Fast Correlation Attacks over Extension Fields, Large-unit Linear Approximation and Cryptanalysis of SNOW 2.0

Bin Zhang[†,‡], Chao Xu[†] and Willi Meier[◇]

[†] TCA Laboratory, SKLCS, Institute of Software, Chinese Academy of Sciences
[‡] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China
{zhangbin,xuchao}@tca.iscas.ac.cn
[◇] FHNW, Switzerland
willi.meier@fhnw.ch

Abstract. Several improvements of fast correlation attacks have been proposed during the past two decades, with a regrettable lack of a better generalization and adaptation to the concrete involved primitives, especially to those modern stream ciphers based on word-based LFSRs. In this paper, we develop some necessary cryptanalytic tools to bridge this gap. First, a formal framework for fast correlation attacks over extension fields is constructed, under which the theoretical predictions of the computational complexities for both the offline and online/decoding phase can be reliably derived. Our decoding algorithm makes use of Fast Walsh Transform (FWT) to get a better performance. Second, an efficient algorithm to compute the large-unit distribution of a broad class of functions is proposed, which allows to find better linear approximations than the bitwise ones with low complexity in symmetric-key primitives. Last, we apply our methods to SNOW 2.0, an ISO/IEC 18033-4 standard stream cipher, which results in the significantly reduced complexities all below $2^{164.15}$. This attack is more than 2^{49} times better than the best published result at Asiacrypt 2008. Our results have been verified by experiments on a small-scale version of SNOW 2.0.

Keywords: Stream ciphers, Cryptanalysis, Large-unit, SNOW 2.0, Finite state machine (FSM), Linear feedback shift register (LFSR)

1 Introduction

The design and analysis of any cipher in history have to match well with the computing technologies in a specified period. Fast correlation attacks, introduced by Meier and Staffelbach in 1989 [19], are commonly regarded as classical methods in the cryptanalysis of LFSR-based stream ciphers, which were usually implemented in hardware at that time. In general, fast correlation attacks have been constantly and steadily evolving [4, 5, 11], resulting in more and more powerful decoding methods dedicated to very large linear codes in the presence of a highly noisy channel.

On the other side, with the development of computing facilities, many word-oriented stream ciphers have been proposed, e.g., SNOW 2.0, SNOW 3G [6, 8] and Sosemanuk [2], aiming to combine the merits from the thoroughly studied LFSR theory with a fast implementation in software. Due to the complex form of the reduced LFSR recursion from the extension field to GF(2) (many taps and a large number of state variables), the previous bitwise fast correlation attacks do not work so well as expected in these cases. This motivates us to study the security of these word-oriented primitives against a new form of fast correlation attacks that works on some larger data unit.

Our Contributions. First, a formal framework for fast correlation attacks over extension fields is constructed, under which the theoretical predictions of the computational complexities for both the offline and online/decoding phase can be reliably derived. This gives an answer to the open problem of Meier in [18] at FSE 2011. We adapt the k-tree algorithm [24] to generate the desirable parity check equations in the pre-computation phase and propose a fast decoding algorithm for the online phase. Second, an efficient algorithm to compute the large-unit distributions of the generalized pseudo-linear functions modulo 2^n (GPLFM), which includes all the previously studied relevant topics [17] in an unified framework, is proposed. This technique, serving as a basis to the first one, generalizes the algorithm in [22] and has the value in its own right. It can compute the noise distributions of the linear approximations of the GPLFM (including the addition modulo 2^n) in a larger alphabet of m-bit ($m > 1$) size when m is divisible by n with a low complexity, e.g., for $n = 32$, the $2, 4, 8, 16$-bit linear approximations can be found efficiently with a slice size depending on the structure of the primitive. Last, we apply our methods to SNOW 2.0, an ISO/IEC 18033-4 standard and a benchmark stream cipher in the European eSTREAM project. We build the byte-wise linear approximation of the FSM by further generalizing the G-PLFM to include the S-boxes and restore the initial state of the LFSR (thus the key) with a fast correlation attack over GF(2^8). The time/memory/data/pre-computation complexities of this attack are all below $2^{186.95}$. Then we further improve our attack by changing the linear mask from GF(2) to GF(2^8), which results in the significantly reduced time/memory/data/pre-computation complexities all below $2^{164.15}$. This attack is more than 2^{49} times better than the best published result at Asiacrypt 2008[1]. Table 1 presents a comparison of our attack on SNOW 2.0 with the best previous ones. Our results have been verified

Table 1. Comparison of the attacks on SNOW 2.0

	type	data	time
[22]	Distinguishing attack	2^{174}	2^{174}
[13]	Key recovery attack	$2^{198.77}$	$2^{212.38}$
this paper	Key recovery attack	$2^{163.59}$	$2^{164.15}$

[1] Note that in the Asiacrypt 2008 paper [13], the complexity is written as $2^{204.38}$ in the abstract, while from the formula in Section 6, this complexity is $2^{212.38}$.

on a small-scale version of SNOW 2.0 with 16-bit word size in experiments.

Outline. We present some preliminaries relevant to our work in Section 2. In Section 3, the framework of fast correlation attacks over extension fields is established with detailed theoretical justifications. The new algorithm to accurately and efficiently compute the large-unit distribution of the GPLFM is provided in Section 4. The application of our approaches to SNOW 2.0 is given in Section 5. The improved attack using finite field linear masks is described in Section 6 with the experimental results. Finally, some conclusions are made and future work is pointed out in Section 7.

2 Preliminaries

In this section, some notations and basic definitions are presented. Denote the set of real numbers by $\mathbf{R}$. The binary field is denoted by GF(2) and the m-dimensional extension field of GF(2) is denoted by GF(2^m). The modular addition is $\boxplus$ and the usual xor operation is $\oplus$. The inner product of two n-dimensional vectors a and b over GF(2^m) is defined as $\langle a, b\rangle = \langle (a_0, \cdots, a_{n-1}), (b_0, \cdots, b_{n-1})\rangle := \bigoplus_{i=0}^{n-1} a_i b_i$. As usual, a function $f : \mathrm{GF}(2^n) \to \mathrm{GF}(2)$ is called a Boolean function and a function $g = (g_1, \cdots, g_m) : \mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$ with each g_i $(1 \le i \le m)$ being a Boolean function is called a m-dimensional vectorial Boolean function.

Definition 1 *Let X be a binary random variable, the correlation between X and zero is defined as $c(X) = Pr\{X = 0\} - Pr\{X = 1\}$. The correlation of a Boolean function $f : GF(2^n) \to GF(2)$ to zero is defined as $c(f) = Pr\{f(X) = 0\} - Pr\{f(X) = 1\}$, where $X \in GF(2^n)$ is an uniformly distributed random variable.*

Given a vectorial Boolean function $g : \mathrm{GF}(2^n) \to \mathrm{GF}(2^m)$, define the distribution p_g of $g(X)$ with X uniformly distributed as $p_g(a) = \#\{X | g(X) = a\}/2^n$ for all $a \in \mathrm{GF}(2^m)$.

Definition 2 *As in [1], the Squared Euclidean Imbalance (SEI) of p_g is $\Delta(p_g) = 2^m \sum_{a \in GF(2^m)} (p_g(a) - \frac{1}{2^m})^2$, which measures the distance between the target distribution and the uniform distribution.*

SEI[2] is used to evaluate the efficiency of large-unit linear approximations in this paper. Here by *large-unit*, we refer to the linear approximation whose basic data unit is non-binary. The next definition introduces a powerful tool to compute the correlation of a nonlinear function and to reduce the complexity of the substitution step of a fast correlation attack [5].

Definition 3 *Given a function $f : GF(2^n) \to \mathbf{R}$, for $\omega \in GF(2^n)$, the Walsh Transform of f at point ω is defined as $\hat{f}(\omega) = \sum_{x \in GF(2^n)} f(x)(-1)^{\langle \omega, x\rangle}$.*

[2] SEI is also referred to as capacity of the distribution in [9].

The Walsh Transform of f can be computed efficiently with an algorithm called Fast Walsh Transform (FWT) [25] in $n2^n$ time and 2^n memory. The preparation of f takes 2^n time, thus the total time complexity is $2^n + n2^n$, which is a large improvement compared to 2^{2n}. The following fact [21] is used in our analysis.

Lemma 4 *We consider a vectorial Boolean function $g : GF(2^n) \to GF(2^m)$ with the probability distribution vector p_g. Then $\Delta(p_g) = \sum_{a \in GF(2^m)} c^2(\langle a, g \rangle)$, where $c(\langle a, g \rangle)$ is the correlation of the Boolean function $\langle a, g \rangle$.*

Lemma 4 indicates that we can derive the SEI of distribution p_g with the correlations $c(\langle a, g \rangle)$ for $a \in \mathrm{GF}(2^m)$. Therefore, computing the SEI of the large data unit distribution can be reduced to the problem of looking for bitwise linear approximations with non-negligible correlations.

