Vectorized linear approximations for attacks on SNOW 3G

Jing Yang[1], Thomas Johansson[1] and Alexander Maximov[2]

[1] Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{jing.yang,thomas.johansson}@eit.lth.se
[2] Ericsson Research, Lund, Sweden
alexander.maximov@ericsson.com

Abstract. SNOW 3G is a stream cipher designed in 2006 by ETSI/SAGE, serving in 3GPP as one of the standard algorithms for data confidentiality and integrity protection. It is also included in the 4G LTE standard. In this paper we derive vectorized linear approximations of the finite state machine of SNOW 3G. In particular, we show one 24-bit approximation with a bias around 2^{-37} and one byte-oriented approximation with a bias around 2^{-40}. We then use the approximations to launch attacks on SNOW 3G. The first approximation is used in a distinguishing attack resulting in an expected complexity of 2^{172} and the second one can be used in a standard fast correlation attack resulting in key recovery in an expected complexity of 2^{177}. If the key length in SNOW 3G would be increased to 256 bits, the results show that there are then academic attacks on such a version faster than the exhaustive key search.

Keywords: SNOW 3G · Stream Cipher · 5G Mobile System Security.

1 Introduction

SNOW 3G is a word-oriented stream cipher being used as the core of 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2 for UMTS and LTE networks [UEA06a]. It is one member of the SNOW family with two predecessors SNOW 1.0 [EJ00] and SNOW 2.0 [EJ02]. SNOW 1.0 was submitted to NESSIE project in 2000 by Ekdahl and Johansson and the improved version SNOW 2.0 was published in 2002 and selected as an ISO standard in 2005. The SNOW ciphers consist of a linearly updated part through an LFSR (Linear Feedback Shift Register) and a non-linear part referred to as a FSM (Finite State Machine). They are all based on operations on 32-bit words, making them quite efficient in both software and hardware environment. SNOW 3G differs from SNOW 2.0 by introducing a third 32-bit register in the FSM and a second 32-bit S-box application to update this register. This presumably makes SNOW 3G a much harder target in an attack compared to SNOW 2.0.

Just as for other stream ciphers, the class of linear approximation attacks, like distinguishing attacks and correlation attacks, are a main threat to the SNOW ciphers. The basic idea for these attacks is to approximate the nonlinear blocks used in the cipher with linear expressions and then derive linear relationships between output values from different time instances. In a distinguishing attack, a cryptanalyst tries to derive some samples from the keystream and find evidence that such a sample sequence is not behaving like a truly random sequence using some statistical tools, e.g., hypothesis testing. When the linear relationship also involves symbols from the LFSR states, some correlation between the keystream and the LFSR states can be explored to recover the key, which is the foundation of a correlation attack.

Several distinguishing attacks and correlation attacks have been proposed on SNOW 1.0 and SNOW 2.0, where the basic idea is to approximate the FSM part. In [CHJ02], a distinguishing attack on SNOW 1.0 with complexity 2^{100} was proposed using linear masking to get a binary approximation with a bias $2^{-8.3}$, which became one reason of the rejection of SNOW 1.0 from the NESSIE project. To resist against this attack, the authors improved the design and proposed SNOW 2.0, which is, however, still vulnerable to some distinguishing attacks. A distinguishing attack based on a linear masking method with complexity 2^{230} and an improved version with complexity 2^{174} were proposed in [WBDC03] and [NW06a], respectively. In [LLP08] and [ZXM15], correlation attacks on SNOW 2.0 were proposed with complexity $2^{204.38}$ and $2^{164.15}$. For SNOW 3G, however, no significant attack of this type has ever been published.

The cryptographic security of SNOW 3G has been studied in depth. As an algorithm appearing in a main standard, it has been thoroughly evaluated by the standardization consortium before its adoption. Some evaluating information appears in [UEA06b]. There are several side-channel attacks and fault attacks, targeting specific implementations of the algorithm, such as the attacks in [DC09] and [BHNS10]. There are also attacks targeting the initialization phase on versions of SNOW 3G with reduced number of initialization rounds [BPSZ10b, BPSZ10a]. At FSE 2006, Nyberg and Wallén [NW06b] examined linear distinguishing attacks on SNOW 2.0, but devoted one section to SNOW 3G. The best linear approximation of the FSM they found had a bias of 2^{-274} and they also derived an upper bound on 2^{-96} for any binary linear approximation. Here the bias as given in the paper was recalculated, now expressed using *Squared Euclidean Imbalance*, as is commonly used for non-binary linear approximations. Note that [NW06b] considered only binary approximations and the key to improvements is to use approximations over larger alphabets.

In this paper, we give one distinguishing attack and one correlation attack on SNOW 3G by finding efficient linear approximations of the nonlinear part of the FSM. We derive a 24-bit linear approximation by truncating and masking three consecutive keystreams with the bias $2^{-37.37}$ and we further derive an 8-bit approximation from the 24-bit one with the bias $2^{-40.97}$. The 24-bit approximation is then employed to launch a distinguishing attack requiring a keystream length of around 2^{172}. This strongest and largest 24-bit approximation cannot be used in a correlation attack, but the derived 8-bit approximation, which is linear over $GF(2^8)$ can be used to give a correlation attack which has complexity around 2^{177}. This is to the best of our knowledge the first significant result on attacking the full SNOW 3G. In particular, if the key length in SNOW 3G would be increased to 256 bits, the results show that there are then academic attacks on such a version faster than the exhaustive key search.

The distributions for the noise related to the different linear approximations given in the paper will be publicly available to download when this work has been accepted for publication.

The rest of this paper is organized as follows. We briefly describe the design and structure of SNOW 3G in Section 2 and then elaborate the process of finding linear approximations of the FSM in Section 3. In Section 4, we give the experimental verification of the approximations by running the cipher to get a large number of samples. In Section 5, we give the attacks, a distinguishing attack and a correlation attack, based on the vectorized linear approximations derived in Section 3 and in Section 6, we conclude the paper.

2 Description of SNOW 3G

In this section, we present a brief description of the SNOW 3G algorithm. We first note that a stream cipher like SNOW 3G takes as input a secret key K, which is a 128-bit value, and an public value known as the IV(initial vector) value, which is also of length 128 bits.

For each such pair of key and IV, (K, IV), the algorithm produces an output sequence, usually called the *keystream*, denoted $z^{(t)}, t = 1, 2, \ldots$. In SNOW 3G, each keystream symbol is a 32-bit word, so we write $z^{(t)} \in GF(2^{32}), t = 1, 2, \ldots$. Furthermore, each pair should produce a unique keystream sequence, and the typical operation of such a stream cipher is to produce many different keystreams for many different public IV values, while keeping the key fixed.

The overall schematic of SNOW 3G algorithm is shown in Figure 1. Just as SNOW 1.0 and SNOW 2.0, it consists of a linear part, the LFSR, and a nonlinear part, the FSM. The FSM is used to break the linearity of the LFSR contribution. For more details on the design of SNOW 3G, we refer to the original design document [UEA06a].

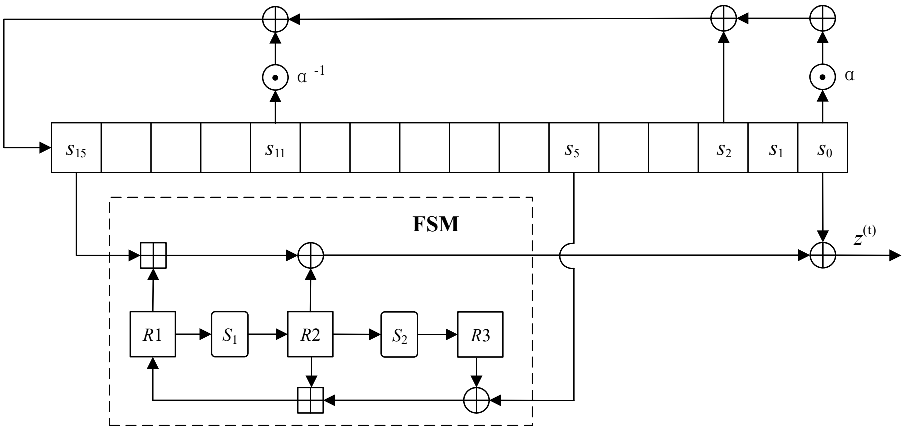

Figure 1: The keystream generation phase of the SNOW-3G stream cipher

The LFSR part consists of 16 cells denoted $(s_0, s_1, ..., s_{15})$ each containing 32 bits and giving 512 bits in total. Every value in a cell is considered as an element from $GF(2^{32})$ and the LFSR sequence is defined by the generating polynomial

$$P(x) = \alpha x^{16} + x^{14} + \alpha^{-1}x^5 + 1 \in GF(2^{32})[x],$$

where α is a root of the polynomial $x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239} \in GF(2^8)[x]$ and β is a root of $x^8 + x^7 + x^5 + x^3 + 1 \in GF(2)[x]$. If we denote the state at clock t as $(s_0^{(t)}, s_1^{(t)}, ..., s_{15}^{(t)})$, then at the next clock $t+1$, $s_i^{(t)}$ is shifted to $s_{i-1}^{(t+1)}$, i.e., $s_i^{(t)} = s_{i-1}^{(t+1)}$, for $1 \le i \le 15$, while $s_{15}^{(t+1)}$ is updated by:

$$s_{15}^{(t+1)} = \alpha^{-1}s_{11}^{(t)} \oplus s_2^{(t)} \oplus \alpha s_0^{(t)},$$

where $\oplus$ denotes a bitwise XOR operation. Note that α and α^{-1} are two constants in $GF(2^{32})$ and the update consequently includes two multiplications in this field.

