Fast Near Collision Attack on the Grain v1 Stream Cipher

Bin Zhang[1,2,3,4(✉)], Chao Xu[1,2], and Willi Meier[5]

[1] TCA Laboratory, SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China
{zhangbin,xuchao}@tca.iscas.ac.cn
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] University of Chinese Academy of Sciences, Beijing 100049, China
[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[5] FHNW, Windisch, Switzerland
willi.meier@fhnw.ch

Abstract. Modern stream ciphers often adopt a large internal state to resist various attacks, where the cryptanalysts have to deal with a large number of variables when mounting state recovery attacks. In this paper, we propose a general new cryptanalytic method on stream ciphers, called fast near collision attack, to address this situation. It combines a near collision property with the divide-and-conquer strategy so that only subsets of the internal state, associated with different keystream vectors, are recovered first and merged carefully later to retrieve the full large internal state. A self-contained method is introduced and improved to derive the target subset of the internal state from the partial state difference efficiently. As an application, we propose a new key recovery attack on Grain v1, one of the 7 finalists selected by the eSTREAM project, in the single-key setting. Both the pre-computation and the online phases are tailored according to its internal structure, to provide an attack for any fixed IV in $2^{75.7}$ cipher ticks after the pre-computation of $2^{8.1}$ cipher ticks, given 2^{28}-bit memory and about 2^{19} keystream bits. Practical experiments on Grain v1 itself whenever possible and on a 80-bit reduced version confirmed our results.

Keywords: Cryptanalysis · Stream ciphers · Grain · Near collision

1 Introduction

As a rule of thumb, the internal state size of modern stream ciphers is at least twice as large as the key size, e.g., all the eSTREAM finalists follow this principle, which considerably complicates cryptanalysis. As a typical case, Grain v1, designed by Hell et al. [8,10], has an internal state size of 160 bits with a 80-bit key. Grain v1 has successfully withstood huge cryptanalytic efforts thus far in the single key model [2,5,15,18].

J. B. Nielsen and V. Rijmen (Eds.): EUROCRYPT 2018, LNCS 10821, pp. 771–802, 2018.
https://doi.org/10.1007/978-3-319-78375-8_25

In this paper, we propose a general new cryptanalytic framework on stream ciphers, called fast near collision attack (FNCA). Given a keystream prefix, finding a corresponding internal state of the stream cipher that generates it is equivalent to determining a preimage of the output of a specific function F. For our purpose, it is assumed that each component function of F can be rewritten in a way that depends on only few variables or combinations of the original variables, which is indeed the case for many stream ciphers. The new strategy is based on a combination of the birthday paradox with respect to near collisions and local differential properties of the component functions. To deal with the situation that F has a large number of variables, the near collision property is combined with a divide-and-conquer strategy so that only subsets of the internal state, associated with different keystream vectors, are restored first and merged carefully later to retrieve the full large internal state. The subset of the internal state associated with a specified keystream vector is called the restricted internal state of the corresponding keystream vector. It is observed that the keystream segment difference (KSD) of a given keystream vector *only* depends on the internal state difference (ISD) and the value of the restricted internal state, i.e., only the differences and the values in the restricted internal state can affect the KSD of a specified keystream vector, whatever the difference distribution and state values in the other parts of the full internal state. Thus, we could apply the near collision idea to this restricted internal state, rather than to the whole internal state. Then a self-contained method [16] is introduced and improved to derive the target subset of the internal state from the partial state difference efficiently. The observation here is that instead of collecting two keystream vector sets to find a near collision state pair, we only collect one set and virtualize the other by directly computing it. An efficient distilling technique is suggested to properly maintain the size of the candidates subset so that the correct candidate is contained in this subset with a higher probability than in a purely random situation. The attack consists of two phases. In the pre-computation phase, we prepare a list of differential look-up tables which are often quite small because of the local differential properties of F. Thus the preprocessing complexity is significantly reduced due to the relatively small number of involved variables. These small tables are carefully exploited in the online phase to determine a series of local candidate sets, which are merged carefully to cover a larger partial state. From this partial state, we aim to recover the full internal state according to the concrete structure of the primitive.

As our main application, we mount a fast near collision attack against Grain v1, one of the 3 finalists selected by the eSTREAM project for restricted hardware environments. In addition to the above general strategies, we further reduce the number of variables associated with a specified keystream vector by rewriting variables according to the internal structure of Grain v1 and making some state variables linearly dependent on the others, similar to the linear enumeration procedure in the BSW sampling in [4]. We first focus on the state recovery of the LFSR together with some partial information on the NFSR. Then a new property in the keystream generation of Grain v1 is exploited in the proposed

Z-technique: given the keystream and the LFSR, it is possible to construct a number of keystream chains to efficiently find out some linear equations on the original NFSR variables, which further reduce the number of unknown variables in the NFSR initial state. Given the LFSR part, Grain v1 degrades into a dynamically linearly filtered NFSR in forward direction and a pure linearly filtered NFSR in backward direction. In both cases, all the NFSR internal state variables can be formally expressed as a linear combination of the initial state variables and of some keystream bits [3]. Taking into account that the best linear approximation of the NFSR feedback function in Grain v1 has a bias of $\frac{41}{512}$, we could construct a system of parity-checks of weight 2 on an even smaller number of the initial NFSR variables with a low complexity. These parity-checks need not to be solved, but can be used as a distinguisher via the Fast Walsh Transform (FWT), called the Walsh distinguisher. The correct LFSR candidate could be identified directly from a glance at the distribution of the Walsh spectrum. Thus, we determine the LFSR part in Grain v1 *independent of* the NFSR state, which releases the complexity issue if the whole internal state is treated together. Finally, the left NFSR state could be restored easily by an algebraic attack with a complexity much lower than the above dominated step and the list of remaining candidates could be tested with the consistency of the available keystream to yield the correct one. As a result, both the pre-computation and the online attack phases are tailored to provide a state/key recovery attack[1] on Grain v1 in the single-key setting with an arbitrary known IV in $2^{75.7}$ cipher ticks after a pre-computation of $2^{8.1}$ cipher ticks, given 2^{28}-bit memory and around 2^{19} keystream bits, which is the best key recovery attack against Grain v1 so far and manages to remove the two unresolved assumptions in complexity manipulation in the previous near collision attack at FSE 2013. This attack is about $2^{11.7}$ times faster than the exhaustive search[2]. Our results have been verified both on Grain v1 itself whenever possible and on a reduced version of Grain v1 with a 40-bit LFSR and a 40-bit NFSR in experiments. A comparison of our attack with the exhaustive search is depicted in Table 1. In summary, though the whole structure of Grain v1 is sound, here we list some properties that facilitate our attack.