3 Fast Correlation Attacks over Extension Fields

In this section, we will describe a formal framework for fast correlation attacks over $\mathrm{GF}(2^n)$, which is the *first* comprehensive answer to the open problem how to amount fast correlation attack over the extension fileds proposed in [3] and [18]. Let us first define the notations used hereafter.

- N is the number of available output words.
- l is the word-length of the LFSR over $\mathrm{GF}(2^n)$.
- l' is the number of target words in decoding phase.
- G is the $l \times N$ generator matrix of a $[N, l]$ linear code $\mathcal{C}_1$ over $\mathrm{GF}(2^n)$.
- $u_i \in \mathrm{GF}(2^n)$ is the i-th output word of the LFSR.
- $z_i \in \mathrm{GF}(2^n)$ is the i-th output word of the keystream generator.
- $e_i \in \mathrm{GF}(2^n)$ is the i-th noisy variable of a Discrete Memoryless Channel (DMC).

3.1 Model for Fast Correlation Attacks over Extension Fields

The fast correlation attack over extension fields is also modelled as a decoding problem, i.e., the keystream segment $\mathbf{z} = (z_1, z_2, \cdots, z_N)$ can be seen as the transmission result of the LFSR sequence $\mathbf{u} = (u_1, u_2, \cdots, u_N)$ through a DMC with the noisy variables $\mathbf{e} = (e_1, e_2, \cdots, e_N)$, as shown in Fig.1. From this model,

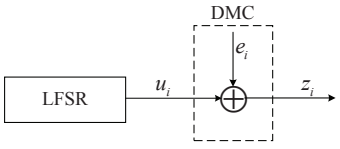

Fig. 1. Model for fast correlation attacks over $\mathrm{GF}(2^n)$

we can represent the received symbols z_i as $z_i = u_i \oplus e_i$, where the noise variable

e_i is non-uniformly distributed for $i = 1, \cdots, N$. The capacity of the DMC is $C_{\mathrm{DMC}} = \log(2^n) + \sum_{e \in GF(2^n)} \Pr\{e_i = e\} \cdot \log(\Pr\{e_i = e\})$, where the maximum capacity is reached when $\Pr\{e_i = e\} = 1/2^n$ for all $e \in \mathrm{GF}(2^n)$. Then the above decoding problem is converted into decoding a $[N, l]$ linear code $\mathcal{C}_1$ over $\mathrm{GF}(2^n)$, where N is the code length and l is the symbol-length of information, with the code rate $R = \log(2^n) \cdot l/N$. Using Taylor series at order two, we achieve the following theorem, which theoretically connects the capacity of the DMC with the SEI of the noise distribution.

Theorem 5 *Let C_{DMC} be the capacity of a DMC over $GF(2^n)$ and the noise variable $e_i \in GF(2^n)$, whose distribution is denoted by $p_{e_i} = (Pr\{e_i = 0\}, \cdots, Pr\{e_i = 2^n - 1\})$. Then the theoretical relation between the capacity C_{DMC} and the SEI of p_{e_i}, i.e., $\Delta(p_{e_i})$, is $C_{\mathrm{DMC}} \approx \frac{\Delta(p_{e_i})}{2\ln(2)}$.*

This theorem provides a tool for bridging the theory based on Shannon theory and that based on the SEI measure. Theorem 5 is the basis of our framework, enabling us to derive a lower bound on the keystream length required for a successful attack. Actually, a $[N, l]$ linear code over $\mathrm{GF}(2^n)$ can be successfully decoded only if its code rate does not exceed the capacity of the transmission channel, pioneered in [23].

Theorem 5 and Shannon Theorem are combined together in our framework to give a theoretical analysis of the new fast correlation attacks over extension fields. Under this theoretical framework, we can assure that the fast correlation attack succeeds with a high probability, i.e., $0.5 < P_{succ} \leq 1$, if $R < C_{\mathrm{DMC}}$.

3.2 General Description of Fast Correlation Attacks over Extension Fields

Our new algorithm is extracted from the previous work in [10, 12] by addressing some important unsolved problems therein. First, the pre-computation algorithm in [10, 12] uses the straight forward method to find all the possible collisions over extension fields, whose complexity is too high to be applied in cryptanalysis. Second, in Fig.1, only a DMC with the following properties is considered, i.e., the distribution of the noise variable e_i satisfies $\Pr\{e_i = 0\} = 1/2^n + \delta$ and $\Pr\{e_i = e\} = 1/2^n - \delta/(2^n - 1), \forall e \in \mathrm{GF}(2^n), e \neq 0$, which is *not* the general case. Usually in the practice of correlation attacks, the distribution of noisy variable does not necessarily satisfy this condition. Third, the straightforward method is used to identify the correct key in the online phase, i.e., by evaluating parity-checks one by one for each possible codeword, which is inappropriate for cryptanalytic purposes. Last, a comprehensive theoretical justification is missing, which will assure the decoding reliability when simulations are infeasible.

Preprocessing. As in [4, 10, 12], we convert the original code $\mathcal{C}_1$ directly derived from the primitive to a new code $\mathcal{C}_2$, which is expected to be easier to decode by some fast decoding algorithm later devised. Precisely, let the length of the LFSR be l-word. Then we have $\mathbf{u} = (u_1, u_2, \cdots, u_l) \cdot G$, where $(u_1, u_2, \cdots, u_l)$ is the initial state of the LFSR. Let $(\cdot, \cdots, \cdot)^T$ be the transpose of a vector,

we rewrite the matrix G in column vectors as $G = (\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_N)$, where $\mathbf{g}_i = (g_i^1, g_i^2, \cdots, g_i^l)^T$ $(1 \le i \le N)$ is the i-th column vector. In order to reduce the decoding complexity, we build a new code $\mathcal{C}_2$ with a smaller number of information symbols $\widehat{\mathbf{u}} = (u_1, u_2, \cdots, u_{l'})$ for a certain $l' < l$ as follows. We first look for some k-tuple column vectors $(\mathbf{g}_{i_1}, \mathbf{g}_{i_2}, \cdots, \mathbf{g}_{i_k})$ satisfying $\mathbf{g}_{i_1} \oplus \mathbf{g}_{i_2} \oplus \cdots \oplus \mathbf{g}_{i_k} = (c_1, c_2, \cdots, c_{l'}, 0, \cdots, 0)^T$. For each k-tuple, we have

$$\bigoplus_{j=1}^{k} u_{i_j} = (u_1, u_2, \cdots, u_l) \bigoplus_{j=1}^{k} \mathbf{g}_{i_j} = c_1 u_1 \oplus c_2 u_2 \oplus \cdots \oplus c_{l'} u_{l'}. \tag{1}$$

This equation is called the *parity check* for $u_1, \cdots, u_{l'}$. Since $z_i = u_i \oplus e_i$, we rewrite it as $\bigoplus_{j=1}^{k} z_{i_j} = c_1 u_1 \oplus c_2 u_2 \oplus \cdots \oplus c_{l'} u_{l'} \oplus \bigoplus_{j=1}^{k} e_{i_j}$. Collect a desirable number of such k-tuples and denote the number of such derived equations by m_k. Denote the indices of t-th such tuple of columns by $\{i_1^{(t)}, i_2^{(t)}, \cdots, i_k^{(t)}\}$. Let $U_t = \bigoplus_{j=1}^{k} u_{i_j^{(t)}}$, $1 \le t \le m_k$. Thus we have constructed an $[m_k, l']$-code $\mathcal{C}_2$, i.e., $\mathbf{U} = (U_1, U_2, \cdots, U_{m_k})$.

Processing. Denote the received sequence by $\mathbf{Z} = (Z_1, Z_2, \cdots, Z_{m_k})$, where $Z_t = \bigoplus_{j=1}^{k} z_{i_j^{(t)}}$. We first use the keystream words $z_1, z_2, \cdots, z_N$ to compute $\mathbf{Z}$. Then decode the code $\mathcal{C}_2$ using the algorithm in the following subsection and output $(u_1, u_2, \cdots, u_{l'})$. Using the DMC model and assuming that all the e_is are independent random values over GF(2^n), it is easy to see that the distribution of the folded noisy variable $E_t = \bigoplus_{j=1}^{k} e_{i_j^{(t)}}$ can be computed by the convolution property via FWT. The new noise sequence can be represented as $\mathbf{E} = (E_1, E_2, \cdots, E_{m_k})$.