For the FSM part, it has three internal 32-bit registers $R1$, $R2$ and $R3$, connected by some linear and nonlinear operations. The FSM takes two words from the LFSR part as the inputs, s_{15} and s_5, and outputs a 32-bit keystream word by xoring with s_0, giving the following formula for the generation of the keystream,

$$z^{(t)} = (R1^{(t)} \boxplus_{32} s_{15}^{(t)}) \oplus R2^{(t)} \oplus s_0^{(t)}.$$

Here $\boxplus_{32}$ denotes integer addition modulo 2^{32}. Then the registers in FSM are updated through the following steps:

$$\begin{aligned}
R2^{(t+1)} &= S_1(R1^{(t)}),\\
R3^{(t+1)} &= S_2(R2^{(t)}),\\
R1^{(t+1)} &= R2^{(t)} \boxplus_{32} (R3^{(t)} \oplus s_5^{(t)}),
\end{aligned}$$

where S_1, S_2 are substitution boxes (S-boxes) composed of four bytewise substitutions followed by the MixColumn operation of Rijndael. Below we give the details of how they are constructed by using Little-endian style. The 32-bit registers in FSM could be expressed as four parallel bytes. Let $W = (w_0, w_1, w_2, w_3)$ be the input to the substitution boxes with w_0 being the least and w_3 the most significant byte. The operations of the two S-boxes are as follows.

S-Box S_1: S_1 is a 32-bit to 32-bit mapping operating on four bytes. Bytes are interpreted as elements of GF(2^8) defined by the polynomial $x^8 + x^4 + x^3 + x + 1$. The underlaying 8-bit SBox $S_R(x)$ is the Rijndael AES SBox [DR13]. In general, S_1 is described by

$$S_1(W) = L_1 \cdot S_R(W),$$

which can be expressed in more details as follows. Let $R = (r_0, r_1, r_2, r_3)$ be the four byte output through $R = S_1(W)$. Then

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \cdot \begin{pmatrix} S_R(w_0) \\ S_R(w_1) \\ S_R(w_2) \\ S_R(w_3) \end{pmatrix}. \tag{1}$$

S-Box S_2: S_2 is also a 32-bit to 32-bit mapping operating on four bytes. Bytes are again interpreted as elements of GF(2^8) but this time defined by the polynomial $y^8 + y^6 + y^5 + y^3 + 1$. The underlaying 8-bit SBox $S_Q(x)$ is another 8-to-8 bit S-box derived from the Dickson polynomials. In general, S_2 is described by

$$S_2(W) = L_2 \cdot S_Q(W),$$

and in more details $S_2(W)$ is:

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} y & y+1 & 1 & 1 \\ 1 & y & y+1 & 1 \\ 1 & 1 & y & y+1 \\ y+1 & 1 & 1 & y \end{pmatrix} \cdot \begin{pmatrix} S_Q(w_0) \\ S_Q(w_1) \\ S_Q(w_2) \\ S_Q(w_3) \end{pmatrix}. \tag{2}$$

Like other stream ciphers, SNOW 3G uses an initialization phase during which the cipher is clocked without producing output to fully mix the key and IV into the LFSR state and the FSM registers. During the initialization phase, the key and the IV, each consisting of four 32-bit words, are first loaded into the LFSR state and the registers in the FSM are initialized to be zero. Then the cipher runs 32 times with the output from the FSM feeding back to the LFSR instead of giving a keystream output. After the initialization, the cipher enters the keystream mode, with the first output word from FSM to be discarded and the following FSM outputs forming the keystream. Since the attacks in this paper only uses the keystream mode, we do not give the details of the initilization mode, but refer to the design document if interested[UEA06a].

3 Approximations of the FSM

A main class of attacks on stream ciphers are the so-called linear distinguishing attacks and (fast) correlation attacks. They both build on the idea of approximating some nonlinear operations as linear ones, thereby introducing some approximation noise. The most simple form simply uses bitwise approximations and has a strong connection to linear cryptanalysis of block ciphers. Recent work in cryptanalysis of stream ciphers have shown that approximations on larger alphabets can improve the attacks considerably, e.g. [ZXM15] used the terminology large-unit linear approximations.

For stream ciphers built around an LFSR, it makes sense to provide approximations that are linear with respect to some binary algebraic structure, such as $GF(2)^n$ or $GF(2^n)$. Since the LFSR part is linearly described in $GF(2)^n$ or possibly $GF(2^n)$, the main obstacle is to approximate the FSM. We will return in Section 5 to the case of how to use an approximation in attacks on the full cipher.

The FSM part in SNOW 3G takes inputs from $s_{15}^{(t)}, s_5^{(t)}, s_0^{(t)}$ and outputs $z^{(t)}$, with t varying. It also contains three unknown values in the registers $R1$, $R2$ and $R3$. As such, they need to be cancelled and a linear approximation of the FSM can thus be described as a linear expression including only $s_{15}^{(t)}, s_5^{(t)}, s_0^{(t)}$ and $z^{(t)}$ for different t values. Such an expression is a good approximation if the corresponding expression has a distribution that is biased. So in general, we are interested in finding an expression of the form

$$\bigoplus_{i\in I}(c_z^{(t+i)}z^{(t+i)} \oplus c_{15}^{(t+i)}s_{15}^{(t+i)} \oplus c_5^{(t+i)}s_5^{(t+i)} \oplus c_0^{(t+i)}s_0^{(t+i)}),$$

for some time set I, where operations are in $GF(2)^n$ (or $GF(2^n)$), and $c_z^{(t+i)}, c_{15}^{(t+i)}, c_5^{(t+i)}, c_0^{(t+i)}$ are now m-dimensional matrices and the inputs are considered as column vectors. In order to determine the quality of an approximation, we consider m-bit random variables $E^{(t)}$, defined as the above expression, i.e.,

$$E^{(t)} = \bigoplus_{i\in I}(c_z^{(t+i)}z^{(t+i)} \oplus c_{15}^{(t+i)}s_{15}^{(t+i)} \oplus c_5^{(t+i)}s_5^{(t+i)} \oplus c_0^{(t+i)}s_0^{(t+i)}).$$

Each such random variable has the same distribution, denoted D. The quality of the linear approximation is measured by the bias of the distribution, which can be defined in many ways. Using the SEI (Squared Euclidean Imbalance) as defined in [BJV04], the bias for the distribution D is computed as

$$\epsilon(D) = |D| \cdot \sum_{x=0}^{|D|-1}\left(D(x) - \frac{1}{|D|}\right)^2.$$

We note that when the bias is measured in SEI, the number of samples required to distinguish samples drawn from D from the uniform distribution is in the order of $1/\epsilon(D)$. We are now ready to investigate how to find expressions of the above form with a large bias.

3.1 A 24-bit linear approximation of the FSM

In this first approach, we are targeting an approximation with as large alphabet size as possible, in order to get a bias as large as possible. The novel parts consist in determining how to build the approximation and how to efficiently compute the bias when the alphabet size and the number of involved variables are large.

Let $\hat{R}1, \hat{R}2, \hat{R}3$ be the content of the FSM registers at time t. Then consider the following word-oriented expressions on three consecutive keystream words:

$$\begin{aligned} z^{(t-1)} &= (S_R^{-1}(L_1^{-1}\hat{R}2) \boxplus s_{15}^{(t-1)}) \oplus S_Q^{-1}(L_2^{-1}\hat{R}3) \oplus s_0^{(t-1)}, \\ z^{(t)} &= (\hat{R}1 \boxplus s_{15}^{(t)}) \oplus \hat{R}2 \oplus s_0^{(t)}, \\ L_1^{-1}z^{(t+1)} &= L_1^{-1}(\hat{R}2 \boxplus (\hat{R}3 \oplus s_5^{(t)}) \boxplus s_{15}^{(t+1)}) \oplus S_R(\hat{R}1) \oplus L_1^{-1}s_0^{(t+1)}. \end{aligned} \tag{3}$$

Let us introduce the following notation that applies to three byte-oriented 32-bit vectors,

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix}_{[i,j,k]} = \begin{pmatrix} A[i] \\ B[j] \\ C[k] \end{pmatrix},$$

where i, j, k are corresponding bytes of A, B, and C, respectively. So $A[i]$ denotes the i-th byte of a 32-bit byte-oriented vector, for $i = 0, 1, 2$ or 3.