- The state size is exactly 160 bits with respect to the 80-bit security.
- The whole system will degrade into a linearly filtered NFSR after knowing the LFSR.
- There is a good linear approximation of the updating function of the NFSR.
- The 2-bit keystream vector depends on a relatively small number of variables after rewriting variables.

[1] Due to the invertible state updating, a state recovery attack on Grain v1 could be converted into a key recovery attack directly.

[2] The brute force attack with an expected complexity of $2^{87.4}$ cipher ticks is shown in [18]. Besides, NCA-2.0 [18] requires a huge pre-computation and memory complexities; while NCA-3.0 [18] is based on two assumptions which remains to be verified on Grain v1 itself.

Table 1. Comparison with the best previous attack on the full Grain v1

Attack	Complexities			
	Pre-comp	Data	Memory	Time
Brute force	-	$2^{7.4}$	$2^{7.4}$	$2^{87.4}$
NCA-2.0 [18]	$2^{83.4}$	2^{62}	$2^{65.9}$	$2^{76.1}$
This paper	$2^{8.1}$	2^{19}	2^{28}	$2^{75.7}$

The time complexity unit here is 1 cipher tick as in [18] and the data/memory complexity unit is 1 bit.

Outline. A brief description of the Grain v1 stream cipher is presented in Sect. 2. Then, some preliminaries relevant to our work are presented in Sect. 3 together with a brief review of the previous near collision attack in [18]. The framework of FNCA is established with the theoretical analysis in Sect. 4 and then applied to Grain v1 in Sect. 5, respectively. In Sect. 6, practical simulations both on Grain v1 itself and on the reduced version are provided. Finally, some conclusions are drawn and future work is pointed out in Sect. 7.

2 Description of Grain v1

Grain v1 is a bit-oriented stream cipher, which consists of a pair of linked 80-bit shift registers, one is a linear feedback shift register (LFSR) and another is a non-linear feedback shift register (NFSR), whose states are denoted by $(l_i, l_{i+1}, ..., l_{i+79})$ and $(n_i, n_{i+1}, ..., n_{i+79})$ respectively. The updating function of the LFSR is $l_{i+80} = l_{i+62} \oplus l_{i+51} \oplus l_{i+38} \oplus l_{i+23} \oplus l_{i+13} \oplus l_i$ and the updating function of the NFSR is

$$\begin{aligned} n_{i+80} = \; & l_i \oplus n_{i+62} \oplus n_{i+60} \oplus n_{i+52} \oplus n_{i+45} \oplus n_{i+37} \oplus n_{i+33} \oplus n_{i+28} \oplus n_{i+21} \\ & \oplus n_{i+14} \oplus n_{i+9} \oplus n_i \oplus n_{i+63}n_{i+60} \oplus n_{i+37}n_{i+33} \oplus n_{i+15}n_{i+9} \\ & \oplus n_{i+60}n_{i+52}n_{i+45} \oplus n_{i+33}n_{i+28}n_{i+21} \oplus n_{i+63}n_{i+45}n_{i+28}n_{i+9} \\ & \oplus n_{i+60}n_{i+52}n_{i+37}n_{i+33} \oplus n_{i+63}n_{i+60}n_{i+21}n_{i+15} \\ & \oplus n_{i+63}n_{i+60}n_{i+52}n_{i+45}n_{i+37} \oplus n_{i+33}n_{i+28}n_{i+21}n_{i+15}n_{i+9} \\ & \oplus n_{i+52}n_{i+45}n_{i+37}n_{i+33}n_{i+28}n_{i+21}. \end{aligned}$$

The keystream generation phase, shown in Fig. 1, works as follows. The combined NFSR-LFSR internal state is filtered by a non-linear boolean function $h(x) = x_1 \oplus x_4 \oplus x_0x_3 \oplus x_2x_3 \oplus x_3x_4 \oplus x_0x_1x_2 \oplus x_0x_2x_3 \oplus x_0x_2x_4 \oplus x_1x_2x_4 \oplus x_2x_3x_4$, which is chosen to be balanced and correlation immune of the first order with the variables x_0, x_1, x_2, x_3 and x_4 corresponding to the tap positions l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64} and n_{i+63} respectively. The output z_i is taken as $z_i = \bigoplus_{k \in \mathcal{A}} n_{i+k} \oplus h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63})$, where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$. The details of the initialization phase are omitted here, the only property relevant to our work is that the initialization phase is invertible.

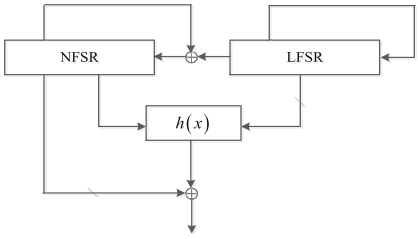

Fig. 1. Keystream generation of Grain v1

3 Preliminaries

In this section, some basic definitions and lemmas are presented with a brief review of the previous near collision attack on Grain v1 in [18]. The following notations are used hereafter.

- $w_H(\cdot)$: the Hamming weight of the input argument.
- d: the Hamming weight of the internal state difference (ISD).
- l: the bit length of the keystream vector.
- n: the bit length of the internal state, whether restricted or not.
- Δx: the value of the ISD, whether restricted or not.
- $V(n,d)$: the total number of the ISDs with $w_H(\Delta x) \leq d$.
- Ω: the number of CPU-cycles to generate 1 bit keystream in Grain v1 in software.

3.1 Basic Conceptions and Lemmas

Let $B_d = \{\Delta x \in \mathbb{F}_2^n | w_H(\Delta x) \leq d\} = \{\Delta x_1, \Delta x_2, ..., \Delta x_{V(n,d)}\}$ and $|B_d| = V(n,d) = \sum_{i=0}^{d} \binom{n}{i}$, where $|\cdot|$ denotes the cardinality of a set. Two n-bit strings s, s' are said to be d-near-collision, if $w_H(s \oplus s') \leq d$ holds. Similar to the birthday paradox, which states that two random subsets of a space with 2^n elements are expected to intersect when the product of their sizes exceeds 2^n, we present the following generalized lemma, which includes the d-near-collision Lemma in [18] as a special case.