3.3 Preprocessing Stage: Generating the Parity Checks

Now we present an algorithm to compute the desirable k-tuple parity checks with a relatively low complexity, while the straight forward method in [12] needs a complexity of $O(N^k)$. First look at the case of $k = 2$. Eq.(1) indicates that $(g_{i_1}^{l'+1}, g_{i_1}^{l'+2}, \cdots, g_{i_1}^{l})^T \oplus (g_{i_2}^{l'+1}, g_{i_2}^{l'+2}, \cdots, g_{i_2}^{l})^T = (0, \cdots, 0)^T$. Thus the construction of parity checks is equivalent to the searching of $n(l - l')$-bit collision, i.e., just split $(g_i^{l'+1}, g_i^{l'+2}, \cdots, g_i^{l})$ for $i = 1, \cdots, N$ into two lists L_1 and L_2, and look for $x_1 \in L_1, x_2 \in L_2$ such that $x_1 \oplus x_2 = 0$. Hence, by searching for collisions through these two lists, 2-tuple parity checks in our attack can be constructed.

Note that the crucial difference between GF(2^n) and GF(2) requires that the length of the partial collision positions cannot be arbitrary and should be a multiple of n. In general, we can split the truncated matrix columns of G into k lists and search for $x_i \in L_i$ for $1 \le i \le k$ such that $\bigoplus_{i=1}^{k} x_i = 0$ holds for $1 \le i \le k$. This problem can be transformed into the well known k-sum problem.

Problem 1. (The k-sum problem) Given k lists $L_1, \cdots, L_k$, each of length α and containing elements drawn uniformly and independently at random from $\{0,1\}^{n(l-l')}$, find $x_1 \in L_1, \cdots, x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$.

Fortunately, this problem can be efficiently solved by the k-tree algorithm in [24]. It is shown that the k-tree algorithm requires $O(k2^{n(l-l')/(1+\log k)})$ time and space and uses lists of size $O(2^{n(l-l')/(1+\log k)})$. The k-tree algorithm can also find many solutions to the k-sum problem. It can find $\beta^{1+\log k}$ solutions to the k-sum problem with β times as much work as finding a single solution, as long as $\beta \leq 2^{n(l-l')/(\log k(1+\log k))}$. Thus the total time/space complexities are $O(\beta k2^{n(l-l')/(1+\log k)})$ and the size of each list is $O(\beta 2^{n(l-l')/(1+\log k)})$.

Now we show how to generate the m_k k-tuple parity checks. Precisely, we denote the truncated partial vector of $\mathbf{g}_i$ by $\mathbf{x}_i = (g_i^{l'+1}, \cdots, g_i^l)$ for $i = 1, \cdots, N$. Then disjoin $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N)$ into k lists $L_1, \cdots, L_k$, each of length $\alpha = N/k$. We want to find $\mathbf{x}_1 \in L_1, \cdots, \mathbf{x}_k \in L_k$ satisfying $\mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \cdots \oplus \mathbf{x}_k = 0$. This is exactly the same case as the k-sum problem, so we can adopt the k-tree algorithm in [24] to find the required number of desirable parity checks.

3.4 Processing Stage: Decoding the Code $\mathcal{C}_2$

It is well-known that decoding a random linear code over an extension field is a NP-hard problem. Here we present a fast decoding algorithm, which can be seen as a solution to this problem.

As shown in [5, 14], FWT can be used to accelerate the decoding process for the linear codes over GF(2). Here we derive a method based on Lemma 4 to exploit FWT for decoding linear codes over GF(2^n).

Let us denote the guessed value of the partial initial state $\widehat{\mathbf{u}} = (u_1, \cdots, u_{l'})$ by $\widehat{\mathbf{u}}' = (u'_1, \cdots, u'_{l'})$. After pre-computation, we construct a distinguisher $I(\widehat{\mathbf{u}}') = c_1^{(t)}(u_1 \oplus u'_1) \oplus \cdots \oplus c_{l'}^{(t)}(u_{l'} \oplus u'_{l'}) \oplus E_t = Z_t \oplus c_1^{(t)}u'_1 \oplus \cdots \oplus c_{l'}^{(t)}u'_{l'}$, to find the correct partial state $\widehat{\mathbf{u}}$. If the guessed value $\widehat{\mathbf{u}}'$ is correct, I is expected to be biased; otherwise it approximates an uniform distribution.

Next, let us give a description on how to compute the SEI of $I(\widehat{\mathbf{u}}')$, which is the crucial part of our algorithm. We need to substitute the z_is into the parity check equations and evaluate the SEI of I for each possible $\widehat{\mathbf{u}}'$. Combining Lemma 4 in Section 2.2 with FWT, we have the following method. Precisely, the SEI of $I(\widehat{\mathbf{u}}')$ can be computed by the correlations $c(\langle \gamma, I \rangle)$, where $\langle \gamma, I \rangle$ is a boolean function and $\gamma \in \mathrm{GF}(2)^n$. We can divide the vectorial boolean function I into n linearly independent boolean functions $I_1, \cdots, I_n$ and each boolean function can be expressed as $I_i = \langle w_i, \widehat{\mathbf{u}}' \rangle \oplus \langle v_i, Z_t \rangle$, where $w_i \in \mathrm{GF}(2)^{nl'}, v_i \in \mathrm{GF}(2)^n$ are two binary coefficient vectors. Let $Q = \mathrm{span}\{I_1, \cdots, I_n\}$ such that Q is a set of approximations generated by these n approximations I_i. Now the advantage is that FWT can be used to compute the correlation of each approximation I_i for $i = 1, \cdots, n$, as described in [14].

Preciously, assume that we have m_k n-bit parity checks over GF(2^n) with the same distribution. Then for each I_i there are m_k bitwise parity checks denoted by $I_i^{(t)}$ for $1 \leq t \leq m_k$. In order to evaluate these m_k bitwise parity checks $I_i^{(t)} = \langle w_i^{(t)}, \widehat{\mathbf{u}}' \rangle \oplus \langle v_i^{(t)}, Z_t \rangle$ for each $\widehat{\mathbf{u}}'$, we introduce an integer-valued function,

$$h(\widehat{\mathbf{u}}') = \sum_{1 \leq t \leq m_k : \widehat{\mathbf{u}}' = w_i^{(t)}} (-1)^{\langle v_i^{(t)}, Z_t \rangle},$$

for all $\widehat{\mathbf{u}}' \in \mathrm{GF}(2^{nl'})$. We compute the Walsh transform of h and then we can get an $2^{nl'}$-dimensional array storing the correlation $c(I_i)$ indexed by $\widehat{\mathbf{u}}'$. The total time complexity for computing $c(I_1), \cdots, c(I_n)$ is $O(n(m_k + l'n2^{l'n}))$ and memory complexity is $O(n2^{l'n})$. In addition, the correlations of the other $2^n - n - 1$ linear approximations can be computed by the Piling-up Lemma [16]. Thus, we have got all the correlations for different guessed values of $\widehat{\mathbf{u}}$. Again from Lemma 4, we can easily compute $\Delta(I(\widehat{\mathbf{u}}'))$ for each possible $\widehat{\mathbf{u}}'$. Then, we can use a distinguisher described in [1] to recover the correct initial state. In total, the time complexity of decoding $\mathcal{C}_2$ in such a way is $O(n(m_k + l'n2^{l'n}) + 2^n 2^{l'n})$.

Now we give the theoretical justifications of our algorithm. Assume the noisy distribution of E_t over $\mathrm{GF}(2^n)$ is $p_{E_t} = (\Pr\{E_t = 0\}, \cdots, \Pr\{E_t = 2^n - 1\})$ and the code length of $\mathcal{C}_2$ is m_k. According to the k-tree algorithm, using k lists, each of which has size of $\alpha = \beta 2^{n(l-l')/(1+\log k)}$, we can find $\beta^{1+\log k}$ parity checks.

Since the number of parity checks pre-computed is m_k, thus we have $m_k = \beta^{1+\log k}$. Further, for the decoding to succeed, the code rate $R = l' \cdot \log(2^n)/m_k$ of $\mathcal{C}_2$ must satisfy $R < C_{\mathrm{DMC}}$. Then by Theorem 5, the value of m_k can be calculated as $m_k \approx (2l'n \ln 2)/\Delta(p_{E_i})$. The following theorem gives the required length N of the observed keystream segment for successfully decoding code $\mathcal{C}_1$.

Theorem 6 *Given a $[N, l]$ linear code $\mathcal{C}_1$ over GF(2^n). After applying the pre-computation of our algorithm, we get a new $[m_k, l']$ linear code $\mathcal{C}_2$, which is transmitted through a 2^n-ary DMC with the noise distribution p_{E_i}. The required length N of the observed keystream segment for the algorithm to succeed is $N \approx k 2^{\frac{n(l-l')}{\theta}} (2l'n \ln 2)^{\frac{1}{\theta}} \Delta(p_{E_i})^{-\frac{1}{\theta}}$, where $\theta = 1 + \log k$.*

4 Large-unit Linear Approximation and Its Distribution

In this section, an efficient algorithm to accurately compute the large-unit distribution of the GPLFM is proposed. This is desired when the decoding algorithm is available.

