\log(\lvert\epsilon(e_2)\rvert)$ | $\log(\lvert\epsilon(e_4)\rvert)$ | $\log(\lvert\epsilon_{\mathrm{FSM}}\rvert)$ |
|---|---|---|---|---|---|---|
| (1) | `0x00000020` | `0x00000001` | −14.06 | −3.42 | −3.00 | −20.48 |
| (2) | `0x00000030` | `0x00000001` | −14.06 | −3.42 | −3.00 | −20.48 |
| (3) | `0x00000001` | `0x00000001` | −13.38 | −3.42 | −4.00 | −20.80 |

**Table 4:** The bitwise masks $(\mathbf{\Phi}, \mathbf{\Gamma})$ for bitwise linear approximation (3) such that $2^{-22} \leq |\epsilon_{\mathrm{FSM}}| < 2^{-21}$ when $\mathbf{\Lambda} =$ `0x1014190f`

| $\mathbf{\Phi}$ | $\mathbf{\Gamma}$ | $\log(\lvert \sum_{\Theta} \epsilon(e_1)\epsilon(e_3)\rvert)$ | $\log(\lvert\epsilon(e_2)\rvert)$ | $\log(\lvert\epsilon(e_4)\rvert)$ | $\log(\lvert\epsilon_{\mathrm{FSM}}\rvert)$ |
|---|---|---|---|---|---|
| `0x0000006c` | `0x00000001` | −14.88 | −3.42 | −3.00 | −21.30 |
| `0x00000006` | `0x00000001` | −13.97 | −3.42 | −4.00 | −21.39 |
| `0x00000020` | `0x00000004` | −14.39 | −4.00 | −3.00 | −21.39 |
| `0x00000030` | `0x00000004` | −14.39 | −4.00 | −3.00 | −21.39 |
| `0x00000073` | `0x00000001` | −15.01 | −3.42 | −3.00 | −21.43 |
| `0x000000b0` | `0x00000001` | −14.14 | −3.42 | −4.00 | −21.56 |
| `0x000000f0` | `0x00000001` | −14.14 | −3.42 | −4.00 | −21.56 |
| `0x00000001` | `0x00000004` | −13.71 | −4.00 | −4.00 | −21.71 |
| `0x00000018` | `0x00000001` | −14.33 | −3.42 | −4.00 | −21.75 |
| `0x000000a0` | `0x00000001` | −15.41 | −3.42 | −3.00 | −21.83 |
| `0x00000017` | `0x00000001` | −14.44 | −3.42 | −4.00 | −21.86 |
| `0x0000001f` | `0x00000001` | −14.44 | −3.42 | −4.00 | −21.86 |
| `0x0000001d` | `0x00000001` | −15.51 | −3.42 | −3.00 | −21.93 |
| `0x0000002d` | `0x00000001` | −15.51 | −3.42 | −3.00 | −21.93 |
| `0x0000004f` | `0x00000001` | −15.57 | −3.42 | −3.00 | −21.99 |
| `0x0000006f` | `0x00000001` | −15.57 | −3.42 | −3.00 | −21.99 |

## 4.2 Byte-wise Linear Approximations of the FSM

Recent work in cryptanalysis of stream ciphers paid more attentions on approximations on larger alphabets, and showed that approximations on larger alphabets can improve the attacks. In [ZXM15], the large-unit approach was used to achieve improvements over the previous attacks on SNOW 2.0. In [YJM19], inspired by the results of [ZXM15], Yang et al. constructed three-round byte-wise linear approximations for the FSM of SNOW 3G and performed the searches for finding actual byte-wise masks that gave high SEI values for the approximations. The byte-wise linear approximations found in [YJM19] were also applied to launch a fast correlation attack against SNOW 3G.

In this section, we will study directly the byte-wise linear approximations of SNOW 3G, following strictly a similar procedure as that in [ZXM15] for approximating the FSM part. In this part, all the 32-bit words are divided into 4 bytes and regarded as 4-dimensional vectors over $\mathbf{F}_{2^8}$. As defined in Section 2.1, for two 4-dimensional vectors

$\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$ over $\mathbf{F}_{2^8}$ with the defining polynomial $p(z)$, we have $\mathbf{x} * \mathbf{y} = \mathbf{x}_0\mathbf{y}_0 \oplus \mathbf{x}_1\mathbf{y}_1 \oplus \mathbf{x}_2\mathbf{y}_2 \oplus \mathbf{x}_3\mathbf{y}_3$ with the multiplications $\mathbf{x}_i\mathbf{y}_i$ taken over $\mathbf{F}_{2^8} = \mathbf{F}_2[z]/\langle p(z)\rangle$.

To build the byte-wise linear approximation for the FSM of SNOW 3G, we first apply the *4-byte* linear masks $\mathbf{Q}$, $\mathbf{T}$ and $\mathbf{N}$ to $\mathbf{z}_{t-1}$, $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ respectively,

$$\begin{aligned}
\mathbf{Q} * (\mathbf{z}_{t-1} \oplus \mathbf{s}_{t-1}) &= \mathbf{Q} * (\mathbf{s}_{t+14} \boxplus \mathbf{R1}_{t-1}) \oplus \mathbf{Q} * \mathbf{R2}_{t-1} \\
\mathbf{T} * (\mathbf{z}_t \oplus \mathbf{s}_t) &= \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathbf{R1}_t) \oplus \mathbf{T} * \mathbf{R2}_t \\
\mathbf{N} * (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1}) &= \mathbf{N} * (\mathbf{s}_{t+16} \boxplus \mathbf{R1}_{t+1}) \oplus \mathbf{N} * \mathbf{R2}_{t+1}
\end{aligned}$$

Let $\mathbf{u}_t = \mathbf{R1}_{t-1}$, $\mathbf{v}_t = \mathbf{R2}_{t-1}$ and $\mathbf{w}_t = \mathbf{R1}_t$. Applying the FSM update algorithm, we have

$$\begin{aligned}
\mathbf{Q} * (\mathbf{z}_{t-1} \oplus \mathbf{s}_{t-1}) &= \mathbf{Q} * (\mathbf{s}_{t+14} \boxplus \mathbf{u}_t) \oplus \mathbf{Q} * \mathbf{v}_t \\
\mathbf{T} * (\mathbf{z}_t \oplus \mathbf{s}_t) &= \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathbf{w}_t) \oplus \mathbf{T} * \mathrm{S}_1(\mathbf{u}_t) \\
\mathbf{N} * (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1}) &= \mathbf{N} * (\mathbf{s}_{t+16} \boxplus (\mathbf{s}_{t+5} \oplus \mathrm{S}_2(\mathbf{v}_t)) \boxplus \mathrm{S}_1(\mathbf{u}_t)) \oplus \mathbf{N} * \mathrm{S}_1(\mathbf{w}_t)
\end{aligned}$$

Regarding to the internal states and keystream words, we consider the following four linear approximations by introducing a *4-byte* intermediate linear mask $\mathbf{\Omega}$, and write them as follows:

(1) $\mathbf{Q} * (\mathbf{s}_{t+14} \boxplus \mathbf{u}_t) = \mathbf{Q} * \mathbf{s}_{t+14} \oplus \mathbf{\Omega} * \mathrm{S}_1(\mathbf{u}_t) \oplus \mathbf{e}_1^{(t)}$,

(2) $\mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathbf{w}_t) = \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N} * \mathrm{S}_1(\mathbf{w}_t) \oplus \mathbf{e}_2^{(t)}$,

(3) $\mathbf{N} * (\mathbf{s}_{t+16} \boxplus (\mathbf{s}_{t+5} \oplus \mathrm{S}_2(\mathbf{v}_t)) \boxplus \mathrm{S}_1(\mathbf{u}_t))$
$= \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{N} * (\mathbf{s}_{t+5} \oplus \mathrm{S}_2(\mathbf{v}_t)) \oplus (\mathbf{\Omega} \oplus \mathbf{T}) * \mathrm{S}_1(\mathbf{u}_t) \oplus \mathbf{e}_3^{(t)}$,

(4) $\mathbf{N} * \mathrm{S}_2(\mathbf{v}_t) = \mathbf{Q} * \mathbf{v}_t \oplus \mathbf{e}_4^{(t)}$,

where $\mathbf{e}_j^{(t)}$ for $j = 1, 2, 3, 4$ are 8-bit noises introduced by the above linear approximations. With these relations, the *byte-wise* linear approximations of the FSM of SNOW 3G have the following form,

$$\begin{aligned}
&\mathbf{Q} * \mathbf{z}_{t-1} \oplus \mathbf{T} * \mathbf{z}_t \oplus \mathbf{N} * \mathbf{z}_{t+1} \\
&= \mathbf{Q} * (\mathbf{s}_{t-1} \oplus \mathbf{s}_{t+14}) \oplus \mathbf{T} * (\mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{N} * (\mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16}) \oplus \mathbf{e}^{(t)},
\end{aligned} \tag{4}$$

where $\mathbf{e}^{(t)} = \mathbf{e}_1^{(t)} \oplus \mathbf{e}_2^{(t)} \oplus \mathbf{e}_3^{(t)} \oplus \mathbf{e}_4^{(t)}$ is the folded noise introduced by the above four linear approximations.

We will try to find the byte-wise mask tuples $(\mathbf{Q}, \mathbf{T}, \mathbf{N})$ for linear approximation (4) such that the SEIs of the distributions of $\mathbf{e}^{(t)}$ are as large as possible, and then they can be used in the fast correlation attack over $\mathbf{F}_{2^8}$. It is important to note that the field $\mathbf{F}_{2^8}$ involved in $\mathbf{e}^{(t)}$ have three different defining polynomials, they are

- $z^8 + z^7 + z^5 + z^3 + 1 \in \mathbf{F}_2[z]$ (field constant `0xA9`), used for defining the LFSR of SNOW 3G. We denote the corresponding field by $\mathcal{G} = \mathbf{F}_2[z]/\langle z^8 + z^7 + z^5 + z^3 + 1\rangle$. For two 4-dimensional vectors $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$, where $\mathbf{x}_i, \mathbf{y}_i \in \mathcal{G}$, the inner product $\mathbf{x} * \mathbf{y} = \mathbf{x}_0\mathbf{y}_0 \oplus \mathbf{x}_1\mathbf{y}_1 \oplus \mathbf{x}_2\mathbf{y}_2 \oplus \mathbf{x}_3\mathbf{y}_3$ is operated with the multiplications $\mathbf{x}_i\mathbf{y}_i$ taken over $\mathcal{G}$.

- $z_1^8 + z_1^4 + z_1^3 + z_1 + 1 \in \mathbf{F}_2[z_1]$ (field constant `0x1B`), used for defining the AES MixColumn operation $\mathbf{M}_1$ of the mapping $\mathrm{S}_1(\cdot)$ in the FSM part. We denote the corresponding field by $\mathcal{G}_1 = \mathbf{F}_2[z_1]/\langle z_1^8 + z_1^4 + z_1^3 + z_1 + 1\rangle$. For $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$, where $\mathbf{x}_i, \mathbf{y}_i \in \mathcal{G}_1$, we will use $\mathbf{x} *_1 \mathbf{y}$ as a substitute for $\mathbf{x} * \mathbf{y}$, where the inner product is operated with the multiplications $\mathbf{x}_i\mathbf{y}_i$ taken over $\mathcal{G}_1$.