Lemma 1. *Given two random sets A and B consisting of n-bit elements and a condition set D, then there exists a pair $(a,b) \in A \times B$ satisfying one of the conditions in D if*

$$|A| \cdot |B| \geq \frac{c \cdot 2^n}{|D|} \tag{1}$$

holds, where c is a constant that determines the existence probability of one good pair (a,b).

Proof. We regard each $a_i \in A$ and $b_j \in B$ as an uniformly distributed random variable with the realization values in $\mathbb{F}_2^n$. Let $A = \{a_1, a_2, ..., a_{|A|}\}$ and $B =$

$\{b_1, b_2, ..., b_{|B|}\}$, we represent the event that a pair $(a_i, b_j) \in A \times B$ satisfies one of the conditions in D briefly as $(a_i, b_j) \in D$. Let ϕ be the characteristic function of the event $\phi((a_i, b_j) \in D)$, i.e.,

$$\phi((a_i, b_j) \in D) = \begin{cases} 1 & \text{if } (a_i, b_j) \in D \\ 0 & \text{otherwise.} \end{cases}$$

For $1 \leq i \leq |A|$ and $1 \leq j \leq |B|$, the number $N_{A,B}(D)$ of good pairs (a_i, b_j) satisfying $(a_i, b_j) \in D$ is $N_{A,B}(D) = \sum_{i=1}^{|A|} \sum_{j=1}^{|B|} \phi((a_i, b_j) \in D)$. Thus, the expected value of $N_{A,B}(D)$ of the pairwise independent random variables can be computed as $E(N_{A,B}(D)) = |A| \cdot |B| \cdot \frac{|D|}{2^n}$. Therefore, if we choose the sizes of A and B satisfying Eq. (1), the expected number of good pairs is at least c. □

While when $D = B_d$, Lemma 1 reduces to the d-near-collision Lemma in [18], Lemma 1 itself is much more general in the sense that D could be an arbitrary condition set chosen by the adversary, which provides a lot of freedom for cryptanalysis. Another issue is the choice of c. In [16], the relation between the choice of the constant c and the existence probability of a d-near-collision pair is illustrated as follows for random samples:

$$\Pr(\text{d-near-collision}) = \begin{cases} 0.606 & \text{if } c = 1 \\ 0.946 & \text{if } c = 3 \\ 0.992 & \text{if } c = 5. \end{cases}$$

As stated in [16], these relations are obtained from the random experiments with a modest size, i.e., for each c value, 100 strings of length 40 to 49 for d-values from 10 to 15 are generated, *not* in a real cipher setting.

Remarks. In a concrete primitive scenario, it is found that the constant c sometimes needs to be even larger to assure a high existence probability of near collision good pairs. In our experiments, we find that the above relation does not hold for Grain v1 and its reduced versions. In these cases, we have to set $c = 8$ or even $c = 10$ to have an existence probability as high as desirable for the subsequent attack procedures. We believe that for each cipher, the choice of the constant c and its correspondence to the existence probability of a near collision pair is a fundamental measure related to the security of the primitive. The following fact is used in our new attack.

Corollary 1. *For a specified cipher and a chosen constant c, let A and B be the internal state subsets associated with the observable keystream vectors, where each element of A and B is of n-bit length. If we choose* $|A| = 1$ *and* $|B| \geq c \cdot \frac{2^n}{|D|}$, *then there exists an element* $b_i \in B$ *such that the pair* (a, b_i) *with the only element* $a \in A$ *forms a d-near collision pair with a probability dependent on c.*

Note that in the near collision setting, the bits in the restricted internal state may be nonadjacent. These bit positions are determined by the tap positions of the composition mapping of the output function if only a short prefix is chosen by the adversary and also dependent on the state updating function, whether

linear or non-linear, if some inconsecutive, or even far away, keystream bits are chosen to be considered together.

In [18], the two sets A and B are chosen to be of equal size, i.e., $|A| = |B|$ to minimize the data complexity, in which case the adversary has to deal with all the candidate state positions one-by-one. Instead, Corollary 1 is used in our new attack on Grain v1 via the self-contained method introduced later in Sect. 4.3, to restore the restricted internal state defined below, at a *specified* chosen position along the keystream segment under consideration.

Definition 1. *For a specified cipher, the subset* $\mathbf{x} = (x_{i_0}, x_{i_1}, \ldots, x_{i_{n-1}})$ *of the full internal state associated with a given keystream vector* $\mathbf{z} = (z_{j_0}, z_{j_1}, \ldots, z_{j_{l-1}})$ *is called the restricted internal state associated with* $\mathbf{z}$.

We choose the following definition of the restricted BSW sampling resistance in stream ciphers.

Definition 2. *Let* $\mathbf{z} = (z_{j_0}, z_{j_1}, \ldots, z_{j_{l-1}})$ *be the known keystream vector selected by the adversary, if* l *internal state bits in the restricted internal state* $\mathbf{x}$ *associated with* $\mathbf{z}$ *could be represented explicitly by* $\mathbf{z}$ *and the other bits in* $\mathbf{x}$*,* l *is called the restricted BSW sampling resistance corresponding to* $(\mathbf{x}, \mathbf{z})$.

It is well known that Grain v1 has a sampling resistance of at least 18 [18], thus from Definition 2, we have $l \leq 18$. Actually, we prefer to consider small values of l in our analysis to reduce the memory complexity and to facilitate the verification of theoretical predictions. Note that here the indices $j_0, j_1, \ldots, j_{l-1}$, either consecutive or inconsecutive, could be chosen arbitrarily by the adversary. The restricted BSW sampling inherits the linear enumeration nature of the classical BSW sampling in [4], but unlike the classical BSW sampling, the new sampling does not try to push this enumeration procedure as far as possible, it just enumerates a suitable number of steps and then terminates.

3.2 The Previous Near Collision Attack

At FSE 2013, a near collision attack on Grain v1 was proposed in [18], trying to identify a near collision in the whole internal state at different time instants and to restore the two involved states accordingly. For such an inner state pair, the keystream prefixes they generate will be similar to each other and the distribution of the KSDs are non-uniform.