4.1 Large-unit Linear Approximations

Most of the previous work only study how to use the bitwise linear approximations to constitute a vector, here we directly focus on the non-binary linear approximations whose basic data unit is over $\mathrm{GF}(2^m)$ ($m > 1$) and such non-binary unit linear approximations are called the *large-unit linear approximations* throughout this paper[3]. Let $H(X_1, X_2, \cdots, X_d)$ be a non-linear function, where the output and the input X_is are all random variables over $\mathrm{GF}(2^n)$. Our task is to accurately compute the m-bit large-unit distribution of some linear approximation of H. In practice, the choice of m cannot be arbitrary and is usually determined by the structure of the primitive and the underlying building blocks, e.g., the LFSR structure and the S-box size. When m is fixed, the output of H

[3] As we can see, when $m = 1$ it is just the bitwise approximation of F, while when $m = n$ it becomes the n-bit linear approximation, discussed in [7, 17].

and each input $X_i (1 \le i \le d)$ can all be regarded as some $\frac{n}{m}$-dimensional vectors over GF(2^m). In this setting, the definition of a binary linear mask is as follows.

Definition 7 *Let $X \in GF(2^n)$ and $\Omega = (\omega_{\frac{n}{m}}, \cdots, \omega_2, \omega_1)$ be a $\frac{n}{m}$-dimensional binary vector, then X can be transformed to a $\frac{n}{m}$-dimensional vector $X = (x_{\frac{n}{m}}, \cdots, x_2, x_1)$ over $GF(2^m)$ with $x_i \in GF(2^m)$ for $1 \le i \le \frac{n}{m}$. The inner product between these two vectors is defined as $\Omega \cdot X = \omega_{\frac{n}{m}} x_{\frac{n}{m}} \oplus \cdots \oplus \omega_1 x_1 \in GF(2^m)$, where Ω is called the $\frac{n}{m}$-dimensional binary linear mask of X over $GF(2^m)$.*

4.2 The Generalized Pseudo-Linear Function Modulo 2^n

Now we first generalize the pseudo-linear function modulo 2^n (PLFM) in [17] to GPLFM by introducing the binary mask with the inner product in Definition 7. Note that in [17], the distribution of some class of functions called PLFM over GF(2^n) is computed, here we consider similar problems of GPLFM in a smaller field GF(2^m) with $m < n$.

Assume the large-unit is of m-bit size. Let $\mathcal{X} = \{X_1, X_2, \cdots, X_d\}$ be a set of d uniformly distributed n-bit random variables with $X_i \in$ GF(2^n) for $1 \le i \le d$, $\mathcal{C} = \{C_1, \cdots, C_g\}$ be a set of n-bit constants and $\mathcal{M}$ be a set of $\frac{n}{m}$-dimensional binary masks of $\mathcal{X}$ and $\mathcal{C}$. Now each element in $\mathcal{X}$ and $\mathcal{C}$ can be regarded as a $\frac{n}{m}$-dimensional vector over GF(2^m). We denote some symbol or expression on $\mathcal{X}$ and $\mathcal{C}$ by T_i. The following two definitions introduce the GPLFM, which generalizes the definition of PLFM in [17].

Definition 8 *Given three sets $\mathcal{X}$, $\mathcal{C}$ and $\mathcal{M}$, we have:*

1. *$\mathcal{A}$ is an arithmetic term[4], if it has only the operation of arithmetic $\boxplus$, e.g., $\mathcal{A} = T_1 \boxplus T_2 \boxplus \cdots$.*
2. *$\mathcal{B}$ is a Boolean term, if it only involves Boolean operations such as OR, AND, XOR, and others, e.g., $\mathcal{B} = (T_1 \oplus T_2)$ & T_3.*
3. *$\mathcal{S}$ is a simple term, if it is a symbol either from $\mathcal{X}$ or $\mathcal{C}$.*
4. *$\Omega \cdot X$ for $X \in \{\mathcal{A}, \mathcal{B}, \mathcal{S}\}$ is the inner product result of the term X with the binary mask $\Omega \in \mathcal{M}$.*

Definition 9 *$F(X_1, X_2, \cdots, X_d)$ is called a generalized pseudo-linear function modulo 2^n (GPLFM) on $\mathcal{X}$, if it can recursively be expressed in $\Omega \cdot X$ for $X \in \{\mathcal{A}, \mathcal{B}, \mathcal{S}\}$ combined by the Boolean operations.*

It can be easily seen that the PLFM studied in [17] forms a subset of the GPLFM, which only satisfies the conditions $1 \sim 3$ in Definition 8. In our large-unit linear approximation of SNOW 2.0 in Section 5 and 6, we actually further generalize the GPLFM functions by considering *parallel* boolean functions, i.e., the S-boxes and multiplication over finite fields are included in our framework.

[4] An arithmetic subtraction $\boxminus$ can be substituted by $\boxplus$ using $X \boxminus Y = X \boxplus (\bar{Y}) \boxplus 1 \bmod 2^n$, where $\bar{\cdot}$ is the complement operation.

4.3 Algorithm for Computing the Distribution of a GPLFM

Assume the basic large-unit is of m-bit size. Let $F(X_1, \cdots, X_d)$ be a GPLFM with $\mathcal{X}$, $\mathcal{C}$ and $\Omega \in \mathcal{M}$, where $X_i \in \mathrm{GF}(2^n)$ $(1 \le i \le d)$ and the binary masks are $\frac{n}{m}$-dimensional vectors. We want to calculate the distribution of F in an efficient way for some large n. Note that if $n \ge 32$ and $d \ge 2$, the distribution p_F is impossible to implement in practice with the straight forward method, which needs 2^{nd} operations. Further, the algorithm in [17] cannot be applied to this problem due to the inner product operation inherent in the GPLFM over a smaller field $\mathrm{GF}(2^m)$. Here we propose Algorithm 1 to fulfill this task.

Our basic idea is as follows. Since each coordinate of the binary mask can only take the value of 0 or 1, it actually selects which parts of the data arguments will take effect in the approximation. According to the binary mask Ω, we can split each variable $X_i \in \mathrm{GF}(2^n)$ for $i = 1, \cdots, d$ into $\frac{n}{m}$ blocks and each block has m bits, i.e., $X_i = (X_i^{\frac{n}{m}}, \cdots, X_i^2, X_i^1)$, where $X_i^j \in \mathrm{GF}(2^m)$ for $1 \le j \le \frac{n}{m}$. Since each block of the input variable is mutually independent, the function F can be split into $\frac{n}{m}$ sub-functions F_i $(1 \le i \le \frac{n}{m})$, which can be evaluated over a smaller space $\mathrm{GF}(2^m)$. Each sub-function F_i can be seen as a PLFM over $\mathrm{GF}(2^m)$, whose distribution can be efficiently calculated by the algorithm in [17].

Algorithm 1 Computing the m-dimensional distribution of a GPLFM over $\mathrm{GF}(2^n)$

Parameters:

M_1 and M_2: two consecutive connection matrices of size $2^m \times |Cr_{\max}|$;

Processing:

1: Split the function F into $\frac{n}{m}$ sub-functions F_i according to the binary masks.
2: $M_1 \leftarrow$ **ComputePLFM**$(F_1(0, 0, X_1^1, \cdots, X_d^1), M_1, 1)$, shown in Appendix A
3: **for** $i = 2, \cdots, \frac{n}{m}$ **do**
4: Initialize M_2 with zeros.
5: **for** $Cr_{i-1} = (0, 0, \cdots, 0)$ **to** $(d_1^+, d_2^+, \cdots, d_s^+)$ **do**
6: **for** $B_{i-1} = 0$ **to** $2^m - 1$ **do**
7: $M_2 \leftarrow$ **ComputePLFM**$(F_i(B_{i-1}, Cr_{i-1}, X_1^i, \cdots, X_d^i), M_2, M_1[B_{i-1}][|Cr_{i-1}|])$;
8: **end for**
9: **end for**
10: $M_1 \leftarrow M_2/(2^m \cdot |Cr_{\max}|)$;
11: **end for**

Output: $p_F(i) = M_1[i][0] + M_1[i][1] + \cdots + M_1[i][|Cr_{\max}| - 1]$

On the other hand, the sub-function F_is are connected with each other by the one direction information propagation from the least significant function F_1 to the most significant $F_{\frac{n}{m}}$, caused by the carry bit introduced by $\boxplus$ and the output of F_i, shown in Fig.2. Therefore, we can use a connection matrix to characterize this propagation process.