- $z_2^8 + z_2^6 + z_2^5 + z_2^3 + 1 \in \mathbf{F}_2[z_2]$ (field constant `0x69`), used for defining the AES MixColumn operation $\mathbf{M}_2$ of the mapping $\mathrm{S}_2(\cdot)$ in the FSM part. We denote the corresponding field by $\mathcal{G}_2 = \mathbf{F}_2[z_2]/\langle z_2^8 + z_2^6 + z_2^5 + z_2^3 + 1\rangle$. For $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$, where $\mathbf{x}_i, \mathbf{y}_i \in \mathcal{G}_2$, we will use $\mathbf{x} *_2 \mathbf{y}$ as a substitute for $\mathbf{x} * \mathbf{y}$, where the inner product is operated with the multiplications $\mathbf{x}_i\mathbf{y}_i$ taken over $\mathcal{G}_2$.

We need to unify all the involved multiplications in $\mathbf{F}_{2^8}$ by specifying only one defining polynomial. Different from the method in [ZXM15] for unifying two fields for SNOW 2.0, here we will unify three fields for SNOW 3G during the course of approximating the FSM. We will use $\mathcal{G} = \mathbf{F}_2[z]/\langle z^8 + z^7 + z^5 + z^3 + 1\rangle$ as the specified field, that means all the masks are defined over $\mathcal{G}$, and the multiplications over $\mathbf{F}_{2^8}$ are taken by modulo the polynomial $z^8 + z^7 + z^5 + z^3 + 1$. Let $\mathbf{M}$ denote the AES MixColumn operation defined over $\mathcal{G}$, then

- For any 4-byte value $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3)$, let $\mathbf{W} = (\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3)$ be the output of four parallel AES S-boxes $S_R(\cdot)$, i.e., $\mathbf{W} = \mathrm{Sbox}(\mathbf{w}) = (S_R(\mathbf{w}_0), S_R(\mathbf{w}_1), S_R(\mathbf{w}_2), S_R(\mathbf{w}_3))$. According to the definition of $\mathrm{S}_1(\cdot)$, we have $\mathrm{S}_1(\mathbf{w}) = \mathbf{M}_1 *_1 \mathbf{W}$, where $\mathbf{M}_1$ is the AES MixColumn operation defined over $\mathcal{G}_1$ and the operation "$*_1$" is taken over $\mathcal{G}_1$. Let $\mathbf{W}' \triangleq l_1(\mathbf{W})$ be another 4-byte value such that $\mathbf{M} * \mathbf{W}' = \mathbf{M}_1 *_1 \mathbf{W}$, where the involved multiplications in the left and right sides are taken over $\mathcal{G}$ and $\mathcal{G}_1$ respectively. Then we derive $\mathrm{S}_1(\mathbf{w}) = \mathbf{M} * l_1(\mathbf{W})$. We refer to Equation (12) in Appendix C on how to compute $\mathbf{W}'$ from $\mathbf{W}$.

- For any 4-byte value $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, let $\mathbf{V} = (\mathbf{V}_0, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3)$ be the output of four parallel 8-bit to 8-bit substitutions $S_Q(\cdot)$, i.e., $\mathbf{V} = \mathrm{Sbox}'(\mathbf{v}) = (S_Q(\mathbf{v}_0), S_Q(\mathbf{v}_1), S_Q(\mathbf{v}_2), S_Q(\mathbf{v}_3))$. According to the definition of $\mathrm{S}_2(\cdot)$, we have $\mathrm{S}_2(\mathbf{v}) = \mathbf{M}_2 *_2 \mathbf{V}$, where $\mathbf{M}_2$ is the AES MixColumn operation defined over $\mathcal{G}_2$ and the operation "$*_2$" is taken over $\mathcal{G}_2$. Let $\mathbf{V}'' \triangleq l_2(\mathbf{V})$ be another 4-byte value such that $\mathbf{M} * \mathbf{V}'' = \mathbf{M}_2 *_2 \mathbf{V}$, where the involved multiplications in the left and right sides are taken over $\mathcal{G}$ and $\mathcal{G}_2$ respectively. Then we derive $\mathrm{S}_2(\mathbf{v}) = \mathbf{M} * l_2(\mathbf{V})$. We refer to Equation (13) in Appendix D for the computation of $\mathbf{V}''$ from $\mathbf{V}$.

After unifying all the involved operations to $\mathcal{G} = \mathbf{F}_2[z]/\langle z^8 + z^7 + z^5 + z^3 + 1\rangle$, we now have the following illustrations for the above four linear approximation relations.

1. For the linear approximation relation (1), we let $\mathbf{U}_t = \mathrm{Sbox}(\mathbf{u}_t)$ be the output of four parallel AES S-boxes $S_R(\cdot)$, then $\mathrm{S}_1(\mathbf{u}_t) = \mathbf{M}_1 *_1 \mathbf{U}_t$. Let $\mathbf{U}'_t = l_1(\mathbf{U}_t)$ be the 4-byte value such that $\mathbf{M} * \mathbf{U}'_t = \mathbf{M}_1 *_1 \mathbf{U}_t$ computed by Equation (12) in Appendix C, we then have

$$\mathbf{e}_1^{(t)} = \mathbf{Q} * (\mathbf{s}_{t+14} \boxplus \mathrm{Sbox}^{-1}(\mathbf{U}_t)) \oplus \mathbf{Q} * \mathbf{s}_{t+14} \oplus \mathbf{\Omega}' * l_1(\mathbf{U}_t),$$

   where $\mathbf{\Omega}' = \mathbf{\Omega} * \mathbf{M}$, with the involved multiplications taken over the unified field $\mathcal{G}$.

2. Similarly, for the linear approximation relation (2), we let $\mathbf{W}_t = \mathrm{Sbox}(\mathbf{w}_t)$, and thus $\mathrm{S}_1(\mathbf{w}_t) = \mathbf{M}_1 *_1 \mathbf{W}_t$. Let $\mathbf{W}'_t = l_1(\mathbf{W}_t)$ be the 4-byte value such that $\mathbf{M} * \mathbf{W}'_t = \mathbf{M}_1 *_1 \mathbf{W}_t$ computed by Equation (12) in Appendix C, then

$$\mathbf{e}_2^{(t)} = \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathrm{Sbox}^{-1}(\mathbf{W}_t)) \oplus \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N}' * l_1(\mathbf{W}_t),$$

   where $\mathbf{N}' = \mathbf{N} * \mathbf{M}$, with the involved multiplications taken over $\mathcal{G}$.

3. For the linear approximation relation (3), we set $\xi_t = \mathbf{s}_{t+5} \oplus \mathrm{S}_2(\mathbf{v}_t)$ and $\eta_t = \mathrm{S}_1(\mathbf{u}_t)$, then

$$\mathbf{e}_3^{(t)} = \mathbf{N} * (\mathbf{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{N} * \xi_t \oplus (\mathbf{\Omega} \oplus \mathbf{T}) * \eta_t.$$

4. For the linear approximation relation (4), we let $\mathbf{V}_t = \mathrm{Sbox}'(\mathbf{v}_t)$ be the output of four parallel 8-bit to 8-bit substitutions $S_Q(\cdot)$, then $\mathrm{S}_2(\mathbf{v}_t) = \mathbf{M}_2 *_2 \mathbf{V}_t$. Let $\mathbf{V}''_t = l_2(\mathbf{V}_t)$ be the 4-byte value such that $\mathbf{M} * \mathbf{V}''_t = \mathbf{M}_2 *_2 \mathbf{V}_t$ computed by Equation (13) in Appendix D. Then we have

$$\mathbf{e}_4^{(t)} = \mathbf{N}' * l_2(\mathbf{V}_t) \oplus \mathbf{Q} * \mathrm{Sbox}'^{-1}(\mathbf{V}_t),$$

where $\mathbf{N}' = \mathbf{N} * \mathbf{M}$, with the involved multiplications taken over $\mathcal{G}$.

To sum up, the four linear approximation relations can be rewritten as follows:

$$\begin{aligned}
\mathbf{e}_1^{(t)} &= \mathbf{Q} * (\mathbf{s}_{t+14} \boxplus \mathrm{Sbox}^{-1}(\mathbf{U}_t)) \oplus \mathbf{Q} * \mathbf{s}_{t+14} \oplus \mathbf{\Omega}' * l_1(\mathbf{U}_t),\\
\mathbf{e}_2^{(t)} &= \mathbf{T} * (\mathbf{s}_{t+15} \boxplus \mathrm{Sbox}^{-1}(\mathbf{W}_t)) \oplus \mathbf{T} * \mathbf{s}_{t+15} \oplus \mathbf{N}' * l_1(\mathbf{W}_t),\\
\mathbf{e}_3^{(t)} &= \mathbf{N} * (\mathbf{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \mathbf{N} * \mathbf{s}_{t+16} \oplus \mathbf{N} * \xi_t \oplus (\mathbf{\Omega} \oplus \mathbf{T}) * \eta_t,\\
\mathbf{e}_4^{(t)} &= \mathbf{N}' * l_2(\mathbf{V}_t) \oplus \mathbf{Q} * \mathrm{Sbox}'^{-1}(\mathbf{V}_t).
\end{aligned}$$

As described in Section 3.2, two-round byte-wise linear approximations for the FSM of SNOW 2.0 were constructed and searches were performed for finding byte-wise masks in [ZXM15]. Based on this, we have provided two improved algorithms in Sections 3.2.1 and 3.2.2, namely Algorithm 1 and Algorithm 2, to compute the distributions of two types of byte-wise linear approximations, whose complexities are $2^{20.25}$ and $2^{27.33}$ respectively. For SNOW 3G, we have computed the SEI of the distributions[3] of $\mathbf{e}_j$ for $j = 1, 2$, i.e., $\Delta(\mathbf{e}_j)$, by modifying Algorithm 1, due to the introduction of the linear transform $l_1(\cdot)$ when unifying the field $\mathbf{F}_{2^8}$, and computed $\Delta(\mathbf{e}_3)$ by Algorithm 2. For $\mathbf{e}_4$, since $\mathrm{Sbox}'^{-1}(\cdot)$ consists of four parallel applications of $S_Q^{-1}$, which do not affect the independency among the bytes of $\mathbf{V}_t$, the distribution of $\mathbf{e}_4$ can be derived by using similar method as that in Algorithm 1 and Algorithm 2. After this, the SEI of the distribution of $\mathbf{e}$, i.e., $\Delta(\mathbf{e})$, can then be obtained by the convolution of the above four distributions.

**Results.** We will sketch some ideas on how to compute the above noise distributions. From our experiments, we have found some byte-wise mask tuples $(\mathbf{Q}, \mathbf{T}, \mathbf{N})$ such that $\Delta(\mathbf{e})$ have high values, which are listed in Table 5.

**Table 5:** The best byte-wise masks $(\mathbf{Q}, \mathbf{T}, \mathbf{N})$ for the linear approximation (4)

| No. | $\mathbf{Q}$ | $\mathbf{T}$ | $\mathbf{N}$ | $\log(\Delta(\mathbf{e}))$ |
|---|---|---|---|---|
| (1) | $(0x0c, 0x00, 0x00, 0x00)$ | $(0x01, 0x00, 0x00, 0x00)$ | $(0x8b, 0x2f, 0x70, 0x1a)$ | $-40.958$ |
| (2) | $(0x16, 0x00, 0x00, 0x00)$ | $(0x01, 0x00, 0x00, 0x00)$ | $(0x8b, 0x2f, 0x70, 0x1a)$ | $-40.958$ |
| (3) | $(0x01, 0x00, 0x00, 0x00)$ | $(0x01, 0x00, 0x00, 0x00)$ | $(0x8b, 0x2f, 0x70, 0x1a)$ | $-41.602$ |

## 4.3 Examples of Relations Between Large-unit and Bitwise Linear Approximations

For SNOW 3G, we let $G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}$ denote the byte-wise linear approximation with the 4-byte mask tuple $(\mathbf{Q}, \mathbf{T}, \mathbf{N})$ such that $G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}(\cdot) = \mathbf{Q} * (\mathbf{z}_{t-1} \oplus \mathbf{s}_{t-1} \oplus \mathbf{s}_{t+14}) \oplus \mathbf{T} * (\mathbf{z}_t \oplus \mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{N} * (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16})$, and $g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})}$ denote the bitwise linear approximation with the 32-bit mask tuple $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})}(\cdot) = \mathbf{\Phi} \cdot (\mathbf{z}_{t-1} \oplus \mathbf{s}_{t-1} \oplus \mathbf{s}_{t+14}) \oplus \mathbf{\Gamma} \cdot (\mathbf{z}_t \oplus \mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{\Lambda} \cdot (\mathbf{z}_{t+1} \oplus \mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16})$.