The adversary first intends to store the mapping from the specific KSDs to the possible ISDs of the *full* inner state with the sorted occurring probabilities in the pre-computed tables. Then in the online phase, he/she tries to recover the correct ISD by utilizing the pre-computed tables, then the two internal states from the determined ISD. The crucial problem here is to examine a large number of possible pairs whether they are truly near collision or not. In this process, strong wrong-candidate filter with a low complexity is needed, while in its form in [18], the reducing effect is not so satisfactory. In order to overcome this problem, the BSW sampling property and the special table techniques are briefly outlined

based on two assumptions, which are essential for the complexity manipulation from the 64-bit reduced version experiments to the full version theoretical attack. In [18], the examination of the candidate state pairs is executed by first recovering the two internal states from the specified ISD. For the LFSR part, this is of no problem since the LFSR updates independently; but for the NFSR part, it is really a problem in [18]. Though the adversary knows the two specified keystream vectors and their corresponding ISD, it is still difficult to restore the full 80-bit NFSR state in such an efficient way that this routine could be invoked a large number of times. Besides, the special table technique assumes that about 50% of all the possible ISDs could be covered on average, which is very hard to verify for the full Grain v1, thus the successful probability of this attack cannot be guaranteed.

In the following, we will show that the adversary need not to recover the full internal state at once when making the examination, actually specified subsets of the internal state could be restored more efficiently than previously thought to be possible, thus the time/memory complexities of the new attack can be considerably reduced with an assured success probability and *without* any assumption in the complexity manipulation.

4 Fast Near Collision Attacks

In this section, we will describe the new framework for fast near collision attacks, including both the pre-computation phase and the online attack phase, with the theoretical justifications.

4.1 General Description of Fast Near Collision Attacks

The new framework is based on the notion of the restricted internal state corresponding to a fixed keystream vector, which is presented in Definition 1 above. Given $\mathbf{z} = (z_{j_0}, z_{j_1}, \ldots, z_{j_{l-1}})$ with z_{j_i} $(0 \le i \le l-1)$ not necessarily being consecutive in the real keystream, the corresponding restricted internal state $\mathbf{x}$ for $\mathbf{z}$ is determined by the output function f together with its tap positions, and the state updating function g of the cipher, i.e., induced by the intrinsic structure of the cipher. Besides, from the keystream vector $\mathbf{z}$, it is natural to look at the augmented function for $\mathbf{z}$.

Definition 3. *For a specified cipher with the output function f and the state updating function g, which outputs one keystream bit in one tick, the lth-order augmented function $Af : \mathbb{F}_2^{|\mathbf{x}|} \to \mathbb{F}_2^l$ for a given $(\mathbf{x}, \mathbf{z})$ pair is defined as $Af(\mathbf{x}) = (f(\mathbf{x}), f(g^{i_1}(\mathbf{x})), \ldots, f(g^{i_{l-1}}(\mathbf{x})))$.*

Note that the definition of the augmented function here is different from the previous ones in [1,11]. In Definition 3, the augmented function is defined on a *subset* of the whole internal state; while in [1,11], similar functions are usually defined on the full internal state. As can be seen later, this difference will make sense in launching a near collision attack, i.e., we need not target the full internal

state, which is usually quite large, at once any more, now we just look for near collision in the sub-states chosen by the attacker. A high-level description of fast near collision attack is depicted in Algorithm 1.

Algorithm 1. FNCA

Parameters: `index`: the concrete value of a KSD
`prefix`: the concrete value of a keystream vector
Offline: **for** each combination of (`index`, `prefix`) **do**
Construct the table T[`index`, `prefix`], projecting from the KSD `index` to all the possible ISDs sorted by the occurring rates
end for
Input: A keystream segment $\mathbf{z}_{\text{total}} = (z_{j_0}, z_{j_1}, \ldots, z_{j_{l-1}}, z_{j_l}, \ldots, z_{j_{l+\gamma}})$
Online: Recover the full internal state $\mathbf{x}_{\text{full}}$ matching with $\mathbf{z}_{\text{total}}$
1: Divide $\mathbf{z}_{\text{total}}$ into α overlapping parts $\mathbf{z}_i$ $(1 \leq i \leq \alpha)$ and a suffix $\mathbf{z}_\mu$
2: **for** $i = 1$ **to** α **do**
3: get the candidates list L_i of the restricted internal state $\mathbf{x}_i$ for $\mathbf{z}_i$
4: **end for**
5: Merge L_is to get a candidate list for the possible partial state $\mathbf{x}_{\text{merge}}$
6: **for** each candidate of $\mathbf{x}_{\text{merge}}$ **do**
7: restore $\mathbf{x}_{\text{full}}$ and test the consistency with the suffix $\mathbf{z}_\mu$

The following proposition provides new insights on what influence the whole internal state size has on the feasibility of a near collision attack.

Proposition 1. *For a specified cipher and two keystream vectors* $\mathbf{z}$ *and* $\mathbf{z}'$*, the KSD* $\Delta\mathbf{z} = \mathbf{z} \oplus \mathbf{z}'$ *only depends on the ISD* $\Delta\mathbf{x} = \mathbf{x} \oplus \mathbf{x}'$ *and the values of* $\mathbf{x}$ *and* $\mathbf{x}'$*, whatever the difference and the values in* $\bar{\mathbf{x}}$*, the other parts of the whole internal state.*

Proof. It suffices to see the algebraic expressions of the keystream bits under consideration. By taking a look at the input variables, we have the claim. □

Offline. Proposition 1 makes the pre-computation phase in FNCA quite different from and much more efficient than that in the previous NCA in [18]. Now we need not to exhaustively search through all the possible ISDs over the *full* internal state, which is usually quite large, instead we just search through all the possible ISDs over a specified restricted internal state corresponding to a given keystream vector, which is usually much shorter compared to the full internal state. In Algorithm 1, we use two parameters `index` and `prefix` to characterise this difference with `index` being the KSD and `prefix` being the *value* of one of the two specified keystream vectors. For each possible combination of (`index`, `prefix`), we construct an individual table for the pair. Thus, many relatively small tables are built instead of one large pre-computed table, which greatly reduces the time/memory complexities of the offline phase, improves the accuracy of the pre-computed information, and finally assures a high success rate of the new attack.