Now, we compute the distribution p_F by calculating the F_is one-by-one from 1 to $\frac{n}{m}$, as depicted in Fig.2. Here $B_{i-1} \in \mathrm{GF}(2^m)$ is the output of sub-function F_{i-1} and Cr_{i-1} is the carry *vector* of F_{i-1} that will be propagated to F_i, generated by the arithmetic terms in F_{i-1}. If there are s arithmetic terms $\mathcal{A}_j$ $(1 \le j \le s)$

in F (thus in each F_i), then we have $Cr_i = (cr_i^1, \cdots, cr_i^s)$, where each cr_i^j is the corresponding local carry value of the A_j $(1 \leq j \leq s)$ when the inputs are truncated to the ith block. Note that though each block is m-bit size, the modular addition is still calculated bit-by-bit, thus the maximum local carry value is d_j^+, where d_j^+ is the number of modular additions in $\mathcal{A}_j$ $(1 \leq j \leq s)$. Emphatically, Cr_i contains all the carry information of F_j for $j < i$, since the carry information is propagated from F_1 to F_i. It is proved in [17] that for

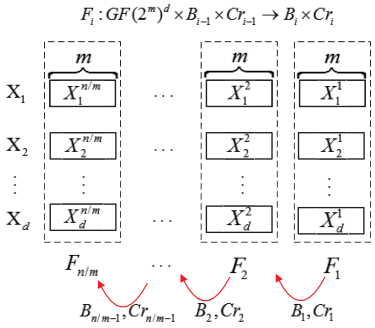

Fig. 2. The basic idea of our algorithm

any arithmetic term $\mathcal{A}_j$, the maximum local carry value is d_j^+ (the additions of carry value are not included). Similarly, denote the cardinality of Cr_i by $|Cr_i| = ((cr_i^1 \cdot (d_2^+ + 1) + cr_i^2)(d_3^+ + 1) + cr_i^3) \cdot \ldots$, which is a one-to-one index mapping function from $(cr_i^1, \cdots, cr_i^s)$ to $[0, \ldots, |Cr_{\max}| - 1]$, where $|Cr_{\max}| = \prod_{j=1}^{s}(d_j^+ + 1)$ is the maximal possible cardinality of the carry vector Cr_i. We use a $2^m \times |Cr_{\max}|$ matrix M_i to store the information of the F_js $(j \leq i)$, where the matrix element $M_i[B_i][|Cr_i|]$ for $0 \leq B_i \leq 2^m - 1$ and $0 \leq |Cr_i| \leq |Cr_{\max}| - 1$ represents the total number of the inputs $(X_1^i, X_2^i, \cdots, X_d^i)$ of F_i that result in the F_i output B_i and the carry vector Cr_i. Thus, the evaluation of F_i is converted into the computation of the matrix M_i. M_i stores all the output and carry information of F_i. Here we call it the **connection matrix**.

Now we need to evaluate the function F_i based on the connection matrix M_{i-1}, to obtain the next matrix M_i. It depends on the carry vector Cr_{i-1} and the output value B_{i-1} of F_{i-1}. For $m > 1$, since the sub-function F_i can be seen as a PLFM over GF(2^m), which is recursively expressed in $\mathcal{A}, \mathcal{B}, \mathcal{S}$, we can use a sub-algorithm called **ComputePLFM** (Appendix A) to compute the matrix M_i (M_2 in Algorithm 1) for all the possible values of B_{i-1} and Cr_{i-1}. Hereafter, when applying the Algorithm 1 we always assume that $m > 1$. The initial values are $Cr_0 = (0, 0, \cdots, 0)$ and $B_0 = 0$, i.e., the initial matrix M_0 is set to be zero matrix. Our algorithm to compute the full m-dimensional distribution $p_F = (p_F(0), p_F(1), \cdots, p_F(2^m - 1))$ of a GPLFM F over GF(2^n) is shown in the Algorithm 1 diagram. Note that in Algorithm 1, only two connection matrices M_1 and M_2 are used to store the propagation information alternatively. The complexity analysis of Algorithm 1 is as follows. First look at the complexity of Algorithm 2 in Appendix A. Step 1 in Algorithm 2 needs a time complexity of

$O(m \cdot |Cr_{\max}| \cdot 2^d)$ from [17]. Step 2 to step 8 needs a complexity of $O(2^m \cdot m \cdot |Cr_{\max}|)$. Thus the complexity of Algorithm 2 is $O(m \cdot |Cr_{\max}| \cdot (2^d + 2^m))$ and the total time complexity of our algorithm is $O(n \cdot 2^m \cdot |Cr_{\max}|^2 \cdot (2^d + 2^m))$.

5 A Key Recovery Attack on SNOW 2.0

In this section, we demonstrate a state recovery attack against SNOW 2.0. The description of SNOW 2.0 is detailed in [6]. Our new attack is based on the byte-wise linear approximation and utilizes the fast correlation attack over GF(2^8) to recover the correct initial state with much lower complexities.

5.1 The Byte-wise Linear Approximation of SNOW 2.0

In SNOW 2.0, denote the AES S-box and the Mixcolumn matrix in the S transform of FSM by S_R and M respectively. Since S_R is a 8-bit S-box, we let $n = 32$ and $m = 8$. As SNOW 2.0 has two 32-bit memory registers $R1$ and $R2$ in the FSM, it is necessary to consider at least two consecutive steps of the FSM to eliminate these two registers in the approximation. Here we denote the two binary masks by $\Gamma, \Lambda \in \mathrm{GF}(2)^4$ respectively, thus the 32-bit word can be divided into 4 bytes and be regarded as a 4-dimensional vector over GF(2^8). For example,

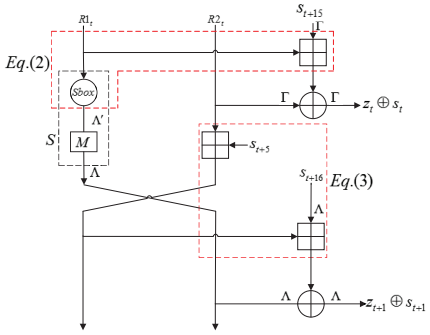

Fig. 3. The linear approximation of the FSM in SNOW 2.0

let the binary mask $\Gamma = (1, 0, 1, 0)$ and $X = (x_4, x_3, x_2, x_1)$ be a 32-bit word of SNOW 2.0 in byte-wise form, thus $\Gamma \cdot X = x_4 \oplus x_2$. Applying Γ and Λ to z_t and z_{t+1} respectively, we have

$$\Gamma \cdot z_t = \Gamma \cdot s_t \oplus \Gamma \cdot (R1_t \boxplus s_{t+15}) \oplus \Gamma \cdot R2_t,$$
$$\Lambda \cdot z_{t+1} = \Lambda \cdot s_{t+1} \oplus \Lambda \cdot (s_{t+16} \boxplus R2_t \boxplus s_{t+5}) \oplus \Lambda \cdot S(R1_t).$$

Let $y_t = \mathrm{Sbox}(R1_t) = (S_R(R1_t^4), S_R(R1_t^3), S_R(R1_t^2), S_R(R1_t^1))$ be the output of S-box. Since the Mixcolumn matrix M is a linear transformation over GF(2^8), we have $\Lambda \cdot S(R1_t) = \Lambda \cdot (My_t) = \Lambda' \cdot y_t$. We can rewrite the above two equations

as

$$\Gamma \cdot z_t = \Gamma \cdot s_t \oplus \Gamma \cdot (\mathrm{Sbox}^{-1}(y_t) \boxplus s_{t+15}) \oplus \Gamma \cdot R2_t,$$
$$\Lambda \cdot z_{t+1} = \Lambda \cdot s_{t+1} \oplus \Lambda \cdot (s_{t+16} \boxplus R2_t \boxplus s_{t+5}) \oplus \Lambda' \cdot y_t.$$

Now we have a new byte-wise linear approximation of SNOW 2.0, depicted in Fig.3. Note that in Fig.3, the S transform of the FSM is dissected to have an efficient approximation. Here we use two linear approximations

$$\Gamma \cdot (\mathrm{Sbox}^{-1}(y_t) \boxplus s_{t+15}) = \Gamma \cdot s_{t+15} \oplus \Lambda' \cdot y_t \oplus \mathbf{N}_1(t), \quad (2)$$
$$\Lambda \cdot (s_{t+16} \boxplus s_{t+5} \boxplus R2_t) = \Lambda \cdot s_{t+16} \oplus \Lambda \cdot s_{t+5} \oplus \Lambda \cdot R2_t \oplus \mathbf{N}_2(t). \quad (3)$$

The linear approximation (3) is a GPLFM, thus we can adopt Algorithm 1 to compute the distribution of $\mathbf{N}_2(t)$. For the linear approximation (2), it is not a GPLFM in Definition 8 and 9, thus we cannot use Algorithm 1 directly. But note that the four S_Rs do not affect the independency among the bytes of y_t, thus we can revise Algorithm 1 to compute the distribution of $\mathbf{N}_1(t)$ as shown in Algorithm 3.