Now we consider a three-byte sampling of the following form:

$$\underbrace{\begin{pmatrix} z^{(t-1)} \\ z^{(t)} \\ (L_1^{-1}z^{(t+1)}) \end{pmatrix}_{[0,0,0]}}_{\text{Sampling in time } (t)} = \underbrace{\begin{pmatrix} (S_R^{-1}(L_1^{-1}\hat{R}2) \boxplus s_{15}^{(t-1)}) \oplus s_{15}^{(t-1)} \oplus S_Q^{-1}(L_2^{-1}\hat{R}3) \\ \hat{R}2 \\ L_1^{-1}[(\hat{R}2 \boxplus (\hat{R}3 \oplus s_5^{(t)}) \boxplus s_{15}^{(t+1)}) \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}] \end{pmatrix}_{[0,0,0]}}_{\text{24-bit Noise } N2}$$

$$\oplus \underbrace{\begin{pmatrix} 0 \\ (\hat{R}1 \boxplus s_{15}^{(t)}) \oplus s_{15}^{(t)} \\ S_R(\hat{R}1) \end{pmatrix}_{[0,0,0]}}_{\text{24-bit Noise } N1} \oplus \underbrace{\begin{pmatrix} s_0^{(t-1)} \oplus s_{15}^{(t-1)} \\ s_{15}^{(t)} \oplus s_0^{(t)} \\ L_1^{-1}[s_0^{(t+1)} \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}] \end{pmatrix}_{[0,0,0]}}_{\text{Contribution from the LFSR}}.$$

Basically, we want to achieve a linear approximation where the bias is as large as possible and the choice of multiplying $z^{(t+1)}$ with L_1^{-1} is chosen to have a really small influence of the noise related to the $\hat{R}1$ register.

As already indicated in the above formula, our linear approximation is

$$\begin{pmatrix} z^{(t-1)} \\ z^{(t)} \\ (L_1^{-1}z^{(t+1)}) \end{pmatrix}_{[0,0,0]} = \begin{pmatrix} s_0^{(t-1)} \oplus s_{15}^{(t-1)} \\ s_{15}^{(t)} \oplus s_0^{(t)} \\ L_1^{-1}[s_0^{(t+1)} \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}] \end{pmatrix}_{[0,0,0]} \oplus N_{tot}^{(t)},$$

where $N_{tot}^{(t)} = N1^{(t)} \oplus N2^{(t)}$ and

$$N1^{(t)} = \begin{pmatrix} 0 \\ (\hat{R}1 \boxplus s_{15}^{(t)}) \oplus s_{15}^{(t)} \\ S_R(\hat{R}1) \end{pmatrix}_{[0,0,0]},$$

$$N2^{(t)} = \begin{pmatrix} (S_R^{-1}(L_1^{-1}\hat{R}2) \boxplus s_{15}^{(t-1)}) \oplus s_{15}^{(t-1)} \oplus S_Q^{-1}(L_2^{-1}\hat{R}3) \\ \hat{R}2 \\ L_1^{-1}[(\hat{R}2 \boxplus (\hat{R}3 \oplus s_5^{(t)}) \boxplus s_{15}^{(t+1)}) \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}] \end{pmatrix}_{[0,0,0]},$$

Since the distributions of $N_{tot}^{(t)}, N1^{(t)}, N2^{(t)}$ are independent of t, we simplify them by writing $N_{tot}, N1, N2$, respectively. Note also that $N1$ and $N2$ are independent.

3.1.1 Computation of the 24-bit noise distributions for $N1$ and $N2$

Computing the distribution of $N1$ is trivial, we simply run over all possible values of $\hat{R}1[0]$ and $s_{15}^{(t)}[0]$, which is of complexity $O(2^{16})$.

Computation for $N2$ is more tricky and below we explain how we do that. We can rewrite $N2$ as:

$$N2 = \begin{pmatrix} (S_R^{-1}(\underbrace{\textstyle\sum_{i=0}^{3} e_i\hat{R}2[i]}_{\text{Linear part } A}) \boxplus x_0) \oplus x_0 \oplus S_Q^{-1}(\underbrace{\textstyle\sum_{i=0}^{3} d_i\hat{R}3[i]}_{\text{Linear part } B}) \\ \hat{R}2[0] \\ \underbrace{\textstyle\sum_{i=0}^{3} e_i[(\hat{R}2[i] \boxplus (\hat{R}3[i] \oplus y_i) \boxplus w_i \boxplus c_i) \oplus y_i \oplus w_i]}_{\text{Linear part } C} \end{pmatrix},$$

where $x_i = s_{15}^{(t-1)}[i], y_i = s_5^{(t)}[i], w_i = s_{15}^{(t+1)}[i]$, and the coefficients are $e_i = L_1^{-1}[0, i]$ and $d_i = L_2^{-1}[0, i]$, for $i = 0, 1, 2, 3$.

The value c_i is the input carry value that comes from the arithmetical addition of the previous byte(s), i.e., bytes 0 to $i-1$, of the 3 terms: $\hat{R}2[i]$, w_i and $(\hat{R}3[i] \oplus y_i)$. Note that the carry value is $0 \leq c_i \leq 2$ and the first byte has $c_0 = 0$.

The idea behind the computation technique is to compute 256 24-bit distributions of the triple (A, B, C), conditioned on the value of $\hat{R}2[0] \in [0, \ldots, 255]$. Let us denote those distributions as $D_{\hat{R}2[0]}(A, B, C)$. Then, the distribution table of $N2$ is constructed as follows: we initialize the distribution table $N2$ with all zeroes; then, for each combination of $\hat{R}2[0], x_0, A, B, C$ (in total 2^{40} choices) we do:

$$Pr\{N2 = \begin{pmatrix} (S_R^{-1}(A) \boxplus x_0) \oplus x_0 \oplus S_Q^{-1}(B) \\ \hat{R}2[0] \\ C \end{pmatrix}\} += 2^{-16} \cdot D_{\hat{R}2[0]}(A, B, C).$$

Also note that we can actually compute distributions $D_{\hat{R}2[0]}(A, B, C)$ one by one for each value of $\hat{R}2[0]$ and add to the accumulating distribution of $N2$. Thus, we do not need to store all of them in RAM simultaneously.

3.1.2 Computation of sub-noises $D_{\hat{R}2[0]}(A, B, C)$

What remains is to show how to compute $D_{\hat{R}2[0]}(A, B, C)$ for a given value of $\hat{R}2[0]$, which can be done as follows.

Let us introduce intermediate distributions

$$E_{\hat{R}2[0],\ k=0,1,2,3,\ c_k=0,1,2,\ o_k=0,1,2}(A_k, B_k, C_k),$$

which are the distributions of the three bytes (A_k, B_k, C_k), and where the expression for the arithmetical addition in the part of the expression for C has the input carry value c_k, and the output carry value o_k, for a fixed $\hat{R}2[0]$.

Then, the first 3 distributions for $k = 0$ and $o_0 = 0, 1, 2$,

$$E_{\hat{R}2[0],\ k=0,\ c_0=0,\ o_0=0,1,2}(A_0, B_0, C_0),$$

can be computed by trying all possible values of $\hat{R}3[0], y_0, w_0$ in time $O(2^{24})$. Note that the value $\hat{R}2[0]$ is fixed and the input carry $c_0 = 0$.