Online. With the differential tables prepared in the offline phase, the adversary first tries to get some candidates of the target restricted internal state $\mathbf{x_1}$ and

to filter out as much as possible wrong candidates of $\mathbf{x_1}$ in a reasonable time complexity. Then he/she moves to another restricted internal state $\mathbf{x_2}$, possibly overlapped with $\mathbf{x_1}$, but not coincident with $\mathbf{x_1}$, and get some candidates of $\mathbf{x_2}$. This process is repeated until enough internal state bits are recovered in an acceptable time/memory complexities and merge the candidate lists L_i together to get a candidate list of possible partial state $\mathbf{x}_{\text{merge}}$. Finally, from $\mathbf{x}_{\text{merge}}$, he/she tries to retrieve the full internal state and check the candidates by using the consistency with the keystream segment.

There are three essential problems that have to be solved in this process. The first one is how to efficiently get the candidates for each restricted internal state and further to filter out those wrong values as much as possible in each case? The second is how to efficiently merge these partial states together without the overflowing of the number of possible internal state candidates, i.e., we need to carefully control the increasing speed of the possible candidates during the merging phase. At last, we need to find some very efficient method to restore the other parts of the full internal state given $\mathbf{x}_{\text{merge}}$, which lies at the core of the routine. We will provide our solutions to these problems in the following sections.

4.2 Offline Phase: Parameterizing the Differential Tables

Now we explain how to pre-compute the differential tables T[index, prefix], conditioned on the event that the value of one of the two keystream vectors is prefix when the KSD is index. Let $\mathbf{x} = (x_{i_0}, x_{i_1}, \ldots, x_{i_{n-1}})$ be the restricted internal state associated with $\mathbf{z} = (z_{j_0}, z_{j_1}, \ldots, z_{j_{l-1}})$, for such a chosen $(\mathbf{x}, \mathbf{z})$ pair, Algorithm 2 fulfills this task. Algorithm 2 is the inner routine of the pre-computation phase of FNCA in the general case, where N_1 and N_2 are the two random sampling sizes when determining whether a given ISD Δx of the restricted internal state $\mathbf{x}$ could generate the KSD index and what the occurring probability is. Algorithm 2 is interesting in its own right, though not adopted in the state recovery attack on Grain v1 in Sect. 5. It can be applied in the most general case with the theoretical justification when dedicated pre-computation is impossible.

Algorithm 2. Constructing the differential table T[index, prefix]

1: **for** each ISD Δx s.t. $w_H(\Delta x) \leq d$ **do**
2: **for** $i = 1$ **to** N_1 **do**
3: determine whether Δx could generate the specified KSD index
4: **if** yes **then**
5: **for** $j = 1$ **to** N_2 **do**
6: generate random $\mathbf{x}$ s.t. $Af(\mathbf{x}) =$ prefix and form the pair $(\mathbf{x}, \mathbf{x} \oplus \Delta x)$
7: compute $\mathbf{z} = Af(\mathbf{x})$ and $\mathbf{z}' = Af(\mathbf{x} \oplus \Delta x)$
8: count the number of times counter that $\Delta z = \mathbf{z} \oplus \mathbf{z}' =$ index
9: store the ratio counter$/N_2$ with Δx in T[index, prefix]
10: Sort the ISDs according to the occurring rates

From $V(n, d) = \sum_{i=0}^{d} \binom{n}{i}$, the time complexity of Algorithm 2 is $P = 2 \cdot V(n, d) \cdot (N_1 + N_2 \cdot (j_{l-1} + 1))$ cipher ticks and the memory requirement is at

most $V(n,d)\cdot(\lceil\log_2 n\rceil\cdot d+(7+7))$ bits, where a $14=7+7$-bit string is used to store the percentage number, e.g., for 76.025%, we use 7 bits to store the integer part $76<128$ and another 7 bits to save the fractional part 0.025 as 0.0000011. In Table 2, the global information of the 16 T tables for Grain v1 with the 23 original variables when $l=2$ is shown, where Pr_{divs} is defined as follows.

Table 2. The summary of the pre-computation phase of Grain v1 for 2-bit keystream vector with the 23 original variables

(index, prefix)	|T|	Pr_{divs}	(index, prefix)	|T|	Pr_{divs}
(0x0, 0x0)	16126	0.314426	(0x2, 0x0)	16106	0.319008
(0x0, 0x1)	16126	0.314434	(0x2, 0x1)	16106	0.318892
(0x0, 0x2)	16126	0.314504	(0x2, 0x2)	16106	0.318934
(0x0, 0x3)	16126	0.314504	(0x2, 0x3)	16106	0.318955
(0x1, 0x0)	16106	0.318958	(0x3, 0x0)	16044	0.311827
(0x1, 0x1)	16106	0.319050	(0x3, 0x1)	16044	0.311839
(0x1, 0x2)	16106	0.318896	(0x3, 0x2)	16044	0.311979
(0x1, 0x3)	16106	0.318928	(0x3, 0x3)	16044	0.311833

Definition 4. *For each* $T[\texttt{index},\texttt{prefix}]$*, let* $|T|$ *be the number of ISDs in the table, the diversified probability of this table is defined as* $Pr_{divs}=\frac{\sum_{\Delta x\in T} Pr_{\Delta x}}{|T|}$*, where* Δx *ranges over all the possible ISDs in the table.*

The diversified probability of a T[index, prefix] table measures the average reducing effect of this table that for a random restricted internal state $\mathbf{x}$ such that $Af(\mathbf{x})=\texttt{prefix}$, flip the bits in $\mathbf{x}$ according to a $\Delta x\in$ T and get $\mathbf{x}'$, then with probability Pr_{divs}, $Af(\mathbf{x}\oplus\mathbf{\Delta x})=\texttt{prefix}\oplus\texttt{index}$. From Definition 4, the success rate of the new FNCA is quite high, for we have taken each possible ISD into consideration in the attack.

Corollary 2. *From Table 2, if the* index *is fixed, then the* 4 Pr_{divs}*s corresponding to different* prefix*es are approximately the same, i.e., the* 4 *T tables have almost the same reducing effect for filtering out wrong candidates.*

Corollary 2 is the basis of the merging operation in the online attack phase, which assures that with the restricted BSW sampling resistance of Grain v1 and the self-contained method introduced later, the partial state recovery procedure will not be affected when the keytream vector under consideration has changed its value along the actual keystream. If the keystream vector under consideration has the value prefix, the adversary uses the value prefix$\oplus$index in the computing stage of the self-contained method to have the KSD remaining the same.