Algorithm 3 Computing the Distribution in Eq.(2) over GF(2^8)

Parameters:
$\Gamma = (\Gamma_4, \Gamma_3, \Gamma_2, \Gamma_1)$, $s_t = (s_t^4, s_t^3, s_t^2, s_t^1)$,$y_t = (y_t^4, y_t^3, y_t^2, y_t^1)$, $\Lambda' = (\Lambda'_4, \Lambda'_3, \Lambda'_2, \Lambda'_1)$;
Processing:
1: Compute $\Gamma_1(S_R^{-1}(y_t^1) + s_{t+15}^1) \oplus \Gamma_1 s_{t+15}^1 \oplus \Lambda'_1 y_t^1$ and store in M_1
2: **for** $i = 2, \cdots, 4$ **do**
3: Initialize M_2 with zeros.
4: **for** $y_t^i = 0, \cdots, 255$ and $s_{t+15}^i = 0, \cdots, 255$ **do**
5: **for** $Cr_{i-1} = 0, 1$ **do**
6: **for** $B_{i-1} = 0, \cdots, 255$ **do**
7: $B_i \leftarrow B_{i-1} \oplus \Gamma_i(S_R^{-1}(y_t^i) + s_{t+15}^i) \oplus \Gamma_i s_{t+15}^i \oplus \Lambda'_i y_t^i$;
8: $Cr_i \leftarrow (S_R(y_t^i) + s_{t+15}^i + Cr_{i-1})/2^8$;
9: $M_2[B_i][|Cr_i|] \leftarrow M_2[B_i][|Cr_i|] + M_1[B_{i-1}][|Cr_{i-1}|]$;
10: $M_1 \leftarrow M_2/(2^8 \times 2)$;
11: **Output**: The distribution $p_i = M_1[i][0] + M_2[i][1]$ for each $0 \le i \le 255$.

The time complexity of computing the distribution of $\mathbf{N}_1(t)$ has dropped from 2^{64} to $2^{26.58}$, which is a large improvement compared to the straightforward method. We have searched over all the different binary masks over GF(2)^4 and found that when $\Gamma = \Lambda$, these two linear approximations will have larger SEIs. Thus the sum of $\Gamma \cdot (z_t \oplus z_{t+1})$ can be expressed as

$$\Gamma \cdot (z_t \oplus z_{t+1}) = \Gamma \cdot s_t \oplus \Gamma \cdot s_{t+1} \oplus \Gamma \cdot s_{t+5} \oplus \Gamma \cdot s_{t+15} \oplus \Gamma \cdot s_{t+16} \oplus \mathbf{N}(t), \quad (4)$$

where $\mathbf{N}(t) = \mathbf{N}_1(t) \oplus \mathbf{N}_2(t)$ is the folded noise variable introduced by the above two linear approximations, whose distribution can be computed by the convolution of the above two noise distributions. We have searched all the possible Γ and Λ and found that the *strongest* linear approximation of the FSM is as

follows. When $\Gamma = \Lambda = (1, 0, 1, 0)$, the distribution of $\mathbf{N}(t)$ has the value of SEI as $\Delta(\mathbf{N}(t)) = 2^{-43.23}$. Observe that given a noise distribution $\Pr\{\mathbf{N}(t)\}$, the SEI can be *precisely* computed by Definition 2. Now, we have constructed the byte-wise linear approximation, i.e., Eq.(4), of SNOW 2.0. Next, we will use this linear approximation to recover the initial state of SNOW 2.0.

5.2 Fast Correlation Attack on SNOW 2.0

Now we apply the fast correlation attack over $\mathrm{GF}(2^8)$ to SNOW 2.0 to recover the initial state of the LFSR. Let the LFSR state be $(s_{t+15}, \cdots, s_t) \in \mathrm{GF}(2^{32})^{16}$, here the LFSR is interpreted as a 64-byte LFSR over $\mathrm{GF}(2^8)$, i.e.,$(s^4_{t+15}, s^3_{t+15}, s^2_{t+15}, s^1_{t+15}, \cdots, s^4_t, s^3_t, s^2_t, s^1_t) \in \mathrm{GF}(2^8)^{64}$. With the feedback polynomial we can express the linear approximation (4) in the initial state form as $\Gamma \cdot (z_t \oplus z_{t+1}) = \Psi_t \cdot (s^4_{t+15}, s^3_{t+15}, s^2_{t+15}, s^1_{t+15}, \cdots, s^4_t, s^3_t, s^2_t, s^1_t) \oplus N(t)$, where $\Psi_t \in \mathrm{GF}(2^8)^{64}$ is the derived recursion of the LFSR.

For the decoding algorithm, we apply the precomputation algorithm in Section 3 to generate the parity checks with the parameters $l = 64, l' = 17, k = 4$, which are the best parameters we have found in terms of the total complexities. The distribution of the folded noise variables $\mathbf{N}(t_{i_1}) \oplus \mathbf{N}(t_{i_2}) \oplus \mathbf{N}(t_{i_3}) \oplus \mathbf{N}(t_{i_4})$ can be computed by the applications of the convolutional operation twice. The SEI of this new distribution is found to be $2^{-177.3}$. Using 4 lists in the k-tree algorithm, we get about $m_k = \beta^{1+\log k} = 2^{184.86}$ parity check equations. By Theorem 6, the data complexity is $N = 2^{188.95}$ and the time/memory complexities of preprocessing stage are $\beta k 2^{n(l-l')/(1+\log k)} = 2^{188.95}$. Second, we perform the online decoding algorithm on the new code $\mathcal{C}_2$ of the code length $2^{188.95}$. With a computational complexity of $n(m_k + l' n 2^{l'n}) + 2^n 2^{l'n} = 2^{187.86}$, we can recover the $17 \cdot 8 = 136$ bits of the initial state of LFSR, the other bits can be recovered with a much lower complexity.

Therefore, the final time/memory/data/pre-computation complexities are all upper bounded by $2^{186.95}$, which is more than 2^{25} times better than the best previous result at Asiacrypt 2008 [13]. This obviously confirms the superiority of our new techniques.

6 An Improved Key Recovery Attack on SNOW 2.0

Recall in Section 4, we use the $\frac{n}{m}$-dimensional binary linear masks. Here we generalize this definition by making each component $\omega_i \in \mathrm{GF}(2^m)$ rather than $\mathrm{GF}(2)$, i.e., changing the 0/1 coefficients to finite field coefficients, i.e., expressing X by $X = (x_{\frac{n}{m}}, \cdots, x_1) \in \mathrm{GF}(2^m)^{\frac{n}{m}}$ with $x_i \in \mathrm{GF}(2^m)$ and denote the inner product by $\Omega \cdot X = \omega_{\frac{n}{m}} x_{\frac{n}{m}} \oplus \cdots \oplus \omega_1 x_1 \in \mathrm{GF}(2^m)$, where $\omega_i x_i$ is the multiplication over $\mathrm{GF}(2^m)$. Ω is called the linear mask over $\mathrm{GF}(2^m)$ of X. Now these new nonlinear functions are not GPLFM in Definition 8 and 9, for we have changed the linear mask from $\mathrm{GF}(2)$ to $\mathrm{GF}(2^m)$. Thus we cannot apply the Algorithm 1 to compute the distributions of these new functions directly. Instead, we further revise Algorithm 1 to efficiently compute the distributions of such functions in the following analysis of SNOW 2.0.

6.1 Linear Approximations of SNOW 2.0 over GF(2^8)

The process of finding the linear approximations of SNOW 2.0 is nearly the same as in Section 5.In order to find the best linear masks over GF(2^8), let us take a closer look at the details of the S permutation in FSM. Let $\Lambda' = (\Lambda'_4, \Lambda'_3, \Lambda'_2, \Lambda'_1)$ denote the linear mask over GF(2^8) of the 4 byte outputs of the Sbox, where the multiplication is computed in GF(2^8) defined by the AES Mixcolumn. Then, we can express $\Lambda \cdot S(\omega)$ as

$$\underbrace{(\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4) \begin{pmatrix} 2\,3\,1\,1 \\ 1\,2\,3\,1 \\ 1\,1\,2\,3 \\ 3\,1\,1\,2 \end{pmatrix}} \begin{pmatrix} S_R(\omega_1) \\ S_R(\omega_2) \\ S_R(\omega_3) \\ S_R(\omega_4) \end{pmatrix} = (\Lambda'_1, \Lambda'_2, \Lambda'_3, \Lambda'_4) \begin{pmatrix} S_R(\omega_1) \\ S_R(\omega_2) \\ S_R(\omega_3) \\ S_R(\omega_4) \end{pmatrix},$$

where $\Lambda_i, \Lambda'_i \in$ GF(2^8) for $1 \le i \le 4$. We adopt the field GF(2^8) as that defined by the AES Mixcolumn and assume that the linear masks over GF(2^8) are also defined in this field. Here we still use the two linear approximations over GF(2^8), i.e., Eq.(2) and Eq.(3), but the linear masks Γ, Λ are 4-dimensional vectors over GF(2^8). The algorithm to compute the distribution of Eq.(2) is similar as before, except that $\Gamma = (\Gamma_4, \Gamma_3, \Gamma_2, \Gamma_1) \in$ GF(2^8)^4 rather than GF(2)^4, shown in Algorithm 3. The distribution of $\mathbf{N}_2(t)$ with the linear mask $\Lambda \in$ GF(2^8)^4 can be computed by Algorithm 4 in Appendix B. The time complexity is $3 \cdot (2^8)^3 \cdot 2^8 \approx 2^{33.58}$, while the straightforward method needs a complexity of 2^{96}.