For $\mathbf{I} = (1, 0, 0, 0, 0, 0, 0, 0)$, we always have $\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) \leq \Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})})$. As was done in Section 3.3, below we will give some concrete examples of byte-wise mask tuple

[3] Since the distributions of $\mathbf{e}_j^{(t)}$ for $j = 1, 2, 3, 4$ are independent of $t$, we simplify them by writing $\mathbf{e}_j$.

$(\mathbf{Q}, \mathbf{T}, \mathbf{N})$ for the linear approximation $G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}$ such that $\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = \Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})})$ by comparing the bitwise and byte-wise masks in Table 3 and Table 5, meaning that each of these byte-wise linear approximations has a single dominating bitwise linear approximation such that the SEI of the whole thing is almost equal to the squared correlation of the dominating single bitwise approximation.

- For the 4-byte masks $\mathbf{Q} = (0x0c, 0x00, 0x00, 0x00)$, $\mathbf{T} = (0x01, 0x00, 0x00, 0x00)$ and $\mathbf{N} = (0x8b, 0x2f, 0x70, 0x1a)$ numbered (1) in Table 5, and the 32-bit masks $\mathbf{\Phi}$ =`0x00000020`, $\mathbf{\Gamma}$ =`0x00000001` and $\mathbf{\Lambda}$ =`0x1014190f` numbered (1) in Table 3, we have verified that $G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})} = g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})}$. In such a case we have obtained $\Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = 2^{-40.958}$ by computing the byte-wise linear approximations, as shown in Table 5. We also computed the distribution of $G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}$ directly from the distribution of $G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}$, and obtained $\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = 2^{-40.958} = \Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})})$, and thus $|\epsilon(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})})| = \sqrt{\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})})} = 2^{-20.48}$. As shown in Table 3, we have obtained $|\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})| = 2^{-20.48}$ by computing the bitwise linear approximations. These results prove the validity and correctness of our algorithms for computing the byte-wise linear approximation in Section 4.2 and computing the bitwise linear approximations in Section 5.1.

- Similarly, we have $G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})} = g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})}$ for the byte-wise masks $\mathbf{Q} = (0x16, 0x00, 0x00, 0x00)$, $\mathbf{T} = (0x01, 0x00, 0x00, 0x00)$ and $\mathbf{N} = (0x8b, 0x2f, 0x70, 0x1a)$ numbered (2) in Table 5, and the bitwise masks $\mathbf{\Phi}$ =`0x00000030`, $\mathbf{\Gamma}$ =`0x00000001` and $\mathbf{\Lambda}$ =`0x1014190f` numbered (2) in Table 3. In such a case we obtained $\Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = 2^{-40.958}$ by computing the byte-wise approximations, as shown in Table 5, and $|\epsilon(g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})})| = |\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})| = 2^{-20.48}$ by computing the bitwise approximations, as shown in Table 3. Thus we can derive $\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = \Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})})$.

- We also have $G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})} = g_{(\mathbf{\Phi},\mathbf{\Gamma},\mathbf{\Lambda})}$ and $\Delta(G^{\mathbf{I}}_{(\mathbf{Q},\mathbf{T},\mathbf{N})}) = \Delta(G_{(\mathbf{Q},\mathbf{T},\mathbf{N})})$ for the byte-wise masks $\mathbf{Q} = (0x01, 0x00, 0x00, 0x00)$, $\mathbf{T} = (0x01, 0x00, 0x00, 0x00)$ and $\mathbf{N} = (0x8b, 0x2f, 0x70, 0x1a)$ numbered (3) in Table 5, and the bitwise masks $\mathbf{\Phi}$ =`0x00000001`, $\mathbf{\Gamma}$ =`0x00000001` and $\mathbf{\Lambda}$ =`0x1014190f` numbered (3) in Table 3.

- and so on ...

In our experiments, we have found many concrete examples of 8-bit linear approximations for SNOW 3G whose certain 1-dimensional bitwise linear approximations have almost the same SEI as that of the original large-unit ones. As explained in Section 3.3, correlation attacks can be more efficiently implemented using bitwise approximations rather than large-unit approximations. Thus we have an opportunity to improve the attack against SNOW 3G in [YJM19] which mounted the fast correlation attacks by using the 8-bit (vectorized) linear approximations. We will show in Section 6 the detailed process of the bitwise fast correlation attack on SNOW 3G.

# 5 Search for Bitwise Masks of SNOW 3G

In this section, we describe how to search for bitwise mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ of the linear approximation relation (3) for the FSM of SNOW 3G.

## 5.1 Computing the Bitwise Linear Approximations of the FSM

As described in Section 4.1, the bitwise linear approximations for the FSM of SNOW 3G have the form of (3) by using four linear approximations with the following approximation

noise variables,

$$
\begin{aligned}
e_1^{(t)} &= \mathbf{\Phi} \cdot (\mathbf{s}_{t+14} \boxplus \mathrm{Sbox}^{-1}(\mathbf{x}_t)) \oplus \mathbf{\Phi} \cdot \mathbf{s}_{t+14} \oplus \mathbf{\Theta}' \cdot \mathbf{x}_t, \\
e_2^{(t)} &= \mathbf{\Gamma} \cdot (\mathbf{s}_{t+15} \boxplus \mathrm{Sbox}^{-1}(\mathbf{y}_t)) \oplus \mathbf{\Gamma} \cdot \mathbf{s}_{t+15} \oplus \mathbf{\Lambda}' \cdot \mathbf{y}_t, \\
e_3^{(t)} &= \mathbf{\Lambda} \cdot (\mathbf{s}_{t+16} \boxplus \xi_t \boxplus \eta_t) \oplus \mathbf{\Lambda} \cdot \mathbf{s}_{t+16} \oplus \mathbf{\Lambda} \cdot \xi_t \oplus (\mathbf{\Theta} \oplus \mathbf{\Gamma}) \cdot \eta_t, \\
e_4^{(t)} &= \mathbf{\Lambda}'' \cdot \mathrm{Sbox}'(\mathbf{v}_t) \oplus \mathbf{\Phi} \cdot \mathbf{v}_t.
\end{aligned}
$$

For any given bitwise mask tuple $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$, the correlation of the linear approximation relation (3) is computed as $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda}) = \epsilon(e_2)\epsilon(e_4) \sum_{\mathbf{\Theta}} \epsilon(e_1)\epsilon(e_3)$. We try to find $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $|\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})|$ is as large as possible. Accordingly, we need to compute the correlations of $e_1$, $e_2$, $e_3$, $e_4$ for given masks. In the following parts, we will show in detail how to compute these correlations in linear-time by utilizing the existing algorithms in [GZ20, NW06].

### 5.1.1 Computation of the Correlations of $e_1$ and $e_2$.

Note that the noises $e_1$ and $e_2$ have the same form but different 32-bit mask tuples, which is $(\mathbf{\Phi}, \mathbf{\Theta}')$ for $e_1$ and $(\mathbf{\Gamma}, \mathbf{\Lambda}')$ for $e_2$. As was done in [GZ20] for the linear approximation of SNOW 2.0, a certain type of function is derived from the expressions of $e_1$ and $e_2$ as $G : \mathbf{F}_{2^{32}} \times \mathbf{F}_{2^{32}} \to \mathbf{F}_{2^{32}}$ such that

$$
G(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \mathbf{x}^{(1)} \boxplus \mathrm{Sbox}^{-1}(\mathbf{x}^{(2)}),
$$

where $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}$ are both 32-bit (4-byte) random variables, and the notation "$\boxplus$" denotes the addition modulo $2^{32}$. We note that $\epsilon(e_1)$ is exactly the correlation of the linear approximation of $G$ with the output mask $\mathbf{\Phi}$ and the input masks $\mathbf{\Phi}$ and $\mathbf{\Theta}'$, and $\epsilon(e_2)$ equals to the correlation of the linear approximation of $G$ with the output mask $\mathbf{\Gamma}$ and the input masks $\mathbf{\Gamma}$ and $\mathbf{\Lambda}'$, thus we have

$$
\epsilon(e_1) = \epsilon_G(\mathbf{\Phi}; \mathbf{\Phi}, \mathbf{\Theta}'), \qquad \epsilon(e_2) = \epsilon_G(\mathbf{\Gamma}; \mathbf{\Gamma}, \mathbf{\Lambda}').
$$

Let $\mathbf{A}$ be the 32-bit output mask of $G$, and $\mathbf{A}, \mathbf{B}$ be the 32-bit input masks, then the correlation of the linear approximation of $G$ under the bitwise mask tuple $(\mathbf{A}; \mathbf{A}, \mathbf{B})$ is defined as

$$
\begin{aligned}
\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = &\Pr\{\mathbf{A} \cdot G(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) = \mathbf{A} \cdot \mathbf{x}^{(1)} \oplus \mathbf{B} \cdot \mathbf{x}^{(2)}\} \\
&- \Pr\{\mathbf{A} \cdot G(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) \neq \mathbf{A} \cdot \mathbf{x}^{(1)} \oplus \mathbf{B} \cdot \mathbf{x}^{(2)}\}
\end{aligned}
$$

In [GZ20], a linear-time algorithm is proposed to compute the correlation of the linear approximation of $G$ for an arbitrary bitwise mask tuple, and then used to mount attacks on SNOW 2.0 and SNOW 3G. The general idea is to divide the 32-bit values into four 8-bit values according to the specific structure of the underlying function $S_R^{-1}(\cdot)$, and then pre-compute and store some useful matrices independent of the input/output masks, and finally compute the correlation under an arbitrary bitwise mask tuple by doing some matrix multiplications using these pre-computed matrices. To be specific, for any 32-bit mask tuple $(\mathbf{A}; \mathbf{A}, \mathbf{B})$ of $G$, we write $\mathbf{A}$ and $\mathbf{B}$ in bytes as $\mathbf{A} = (\mathbf{A}_0 \parallel \mathbf{A}_1 \parallel \mathbf{A}_2 \parallel \mathbf{A}_3)$ and $\mathbf{B} = (\mathbf{B}_0 \parallel \mathbf{B}_1 \parallel \mathbf{B}_2 \parallel \mathbf{B}_3)$ with $\mathbf{A}_j, \mathbf{B}_j \in \mathbf{F}_{2^8}$ for $j = 0, 1, 2, 3$, then $\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B})$ can be efficiently computed by Theorem 2 of [GZ20] as

$$
\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = \boldsymbol{l}_2 \mathbf{M}_{(\mathbf{A}_3, \mathbf{A}_3, \mathbf{B}_3)} \mathbf{M}_{(\mathbf{A}_2, \mathbf{A}_2, \mathbf{B}_2)} \mathbf{M}_{(\mathbf{A}_1, \mathbf{A}_1, \mathbf{B}_1)} \mathbf{M}_{(\mathbf{A}_0, \mathbf{A}_0, \mathbf{B}_0)} \boldsymbol{e}_0, \tag{5}
$$

where $\mathbf{M}_{(\mathbf{A}_j, \mathbf{A}_j, \mathbf{B}_j)}$ are $2 \times 2$ matrices pre-computed by Algorithm 3 as shown below, $\boldsymbol{l}_2 = (1, 1)$ is a row vector, and $\boldsymbol{e}_0 = (1, 0)^{\mathrm{T}}$ is a column vector.