4.3 Online Phase: Restoring and Distilling the Candidates

Now we come to the online phase of FNCA. The aim is to restore the overlapping restricted internal states one-by-one, merge them together, and finally from the already covered state $\mathbf{x}_{\text{merge}}$ to retrieve the correct full internal state. For a chosen $(\mathbf{x}, \mathbf{z})$ pair, the refined self-contained method is depicted in Algorithm 3.

Algorithm 3. The refined self-contained method

```
Initialize i = 0
while i ≤ c · 2^n / |D| do
    load x with a new random value so that it generates z ⊕ index
    for each possible ISD Δx in T[index, z ⊕ index] do
        compute x' = x ⊕ Δx
        if x' generates z then
            put x' into the candidates list L
        end if
    end for
    i = i + 1
end while
```

The original self-contained method was proposed in [16], whose idea is to make a tradeoff between the data and the online time complexity in such a way that the second set B of keystream vectors in Lemma 1 is generated by the adversary himself, thus he also knows the actual value of the corresponding internal state that generates the keystream vector. Therefore, given the ISD from the pre-computed tables, the adversary could just xor the ISD with the internal state matching with the keystream vector in B to get the candidate internal state for the keystream vectors in A. In Algorithm 3, it is quite possible that although obtained from a different new starting value for the restricted internal state $\mathbf{x}$, some candidates $\mathbf{x}'$ will collide with the already existing element in the list, thus the final number of hitting values in the list is not so much as the number of invoking times $c \cdot \frac{2^n}{|D|}$. The following theorem gives the expected value of the actual hitting numbers.

Theorem 1. *Let b be the number of all the values that can be hit and $a = c \cdot \frac{2^n}{|D|} \cdot |T| \cdot P_{divs}$, then after one invoking of Algorithm 3, the mathematical expectation of the final number r of hitting values in the list is*

$$E[r] = \sum_{r=1}^{a} \frac{\binom{b}{r} \cdot r! \cdot \left\{ {a \atop r} \right\} \cdot r}{b^a}, \tag{2}$$

where $\left\{ {a \atop r} \right\}$ is the Stirling number of the second kind, $\binom{b}{r}$ is the binomial coefficient and $r!$ is the factorial.

Proof. Note that the Stirling number of the second kind [17] $\left\{ {a \atop r} \right\}$ counts the number of ways to partition a set of a objects into r non-empty subsets, i.e., each way is a partition of the a subjects, which coincides with our circumstance. Thus we can model the process of Algorithm 3 as follows.

We throw a balls into b different boxes, and we want to know the probability that there are exactly r boxes having some number of balls in. From this converted model, we can see that the size of the total sample space is b^a, while the number of samples in our expected event can be calculated in the following steps.

1. Choose r boxes to hold the thrown balls, there are $\binom{b}{r}$ ways to fulfill this step.
2. Permute these r boxes, there are $r!$ ways to fulfill this step.
3. Partition the a balls into r non-empty sets, this is just the Stirling number of the second kind $\left\{ {a \atop r} \right\}$.

Following the multiplication principle in combinatorics, the size of our expected event is just the product of the above three. This completes the proof. □

We have made extensive experiments to verify Theorem 1 and the simulation results match the theoretical predictions quite well. Back to the self-contained method setting, a is just the number of valid candidates satisfying the conditions that the KSD is `index` and one of the keystream prefix is `prefix`, of which some may be identical due to the flipping according to the ISDs in T.

In general, the opponent can build a table for the function f, mapping the partial sub-states to the keystream vector z, to get a full list of inputs that map to a given z. In order to reduce the candidate list size in a search, he may somehow choose a smaller list of inputs that map to z, and hope that the correct partial state is still in the list with some probability p, depending on the size of the list. The aim of the distilling phase is to exploit the birthday paradox regarding d-near collisions, local differential properties of f, and the self-contained method to derive smaller lists of input sub-states so that the probability that the correct state is in the list is at least p. From Lemma 1, with a properly chosen constant c, the correct internal state $\mathbf{x}$ will be in the list L with a high probability, e.g., 0.8 or 0.95. For a chosen $(\mathbf{x}, \mathbf{z})$ pair, the candidates reduction process is depicted in Algorithm 4.

Algorithm 4. Distilling the candidates

Parameter: a well chosen constant β

for $i = 1$ to β **do**
    run Algorithm 3 to get the candidates list L_i
end for
Initialize a list U and let $U = L_1$
for $i = 2$ to β **do**
    intersect U with L_i, i.e., $U \leftarrow U \cap L_i$

The next theorem characterizes the number of candidates passing through the distilling process in Algorithm 4.

Theorem 2. *The expected number of candidates in the list U in Algorithm 4 after $\beta - 1$ steps of intersection is $|U_1| \cdot (\frac{E[r]}{b})^{\beta-1}$, where $|U_1| = |L_1|$ is the number of candidates present in the first list L_1 and $E[r]$ is the expected number of hitting values in one single invoking of Algorithm 3.*

Proof. For simplicity, let $|U_i| = f_i$ denote the cardinality of the candidates list after $i-1$ steps of intersection for $1 \le i \le \beta - 1$. Note that in the intersection process, if there are f_i candidates in the current list U, then at the next intersection operation, an element in U has the probability $\frac{E[r]}{b}$ to remain, and the probability $1 - \frac{E[r]}{b}$ to be filtered out.