Now we describe how to find the linear masks Λ, Γ that satisfy Eq.(2) and Eq.(3) with large SEIs. Our strategy is to limit the Hamming weights of the linear masks Λ and Λ' over GF(2^8). Denote the Hamming weight of a vector by $wt(\cdot)$, thus $wt(\Lambda')$ determines the number of active S-boxes in the S-box ensemble S. In the experiments, we found that the SEI of $\mathbf{N}_2(t)$ is dependent on $wt(\Lambda)$. The lower value of $wt(\Lambda)$, the larger $\Delta(\mathbf{N}_2(t))$. We have searched all the linear masks Λ, Λ' with $wt(\Lambda) \le 3$ and $wt(\Lambda') \le 3$ and found 255 different linear masks having the same largest value of $\Delta(\mathbf{N}(t))$. For example, when $\Gamma = \Lambda =$ (0x00, 0x01, 0x00, 0x03), we get the best linear approximation with the noise distribution $\mathbf{N}(t)$ having a SEI of $\Delta(\mathbf{N}(t)) = 2^{-29.23}$.

Please see Appendix C for unifying the two fields before decoding. Then we launch the fast correlation attack over GF(2^8) with the parameters $n = 32, l = 64, l' = 19, k = 4$. The data complexity is $N \approx 2^{163.59}$, while the time/memory complexities of the pre-computation is $2^{163.59}$. After pre-computation, we can acquire about $m_k = 2^{124.79}$ parity checks. For the online decoding algorithm, the time complexity is $2^{162.52}$ with the above parameter configuration. Note that here all the complexities are below $2^{164.15} \approx 2^{162.52} + 2^{163.59}$, which are considerably reduced compared to the binary mask case. The reason is that the linear masks with the finite field coefficients greatly extend the searching space and can well exploit the algebraic structure of the two finite fields (one defined in a tower manner in the LFSR and the other in the Mixcolumn) inherent in SNOW 2.0.

6.2 Experimental Results

We have verified each step of our new techniques in simulations to support the theoretical analysis. We have used the GNU Multiple Precision Arithmetic Library in Linux system to verify the exact distribution of each linear approximation that has been found, thus the SEI of our large-unit linear approximation is precisely evaluated *without* any precision error. Then we have run extensive experiments on a small-scale version of SNOW 2.0, called s-SNOW 2.0 described in Appendix D, that have verified our approach.

We have computed the 4-bit linear approximation of the s-SNOW 2.0 with Algorithm 1 in theory and verified it in practice. Then we executed the experiments on the decoding algorithm in Section 3.4. We randomly fixed the values of 60 bits of the initial state of the LFSR and tried to recover the remaining 20-bit by our method. The chosen parameters are $l' = 20, m_k = 2^{15.39}$. We first use s-SNOW 2.0 to generate 2^{17} keystream bits z_t for a randomly generated 80-bit initial state. Then we store z_t and s_t in two arrays for $t = 1, \cdots, 2^{17}$. Thus we can construct 2^{17} parity checks $I^{(t)} = \Gamma \cdot (z_t \oplus z_{t+1}) \oplus \Gamma \cdot s_t \oplus \Gamma \cdot s_{t+1} \oplus \Gamma \cdot s_{t+3} \oplus \Gamma \cdot s_{t+4} \oplus \Gamma \cdot s_t$, for $t = 0, \cdots, 2^{17} - 1$. Second, for each parity check I^t, we use the LFSR feedback polynomial to express each s_t for $t > 4$ as a linear combination of the LFSR initial state variables. Now we get 2^{17} parity checks only containing $(s_0, s_1, s_2, s_3, s_4)$ after fixing 60-bit in the state. Third, we divide the 4-bit linear approximation $I^{(t)}$ into four bitwise linear approximations, i.e., $I_1^{(t)} = \langle (0,0,0,1), I^t \rangle, I_2^{(t)} = \langle (0,0,1,0), I^{(t)} \rangle, I_3^{(t)} = \langle (0,1,0,0), I^{(t)} \rangle, I_4^{(t)} = \langle (1,0,0,0), I^{(t)} \rangle$. For each possible 20-bit initial state, we use FWT to compute the correlations $c(I_i^{(t)})$ for $i = 1, \cdots, 4$. Fourth, we apply Lemma 4 to compute the SEI of p_I for each possible initial state. Then we use the SEI to distinguish the correct initial state. We ran the experiments for randomly generated values 100 times with *different* fixed values at *different* positions in the LFSR state, and we found that the correct key always ranks in the top 10 in the candidates list. These 10 candidates have $\Delta(p_I)$ around $2^{-10.6}$, which verified the theoretical analysis.

Therefore, the experimental results have provided a solid support to our decoding algorithm and we can get reliable predictions from our theoretical analysis when the simulation is infeasible to perform. Further, our decoding method is essentially the LLR method in linear cryptanalysis, whose validity can be guaranteed by the theory of linear cryptanalysis.

7 Conclusions

In this paper, we have developed two new cryptanalytic tools to bridge the gap between the widely used primitives employing word-based LFSRs and the current mainstream bitwise fast correlation attacks. The first one, a formal framework for fast correlation attacks over extension fields with a thorough theoretical analysis, is the *first* comprehensive answer to the corresponding open problem in the field of correlation attacks. The second technique, serving as a basis to the first one, allows to efficiently compute the bias distributions of large-unit linear

approximations of the flexibly derived GPLFM, which includes all the previously studied topics in the open literature in an unified framework. The size of the data unit is usually chosen according to the structure of the underlying primitive and the building blocks, which greatly extends the freedom of the adversary in the cryptanalysis of many symmetric-key primitives. As an application, we adapted these two techniques to SNOW 2.0, an ISO/IEC 18033-4 standard and a benchmark stream cipher in the European eSTREAM project, and achieved the best key recovery attacks known so far. The new methods are generic and are applicable to other symmetric-key primitives as well, e.g., SNOW 3G, Sosemanuk, Dragon, and some CAESAR candidates. It is our future work to study the large-unit linear approximations of these primitives and launch various attacks accordingly.

Acknowledgments. This work is supported by the National Grand Fundamental Research 973 Program of China (Grant No. 2013CB338002), and the programs of the National Natural Science Foundation of China (Grant No. 60833008, 60603018, 61173134, 91118006, 61272476). The third author was supported in part by the Research Council KU Leuven: a senior postdoctoral scholarship S-F/14/010 linked to the GOA TENSE (GOA/11/007).

References

1. Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Lee, editor, *Advances in Cryptology–ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 113–128. Springer Berlin / Heidelberg, 2004.
2. Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Sosemanuk, a fast software-oriented stream cipher. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 98–118. Springer Berlin Heidelberg, 2008.
3. Anne Canteaut. Fast correlation attacks against stream ciphers and related open problems. In *2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security*, pages 49–54, 2005.
4. Vladimor V. Chepyzhov, Thomas Johansson, and Ben Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption–FSE 2001*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer Berlin Heidelberg, 2001.
5. Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In Lars Knudsen, editor, *Advances in Cryptology–EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer Berlin / Heidelberg, 2002.
6. Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher snow. In Kaisa Nyberg and Howard Heys, editors, *Selected Areas in Cryptography–SAC 2003*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin Heidelberg, 2003.