The next 3×3 distributions for $k = 1$ and $c_1, o_1 \in [0, 1, 2]$,

$$E_{\hat{R}2[0],\ k=1,\ c_1=0,1,2,\ o_1=0,1,2}(A_1, B_1, C_1),$$

can be computed by trying all possible values of $c_1, \hat{R}2[1], \hat{R}3[1], y_1, w_1$ in time $O(3 \cdot 2^{32})$. We continue this way to compute all these distributions. Then, the distribution $D_{\hat{R}2[0]}(A, B, C)$

can be derived through a series of convolutions of distributions that are matching in the input/output carries, and summations as follows:

$$\begin{aligned} D_{\hat{R}2[0]}(A|B|C) = \sum_{v_0=0}^{2}\sum_{v_1=0}^{2}\sum_{v_2=0}^{2}\sum_{v_3=0}^{2} &[E_{\hat{R}2[0],\ k=0,\ c_0=0,\ o_0=v_0}(A_0, B_0, C_0) \\ &\times E_{\hat{R}2[0],\ k=1,\ c_1=v_0,\ o_1=v_1}(A_1, B_1, C_1) \\ &\times E_{\hat{R}2[0],\ k=2,\ c_2=v_1,\ o_2=v_2}(A_2, B_2, C_2) \\ &\times E_{\hat{R}2[0],\ k=3,\ c_3=v_2,\ o_3=v_3}(A_3, B_3, C_3)] \end{aligned}$$

where × denotes a XOR-convolution of the 24-bit distribution tables, and $\sum$ is a point-wise arithmetical summation of the probabilities. A single XOR-convolution can be done with Fast Walsh-Hadamard Transform (FWHT) having time complexity $O(N \log N)$ where, in our case, we have $N = 2^{24}$.

We can speed up the computations as follows. Note that for the first $k = 0$ we only have three distribution tables $E_{k=0,*,o_0=0,1,2}$ corresponding to three possible output carry values. After we have computed the distributions for $k = 1$, we can actually perform the partial convolutions and combine $E_{k=0,*}$ and $E_{k=1,*}$ such that we again have only three distributions to keep – corresponding to three possible values of the output carry to the third byte from the first two bytes. I.e., $E_{k=0-1,*,o_1=0,1,2}$ combines the distributions over the first two bytes. We do that until $k = 3$, and then the total distribution D_* is the point-wise sum of the final three distributions $E_{k=0-3,*,o_3=0,1,2}$.

Also note that FWHT is a linear transformation. So convolution in the time domain corresponds to point-wise multiplication in the frequency domain; and, summation in the time domain corresponds to summation in the frequency domain. Thus, there is no need to switch between the time and frequency domain for mixed summation and convolution operations, we can do most of the above in the frequency domain without switching.

3.1.3 Computation results and bias values

We have implemented the above computation method, computed the distribution for $N1$ and $N2$, and then the 24-bit total noise distribution for $N_{tot} = N1 \oplus N2$ for the proposed approximation. Note that the approximation takes into account the full 32-bit adders $\boxplus$. We have got the following results regarding corresponding biases:

$$\begin{aligned} \epsilon(N1) &> 1, \\ \epsilon(N2) &\approx 2^{-29.391880}, \\ \epsilon(N_{tot}) &\approx 2^{-37.374659}, \\ \epsilon(2 \times N_{tot}) &\approx 2^{-80.212670}, \\ \epsilon(3 \times N_{tot}) &\approx 2^{-121.663520}, \\ \epsilon(4 \times N_{tot}) &\approx 2^{-162.757498}. \end{aligned}$$

Here $\epsilon(i \times N_{tot})$ is the notion for the bias of the resulting distribution when summing i random variables distributed as N_{tot} using bitwise XOR.

3.2 An 8-bits approximation

As will be clear later, the 24-bit approximation cannot be used in a straight-forward manner in a fast correlation attack. For such a case, one would need an approximation that can be completely described over a finite field.

From the 24-bit noise distribution that we computed in the previous subsection, we can further derive an 8-bit approximation with operations in the Rijndael field $GF(2^8)$, with the noise now denoted N'_{tot}. The approximation has the following form,

$$\begin{aligned}\Lambda z^{(t-1)}[0] \oplus z^{(t)}[0] \oplus \Gamma(L_1^{-1} z^{(t+1)})[0] = N'_{tot} \\ \oplus (\Lambda(s_0^{(t-1)} \oplus s_{15}^{(t-1)}) \oplus s_0^{(t)} \oplus s_{15}^{(t)} \oplus \Gamma L_1^{-1}[s_0^{(t+1)} \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}])[0],\end{aligned} \quad (4)$$

where Γ, Λ are some nonzero constants in $GF(2^8)$. For each possible choice of $\Gamma, \Lambda \in GF(2^8)$ we compute the 8-bit distribution of N'_{tot} directly from the given 24-bit distribution of N_{tot} and then we compute the corresponding bias,

$$N'_{tot} = \Lambda N_{tot}[0] \oplus N_{tot}[1] \oplus \Gamma N_{tot}[2].$$

Searching through all choices of Γ and Λ would normally imply the computational complexity $O(2^{48})$, but we can reduce it down to $O(\sim 2^{40})$ by the following technique. In the loop for Γ, we first precompute the joint 16-bit distribution of $(N_{tot}[0], N_{tot}[1] \oplus \Gamma N_{tot}[2])$ with complexity $O(2^{24})$, then we loop for Λ and use the precomputed joint distribution to derive the 8-bit distribution of N'_{tot}. The best choice for constants appears to be $\Gamma =$ `0x9c` and $\Lambda =$ `0x08` (alternatively, $\Lambda =$ `0x18` also gives the best approximation) and the resulting bias of N'_{tot} is:

$$\begin{aligned}\epsilon(N'_{tot}) &\approx 2^{-40.970689}, \\ \epsilon(2 \times N'_{tot}) &\approx 2^{-81.940519}, \\ \epsilon(3 \times N'_{tot}) &\approx 2^{-122.909464}, \\ \epsilon(4 \times N'_{tot}) &\approx 2^{-163.877500}.\end{aligned}$$

4 Experimental verification

In this section, we experimentally verify the correctness of the bias derived in the previous section. The experimental verification can be done by running the cipher and collecting a large number of samples. For distributions of smaller sizes one can fully verify them by experimentally determining the exact distribution, i.e., every probability value in the distribution is correct; while for larger distributions, this might not be computationally possible. Instead, we can use them in a hypothesis testing and in this way demonstrate that it can be used in a distinguisher. This will be the case for the 24-bit approximation from Section 3.

We consider deciding the sample distribution between the uniform distribution and the noise distribution derived in Section 3 by hypothesis testing. We will follow the hypothesis testing approach as formulated in information theory, see [CT12]. It is centered around the divergence (or relative entropy, or Kullback Leibler distance), denoted $D(P_X||P_Y)$, between two distributions P_X and P_Y over the same alphabet and defined as $D(P_X||P_Y) = \sum_i P(X=i)\log\frac{P(X=i)}{P(Y=i)}$. The relative entropy is used to measure the distance between two distributions: the closer the distributions are, the smaller $D(P_X||P_Y)$ would be. If the ditsributions are the same, $D(P_X||P_Y) = 0$.

Furthermore, if we have a sequence of n sample symbols $\mathbf{x} = x_0, x_1, \ldots, x_{n-1}$ from the same alphabet $\mathcal{A}$ then we can count the number of occurrences of each symbol $a \in \mathcal{A}$, denoted $N(a|\mathbf{x})$ and forming the *type* (or empirical distribution; or sample distribution) by assigning $P(X = a) = N(a|\mathbf{x})/n$.

Let us denote the uniform distribution as P_U, the noise distribution derived in Section 3 as P_N. Assume we collect n samples $\mathbf{x}$ from an unknown distribution P_X. Then the hypothesis testing can be modeled as below with two hypotheses:

$$\begin{cases} H_0 : P_X = P_N, \\ H_1 : P_X = P_U. \end{cases} \tag{5}$$

In our case we are considering 24-bit distributions, so $|\mathcal{A}| = 2^{24}$ and the sample distribution is denoted P_{X^n}, with n as the length of the sample symbols.

We use the Neyman-Pearson lemma to make the optimum decision for the hypothesis testing, according to the distances from P_{X^n} to P_U and P_N, respectively. The decision problem is a maximum-likelihood test and the log-likelihood ratio can be written as

$$L = nD(P_{X^n}||P_U) - nD(P_{X^n}||P_N).$$

Then we define the decision rule as below

$$P_X = \begin{cases} P_N & if\ D(P_{X^n}||P_U) > D(P_{X^n}||P_N), \\ P_U & if\ D(P_{X^n}||P_U) < D(P_{X^n}||P_N). \end{cases} \tag{6}$$

With the hypothesis-testing problem defined above and P_N being the 24-bit noise distribution from the previous section, we build a distinguisher to decide the underlying sample distribution. We run 64 parallel SNOW 3G instances with random initial states, each clocking 2^{40} times and collect the targeted samples. At each clock t, we combine $(z^{(t-1)} \oplus s_0^{(t-1)} \oplus s_{15}^{(t-1)})[0], (z^{(t)} \oplus s_{15}^{(t)} \oplus s_0^{(t)})[0], (L_1^{-1}[z^{(t+1)} \oplus s_0^{(t+1)} \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}])[0]$, which is exactly the xor-sum of the keystream and LFSR part in (3), to make a 24-bit integer and increase the occurrence of the corresponding entry in the distribution table. We also collect the least three significant bytes of $z^{(t)}$ and concatenate them into a 24-bit variable, which is regarded as a comparison sample drawn from a uniform distribution.