Let $f_i \to f_{i+1}$ denote the event that there are f_{i+1} elements left after one intersection operation on the f_i elements in the current U. The expected value of f_{i+1} is $E[f_{i+1}] = \sum_{j=0}^{f_i} \binom{f_i}{j} \cdot (\frac{E[r]}{b})^j \cdot (1 - \frac{E[r]}{b})^{f_i - j} \cdot j = f_i \cdot \frac{E[r]}{b}$. Thus we have the following recursion

$$f_{\beta-1} = f_{\beta-t-1} \cdot \prod_{i=1}^{t} (\frac{E[r]}{b}) = |U_1| \cdot \underbrace{(\frac{E[r]}{b}) \cdot (\frac{E[r]}{b}) \cdot \cdots \cdot (\frac{E[r]}{b})}_{\beta-1} = |U_1| \cdot (\frac{E[r]}{b})^{\beta-1},$$

which completes the proof. □

Algorithm 5. Improving the existence probability of the correct $\mathbf{x}$

Parameter: a well chosen constant γ

1: **for** $i = 1$ to γ **do**
2: run Algorithm 4 to get the candidates list U_i
3: **end for**
4: Initialize a list V and let $V = U_1$
5: **for** $i = 2$ to γ **do**
6: union V with U_i, i.e., $V \leftarrow V \cup U_i$

Theorem 2 partially characterizes the distilling process in theory. Now the crucial problem is what the *reduction effect* of this process is, which is determined by the choice of the constant c intrinsic to each primitive and the number of variables involved in the current augmented function. From the cryptanalyst's point of view, the larger β is, the lower the probability that the correct $\mathbf{x}$ is involved in each generated list, thus it is better for the adversary to make some tradeoff between β and this existence probability. Algorithm 5 provides a way to exploit this tradeoff to get some higher existence probability of the correct restricted internal state $\mathbf{x}$.

In Algorithm 5, several candidate lists are first generated by Algorithm 4 with a number of intersection operations for each list, then these lists are unified together to form a larger list so that the existence probability of the correct $\mathbf{x}$ becomes higher compared to that of each component list.

Theorem 3. *Let the expected number of candidates in list V in Algorithm 5 after i $(1 \le i \le \gamma)$ steps of union be F_i, then the following relation holds*

$$F_{i+1} = F_i + |U_{i+1}| - \sum_{j=0}^{|U_{i+1}|} \frac{\binom{F_i}{j} \cdot \binom{F_{i+1}-F_i}{|U_{i+1}|-j}}{\binom{F_{i+1}}{|U_{i+1}|}} \cdot j \ , \ 1 \le i \le \gamma - 1 \qquad (3)$$

where $|F_1| = |U_1|$.

Proof. Note that when a new U_{i+1} is unified into V, we have $F_{i+1} = F_i + |U_{i+1}| - |F_i \cap U_{i+1}|$. It suffices to note that Eq.(3) can be derived from the hypergeometric distribution for the $|F_i \cap U_{i+1}|$ part, which completes the proof. □

Theorem 3 provides a theoretical estimate of $|V|$, which is quite close to the experimental results. After getting V for $\mathbf{x}_1$, we move to the next restricted internal state $\mathbf{x}_2$, as depicted in Algorithm 1 until we recovered all the α restricted internal states. Then we merge the restored partial states $\mathbf{x}_i$ for $1 \leq i \leq \alpha$ to cover a larger part of the full internal state. We have to recover the full internal state conditioned on $\mathbf{x}_{\text{merge}}$. The next theorem describes the reduction effect when merging the candidate lists of two restricted internal states.

Theorem 4. *Let the candidates list for $\mathbf{x}_i$ be V_i, then when merging the candidates list V_i for $\mathbf{x}_i$ and V_{i+1} for $\mathbf{x}_{i+1}$ to cover an union state $\mathbf{x}_i \cup \mathbf{x}_{i+1}$, the expected number of candidates for the union state $\mathbf{x}_i \cup \mathbf{x}_{i+1}$ is*

$$E[|V_{\mathbf{x}_i \cup \mathbf{x}_{i+1}}|] = \frac{|V_i| \cdot |V_{i+1}|}{|V_i \cap V_{i+1}|},$$

where $V_{\mathbf{x}_i \cup \mathbf{x}_{i+1}}$ is the candidates list for the union state $\mathbf{x}_i \cup \mathbf{x}_{i+1}$.

Proof. Denote the bits in $\mathbf{x}_i \cap \mathbf{x}_{i+1}$ by $\mathbf{I} = I_0, I_1, \cdots, I_{|\mathbf{x}_i \cap \mathbf{x}_{i+1}|-1}$ when merging the two adjacent restricted internal states $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, then we can group V_i and V_{i+1} according to the $|\mathbf{x}_i \cap \mathbf{x}_{i+1}|$ concrete values of $\mathbf{I}$. For the same value pattern of the common bits in $\mathbf{I}$, we can just merge the two states $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ by concatenating the corresponding candidate states together directly. Thus, the expected number of candidates for the union state is

$$E[|V_{\mathbf{x}_i \cup \mathbf{x}_{i+1}}|] = \frac{|V_i|}{|V_i \cap V_{i+1}|} \cdot \frac{|V_{i+1}|}{|V_i \cap V_{i+1}|} \cdot |V_i \cap V_{i+1}| = \frac{|V_i| \cdot |V_{i+1}|}{|V_i \cap V_{i+1}|},$$

which completes the proof. □

Corollary 3. *In the merging process of Algorithm 1, let M_A and M_B be two partial internal states, each merged from possibly several restricted internal states respectively, then when merging M_A and M_B together, the expected number of candidates for the union state $M_A \cup M_B$ is $E[|M_A \cup M_B|] = \frac{|M_A| \cdot |M_B|}{|M_A \cap M_B|}$.*

Proof. It suffices to note the statistical independence of each invoking of Algorithm 5 and Theorem 4. □

Finally, we present the theorem on the success probability of Algorithm 5.

Theorem 5. *Let the probability that the correct value of the restricted internal state $\mathbf{x}$ will exist in V be $Pr_{\mathbf{x}}$, then we have $Pr_{\mathbf{x}} = 1 - (1 - (P_c)^{\beta})^{\gamma}$, where P_c is the probability that the correct value of the restricted internal state $\mathbf{x}$ exist in U for one single invoking of Algorithm 3.*

Proof. From Algorithms 4 and 5, the probability that for all the γ U_is, the correct value of $\mathbf{x}$ does not exist in the list is $(1 - (P_c)^{\beta})^{\gamma}$, thus the opposite event has the probability given above. This completes the proof. □

Based on the above theoretical framework of FNCA, we will develop a state recovery attack against Grain v1 in the next section, taking into account the dedicated internal structure of the primitive.

5 State Recovery Attack on Grain v1

Now we demonstrate a state recovery attack on the full Grain v1. The new attack is based on the FNCA framework described in Sect. 4 with some techniques to control the attack complexities.