7. Hakan Englund and Alexander Maximov. Attack the dragon. In Subhamoy Maitra, C.E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *Progress in Cryptology–INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 130–142. Springer Berlin Heidelberg, 2005.
8. ETSI/SAGE. Specification of the 3gpp confidentiality and integrity algorithms uea2 & uia2. In *Document 2: SNOW 3G Specification, version 1.1*. http://www.3gpp.org/ftp/., September 2006.
9. Miia Hermelin, JooYeon Cho, and Kaisa Nyberg. Multidimensional extension of matsui's algorithm 2. In Orr Dunkelman, editor, *Fast Software Encryption–FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 209–227. Springer Berlin Heidelberg, 2009.
10. Fredrik Jönsson and Thomas Johansson. Correlation attacks on stream ciphers over GF(2^n). In *2001 IEEE International Symposium on Information Theory–ISIT'2001*, page 140, 2001.
11. Thomas Johansson and Fredrik Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In Mihir Bellare, editor, *Advances in Cryptology–CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer Berlin / Heidelberg, 2000.
12. Fredrik Jönsson. Some results on fast correlation attacks. *PhD Thesis, Lund University, Sweden* 2002.
13. Jung-Keun Lee, DongHoon Lee, and Sangwoo Park. Cryptanalysis of sosemanuk and snow 2.0 using linear masks. In Josef Pieprzyk, editor, *Advances in Cryptology–ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 524–538. Springer Berlin Heidelberg, 2008.
14. Yi Lu and Serge Vaudenay. Faster correlation attack on bluetooth keystream generator e0. In Matt Franklin, editor, *Advances in Cryptology–CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 35–49. Springer Berlin / Heidelberg, 2004.
15. J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT-15 (1), pages 122–127.
16. Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT' 93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer Berlin Heidelberg, 1994.
17. Alexander Maximov and Thomas Johansson. Fast computation of large distributions and its cryptographic applications. In Bimal Roy, editor, *Advances in Cryptology–ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 313–332. Springer Berlin Heidelberg, 2005.
18. Willi Meier. Fast correlation attacks: Methods and countermeasures. In Antoine Joux, editor, *Fast Software Encryption*, volume 6733 of *Lecture Notes in Computer Science*, pages 55–67. Springer Berlin Heidelberg, 2011.
19. Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1:159–176, 1989.
20. Mohammad A Musa, Edward F Schaefer, and Stephen Wedig. A simplified aes algorithm and its linear and differential cryptanalyses. *Cryptologia*, 27(2):148–177, 2003.
21. K. Nyberg and M. Hermelin. Multidimensional walsh transform and a characterization of bent functions. In *Information Theory for Wireless Networks, 2007 IEEE Information Theory Workshop on*, pages 1–4, 2007.
22. Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for snow 2.0. In Matthew Robshaw, editor, *Fast Software Encryption-FSE 2006*, volume 4047 of

Lecture Notes in Computer Science, pages 144–162. Springer Berlin Heidelberg, 2006.
23. Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.
24. David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology–CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin Heidelberg, 2002.
25. RK Rao Yarlagadda and John E Hershey. *Hadamard matrix analysis and synthesis with applications to communications and signal/Image processing*. Kluwer Academic Publishers, 1997.

A Diagrams of the Invoked Algorithm 2

Algorithm 2 ComputePLFM(F_i, M, D_{i-1})

Parameters: D_{i-1}: a probability of F_{i-1} that results in B_{i-1}, Cr_{i-1}

Temporary Variable: B_i, Cr_i: the output and carry vector of F_i

$\mathbf{v} = (v_0, \cdots, v_{|Cr_{\max}|-1})$: a $|Cr_{\max}|$-dimensional vector

Processing:

1: Submit F_i to the precomputed algorithm in [17] and get $2m$ matrices $M_{(B_i)_t|t}$;
2: **for** $B_i = 0$ **to** $2^m - 1$ **do**
3: $\quad \mathbf{v} = (\prod_{t=m-1}^{0} M_{(B_i)_t|t}) \times (1, 0, \cdots, 0)^T$;
4: $\quad$ **for** $j = 0$ **to** $|Cr_{\max}| - 1$ **do**
5: $\quad\quad M[B_i][j] = M[B_i][j] + (v_j/2^{md}) D_{i-1}$;

Return: M

In the diagram of Algorithm 2, $(B_i)_t$ is the t-th bit of B_i and the matrices in [17] have the similar meaning with our connection matrix, which store the carry information for each bit.

B Computing the Distribution in Eq.(3) over GF(2^8)

Algorithm 4 Computing the distribution of $\mathbf{N}_2(t)$

Parameters:

$\Lambda = (\Lambda_4, \Lambda_3, \Lambda_2, \Lambda_1) \in \mathrm{GF}(2^8)^4$, $s_t = (s_t^4, s_t^3, s_t^2, s_t^1)$,$R2_t = (R2_t^4, R2_t^3, R2_t^2, R2_t^1)$;

Processing:

1: Compute $\Lambda_1(s_{t+16}^1 \boxplus s_{t+5}^1 \boxplus R2_t^1) \oplus \Lambda_1 s_{t+16}^1 \oplus \Lambda_1 s_{t+5}^1 \oplus \Lambda_1 R2_t^1$ and store in M_1
2: **for** $i = 2, \cdots, 4$ **do**
3: $\quad$ Initialize M_2 with zeros.
4: $\quad$ **for** $s_{t+16}^i = 0, \cdots, 255$ and $s_{t+5}^i = 0, \cdots, 255$ and $R2_t^i = 0, \cdots, 255$ **do**
5: $\quad\quad$ **for** $Cr_{i-1} = 0, 1, 2$ **do**
6: $\quad\quad\quad$ **for** $B_{i-1} = 0, \cdots, 255$ **do**
7: $\quad\quad\quad\quad B_i \leftarrow B_{i-1} \oplus \Lambda_i(s_{t+16}^i + s_{t+5}^i + R2_t^i) \oplus \Lambda_i s_{t+16}^i \oplus \Lambda_i s_{t+5}^i \oplus \Lambda_i R2_t^i$;
8: $\quad\quad\quad\quad Cr_i \leftarrow (s_{t+16}^i + s_{t+5}^i + R2_t^i + Cr_{i-1})/2^8$;
9: $\quad\quad\quad\quad M_2[B_i][|Cr_i|] \leftarrow M_2[B_i][|Cr_i|] + M_1[B_{i-1}][|Cr_{i-1}|]$;
10: $\quad M_1 \leftarrow M_2/(2^8 \times 3)$;
15: **Output**: $p_i = M_1[i][0] + M_2[i][1] + M_2[i][2]$ for each $0 \le i \le 255$.

C Unifying the Two Fields

Note that in Eq.(4), the mask $\Gamma = (\texttt{0x00}, \texttt{0x01}, \texttt{0x00}, \texttt{0x03})$ is defined over the Mixcolumn field GF(2^8), which is different from the corresponding field of the LFSR. We need to first unify the two fields for an efficient decoding, otherwise there will be the folded noise introduced by whether xoring the two field constants or not. Here we adopt the following routine to solve this problem. To facilitate the decoding phase, we first find an equivalent representation of the LFSR part theoretically so that it is defined over the new GF(2^{32}) field, which is derived as follows.

We first substitute the low-level GF(2^8) field of the LFSR defined by $x^8 + x^7 + x^5 + x^3 + 1$ (field constant `0xa9`) with the GF(2^8) field defined in Mixcolumn by $x^8 + x^4 + x^3 + x + 1$ (field constant `0x1b`), and then randomly select a primitive polynomial of degree 4 over this new field to construct the new GF(2^{32}) field. Let $\{s_i\}_{i=0}^{\infty}$ be the sequence generated by the LFSR defined over the original GF(2^{32}) field in SNOW 2.0, our observation is that the sequence itself is just a string of bits and is independent of the definition of the underlying field, thus once one segment of sufficient length of the sequence is produced from a LFSR over the field associated with one definition, we can use the classical Berlekamp-Massey algorithm [15] over the field with another definition to reconstruct the LFSR feedback polynomial over the latter field and as a by product, the equivalent state conversion relation between the two field definitions can be obtained. Note that the LFSR sequence $\{s_i\}_{i=0}^{\infty}$ in SNOW 2.0 is primitive, thus the new generated LFSR over the new GF(2^{32}) field is also of length 16. Compared with the other parts of our attack, the complexity of computing the equivalent representation of the LFSR part defined in the new field is negligible. The overall complexity of our attack is dominated by the complexity of the decoding phase given below.

D A Small Scale Version of SNOW 2.0

The LFSR consists of 5 units and each unit is a 16-bit word in GF(2^{16}). The feedback polynomial is $\pi(x) = \alpha x^5 + \alpha^{-1}x^3 + x^2 + 1 \in \mathrm{GF}(2^{16})[x]$, where α is a root of $x^4 + \beta^{10}x^3 + \beta^6 x^2 + x + \beta^{11}$, and β is a root of $x^4 + x + 1 \in \mathrm{GF}(2)[x]$. The FSM has two 16-bit registers $R1$ and $R2$ updated by $R1_{t+1} = (s_{t+3} \boxplus R2_t) \bmod 2^{16}$ and $R2_{t+1} = S(R1_t)$. The function S is composed of four parallel Nibble S-boxes followed by the following MixColumn.

$$S(s_i) = \begin{pmatrix} 1 & 4 \\ 4 & 1 \end{pmatrix} \begin{pmatrix} S_R(s_i^1) & S_R(s_i^3) \\ S_R(s_i^2) & S_R(s_i^4) \end{pmatrix},$$

where S_R is the Nibble S-box in Small AES [20]. The output of FSM is $F_t = (s_{t+4} \boxplus R1_t) \oplus R2_t$. The generated keystream is $z_t = F_t \oplus s_t$.