After this process, we get the tables of occurrences of all possible 24-bit values and their corresponding probabilities. Then these sample distributions are tested by the decision rule given in (6) to get the answer to which distribution they follow, by calculating the distances to the uniform distribution P_U and noise distribution P_N, respectively. There are two types of errors to the correctness of the distinguisher: TYPE I errors, the errors of guessing a noise distribution as random; and TYPE II errors, the errors of falsely guessing a uniform distribution as the biased one.

Figure 2 shows the distances of one sample sequence to a uniform distribution and the noise distribution and their differences under different lengths of samples. We can see that with an increase in the length of samples, the distances to the uniform and noise distribution, which are quite close, are both decreasing. Even so, we could still distinguish the distribution by the difference of the two distances, i.e., $D(P_X||P_U) - D(P_X||P_N)$. While fluctuating around 0 in the beginning, the difference becomes stably positive after length $2^{39.58}$, indicating that the sample distribution is closer to the noise distribution and we get the correct guess.

Figure 3 shows the TYPE I and II error probabilities for the distinguisher under different lengths of samples. We could see that while fluctuating, the error probabilities are becoming smaller with an increase in the amount of samples, which indicates the guesses are becoming more accurate. From length 2^{40}, we can distinguish the samples with large success probabilities, while at the length $2^{41.5}$, there are no errors in our 21 sample sequences. The result matches well with the bias obtained in Section 3 and the conclusion that $O(1/\epsilon)$ samples are needed to distinguish the distribution from random when the bias is ϵ.

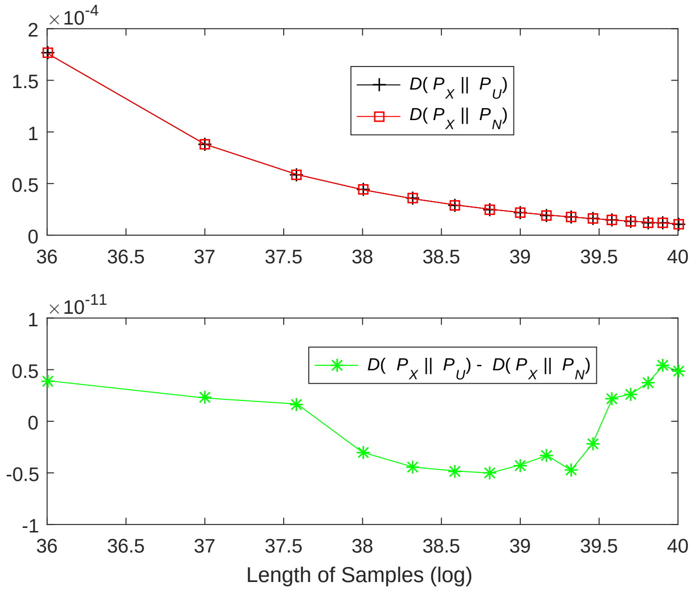

Figure 2: Distances to the uniform distribution and noise distribution

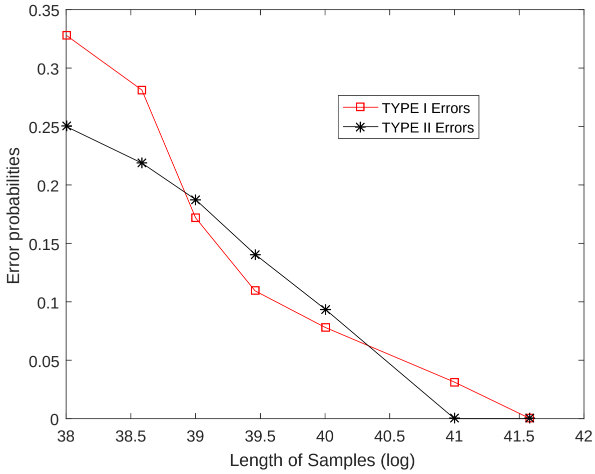

Figure 3: Error probabilities under different lengths of samples

5 Attacks based on the new vectorized linear approximations

We are now ready to use our vectorized linear approximations of the FSM to launch attacks. We recall the approximation on three bytes we derived in Section 3 is of the form below,

$$\left(z^{(t-1)}[0], z^{(t)}[0], (L_1^{-1}z^{(t+1)})[0]\right) = \left((s_{15}^{(t-1)} \oplus s_0^{(t-1)})[0], (s_{15}^{(t)} \oplus s_0^{(t)})[0], (L_1^{-1}s_5^{(t)} \oplus L_1^{-1}s_{15}^{(t+1)} \oplus L_1^{-1}s_0^{(t+1)})[0]\right) + (n_0, n_1, n_2)\,, \quad (7)$$

where (n_0, n_1, n_2) denotes the noise in the 24-bit linear approximation. In Section 3 we have computed the bias of this noise to be about 2^{-37} and experimentally verified it. We now consider how to launch a distinguishing attack with this 24-bit approximation in the next subsection. Fast correlation attacks will be considered in Subsection 5.2.

5.1 A distinguishing attack

In a distinguishing attack we build an algorithm that takes a sequence as input and determines with a small error probability whether the sequence stems from the considered keystream generator, or it is a truly random sequence. A potential application would be a case when only two messages $\mathbf{m}, \mathbf{m}'$ are possible, and from the ciphertext $\mathbf{c}$ only, we would like to determine which message was sent. One then forms a candidate keystream by computing $\mathbf{c} \oplus \mathbf{m}$ and inputs this to the distinguisher. If the distinguisher finds that this candidate keystream is likely to have been generated from the keystream generator, it is likely that the sent message was $\mathbf{m}$.

The basic idea for finding a distinguishing attack in our scenario is to completely remove the contribution from the LFSR part, leaving only a linear function of known output symbols as a sample from a noisy distribution. After collecting enough samples, one can distinguish the considered keystream from a truly random sequence.

Considering the relationship in (7), we would like to cancel the LFSR contribution

$$S^{(t)} = \left((s_{15}^{(t-1)} \oplus s_0^{(t-1)})[0], (s_{15}^{(t)} \oplus s_0^{(t)})[0], (L_1^{-1} s_5^{(t)} \oplus L_1^{-1} s_{15}^{(t+1)} \oplus L_1^{-1} s_0^{(t+1)})[0] \right).$$

Since $s_i^{(j)} = s_0^{(i+j)}$ and for simplicity, we write $s_0^{(t)}$ simply as $s^{(t)}$. It is easy to verify the following theorem.

Theorem 1. *If one can find t_1, t_2, t_3 such that $s^{(0)} \oplus s^{(t_1)} \oplus s^{(t_2)} \oplus s^{(t_3)} = 0$ then*

$$S^{(t)} \oplus S^{(t+t_1)} \oplus S^{(t+t_2)} \oplus S^{(t+t_3)} = (0, 0, 0).$$

Proof. Since $s^{(0)} \oplus s^{(t_1)} \oplus s^{(t_2)} \oplus s^{(t_3)} = 0$, with any time shift the equation would still hold, i.e., $s^{(t)} \oplus s^{(t+t_1)} \oplus s^{(t+t_2)} \oplus s^{(t+t_3)} = 0$. For the other terms in $S^{(t)}$, the xor-sum from the values at $0, t_1, t_2, t_3$ would also be 0. Let us take the term $s_{15}^{(t-1)}$ for example: since $s_{15}^{(t-1)} = s_0^{(t+14)}$, then $\bigoplus_{i=0}^{3} s_{15}^{(t+t_i-1)} = \bigoplus_{i=0}^{3} s_0^{(t+t_i+14)} = 0$. Then we have $\bigoplus_{i=0}^{3} S^{(t+t_i)} = (0, 0, 0)$. □

Assuming that we have found t_1, t_2, t_3 satisfying Theorem 1, we can create samples from a biased distribution by computing samples x_t as

$$x^{(t)} = \sum_{i=0}^{3} \left(z^{(t+t_i-1)}[0], z^{(t+t_i)}[0], (L_1^{-1} z^{(t+t_i+1)})[0] \right),$$

where $t_0 = 0$. The samples $x^{(t)}$ are then drawn from a noisy distribution, which is the distribution of the sum of 4 noise variables like N_{tot}. This was previously computed to have a bias of $\epsilon(4 \times N_{tot}) = 2^{-163}$ and hence it requires in the order of 2^{163} keystream symbols to distinguish the samples from a uniform distribution.