**Algorithm 3** Construction of the matrix $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$

**Parameters**: the partial $m$-bit mask values $\mathbf{a}$ and $\mathbf{b}$ ($m = 8$)

Create a matrix $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$ of size $2 \times 2$;
Create two $2 \times 2$ matrices $\mathbf{N}_0$ and $\mathbf{N}_1$ initialized with zeros;
**for** $i\theta \in \{0,1\}$, $\mathbf{x} \in \mathbf{F}_{2^m}$ and $\mathbf{y} \in \mathbf{F}_{2^m}$ **do**
    compute $u = \mathbf{x} + S_R^{-1}(\mathbf{y})$;
    compute $r = \mathbf{a} \cdot ((u + i\theta) \bmod 2^m) \oplus \mathbf{a} \cdot \mathbf{x} \oplus \mathbf{b} \cdot \mathbf{y}$;
    compute $o\theta = \lfloor (u + i\theta)/2^m \rfloor$;
    $\mathbf{N}_r[o\theta][i\theta] := \mathbf{N}_r[o\theta][i\theta] + 1$;
**for** $i\theta \in \{0,1\}$ and $o\theta \in \{0,1\}$ **do**
    $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}[o\theta][i\theta] := (\mathbf{N}_0[o\theta][i\theta] - \mathbf{N}_1[o\theta][i\theta])/2^{2m}$;

**Output**: the corresponding $2 \times 2$ matrix $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$

**Complexity Analysis.** To compute $\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B})$ by Equation (5) for any 32-bit masks $\mathbf{A}, \mathbf{B}$, we need to pre-compute $2^8 \times 2^8 = 2^{16}$ matrices $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$ by trying all the possibilities of $\mathbf{a}, \mathbf{b} \in \mathbf{F}_{2^8}$. From Algorithm 3, for each $(\mathbf{a}, \mathbf{b})$, the matrix $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$ can be constructed with a time complexity $\mathcal{O}(2^{2m+1})$ and a memory complexity $\mathcal{O}(1)$. Here we have $m = 8$. Thus all the $2^{16}$ matrices $\mathbf{M}_{(\mathbf{a},\mathbf{a},\mathbf{b})}$ are constructed with a time complexity of $2^{16} \times (2^{16} \times 2) = 2^{33}$ and a memory complexity of $2^{16} \times (2 \times 2) = 2^{18}$. Using these pre-computed matrices, the accurate value of $\epsilon(e_1)$ with the given masks $(\mathbf{\Phi}, \mathbf{\Theta}')$ (*Resp.* $\epsilon(e_2)$ with the given masks $(\mathbf{\Gamma}, \mathbf{\Lambda}')$) can be obtained according to Equation (5) by doing 4 matrix multiplications of small size, which costs a *linear-time* complexity.

For the value of $\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B})$ under any given masks $\mathbf{A}, \mathbf{B}$, we have the following conclusion, which will help in finding good bitwise linear approximations for the FSM of SNOW 3G. The proof is given in Appendix E.

**Corollary 1.** *For any 32-bit masks* $\mathbf{A} = (\mathbf{A}_0 \parallel \mathbf{A}_1 \parallel \mathbf{A}_2 \parallel \mathbf{A}_3)$ *and* $\mathbf{B} = (\mathbf{B}_0 \parallel \mathbf{B}_1 \parallel \mathbf{B}_2 \parallel \mathbf{B}_3)$*, suppose* $\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) \neq 0$*. Then we have*

- $\mathbf{A}_3 =$`0x00` *if and only if* $\mathbf{B}_3 =$`0x00`*; In this case, we have*

$$\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = \boldsymbol{l}_2 \mathbf{M}_{(\mathbf{A}_2,\mathbf{A}_2,\mathbf{B}_2)} \mathbf{M}_{(\mathbf{A}_1,\mathbf{A}_1,\mathbf{B}_1)} \mathbf{M}_{(\mathbf{A}_0,\mathbf{A}_0,\mathbf{B}_0)} \boldsymbol{e}_0.$$

- $\mathbf{A}_3 = \mathbf{A}_2 =$`0x00` *if and only if* $\mathbf{B}_3 = \mathbf{B}_2 =$`0x00`*; In this case, we have*

$$\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = \boldsymbol{l}_2 \mathbf{M}_{(\mathbf{A}_1,\mathbf{A}_1,\mathbf{B}_1)} \mathbf{M}_{(\mathbf{A}_0,\mathbf{A}_0,\mathbf{B}_0)} \boldsymbol{e}_0.$$

- $\mathbf{A}_3 = \mathbf{A}_2 = \mathbf{A}_1 =$`0x00` *if and only if* $\mathbf{B}_3 = \mathbf{B}_2 = \mathbf{B}_1 =$`0x00`*; In this case, we have*

$$\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = \boldsymbol{l}_2 \mathbf{M}_{(\mathbf{A}_0,\mathbf{A}_0,\mathbf{B}_0)} \boldsymbol{e}_0.$$

- $\mathbf{A}_3 = \mathbf{A}_2 = \mathbf{A}_1 = \mathbf{A}_0 =$`0x00` *if and only if* $\mathbf{B}_3 = \mathbf{B}_2 = \mathbf{B}_1 = \mathbf{B}_0 =$`0x00`*. In this case, we have* $\epsilon_G(\mathbf{A}; \mathbf{A}, \mathbf{B}) = 1$.

### 5.1.2 Computation of the Correlation of $e_3$.

Note that the correlation of $e_3$ is closely related with the correlation of the addition modulo $2^{32}$ with three inputs. The $k$-input addition modulo $2^n$ is defined as $F : \mathbf{F}_{2^n} \times ... \times \mathbf{F}_{2^n} \to \mathbf{F}_{2^n}$ such that $F(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(k)}) = \mathbf{x}^{(1)} \boxplus \cdots \boxplus \mathbf{x}^{(k)}$, where "$\boxplus$" denotes the addition modulo $2^n$. Let $\epsilon_+(\mathbf{\Gamma}^{(0)}; \mathbf{\Gamma}^{(1)}, ..., \mathbf{\Gamma}^{(k)})$ denote the correlation of $F$ with respect to the $n$-bit output mask $\mathbf{\Gamma}^{(0)}$ and the $n$-bit input masks $\mathbf{\Gamma}^{(1)}, ..., \mathbf{\Gamma}^{(k)}$. In [NW06], the authors have proposed a *linear-time* algorithm to accurately compute the correlation of the linear approximation of $F$ for any given mask tuple, we describe it in the following theorem.

**Theorem 1.** [NW06]. *Let $k > 1$ be a fixed integer. For all $\mathcal{R} \in \mathbf{F}_2^{k+1}$, let $\mathbf{D}_\mathcal{R}$ be the $k \times k$ matrices where the $(oc, ic)$-element is computed as*

$$\begin{aligned}\mathbf{D}_\mathcal{R}[oc][ic] = 2^{-k}(&|\{\mathbf{x} \in \mathbf{F}_2^k : r_0 \cdot f(\mathbf{x}, ic) = \bar{\mathbf{r}} \cdot \mathbf{x},\ g(\mathbf{x}, ic) = oc\}| \\ &-|\{\mathbf{x} \in \mathbf{F}_2^k : r_0 \cdot f(\mathbf{x}, ic) \neq \bar{\mathbf{r}} \cdot \mathbf{x},\ g(\mathbf{x}, ic) = oc\}|)\end{aligned}$$

*where*[4]

$$\begin{aligned}&\mathcal{R} = (r_0, r_1, ..., r_k) \in \mathbf{F}_2^{k+1}, \quad ic, oc \in \{0, ..., k-1\}, \\ &\bar{\mathbf{r}} = (r_1, ..., r_k) \in \mathbf{F}_2^k, \quad \mathbf{x} = (x_1, ..., x_k) \in \mathbf{F}_2^k, \\ &f(\mathbf{x}, ic) = (w_H(\mathbf{x}) + ic) \bmod 2,\ g(\mathbf{x}, ic) = \lfloor (w_H(\mathbf{x}) + ic)/2 \rfloor .\end{aligned}$$

*Let $\boldsymbol{l}_k$ be the row vector of length $k$ with all elements equal to 1, and let $\boldsymbol{e}_0$ be the column vector of length $k$ with a single 1 in 0-th row and 0 otherwise. For any given mask tuple $(\mathbf{\Gamma}^{(0)}, \mathbf{\Gamma}^{(1)}, ..., \mathbf{\Gamma}^{(k)})$ of the $k$-input addition modulo $2^n$, write $\mathbf{\Gamma}^{(i)}$ in bits as $\mathbf{\Gamma}^{(i)} = (\gamma_0^{(i)} \parallel \gamma_1^{(i)} \parallel ... \parallel \gamma_{n-1}^{(i)})$, $i = 0, 1, ..., k$, then we have*

$$\epsilon_+(\mathbf{\Gamma}^{(0)}; \mathbf{\Gamma}^{(1)}, ..., \mathbf{\Gamma}^{(k)}) = \boldsymbol{l}_k \mathbf{D}_{\mathcal{R}_{n-1}} ... \mathbf{D}_{\mathcal{R}_1} \mathbf{D}_{\mathcal{R}_0} \boldsymbol{e}_0,$$

*where $\mathcal{R}_j = (\gamma_j^{(0)}, \gamma_j^{(1)}, ..., \gamma_j^{(k)}) \in \mathbf{F}_2^{k+1}$ for $j = 0, 1, ..., n-1$.*

From the above, we have $\epsilon(e_3) = \epsilon_+(\mathbf{\Lambda}; \mathbf{\Lambda}, \mathbf{\Lambda}, \mathbf{\Theta} \oplus \mathbf{\Gamma})$ with the parameters $n = 32$ and $k = 3$, which can be accurately computed using Theorem 1. Let us write $\mathbf{\Lambda}$, $\mathbf{\Theta}$ and $\mathbf{\Gamma}$ in bits as $\mathbf{\Lambda} = (\lambda_0 \parallel \lambda_1 \parallel ... \parallel \lambda_{31})$, $\mathbf{\Theta} = (\theta_0 \parallel \theta_1 \parallel ... \parallel \theta_{31})$ and $\mathbf{\Gamma} = (\gamma_0 \parallel \gamma_1 \parallel ... \parallel \gamma_{31})$, then we have

$$\epsilon(e_3) = \boldsymbol{l}_3 \mathbf{D}_{(\lambda_{31}, \lambda_{31}, \lambda_{31}, \theta_{31} \oplus \gamma_{31})} ... \mathbf{D}_{(\lambda_1, \lambda_1, \lambda_1, \theta_1 \oplus \gamma_1)} \mathbf{D}_{(\lambda_0, \lambda_0, \lambda_0, \theta_0 \oplus \gamma_0)} \boldsymbol{e}_0. \tag{6}$$

**Complexity Analysis.** For these computations, we need to pre-compute four $3 \times 3$ matrices $\mathbf{D}_{(\alpha,\alpha;\alpha;\beta)}$ for $\alpha, \beta \in \mathbf{F}_2$, corresponding to a time complexity of $4 \times (2^3 \times 3) = 2^{6.58}$ and a memory complexity of $4 \times (3 \times 3) = 2^{5.17}$. Using these pre-computed matrices, the accurate value of $\epsilon(e_3)$ with the given masks $\mathbf{\Lambda}$,$\mathbf{\Theta}$,$\mathbf{\Gamma}$ can be obtained according to Equation (6) by doing 32 matrix multiplications of small size, which costs a *linear-time* complexity.