The remaining problem here is to examine the computational complexity of finding t_1, t_2, t_3 satisfying $s^{(0)} \oplus s^{(t_1)} \oplus s^{(t_2)} \oplus s^{(t_3)} = 0$. The sequence $s^{(t)}$ is generated using the feedback polynomial $P(x) = \alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1 \in GF(2^{32})[x]$. We are thus seeking a weight 4 multiple $K(x)$ of the feedback polynomial where all coefficients are set to one. We may first argue about the expected degree q of such a polynomial. Let us consider all

$t \leq q$, then we can create $\binom{q}{3}$ different combinations of $s^{(0)} \oplus s^{(t_1)} \oplus s^{(t_2)} \oplus s^{(t_3)}$ expressed in the initial state. Since there are 2^{512} possible such combinations, we can expect that we need to go to degree such that roughly $q^3/6 \approx 2^{512}$, resulting in $q \approx 2^{172}$.

Finally, we need an efficient way to find a weight 4 multiple. Here we use a slight generalization of the algorithm proposed by Löndahl and Johansson in [LJ14]. The algorithm solves the problem with computational complexity of only around 2^d, where $d = \log q$, and similar storage. The algorithm uses the idea of duplicating the desired multiple to many instances and then finding one of them with very large probability. The solution to problem associated to the SNOW 3G case can be described as follows:

Assume that $K(x)$ is the weight 4 multiple of the lowest degree and assume that its degree is around 2^d as expected. The Algorithm 1 considers and creates all weight 4 multiples up to degree 2^{d+b} where b is a small integer, but will only find those that include two monomials x^{i_1} and x^{i_2} such that $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$, where $\phi()$ means the d least significant bits.

Algorithm 1 Finding a multiple of $P(x)$ with weight 4 and all nonzero coefficients one

Input Polynomial $P(x)$, a small integer b

Output A polynomial multiple $K(x) = P(x)Q(x)$ of weight 4 and expected degree 2^d with nonzero coefficients set to be one

1. From $P(x)$, create all residues $x^{i_1} \bmod P(x)$, for $0 \leq i_1 < 2^{d+b}$ and put $(x^{i_1} \bmod P(x), i_1)$ in a list $\mathcal{L}_1$. Sort $\mathcal{L}_1$ according to the residue value of each entry.

2. Create all residues $x^{i_1} + x^{i_2} \bmod P(x)$ such that $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$, for $0 \leq i_1 < i_2 < 2^{d+b}$ and put in a list $\mathcal{L}_2$. Here $\phi()$ means the d least significant bits. This is done by merging the sorted list $\mathcal{L}_1$ by itself and keeping only residues $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$. The list $\mathcal{L}_2$ is sorted according to the residue value.

3. In the final step we merge the sorted list $\mathcal{L}_2$ with itself to create a list $\mathcal{L}$, keeping only residues $x^{i_1} + x^{i_2} + x^{i_3} + x^{i_4} = 0 \bmod P(x)$.

As $K(x)$ is of weight 4, any polynomial $x^{i_1}K(x)$ is also of weight 4 and since we consider all weight 4 multiples up to degree 2^{d+b} we will consider $2^{d+b} - 2^d$ such weight 4 polynomials, i.e. about $2^d(2^b - 1)$ duplicates of $K(x)$. As the probability for a single weight 4 polynomial to have the condition $\phi(x^{i_1} + x^{i_2} \bmod P(x)) = 0$ can be approximated to be around 2^{-d}, there will be a large probability that at least one duplicate of $x^{i_1}K(x)$ will survive in Step 2 in Algorithm 1 and will be included in the output. Further details and experimental verification can be found in [LJ14].

Regarding complexity, we note that the tables are all of size around 2^d. Creation of $\mathcal{L}_1$ costs roughly 2^d and creation of $\mathcal{L}_2$ costs about the same as we are only accessing entries in $\mathcal{L}_1$ with the same d least significant bits. For a sufficiently large b, one iteration of Algorithm Algorithm 1 (inner loop) succeds with a high probability.

To conclude, we have described a distinguishing attack on SNOW 3G for which we need a keystream length of around 2^{172} and similar complexity. It uses a precomputation step of complexity around 2^{172} and required memory is of the same size.

5.2 A fast correlation attack

A fast correlation attack is a key recovery attack, which is hence a much stronger attack than a distinguishing attack. It tries to recover the key by exploring the correlation between the keystream and the output of the LFSR states, which always exists for nonlinear functions[Sie84]. It is commonly modeled as a decoding problem in $GF(2)^n$ or $GF(2^n)$, with the observed keystream samples $\mathbf{y} = (y_0, y_1, ..., y_{N-1})$ being the noisy version of the LFSR sequence $\mathbf{u} = (u_0, u_1, ..., u_{N-1})$ through a discrete memoryless channel (DMC) with non-uniform noise $\mathbf{e} = (e_0, e_1, ..., e_{N-1})$, i.e., $y_i = u_i + e_i$ for $1 \leq i \leq N - 1$. It should be noted here that $\mathbf{u}$ and $\mathbf{z}$ might not be the exact output from the LFSR and the keystream of the considered keystream generator, but can be some linear combinations of them. The LFSR sequence $\mathbf{u} = (u_0, u_1, ..., u_{N-1})$ is regarded as a codeword generated from the information words $(u_0, u_1, ..., u_{l-1})$, which is the initial state of LFSR by $\mathbf{u} = (u_0, u_1, ..., u_{l-1})G$, where $G = (\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_{N-1})$ with each element $\mathbf{g}_i$ being a column vector of length l. Then the correlation attack is converted into decoding an $[N, l]$ linear code with N being the code length and l the length of the information word. The main problem is to find methods of efficient decoding as a straight-forward approach of running through all codewords to find the closest one to the received vector requiring a huge computational complexity. A common approach is to find parity checks that can be combined to form partity checks in a lower dimensional code, which is then more efficiently decoded.

The correlation attacks always have two stages: a preprocessing stage, during which as many parity check equations of the $[N, l]$ code as possible are generated and processed; and an online decoding stage, during which the decoding is executed according to these generated parity check equations. Research on correlation attacks have been mainly focused on these two aspects: finding parity check equations with low weights [Pen96], [CT00], typically 3, 4 or 5, and exploring more effecient decoding techniques to solve the decoding process [JJ99b], [JJ99a].

The framework of a fast correlation attack is defined over a finite field. It is usually the binary case but a larger alphabet is possible as long as the operations are in a finite field. Our strongest (24-bit) approximation does not fulfill this condition and it cannot be used in a straight-forward manner. Instead, we rely on the 8-bit approximation from (4), which is an approximation over $GF(2^8)$. Below we first give a correlation attack using the method in [ZXM15] with the byte-based linear approximation in (4). Precisely, the attack employs the k-tree algorithm in [Wag02] to generate parity check equations during the preprocessing stage and a Fast Walsh Transform (FWT) to accelerate the decoding process during the decoding stage.

5.2.1 A fast correlation attack based on an approximation in $GF(2^8)$

We consider the two sequences,

$$y^{(t)} = \Lambda z^{(t-1)}[0] \oplus z^{(t)}[0] \oplus \Gamma(L_1^{-1} z^{(t+1)})[0]$$

and

$$u^{(t)} = (\Lambda(s_0^{(t-1)} \oplus s_{15}^{(t-1)}) \oplus s_0^{(t)} \oplus s_{15}^{(t)} \oplus \Gamma L_1^{-1}[s_0^{(t+1)} \oplus s_5^{(t)} \oplus s_{15}^{(t+1)}])[0],$$

both defined in $GF(2^8)$ and we know from before that $y^{(t)} = u^{(t)} + e^{(t)}$, where $e^{(t)}$ has the same distribution as N'_{tot}. Note that $u^{(t)}$ is an LFSR sequence of length 64 defined over $GF(2^8)$, while the $s^{(t)}$ sequence is defined over an extension field of $GF(2^8)$. It can be seen that the LFSR with state $(s_0, s_1, ..., s_{15}) \in GF(2^{32})^{16}$ can also be described through a length 64 LFSR over $GF(2^8)$ with states $(u_0, u_1, ...u_{62}, u_{63}) \in GF(2^8)^{64}$. Below we give the 8-bit correlation attack based on such an LFSR.