### 5.1.3 Computation of the Correlation of $e_4$.

Note that the correlation of $e_4$ is exactly the correlation of the function $\mathrm{Sbox}'(\cdot)$ with respect to a linear output mask $\mathbf{\Lambda}''$ and a linear input mask $\mathbf{\Phi}$, i.e., $\epsilon(e_4) = \epsilon_{\mathrm{Sbox}'}(\mathbf{\Lambda}''; \mathbf{\Phi})$. Let $\mathbf{\Lambda}'' = (\mathbf{\Lambda}_0'' \parallel \mathbf{\Lambda}_1'' \parallel \mathbf{\Lambda}_2'' \parallel \mathbf{\Lambda}_3'')$ and $\mathbf{\Phi} = (\mathbf{\Phi}_0 \parallel \mathbf{\Phi}_1 \parallel \mathbf{\Phi}_2 \parallel \mathbf{\Phi}_3)$, where $\mathbf{\Lambda}_j'', \mathbf{\Phi}_j \in \mathbf{F}_{2^8}$ for $j = 0, 1, 2, 3$. Since $\mathrm{Sbox}'(\cdot)$ is composed of four parallel applications of $S_Q$, according to the Piling-up Lemma, we have

$$\epsilon(e_4) = \epsilon_{S_Q}(\mathbf{\Lambda}_3''; \mathbf{\Phi}_3)\epsilon_{S_Q}(\mathbf{\Lambda}_2''; \mathbf{\Phi}_2)\epsilon_{S_Q}(\mathbf{\Lambda}_1''; \mathbf{\Phi}_1)\epsilon_{S_Q}(\mathbf{\Lambda}_0''; \mathbf{\Phi}_0) \tag{7}$$

**Complexity Analysis.** We need to pre-compute a linear approximation table (LAT) to store all the linear approximations of $S_Q$ by trying all the possibilities of $\mathbf{a}, \mathbf{b}$ values, i.e., all the values $\epsilon_{S_Q}(\mathbf{a}; \mathbf{b})$ for all $\mathbf{a}, \mathbf{b} \in \mathbf{F}_{2^8}$ are stored in the row of LAT indexed by $(\mathbf{a}, \mathbf{b})$. For this, we loop for all $\mathbf{x} \in \mathbf{F}_{2^8}$, and compute $\mathbf{a} \cdot S_Q(\mathbf{x}) \oplus \mathbf{b} \cdot \mathbf{x}$, which requires a time complexity of $2^8 \times 2^8 \times 2^8 = 2^{24}$. Using the LAT, the accurate value of $\epsilon(e_4)$ with the given masks $\mathbf{\Lambda}''$ and $\mathbf{\Phi}$ can be obtained according to Equation (7) by table lookups 4 times, which costs a *linear-time* complexity.

[4] Note that $w_H(\mathbf{x})$: $0 \leq w_H(\mathbf{x}) \leq k$, denote the Hamming weight of $\mathbf{x}$, which is the number of non-zero components of $\mathbf{x}$.

#### 5.1.4 Computation of $\epsilon_{\mathbf{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$.

For any fixed mask tuple $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ and any intermediate linear mask $\mathbf{\Theta}$, we write $\mathbf{\Phi} = (\mathbf{\Phi}_0 \parallel \mathbf{\Phi}_1 \parallel \mathbf{\Phi}_2 \parallel \mathbf{\Phi}_3)$, $\mathbf{\Gamma} = (\mathbf{\Gamma}_0 \parallel \mathbf{\Gamma}_1 \parallel \mathbf{\Gamma}_2 \parallel \mathbf{\Gamma}_3)$, $\mathbf{\Lambda} = (\mathbf{\Lambda}_0 \parallel \mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2 \parallel \mathbf{\Lambda}_3)$, and $\mathbf{\Theta} = (\mathbf{\Theta}_0 \parallel \mathbf{\Theta}_1 \parallel \mathbf{\Theta}_2 \parallel \mathbf{\Theta}_3)$. Then $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ is computed as follows:

- We compute $\mathbf{\Lambda}' = (\mathbf{\Lambda}'_0 \parallel \mathbf{\Lambda}'_1 \parallel \mathbf{\Lambda}'_2 \parallel \mathbf{\Lambda}'_3)$ from $\mathbf{\Lambda}$ and $\mathbf{\Theta}' = (\mathbf{\Theta}'_0 \parallel \mathbf{\Theta}'_1 \parallel \mathbf{\Theta}'_2 \parallel \mathbf{\Theta}'_3)$ from $\mathbf{\Theta}$ according to (10) in Appendix A, and then derive the accurate values of $\epsilon(e_1) = \epsilon_G(\mathbf{\Phi}; \mathbf{\Phi}, \mathbf{\Theta}')$ and $\epsilon(e_2) = \epsilon_G(\mathbf{\Gamma}; \mathbf{\Gamma}, \mathbf{\Lambda}')$ by Equation (5). This is a *linear-time* procedure.
- We compute the accurate value of $\epsilon(e_3) = \epsilon_+(\mathbf{\Lambda}; \mathbf{\Lambda}, \mathbf{\Lambda}, \mathbf{\Theta} \oplus \mathbf{\Gamma})$ by Equation (6), which costs *linear-time* complexity.
- We compute $\mathbf{\Lambda}'' = (\mathbf{\Lambda}''_0 \parallel \mathbf{\Lambda}''_1 \parallel \mathbf{\Lambda}''_2 \parallel \mathbf{\Lambda}''_3)$ from $\mathbf{\Lambda}$ according to (11) in Appendix B, and then derive the accurate value of $\epsilon(e_4) = \epsilon_{\mathrm{Sbox}'}(\mathbf{\Lambda}''; \mathbf{\Phi})$ using Equation (7) by table lookups 4 times. This is also a *linear-time* procedure.

With the above method, the accurate value of $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ is obtained as

$$\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda}) = \epsilon_G(\mathbf{\Gamma}; \mathbf{\Gamma}, \mathbf{\Lambda}')\epsilon_{\mathrm{Sbox}'}(\mathbf{\Lambda}''; \mathbf{\Phi}) \sum\nolimits_{\mathbf{\Theta}} \epsilon_G(\mathbf{\Phi}; \mathbf{\Phi}, \mathbf{\Theta}')\epsilon_+(\mathbf{\Lambda}; \mathbf{\Lambda}, \mathbf{\Lambda}, \mathbf{\Theta} \oplus \mathbf{\Gamma}), \quad (8)$$

whose complexity is essentially proportional to the number of the terms of the sum over $\mathbf{\Theta}$.

### 5.2 Search for Bitwise Masks

In this part, we hope to find 32-bit mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ for the linear approximation (3) such that $|\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})|$ computed by Equation (8) are as large as possible. Obviously, executing the search for all possible mask values is impractical. Therefore, we consider to use a clever search strategy trying to find some potential linear masks.

For ease of description, we define a subset $\mathcal{S}$ of all the 32-bit masks such that

$$\mathcal{S} = \{\mathbf{\Lambda} = (\mathbf{\Lambda}_0 \| \mathbf{\Lambda}_1 \| \mathbf{\Lambda}_2 \| \mathbf{\Lambda}_3) \colon \mathbf{\Lambda}_0 = \mathtt{0xa} \in \mathbf{F}_{2^8}^* \text{ and } \mathbf{\Lambda}_k = \mathtt{0x00} \text{ for } k = 1, 2, 3\}.$$

There are totally 255 values in $\mathcal{S}$. In our attempt to find good linear approximations, we have observed according to Corollary 1 and our experiments that the term $|\epsilon(e_2)| = |\epsilon_G(\mathbf{\Gamma}; \mathbf{\Gamma}, \mathbf{\Lambda}')|$ is more likely to have high value when both $\mathbf{\Gamma} \in \mathcal{S}$ and $\mathbf{\Lambda}' \in \mathcal{S}$ are satisfied. Besides, taking into consideration the term $\epsilon(e_4) = \epsilon_{S_Q}(\mathbf{\Lambda}''_3; \mathbf{\Phi}_3)\epsilon_{S_Q}(\mathbf{\Lambda}''_2; \mathbf{\Phi}_2)\epsilon_{S_Q}(\mathbf{\Lambda}''_1; \mathbf{\Phi}_1)\epsilon_{S_Q}(\mathbf{\Lambda}''_0; \mathbf{\Phi}_0)$, we have confined the search to $\mathbf{\Lambda}$ and $\mathbf{\Phi}$ such that $\mathbf{\Lambda}'' \in \mathcal{S}$ and $\mathbf{\Phi} \in \mathcal{S}$ to ensure $|\epsilon(e_4)|$ is as large as possible. Furthermore, we deduce from Corollary 1 that $|\epsilon(e_1)| = |\epsilon_G(\mathbf{\Phi}; \mathbf{\Phi}, \mathbf{\Theta}')|$ have nonzero values if and only if $\mathbf{\Theta}' \in \mathcal{S}$.

Above all, we have confined the search to mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ such that $\mathbf{\Phi} \in \mathcal{S}$, $\mathbf{\Gamma} \in \mathcal{S}$, $\mathbf{\Lambda}' \in \mathcal{S}$ and $\mathbf{\Lambda}'' \in \mathcal{S}$, and the terms of the sum over $\mathbf{\Theta}$ such that $\mathbf{\Theta}' \in \mathcal{S}$ are included in Equation (8). According to the computations of $\mathbf{\Lambda}'$ and $\mathbf{\Lambda}''$ from $\mathbf{\Lambda}$ in Appendix A and B, we obtained 31 choices for $\mathbf{\Lambda}$ which are listed in Table 6 of Appendix F.

Based on the above, we compute $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ for all the $255 \times 255 \times 31 \approx 2^{21}$ mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ in the following way.

**Step 1:** We tried all the $255 \times 31$ combinations of the selected $(\mathbf{\Gamma}, \mathbf{\Lambda})$ and compute the values of $\epsilon(e_2) = \epsilon_G(\mathbf{\Gamma}; \mathbf{\Gamma}, \mathbf{\Lambda}')$.

**Step 2:** We tried all the $255 \times 31$ combinations of the selected $(\mathbf{\Phi}, \mathbf{\Lambda})$ and compute the values of $\epsilon(e_4) = \epsilon_{S_Q}(\mathbf{\Lambda}''_0; \mathbf{\Phi}_0)$.

**Step 3:** For all the $255 \times 255 \times 31 \approx 2^{21}$ choices of $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$, we compute the values of $\sum_{\mathbf{\Theta}' \in \mathcal{S}} \epsilon_G(\mathbf{\Phi}; \mathbf{\Phi}, \mathbf{\Theta}')\epsilon_+(\mathbf{\Lambda}; \mathbf{\Lambda}, \mathbf{\Lambda}, \mathbf{\Theta} \oplus \mathbf{\Gamma})$ by including 255 terms of the sum over $\mathbf{\Theta}$.

Following the above procedure, we obtained some newly found mask tuples for the bitwise linear approximation of the FSM of SNOW 3G, some of the results are presented in Table 3 and Table 4 of Section 4.1. We have obtained two best results, which correspond to the linear mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$ where $\mathbf{\Phi}$=0x00000020 or 0x00000030, $\mathbf{\Gamma}$=0x00000001 and $\mathbf{\Lambda}$=0x1014190f, and the best bitwise linear approximations have the correlation $\pm 2^{-20.48}$ and thus the SEI $2^{-40.96}$.

# 6 Bitwise Fast Correlation Attack on SNOW 3G

In this section, we will first present a fast correlation attack on SNOW 3G by using the bitwise linear approximations, and then make a brief discussion about the bitwise approximation attacks compared the large-unit approximation attack in [YJM19].

## 6.1 Using the Bitwise Masks in a Fast Correlation Attack

The bitwise linear approximations of the FSM of SNOW 3G have the following form:

$$\mathbf{\Phi} \cdot \mathbf{z}_{t-1} \oplus \mathbf{\Gamma} \cdot \mathbf{z}_t \oplus \mathbf{\Lambda} \cdot \mathbf{z}_{t+1} = \mathbf{\Phi} \cdot (\mathbf{s}_{t-1} \oplus \mathbf{s}_{t+14}) \oplus \mathbf{\Gamma} \cdot (\mathbf{s}_t \oplus \mathbf{s}_{t+15}) \oplus \mathbf{\Lambda} \cdot (\mathbf{s}_{t+1} \oplus \mathbf{s}_{t+5} \oplus \mathbf{s}_{t+16}) \oplus e^{(t)}.$$

Given a mask tuple $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$, we let $\varphi_t = \mathbf{\Phi} \cdot \mathbf{z}_{t-1} \oplus \mathbf{\Gamma} \cdot \mathbf{z}_t \oplus \mathbf{\Lambda} \cdot \mathbf{z}_{t+1}$. For a given parameter $N$ (to be determined later), set $D = N/2+2$, provided the keystream words $\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_{D-1}$, we can obtain $\varphi_t$ for $t = 1, 2, ..., N/2$. Let $(x_0, x_1, ..., x_{l-1})$ be the LFSR initial state of SNOW 3G in bits ($l = 512$). With the feedback polynomial we can express the above linear approximations in the initial state form as

$$\varphi_t = (x_0, x_1, ..., x_{l-1}) \cdot \mathbf{g}_t \oplus e^{(t)}, \quad t = 1, 2, ..., N/2$$

where $\mathbf{g}_t$ is the corresponding $l$-bit coefficient column vector. For SNOW 3G, we will use the best two bitwise mask tuples for approximations, both yielding the correlation of $\epsilon_{\mathrm{FSM}}(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda}) = \pm 2^{-20.48}$. In such a case we can obtain $N$ parity checks in total written as follows

$$\mathbf{Z} = \mathbf{U} \oplus \mathbf{E} = (x_0, x_1, ..., x_{l-1}) \cdot \mathbf{G} \oplus \mathbf{E}, \tag{9}$$

where $\mathbf{Z}$ is the $N$-bit row vector computed from the given keystream words $\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_{D-1}$, $\mathbf{G}$ is the $l \times N$ generator matrix, and $\mathbf{E}$ is the $N$-bit noise vector with the correlation $\alpha \triangleq \pm 2^{-20.48}$.

We present a high-level description of our attack on SNOW 3G in Algorithm 4. Generally, the fast correlation attack is modelled as a decoding problem, i.e., the keystream segment $\mathbf{Z}$ can be seen as the transmission result of the LFSR sequence $\mathbf{U}$ through a Binary Symmetry Channel (BSC) with the error probability $p$, as shown in Fig. 6.

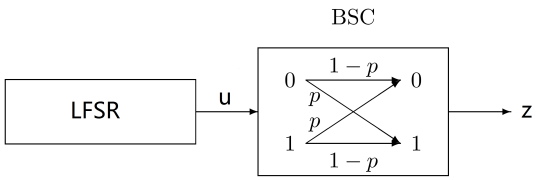


**Figure 6:** Model for a bitwise fast correlation attack

Our bitwise fast correlation attack on SNOW 3G is divided into the preprocessing phase and the processing phase. In the preprocessing phase, we first collect $N$ samples of (9) involving only the keystream words and $l = 512$ LFSR initial state bits, and then try

to reduce the number of the involved LFSR initial state bits to $l'(< l)$ bits at the expense of a folded noise level by employing Wagner's $k$-tree algorithm to generate parity check equations. After this, we enter the processing phase to recover the target $l'$ bits by using the fast Walsh transform as was done in [CJM02, LV04], and further the whole LFSR initial state of SNOW 3G.

**Algorithm 4** the bitwise fast correlation attack on SNOW 3G

**Parameters**: $N$, $m_2$, $l(= 512)$ and $l'(< l)$. Let $D = N/2 + 1$
**Input**:
the keystream words $\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_{D-1}$, and the best two mask tuples $(\mathbf{\Phi}, \mathbf{\Gamma}, \mathbf{\Lambda})$
**1**: Compute $\mathbf{Z}$ and $\mathbf{G}$;
**2**: Follow the ***Preprocessing Phase*** to derive $m_2$ parity check equations involving only $l'$ bits of the LFSR initial state, e.g., $(x_0, x_1, ..., x_{l'-1})$;
**3**: Follow the ***Processing Phase*** to recover $(x_0, x_1, ..., x_{l'-1})$ by using the FWT;
**4**: Recover the remaining $l - l'$ LFSR initial state bits using a similar method;
**5**: Recover the secret key according to the reverse of the initialization;
**Output**: the correct secret key.

Rewriting the matrix $\mathbf{G}$ in column vectors as $\mathbf{G} = (\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_N)$, where $\mathbf{g}_j$ is the $j$-th column vector, we try to find some linear combinations of columns which vanish on $l - l'$ bits to reduce the dimension of the secret (i.e., the number of the involved LFSR initial state bits). For SNOW 3G, the number of folded noise variables is set to 4. Specifically, we look for a number of 4-tuples from $\mathbf{G}$ which add to 0 on their most significant $l - l'$ bits. This is usually solved using Wagner's $k$-tree algorithm [Wag02]. It is stated in [GZ20] that a small technique can be combined when applying the $k$-tree algorithm. Below we illustrate this process using the method in [GZ20].

***Preprocessing Phase.*** 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 these $N$ vectors in one single list $\mathbf{L}$, we carry out the following steps:

**Step 1.** Create a new list $\mathbf{L}_1$ from the original list $\mathbf{L}$ 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 $\mathbf{L}_1$ 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 $\mathbf{L}_1$.

**Step 2.** Create a new list $\mathbf{L}_2$ from $\mathbf{L}_1$ 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 $\mathbf{L}_1$ 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 $\mathbf{L}_2$. Similarly, the expected number of elements in $\mathbf{L}_2$ 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[5] $(\mathbf{g}_{j_1}, \mathbf{g}_{j_2}, \mathbf{g}_{j_3}, \mathbf{g}_{j_4})$ 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 $\alpha^4$ 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 $\mathcal{O}(N + m_1)$.

[5] As illustrated in [GZ20], 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.

***Processing Phase.*** We now enter the process to recover the first $l'$ bits of the LFSR initial state of SNOW 3G. Following a similar method as that in [CJM02, GZ20, LV04], we use the FWT to speed up the evaluation of the $m_2$ parity check equations, and thus recover the value of the target $l'$ bits, which needs a time complexity[6] of $\mathcal{O}(m_2 + l'2^{l'})$ and a memory complexity of $\mathcal{O}(2^{l'})$. To guarantee a high success rate, we set $m_2 = 2l' \ln 2/(\alpha^4)^2$. Since $m_1 = N^2 2^{-(l_1+1)}$ and $m_2 = m_1^2 2^{-(l_2+1)}$, the parameter $N$ is determined to be $N = (m_2 \cdot 2^{2l_1+l_2+3})^{\frac{1}{4}}$, and thus the required number of keystream words can be computed by $D = N/2 + 2$.

**Complexity Analysis.** For SNOW 3G, we follow the above preprocessing phase and processing phase with the parameters $l = 512$, $l' = 166$. In this case, we need to prepare $m_2 = 2l' \ln 2/(\alpha^4)^2 = 2^{171.69}$ approximation relations involving only $x_0, x_1, ..., x_{165}$. By choosing $l_1 = l_2 = 173$, we have $m_1 = 2^{172.84}$ and $N = 2^{173.42}$. Thus it requires a keystream of length $D = N/2 + 2 = 2^{172.42}$, and the time/memory complexity for preparing $m_2$ approximation relations is $\mathcal{O}(N + m_1)$, i.e., $2^{174.16}$. The FWT is utilized to determine the first $l' = 166$ bits of the LFSR initial state, which costs a time complexity $2^{173.77}$ and a memory complexity $2^{166}$. Therefore, all the complexities are all upper bounded by $2^{174.16}$. Once the first 166 bits are recovered, the other LFSR state bits and the FSM state can be recovered by using a similar method and a small-scale exhaustive search with a much lower complexity. Since the initialization of SNOW 3G is a reversible process, the secret key can be recovered once knowing the initial state.

### 6.2 Comparison

Note that the first significant result on SNOW 3G was given in [YJM19] with all the complexities upper bounded by $2^{176.56}$, which used an 8-bit linear approximation in a fast correlation attack over $\mathbf{F}_{2^8}$. Here we improve slightly this result with all the complexities upper bounded by $2^{174.16}$ in a bitwise fast correlation attack using the bitwise linear approximations. Actually, we have found more choices of tradeoff parameters by applying the bitwise fast correlation attack than the attack over $\mathbf{F}_{2^8}$, which can lead to somehow better attacks. Though not a significant improvement, our research results illustrate that we have an opportunity to achieve improvement over the large-unit attacks by using bitwise linear approximations in a linear approximation attack.

## 7 Conclusion

In this paper, we study and compare the large-unit and bitwise linear approximations of SNOW 2.0 and SNOW 3G, and present a bitwise fast correlation attack on SNOW 3G by using our newly found bitwise linear approximations, slightly improving the best known attack on SNOW 3G in [YJM19] which mounted a fast correlation attack by using the 8-bit (vectorized) linear approximation. On one hand, Property 1 and Property 2 indicate that approximations on large-unit alphabets have advantages over all the smaller-unit/bitwise ones in linear approximation attacks, and the results on SNOW 2.0 in [ZXM15] gave the impression that large-unit approximations lead to larger SEI and also to better attacks. However, as shown in Section 3.3 and Section 4.3 for the linear approximations of SNOW 2.0 and SNOW 3G, we have found many concrete examples of byte-wise linear approximations whose certain 1-dimensional/bitwise linear approximations have almost the same SEI as that of the original 8-bit ones. That is, each of these byte-wise approximations is dominated by a single bitwise approximation, and thus the whole SEI is not essentially larger than the SEI of the dominating single bitwise one. Since correlation attacks can be more efficiently implemented using bitwise approximations rather

[6] We refer to [GZ20] for more detailed description of this process.

than large-unit approximations, improvements over the large-unit linear approximation attacks [ZXM15, YJM19] are possible for SNOW 2.0 and SNOW 3G. For SNOW 3G, we have given a fast correlation attack utilizing bitwise linear approximations, with all the complexities upper bounded by $2^{174.16}$, which improve slightly the previous best one in [YJM19]. Though not a significant improvement, our results have illustrated that the bitwise linear approximations may lead to better attacks than the large-unit ones. The cryptanalyst should carefully figure out the internal relation between large-unit linear approximations and the smaller ones, and make his/her best choice of attack parameters according to the concrete structure of the primitives. Note that for the new SNOW stream cipher SNOW-V, the large-unit linear approximations and bitwise ones on several close variants of SNOW-V are studied in [EJMY19] and [GZ21] respectively. It is our future work to study the relation between the large-unit and bitwise linear approximations for these variants, and present the large-unit and bitwise linear approximation attacks on the full SNOW-V.

## Acknowledgements

We would like to thank all reviewers for providing valuable comments to the manuscript. This work is supported by the National Key R&D Research programm (Grant No. 2017YFB0802504), the program of the National Natural Science Foundation of China (Grant No. 61572482), National Cryptography Development Fund (Grant No. MMJJ20170107).

## References

[BBC+08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, et al. Sosemanuk, a fast software-oriented stream cipher. In *New stream cipher designs*, pages 98–118. Springer, 2008.