Preprocessing Stage: Generating the Parity Check Equations

Since decoding an $[N, l]$ linear code involves a large complexity when l is large, the method in [CJS00] can be used to convert the $[N, l]$ code $\mathcal{C}_1$ into a simpler one $[N', l']$ denoted $\mathcal{C}_2$ with $l' < l$. The key point is to find a k-tuple of column vectors $(\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, ..., \mathbf{g}_{i_k})$ from G satisfying $\mathbf{g}_{i_1} \oplus \mathbf{g}_{i_2} \oplus ... \oplus \mathbf{g}_{i_k} = (c_0, c_1, ..., c_{l'-1}, 0, ..., 0)^T$, i.e., the xor-sums of the last $l - l'$ elements are all zero. Then for such a tuple, the following equation holds, which is a parity check for $u_1, ..., u_{l'}$,

$$\bigoplus_{j=1}^{k} u_{i_j} = (u_0, u_1, ...u_{l-1}) \bigoplus_{j=1}^{k} \mathbf{g}_{i_j} = c_0 u_0 \oplus c_1 u_1 \oplus ... \oplus c_{l'-1} u_{l'-1}.$$

Correspondingly,

$$\bigoplus_{j=1}^{k} y_{i_j} = \bigoplus_{j=1}^{k} (u_{i_j} \oplus e_{i_j}) = c_0 u_0 \oplus c_1 u_1 \oplus ... \oplus c_{l'-1} u_{l'-1} \oplus \bigoplus_{j=1}^{k} e_{i_j}.$$

If we denote $Y_i = \bigoplus_{j=1}^{k} y_{i_j}$, $U_i = \bigoplus_{j=1}^{k} u_{i_j}$ and $E_i = \bigoplus_{j=1}^{k} e_{i_j}$, we have $Y_i = U_i \oplus E_i$. If we collect N' such parity checks, we can construct a new $[N', l']$ code, with $\mathbf{U} = (U_0, U_1, ..., U_{N'-1})$ being the output of a converted LFSR with l' states and $\mathbf{Y} = (Y_0, Y_1, ..., Y_{N'-1})$ being the noisy version of U_i through a more noisy channel with noise $\mathbf{E} = (E_0, E_1, ..., E_{N'-1})$. Since $l' < l$, the decoding complexity is reduced. Then the remaining work is to solve the decoding problem efficiently, which we will describe in detail in the processing stage.

As for the complexity for the preprocessing stage, with the k-tree algorithm in [Wag02] employed to find such parity check equations, the time/space complexities are $O(k2^{n(l-l')/(1+\log k)})$ and the sizes of lists are $O(2^{n(l-l')/(1+\log k)})$, with n being the size of the finite field, $n = 8$ in our case. Note that $\rho^{1+\log k}$ such tuples could be found with ρ times as much work as finding a single solution, i.e., $O(\rho k 2^{n(l-l')/(1+\log k)})$ for time and space and $O(\rho 2^{n(l-l')/(1+\log k)})$ for the size of each list, as long as $\rho \leq 2^{n(l-l')/(\log k(1+\log k)}$.

Processing Stage: Decoding the code

We now move to the process of decoding the $[N', l']$ code, following the method in [ZXM15] and using the Fast Walsh Transform (FWT) to accelerate the decoding process. The main idea is to make a distinguisher defined as $I(\hat{\mathbf{u}}) = c_{i_0}(u'_0 \oplus u_0) \oplus ... \oplus c_{i_{l'-1}}(u'_{l'-1} \oplus u_{l'-1}) \oplus E_i = Y_i \oplus c_{i_0} u'_0 \oplus ... \oplus c_{i_{l'-1}} u'_{l'-1}$ for a guess $\hat{\mathbf{u}} = (u'_0, u'_1, ..., u'_{l'-1})$ of the first l' LFSR states, where i denotes the i-th tuple. $I(\hat{\mathbf{u}})$ would be biased for the correct guess since only the noise term E_i remains.

The next step is to check the balancedness of $I(\hat{\mathbf{u}})$ for every guessed $\hat{\mathbf{u}}$ to find the correct key. Firstly, the correlations $c(\langle a, I\rangle)$ of the Boolean function $\langle a, I\rangle$, i.e., the inner product of a and I where $a \in GF(2)^n$, is obtained and then the SEI of $I(\hat{\mathbf{u}})$ can be derived by $\Delta(\hat{\mathbf{u}}) = \sum_{a \in GF(2^m)} c^2(\langle a, I\rangle)$ according to [NH07]. Then we can verify whether $I(\hat{\mathbf{u}})$ is biased or not and further recover the key. To get the correlations $c(\langle a, I\rangle)$ efficiently, the method in [LV04] could be used. Firstly, the vectorial boolean function I can be divided into n linearly independent boolean functions $I_1, ..., I_n$, each expressed as $I_j = \langle w_j, \hat{\mathbf{u}}\rangle \oplus \langle v_j, Y_i\rangle$ where $w_j \in GF(2)^{nl'}, v_j \in GF(2)^n$ are two binary coefficient vectors. Then FWT can be used to compute the correlation of each I_i. It is stated in [ZXM15] that the total correlation can be further derived by the Piling-up Lemma. We refer to other work like [ZXM15] for a more detailed description of this process and we use the complexity formulas from [ZXM15].

For SNOW 3G, we use the 8-bit linear approximation which has a bias of $\epsilon(N'_{tot}) \approx 2^{-40.970689}$. We can first rewrite the LFSR sequence symbols as linear functions of 64 initial state bytes, with a new and more complex generating polynomial. Then use the preprocessing stage described before to generate the parity check equations with parameters $l = 64$, $k = 4$. The SEI of $k = 4$ folded noise variables is $\epsilon(4 \times N'_{tot}) \approx 2^{-163.877500}$. We then tested different choices for l' and found that under $l' = 20$ the total complexity is the lowest. The number of parity check equations m_k required in this case is $m_k = 2^{171.67}$. The time/space complexity of preprocessing is $\rho k 2^{n(l-l')/(1+\log k)} = 2^{176.56}$ and the required length of the keystream is $2^{176.56}$. With complexity of $n(m_k + l'n2^{l'n}) + 2^{n+l'n} = 2^{174.75}$, $20 \cdot 8 = 160$ bits of the LFSR initial states could be recovered. Therefore, the time/memory/data/pre-computation complexities are all upper bounded by $2^{176.56}$.

5.2.2 Potential Correlation Attack using a 16-bit approximation

In Section 3, we got the 24-bit and 8-bit linear approximations with biases 2^{-37} and 2^{-41}, respectively. We would obviously like to use the 24-bit approximation to launch a correlation attack. But as explained before, the 24-bit approximation is not defined over a finite field and cannot be used directly in a fast correlation attack. Now we report some findings on building a 16-bit approximation by an experimental method based on the 8-bit approximation derived before. Specifically, we concatenate two consecutive 8-bit symbols to build one 16-bit symbol, i.e., $(y^{(t)}, y^{(t+1)})$. The theoretical distribution for such a pair of bytes is too computationally consuming to compute, but since we know the bias for a single byte we can get an bound on the bias.

We run a large amount of SNOW 3G instances in parallel with random initial states and collect $(y^{(t)}, y^{(t+1)})$ by (4) at each clock, obtaining 2^{53} 16-bit samples in total. We then record the occurrence of each entry in the distribution table, and got the bias $2^{-36.8293}$. However, the bias here is not very accurate and more samples are needed for full confidence. Even so, we could still get a general estimation of the bias. After we collected these samples, we used the method in Section 4 to distinguish a 16-bit and 8-bit sample between the uniform distribution and the respective 16-bit and 8-bit distributions we derived.

Table 1 shows the TYPE I errors and probabilities for the two cases under different lengths of samples. We can see directly from the table that the error probability for 16-bit distinguishing is much smaller than the 8-bit one, indicating the bias of the 16-bit approximation is larger than the latter. Considering the error probabilities to be 0 for 16-bit and 8-bit distinguishing are after lengths 2^{42} and 2^{46}, respectively, we could make a general estimation that the bias for the 16-bit approximation is around 2^{-38}.