[BJV04] Thomas Baigneres, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 432–450. Springer, 2004.

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 209–221. Springer, 2002.

[CJS00] Vladimor V Chepyzhov, Thomas Johansson, and Ben Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *International Workshop on Fast Software Encryption*, pages 181–195. Springer, 2000.

[CS91] Vladimir Chepyzhov and Ben Smeets. On a fast correlation attack on certain stream ciphers. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 176–185. Springer, 1991.

[CT00] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 573–588. Springer, 2000.

[EJ00] Patrik Ekdahl and Thomas Johansson. SNOW-a new stream cipher. In *Proceedings of First Open NESSIE Workshop, KU-Leuven*, pages 167–168. Citeseer, 2000.

[EJ02] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In *International Workshop on Selected Areas in Cryptography*, pages 47–61. Springer, 2002.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Transactions on Symmetric Cryptology*, pages 1–42, 2019.

[FTIM18] Yuki Funabiki, Yosuke Todo, Takanori Isobe, and Masakatu Morii. Several milp-aided attacks against SNOW 2.0. In *International Conference on Cryptology and Network Security*, pages 394–413. Springer, 2018.

[GZ20] Xinxin Gong and Bin Zhang. 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.

[GZ21] Xinxin Gong and Bin Zhang. Resistance of SNOW-V against fast correlation attacks. *IACR Transactions on Symmetric Cryptology*, pages 378–410, 2021.

[JJ99] Thomas Johansson and Fredrik Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 347–362. Springer, 1999.

[JJ00] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *Annual International Cryptology Conference*, pages 300–315. Springer, 2000.

[LLP08] Jung-Keun Lee, Dong Hoon Lee, and Sangwoo Park. Cryptanalysis of SOSEMANUK and SNOW 2.0 using linear masks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 524–538. Springer, 2008.

[LV04] Yi Lu and Serge Vaudenay. Faster correlation attack on bluetooth keystream generator E0. In *Annual International Cryptology Conference*, pages 407–425. Springer, 2004.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[NH07] Kaisa Nyberg and Miia Hermelin. Multidimensional walsh transform and a characterization of bent functions. In *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, pages 1–4. IEEE, 2007.

[NW06] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for SNOW 2.0. In *International Workshop on Fast Software Encryption*, pages 144–162. Springer, 2006.

[Nyb01] Kaisa Nyberg. Correlation theorems in cryptanalysis. *Discrete Applied Mathematics*, 111(1-2):177–188, 2001.

[SAG] ETSI SAGE. Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2, document 2: SNOW 3G specification, version 1.1, 2006.

[Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information theory*, 30(5):776–780, 1984.

[Sie85] Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Computer Architecture Letters*, 34(01):81–85, 1985.

[Wag02] David Wagner. A generalized birthday problem. In *Annual International Cryptology Conference*, pages 288–304. Springer, 2002.

[WBDC03] Dai Watanabe, Alex Biryukov, and Christophe De Canniere. A distinguishing attack of SNOW 2.0 with linear masking method. In *International Workshop on Selected Areas in Cryptography*, pages 222–233. Springer, 2003.

[YJM19] Jing Yang, Thomas Johansson, and Alexander Maximov. Vectorized linear approximations for attacks on SNOW 3G. *IACR Transactions on Symmetric Cryptology*, pages 249–271, 2019.

[ZGM17] Bin Zhang, Xinxin Gong, and Willi Meier. Fast correlation attacks on grain-like small state stream ciphers. *IACR Transactions on Symmetric Cryptology*, pages 58–81, 2017.

[ZXM15] Bin Zhang, Chao Xu, and Willi Meier. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In *Annual Cryptology Conference*, pages 643–662. Springer, 2015.

# A Computing the mask $\Lambda'$ from the given mask $\Lambda$ such that $\Lambda' \cdot \mathbf{x} = \Lambda \cdot (\mathbf{M_1} \cdot \mathbf{x})$

Let $lin : \mathbf{F}_{2^{32}} \to \mathbf{F}_{2^{32}}$ be a linear transformation with $\mathbf{\Lambda}' = lin(\mathbf{\Lambda})$ being the linear mask computed from $\mathbf{\Lambda}$ by combining the MixColumn matrix $\mathbf{M}_1$, i.e., $\mathbf{\Lambda}' \cdot \mathbf{x} = \mathbf{\Lambda} \cdot (\mathbf{M}_1 \cdot \mathbf{x})$ for all 32-bit $\mathbf{x}$. Let us briefly describe how to compute $\mathbf{\Lambda}'$ from $\mathbf{\Lambda}$. For $\mathbf{r} = (r_0 \parallel r_1 \parallel r_2 \parallel r_3 \parallel r_4 \parallel r_5 \parallel r_6 \parallel r_7) \in \mathbf{F}_{2^8}$, we define $\mathbf{r}' = trans(\mathbf{r}) = (r'_0 \parallel r'_1 \parallel r'_2 \parallel r'_3 \parallel r'_4 \parallel r'_5 \parallel r'_6 \parallel r'_7)$ such that $r'_i = r_{i+1}$ for $i = 0, 1, ..., 6$, and $r'_7 = r_0 \oplus r_1 \oplus r_3 \oplus r_4$. Write $\mathbf{\Lambda} = (\mathbf{\Lambda}_0 \parallel \mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2 \parallel \mathbf{\Lambda}_3)$, $\mathbf{\Lambda}' = (\mathbf{\Lambda}'_0 \parallel \mathbf{\Lambda}'_1 \parallel \mathbf{\Lambda}'_2 \parallel \mathbf{\Lambda}'_3)$, where $\mathbf{\Lambda}_j \in \mathbf{F}_{2^8}$, $\mathbf{\Lambda}'_j \in \mathbf{F}_{2^8}$, we have $(\mathbf{\Lambda}'_0 \parallel \mathbf{\Lambda}'_1 \parallel \mathbf{\Lambda}'_2 \parallel \mathbf{\Lambda}'_3) = lin(\mathbf{\Lambda}_0 \parallel \mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2 \parallel \mathbf{\Lambda}_3)$ such that

$$\begin{aligned}
\mathbf{\Lambda}'_0 &= trans(\mathbf{\Lambda}_0) \oplus \mathbf{\Lambda}_1 \oplus \mathbf{\Lambda}_2 \oplus \mathbf{\Lambda}_3 \oplus trans(\mathbf{\Lambda}_3).\\
\mathbf{\Lambda}'_1 &= trans(\mathbf{\Lambda}_1) \oplus \mathbf{\Lambda}_2 \oplus \mathbf{\Lambda}_3 \oplus \mathbf{\Lambda}_0 \oplus trans(\mathbf{\Lambda}_0),\\
\mathbf{\Lambda}'_2 &= trans(\mathbf{\Lambda}_2) \oplus \mathbf{\Lambda}_3 \oplus \mathbf{\Lambda}_0 \oplus \mathbf{\Lambda}_1 \oplus trans(\mathbf{\Lambda}_1),\\
\mathbf{\Lambda}'_3 &= trans(\mathbf{\Lambda}_3) \oplus \mathbf{\Lambda}_0 \oplus \mathbf{\Lambda}_1 \oplus \mathbf{\Lambda}_2 \oplus trans(\mathbf{\Lambda}_2).
\end{aligned} \tag{10}$$

# B Computing the mask $\Lambda''$ from the given mask $\Lambda$ such that $\Lambda'' \cdot x = \Lambda \cdot (\mathbf{M_2} \cdot \mathbf{x})$

Let $lin' : \mathbf{F}_{2^{32}} \to \mathbf{F}_{2^{32}}$ be another linear transformation with $\mathbf{\Lambda}'' = lin'(\mathbf{\Lambda})$ being the linear mask computed from $\mathbf{\Lambda}$ by combining the MixColumn matrix $\mathbf{M}_2$, i.e., $\mathbf{\Lambda}'' \cdot \mathbf{x} = \mathbf{\Lambda} \cdot (\mathbf{M}_2 \cdot \mathbf{x})$ for all 32-bit $\mathbf{x}$. We now describe how to compute $\mathbf{\Lambda}''$ from $\mathbf{\Lambda}$. For $r = (r_0 \parallel r_1 \parallel r_2 \parallel r_3 \parallel r_4 \parallel r_5 \parallel r_6 \parallel r_7) \in \mathbf{F}_{2^8}$, we define $r'' = trans'(r) = (r''_0 \parallel r''_1 \parallel r''_2 \parallel r''_3 \parallel r''_4 \parallel r''_5 \parallel r''_6 \parallel r''_7)$ such that $r''_i = r_{i+1}$ for $i = 0, 1, ..., 6$, and $r''_7 = r_0 \oplus r_3 \oplus r_5 \oplus r_6$. Writing

$\mathbf{\Lambda} = (\mathbf{\Lambda}_0 \parallel \mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2 \parallel \mathbf{\Lambda}_3)$, $\mathbf{\Lambda}'' = (\mathbf{\Lambda}''_0 \parallel \mathbf{\Lambda}''_1 \parallel \mathbf{\Lambda}''_2 \parallel \mathbf{\Lambda}''_3)$, where $\mathbf{\Lambda}_j \in \mathbf{F}_{2^8}$, $\mathbf{\Lambda}''_j \in \mathbf{F}_{2^8}$, we have $(\mathbf{\Lambda}''_0 \parallel \mathbf{\Lambda}''_1 \parallel \mathbf{\Lambda}''_2 \parallel \mathbf{\Lambda}''_3) = lin'(\mathbf{\Lambda}_0 \parallel \mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2 \parallel \mathbf{\Lambda}_3)$ such that

$$\begin{aligned}
\mathbf{\Lambda}''_0 &= trans'(\mathbf{\Lambda}_0) \oplus \mathbf{\Lambda}_1 \oplus \mathbf{\Lambda}_2 \oplus \mathbf{\Lambda}_3 \oplus trans'(\mathbf{\Lambda}_3),\\
\mathbf{\Lambda}''_1 &= trans'(\mathbf{\Lambda}_1) \oplus \mathbf{\Lambda}_2 \oplus \mathbf{\Lambda}_3 \oplus \mathbf{\Lambda}_0 \oplus trans'(\mathbf{\Lambda}_0),\\
\mathbf{\Lambda}''_2 &= trans'(\mathbf{\Lambda}_2) \oplus \mathbf{\Lambda}_3 \oplus \mathbf{\Lambda}_0 \oplus \mathbf{\Lambda}_1 \oplus trans'(\mathbf{\Lambda}_1),\\
\mathbf{\Lambda}''_3 &= trans'(\mathbf{\Lambda}_3) \oplus \mathbf{\Lambda}_0 \oplus \mathbf{\Lambda}_1 \oplus \mathbf{\Lambda}_2 \oplus trans'(\mathbf{\Lambda}_2).
\end{aligned} \tag{11}$$

# C Computing the 4-byte value $\mathbf{W'} = l_1(\mathbf{W})$ such that $\mathbf{M} * \mathbf{W'} = \mathbf{M_1} *_1 \mathbf{W}$