Table 1: Errors and error probabilities(in the brackets) under different lengths of samples

Length Samples	2^{40} 8192	2^{41} 4096	2^{42} 2048	2^{43} 1024	2^{44} 512	2^{45} 256	2^{46} 128
16-bit	626(0.076)	94(0.023)	3(0.001)	0	0	0	0
8-bit	2895(0.353)	1260(0.308)	445(0.217)	148(0.144)	40(0.078)	7(0.027)	0

Now we briefly explain how to use this 16-bit approximation in a correlation attack. First we point out that any output from the LFSR at clock t, $u^{(t)}$, can be derived from the initial states $(u_0, u_1, ..., u_{63})$ by $u^{(t)} = \bigoplus_{i=0}^{63} c_i^{(t)} u_i$ for $0 \leq i \leq 63$, where $c_i^{(t)} \in GF(2^8)$. Then we have,

$$y^{(t)} = u^{(t)} + e^{(t)} = \bigoplus_{i=0}^{63} c_i^{(t)} u_i \oplus e^{(t)}. \tag{8}$$

At the next clock $t+1$, the value in the $(i+1)$-th cell is shifted to the i-th cell for $0 \leq i \leq 62$ and only the 63-rd cell is updated, which can be expressed as

$$u^{(t+1)} = \bigoplus_{i=0}^{62} c_i^{(t)} u_{i+1} \oplus c_{63}^{(t)} u'_{63},$$

where u'_{63} is the new value for the 63-rd cell updated by $u'_{63} = \bigoplus_{i=0}^{62} \gamma_i u_i$, where γ_i's are the feedback coefficients of the LFSR in $GF(2^8)$. Then $y^{(t+1)}$ can be expressed as,

$$y^{(t+1)} = u^{(t+1)} + e^{(t+1)} = \bigoplus_{i=0}^{62} c_i^{(t)} u_{i+1} \oplus c_{63}^{(t)} \bigoplus_{i=0}^{62} \gamma_i u_i \oplus e^{(t+1)}. \tag{9}$$

Assume by the k-tree algorithm, we have found a k-tuple combination, say $k = 4$, with (t_1, t_2, t_3, t_4) which maps the xor-sum of the output to the first l' 8-bit symbols in the LFSR, i.e.,

$$\bigoplus_{j=1}^{4} y^{(t_j)} = \bigoplus_{j=1}^{4}(\bigoplus_{i=0}^{63} c_i^{(t_j)} u_i \oplus e^{(t_j)}) = \bigoplus_{i=0}^{l'-1} c_i u_i \oplus E,$$

i.e., $c_i^{(t1)} \oplus c_i^{(t2)} \oplus c_i^{(t3)} \oplus c_i^{(t4)} = c_i$ for $0 \leq i \leq l'-1$, while $c_i^{(t1)} \oplus c_i^{(t2)} \oplus c_i^{(t3)} \oplus c_i^{(t4)} = 0$ for $l' \leq i \leq 63$, where $E = \bigoplus_{j=1}^{4} e^{(t_j)}$. We aim to build another part of the 16-bit symbol by getting $\bigoplus_{i=1}^{4} y^{(t_i+1)}$ from (9) . We get

$$\bigoplus_{j=1}^{4} y^{(t_j+1)} \;=\; \bigoplus_{j=1}^{4}(u^{(t_j+1)} \oplus e^{(t_j+1)}) \;=\; \bigoplus_{j=1}^{4}(\bigoplus_{i=0}^{62} c_i^{(t_i)} u_{i+1} \oplus c_{63}^{(t_i)} \bigoplus_{i=0}^{62} \gamma_i u_i \oplus e^{(t_i+1)}).$$

Since $c_i^{(t1)} \oplus c_i^{(t2)} \oplus c_i^{(t3)} \oplus c_i^{(t4)} = 0$ and $c_{63}^{(t1)}\gamma_i \oplus c_{63}^{(t2)}\gamma_i \oplus c_{63}^{(t3)}\gamma_i \oplus c_{63}^{(t4)}\gamma_i = 0$ for $l' \leq i \leq 63$, we could get

$$\bigoplus_{j=1}^{4} y^{(t_j+1)} = \bigoplus_{j=1}^{4}(\bigoplus_{i=0}^{l'-1} c_i^{(t_i)} u_{i+1} \oplus c_{63}^{(t_i)} \bigoplus_{i=0}^{l'-1} \gamma_i u_i \oplus e^{(t_i+1)}).$$

From above, $\bigoplus_{j=1}^{4} y^{(t_j+1)}$ can be mapped to the xor-sum of $(u_0, u_1, ...u_{l'})$ and we express it as

$$\bigoplus_{j=1}^{4} y^{(t_j+1)} = \bigoplus_{i=0}^{l'} c_i' u_i \oplus E',$$

with c_i' being some new coefficients and $E' = \bigoplus_{j=1}^{4} e^{(t_i+1)}$. Then the 16-bit approximation can be expressed as:

$$(\bigoplus_{j=1}^{4} y^{(t_j)}, \bigoplus_{j=1}^{4} y^{(t_j+1)}) = (\bigoplus_{i=0}^{l'-1} c_i u_i, \bigoplus_{i=0}^{l'} c_i' u_i) \oplus (E, E').$$

Here (E, E') can be regarded as following the 16-bit distribution we obtained in the beginning of this subsection. We could see, compared to the 8-bit correlation attack where the output is mapped to the first l' states, here the 16-bit symbol is mapped to the first $l'+1$ states, i.e., one more state is involved. If we find N' tuples like this, we could build a new $[N', l'+1]$ code with the 16-bit approximation as the noisy channel. Then we proceed to the decoding stage to recover the $l'+1$ states.

Next, let us check the complexity. For the preprocessing phase, we could still use the k-tree method to find parity check equations, but through a different DMC with a smaller noise. Using the method before, we could get the best result in terms of complexity is now under $l' = 19$. Now $2^{159.72}$ parity check equations are required with time/space complexity of $2^{175.24}$ and the required length of keystream sample is $2^{175.24}$. For the decoding process, the complexity is $2^{176.06}$. From the result, we can find that while the complexity is still upper bounded by the same order around 2^{176}, the required number of parity check equations reduces from $2^{171.67}$ to $2^{159.72}$. These complexity results are based on only experimental result and are not with full confidence.

6 Conclusion

In this paper, we propose a distinguishing attack and correlation attack to SNOW 3G using new linear approximations over larger alphabets. We first derive a 24-bit and an 8-bit linear approximation of the FSM and verify them by an experimental test using hypothesis testing. Then we used the derived approximations to launch a distinguishing attack and correlation attack. For the distinguishing attack, we find a weight 4 mupltiple of the generating polynomial to cancel out the contribution from the LFSR and distinguish the corresponding keystream sample sequence with complexity around 2^{172}. For the correlation attack, we use the the 8-bit approximation to recover 160 bits of the secret key with complexity around 2^{177}. As far as we know, these are the first distinguishing and correlation attacks on SNOW 3G. If the key length in SNOW 3G would be increased to 256 bits, the results show that there are then academic attacks on such a version faster than the exhaustive key search.

A possible way to improve the results and achieve a higher bias would be to consider even larger alphabets in the approximations, but this would require much more complex simulation tasks to find and verify biases of different choices of approximation. Another interesting question would be to launch distinguishing attacks which are based directly on the weight 4 recurrence relation for the LFSR, which has nonzero coefficients that are not all one. If so, it would remove the requirement of having a very long keystream, but could be applied on a large set of short keystreams generated with different IVs.

References

[BHNS10] Billy Bob Brumley, Risto M Hakala, Kaisa Nyberg, and Sampo Sovio. Consecutive s-box lookups: A timing attack on snow 3g. In *International Conference on Information and Communications Security*, pages 171–185. Springer, 2010.

[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.

[BPSZ10a] Alex Biryukov, Deike Priemuth-Schmid, and Bin Zhang. Differential resynchronization attacks on reduced round snow 3g. In *International Conference on E-Business and Telecommunications*, pages 147–157. Springer, 2010.

[BPSZ10b] Alex Biryukov, Deike Priemuth-Schmid, and Bin Zhang. Multiset collision attacks on reduced-round snow 3g and snow 3g. In *International Conference on Applied Cryptography and Network Security*, pages 139–153. Springer, 2010.

[CHJ02] Don Coppersmith, Shai Halevi, and Charanjit Jutla. Cryptanalysis of stream ciphers with linear masking. In *Annual International Cryptology Conference*, pages 515–532. 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.

[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.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[DC09] Blandine Debraize and Irene Marquez Corbella. Fault analysis of the stream cipher snow 3g. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 103–110. IEEE, 2009.

[DR13] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

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

[JJ99a] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks based on turbo code techniques. In *Annual International Cryptology Conference*, pages 181–197. Springer, 1999.

[JJ99b] 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.

[LJ14] Carl Löndahl and Thomas Johansson. Improved algorithms for finding low-weight polynomial multiples in $F_2[x]$ and some cryptographic applications. *Designs, codes and cryptography*, 73(2):625–640, 2014.

[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.

[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.

[NW06a] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for SNOW 2.0. In *International Workshop on Fast Software Encryption*, pages 144–162. Springer, 2006.

[NW06b] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for snow 2.0. In *International Workshop on Fast Software Encryption*, pages 144–162. Springer, 2006.

[Pen96] Walter T Penzhorn. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes. In *International Workshop on Fast Software Encryption*, pages 159–172. Springer, 1996.

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

[UEA06a] I UEA2&UIA. Specification of the 3GPP confidentiality and integrity algorithms UEA2& UIA2, 2006.

[UEA06b] I UEA2&UIA. Specification of the 3GPP confidentiality and integrity algorithms UEA2& UIA2, document 5: Design and evaluation report, version 1.1, 2006.

[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.

[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.