For any 4-byte value $\mathbf{W} = (\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3)$ and $\mathbf{W}' = l_1(\mathbf{W}) = (\mathbf{W}'_0, \mathbf{W}'_1, \mathbf{W}'_2, \mathbf{W}'_3)$, we can write $\mathbf{W}_i \in \mathbf{F}_{2^8}$ and $\mathbf{W}'_i \in \mathbf{F}_{2^8}$ in bits as

$$\begin{aligned}
\mathbf{W}_i &= (W_{i,0} \parallel W_{i,1} \parallel W_{i,2} \parallel W_{i,3} \parallel W_{i,4} \parallel W_{i,5} \parallel W_{i,6} \parallel W_{i,7}),\\
\mathbf{W}'_i &= (W'_{i,0} \parallel W'_{i,1} \parallel W'_{i,2} \parallel W'_{i,3} \parallel W'_{i,4} \parallel W'_{i,5} \parallel W'_{i,6} \parallel W'_{i,7}),
\end{aligned}$$

where $W_{i,j}, W'_{i,j} \in \mathbf{F}_2$ for $i = 0, 1, 2, 3$ and $j = 0, 1, ..., 7$. We derive

$$\begin{cases}
W'_{i,0} = W_{i,0} \oplus W_{(i+1) mod\, 4,7} \oplus W_{(i+3) mod\, 4,7}\\
W'_{i,1} = W_{i,1} \oplus W_{(i+2) mod\, 4,7} \oplus W_{(i+3) mod\, 4,7}\\
W'_{i,2} = W_{i,2} \oplus W_{(i) mod\, 4,7} \oplus W_{(i+2) mod\, 4,7}\\
W'_{i,3} = W_{i,3} \oplus W_{(i) mod\, 4,7} \oplus W_{(i+2) mod\, 4,7}\\
W'_{i,4} = W_{i,4} \oplus W_{(i+2) mod\, 4,7} \oplus W_{(i+3) mod\, 4,7}\\
W'_{i,5} = W_{i,5} \oplus W_{(i+2) mod\, 4,7} \oplus W_{(i+3) mod\, 4,7}\\
W'_{i,6} = W_{i,6} \oplus W_{(i) mod\, 4,7} \oplus W_{(i+2) mod\, 4,7}\\
W'_{i,7} = W_{i,7} \oplus W_{(i+1) mod\, 4,7} \oplus W_{(i+2) mod\, 4,7}
\end{cases} \tag{12}$$

# D Computing the 4-byte value $\mathbf{V''} = l_2(\mathbf{V})$ such that $\mathbf{M} * \mathbf{V''} = \mathbf{M_2} *_2 \mathbf{V}$

For any 4-byte value $\mathbf{V} = (\mathbf{V}_0, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3)$ and $\mathbf{V}'' = l_2(\mathbf{V}) = (\mathbf{V}''_0, \mathbf{V}''_1, \mathbf{V}''_2, \mathbf{V}''_3)$, we write $\mathbf{V}_i \in \mathbf{F}_{2^8}$ and $\mathbf{V}''_i \in \mathbf{F}_{2^8}$ in bits as

$$\begin{aligned}
\mathbf{V}_i &= (V_{i,0} \parallel V_{i,1} \parallel V_{i,2} \parallel V_{i,3} \parallel V_{i,4} \parallel V_{i,5} \parallel V_{i,6} \parallel V_{i,7}),\\
\mathbf{V}''_i &= (V''_{i,0} \parallel V''_{i,1} \parallel V''_{i,2} \parallel V''_{i,3} \parallel V''_{i,4} \parallel V''_{i,5} \parallel V''_{i,6} \parallel V''_{i,7}),
\end{aligned}$$

where $V_{i,j}, V''_{i,j} \in \mathbf{F}_2$ for $i = 0, 1, 2, 3$ and $j = 0, 1, ..., 7$. We derive

$$\begin{cases}
V''_{i,0} = V_{i,0} \oplus V_{(i) mod\, 4,7} \oplus V_{(i+2) mod\, 4,7}\\
V''_{i,1} = V_{i,1} \oplus V_{(i) mod\, 4,7} \oplus V_{(i+1) mod\, 4,7} \oplus V_{(i+2) mod\, 4,7} \oplus V_{(i+3) mod\, 4,7}\\
V''_{i,2} = V_{i,2}\\
V''_{i,3} = V_{i,3} \oplus V_{(i) mod\, 4,7} \oplus V_{(i+2) mod\, 4,7}\\
V''_{i,4} = V_{i,4} \oplus V_{(i) mod\, 4,7} \oplus V_{(i+1) mod\, 4,7} \oplus V_{(i+2) mod\, 4,7} \oplus V_{(i+3) mod\, 4,7}\\
V''_{i,5} = V_{i,5} \oplus V_{(i) mod\, 4,7} \oplus V_{(i+2) mod\, 4,7}\\
V''_{i,6} = V_{i,6} \oplus V_{(i+2) mod\, 4,7} \oplus V_{(i+3) mod\, 4,7}\\
V''_{i,7} = V_{(i+1) mod\, 4,7}
\end{cases} \tag{13}$$

# E Proof of Corollary 1

*Proof.* We first present some observations on the pre-computed matrices $\mathbf{M}_{(\alpha,\alpha,\beta)}$ for any $\alpha, \beta \in \mathbf{F}_{2^8}$.

- The matrices $\mathbf{M}_{(\alpha,\alpha,0)}$ for any $\alpha \in \mathbf{F}_{2^8}^*$ are always equal to $\begin{pmatrix} \frac{1}{512} & -\frac{1}{512} \\ -\frac{1}{512} & \frac{1}{512} \end{pmatrix}$;
- The matrices $\mathbf{M}_{(0,0,\beta)}$ for any $\beta \in \mathbf{F}_{2^8}^*$ always have the form of $\begin{pmatrix} val & val \\ -val & -val \end{pmatrix}$, where $val$ is a rational number.
- The matrix $\mathbf{M}_{(0,0,0)}$ is equal to $\begin{pmatrix} \frac{257}{512} & \frac{255}{512} \\ \frac{255}{512} & \frac{257}{512} \end{pmatrix}$.

For any 32-bit masks $\mathbf{A} = (\mathbf{a}_0 \parallel \mathbf{a}_1 \parallel \mathbf{a}_2 \parallel \mathbf{a}_3)$ and $\mathbf{B} = (\mathbf{b}_0 \parallel \mathbf{b}_1 \parallel \mathbf{b}_2 \parallel \mathbf{b}_3)$, we have $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = \boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,\mathbf{a}_2,\mathbf{b}_2)}\mathbf{M}_{(\mathbf{a}_1,\mathbf{a}_1,\mathbf{b}_1)}\mathbf{M}_{(\mathbf{a}_0,\mathbf{a}_0,\mathbf{b}_0)}\boldsymbol{e}_0 \neq 0$. According to the above observations,

(1) If $\mathbf{a}_3$=0x00, we must have $\mathbf{b}_3$=0x00. Otherwise $\boldsymbol{l}_2\mathbf{M}_{(0,0,\mathbf{b}_3)} = (0,0)$, and thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = 0$. Similarly, we can prove that, if $\mathbf{b}_3$=0x00, we must have $\mathbf{a}_3$=0x00. When $\mathbf{a}_3 = \mathbf{b}_3$=0x00, we have $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)} = (1,1)$, thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = \boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_2,\mathbf{a}_2,\mathbf{b}_2)}\mathbf{M}_{(\mathbf{a}_1,\mathbf{a}_1,\mathbf{b}_1)}\mathbf{M}_{(\mathbf{a}_0,\mathbf{a}_0,\mathbf{b}_0)}\boldsymbol{e}_0$.

(2) If $\mathbf{a}_3 = \mathbf{a}_2$ =0x00, from the above we have $\mathbf{b}_3$=0x00, and $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)} = \boldsymbol{l}_2$. Besides, we must have $\mathbf{b}_2$=0x00. Otherwise $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(0,0,\mathbf{b}_2)} = \boldsymbol{l}_2\mathbf{M}_{(0,0,\mathbf{b}_2)} = (0,0)$, and thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = 0$. Similarly, we can prove that, if $\mathbf{b}_3 = \mathbf{b}_2$ =0x00, we must have $\mathbf{a}_3 = \mathbf{a}_2$ =0x00. When $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{b}_3 = \mathbf{b}_2$ =0x00, we have $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,0\mathbf{a}_2,\mathbf{b}_2)} = (1,1)$, thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = \boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_1,\mathbf{a}_1,\mathbf{b}_1)}\mathbf{M}_{(\mathbf{a}_0,\mathbf{a}_0,\mathbf{b}_0)}\boldsymbol{e}_0$.

(3) If $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{a}_1$ =0x00, we have $\mathbf{b}_3 = \mathbf{b}_2$ =0x00, and $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,\mathbf{a}_2,\mathbf{b}_2)} = \boldsymbol{l}_2$. Besides, we must have $\mathbf{b}_1$=0x00. Otherwise $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,\mathbf{a}_2,\mathbf{b}_2)}\mathbf{M}_{(0,0,\mathbf{b}_1)} = \boldsymbol{l}_2\mathbf{M}_{(0,0,\mathbf{b}_1)} = (0,0)$, and thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = 0$. Similarly, we can prove that, if $\mathbf{b}_3 = \mathbf{b}_2 = \mathbf{b}_1$ =0x00, we must have $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{a}_1$ =0x00. When $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{a}_1 = \mathbf{b}_3 = \mathbf{b}_2 = \mathbf{b}_1$ =0x00, we have $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,0\mathbf{a}_2,\mathbf{b}_2)}\mathbf{M}_{(\mathbf{a}_1,\mathbf{a}_1,\mathbf{b}_1)} = (1,1)$, thus $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = \boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_0,\mathbf{a}_0,\mathbf{b}_0)}\boldsymbol{e}_0$.

(4) If $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{a}_1 = \mathbf{a}_0$ =0x00, we have $\mathbf{b}_3 = \mathbf{b}_2 = \mathbf{b}_1$ =0x00, and $\boldsymbol{l}_2\mathbf{M}_{(\mathbf{a}_3,\mathbf{a}_3,\mathbf{b}_3)}\mathbf{M}_{(\mathbf{a}_2,\mathbf{a}_2,\mathbf{b}_2)}\mathbf{M}_{(\mathbf{a}_1,\mathbf{a}_1,\mathbf{b}_1)} = \boldsymbol{l}_2$. Besides, we must have $\mathbf{b}_0 \neq$0x00. Otherwise $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = 0$. Similarly, we can prove that, if $\mathbf{b}_3 = \mathbf{b}_2 = \mathbf{b}_1 = \mathbf{b}_0$ =0x00, we must have $\mathbf{a}_3 = \mathbf{a}_2 = \mathbf{a}_1 = \mathbf{a}_0$ =0x00. When $\mathbf{A} = \mathbf{B}$ =0x00000000, we have $\epsilon_G(\mathbf{A};\mathbf{A},\mathbf{B}) = \boldsymbol{l}_2\mathbf{M}_{(0,0,0)}\mathbf{M}_{(0,0,0)}\mathbf{M}_{(0,0,0)}\mathbf{M}_{(0,0,0)}\boldsymbol{e}_0 = 1$.

Above all, we complete the proof. □

# F All 31 choices for $\Lambda$ such that $\Lambda' \in \mathcal{S}$ and $\Lambda'' \in \mathcal{S}$

**Table 6:** All 31 choices for $\boldsymbol{\Lambda}$ such that $\boldsymbol{\Lambda}' \in \mathcal{S}$ and $\boldsymbol{\Lambda}'' \in \mathcal{S}$

| |
|---|
| 0x066ade02, 0x0dd5bd04, 0x0bbf6306, 0x1bab7a09, 0x1dc1a40b, 0x167ec70d, 0x1014190f, 0xa2544e61, 0xa43e9063, 0xaf81f365, 0xa9eb2d67, 0xb9ff3468, 0xbf95ea6a, 0xb42a896c, 0xb240576e, 0xd1aa27b0, 0xd7c0f9b2, 0xdc7f9ab4, 0xda1544b6, 0xca015db9, 0xcc6b83bb, 0xc7d4e0bd, 0xc1be3ebf, 0x73fe69d1, 0x7594b7d3, 0x7e2bd4d5, 0x78410ad7, 0x685513d8, 0x6e3fcdda, 0x6580aedc, 0x63ea70de |