5.1 Rewriting Variables and Parameter Configuration

From the keystream generation of Grain v1, we have $z_i = \bigoplus_{k\in\mathcal{A}} n_{i+k} \oplus h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63})$, where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$, i.e., one keystream bit z_i is dependent on 12 binary variables, of which 7 bits from the NFSR form the linear masking bit $\bigoplus_{k\in\mathcal{A}} n_{i+k}$, 4 bits from the LFSR and n_{i+63} from the NFSR are involved in the filter function h.

For a straightforward FNCA on Grain v1, even considering two consecutive keystream bits, we have to deal with 23 binary variables simultaneously at the beginning of the attack. Thus the number of involved variables will grow rapidly with the running of the attack, and probably overflow at some intermediate point. To overcome this difficulty, we introduce the following two techniques to reduce the number of free variables involved in the keystream vectors.

Let $x_i = n_{i+1} \oplus n_{i+2} \oplus n_{i+4} \oplus n_{i+10} \oplus n_{i+31} \oplus n_{i+43} \oplus n_{i+56}$, then we have

$$z_i = x_i \oplus h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63}). \tag{4}$$

There are only 6 binary variables $x_i, l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63}$ involved in Eq.(4) and if we consider a keystream vector $\mathbf{z} = (z_i, z_{i+1})$, there are only 12 variables now, almost reduced by half compared to the previous number 23. Note that the rewriting technique is known to be useful in [9] before in algebraic attacks on stream ciphers.

Besides, we can still use the linear enumeration procedure as in the BSW sampling case to reduce the variables further. Precisely, from Eq.(4), we have $x_i = z_i \oplus h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63})$, thus for the above keystream vector $\mathbf{z} = (z_i, z_{i+1})$, we could actually deal with 10 binary variables only, making x_i and x_{i+1} dependent on the other 10 variables and (z_i, z_{i+1}).

Algorithm 6. The pre-computation after rewriting variables

Parameter: matrix P_1 of size $2^l \times V(n, d)$ with $P_1[i][j] \neq 0$ if the ISD j could generate the KSD i and 0 otherwise

1: Initialize the table T[`index`, `prefix`]
2: **for** each possible value of **x** **do**
3: **for** each ISD Δx s.t. $w_H(\Delta x) \leq d$ **do**
4: determine whether $f_{sr}(\mathbf{x}) =$ `prefix` and $f_{sr}(\mathbf{x} \oplus \Delta x) =$ `prefix` $\oplus$ `index`
5: **if** yes **then** P_1[`index`][Δx] = P_1[`index`][Δx] + 1
6: **for** each ISD Δx s.t. $w_H(\Delta x) \leq d$ **do**
7: set P_1[`index`][Δx]/$|\mathbf{x}|$ as the occurring rate of Δx
8: Sort the ISDs according to the occurring rates

There is an extra advantage of the above strategy. That is we could now exhaustively search the full input variable space when preparing the differential tables T[index, prefix] for the chosen attack parameters shown in Algorithm 6, which results in the accurately computed occurring probabilities compared to Algorithm 2 in Sect. 4.2, where $f_{sr}(\cdot)$ is the evaluation of the underlying stream cipher. The complete pre-computation table of Grain v1 is listed in Table 3 for $d = 3$, where $*$ indicates that the prefix could take any value from 0x0 to 0x3 due to the same distribution for different prefix values and number denotes the number of ISDs having the corresponding occurring probability.

Table 3. The full pre-computation information of Grain v1 after rewriting variables when $d = 3$

(index, prefix)	(0x0, $*$)					(0x1, $*$)			(0x2, $*$)			(0x3, $*$)		
prob.	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{3}{4}$	$\frac{1}{2}$	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{3}{16}$	$\frac{1}{8}$	$\frac{9}{16}$	$\frac{3}{8}$	$\frac{1}{4}$
number	1	44	69	54	8	3	22	27	63	8	27	8	54	63

From Table 3 and Definition 4, we have the following corollary on the diversified probabilities of different pre-computed tables.

Corollary 4. *For the pre-computation table of Grain v1 after rewriting variables, we have*

$$Pr_{divs} = \begin{cases} 0.269886, & \textit{if } \texttt{index=0x0} \\ 0.293333, & \textit{if } \texttt{index=0x1} \\ 0.293333, & \textit{if } \texttt{index=0x2} \\ 0.324000, & \textit{if } \texttt{index=0x3}. \end{cases}$$

Proof. From Definition 4, we have $Pr_{divs} = \frac{\sum_{\Delta x \in T} \Pr_{\Delta x}}{|T|}$, it suffices to substitute the variables with the values from Table 3 to get the results. □

From Corollary 4, we choose the KSD to be 0x0 in our attack, for in this case the reduction effect is maximized with the minimum $\Pr_{divs}$. Under this condition, we have run extensive experiments to determine the constant c for Grain v1, which is shown in the following table, where P_c is the probability that the correct value of the restricted internal state exists in the resultant list after one single invoking of

Table 4. The correspondence between the constant c and the existence probability for $index = 0x0$

c	5	6	7	8	9	10
P_c	0.757137	0.816551	0.860638	0.89502	0.92114	0.94644
c	11	12	13	14	15	16
P_c	0.95423	0.96573	0.97567	0.98021	0.985524	0.989411

Algorithm 3. Based on Table 4, we have run a number of numerical experiments to determine the appropriate configuration of attack parameters and found that $c = 10$ provides a balanced tradeoff between various complexities.

Precisely, under the condition that $c = 10$ and the $l = 2$-bit keystream vector with 12 variables (either consecutive or non-consecutive to construct the augmented function Af), we find that if $\beta = 21$ and $\gamma = 6$, then we get $\Pr_{\mathbf{x}_i} = 1 - (1 - 0.94644^{21})^6 = 0.896456$ from Theorem 5. We have tested this fact in experiments for 10^6 times, and found that the *average* value of the success rate well matches to the theoretical prediction. Besides, we have also found that under this parameter configuration, the number of candidates in the list V for the current restricted internal state $\mathbf{x}$ is $848 \approx 2^{9.73}$, which is also quite close to the theoretical value $2^{9.732}$ got from Theorem 3.

Corollary 5. *For Grain v1 when $c = 10$ and $l = 2$, the configuration that the resultant candidate list V is of size* 848 *with the average probability of* 0.896456 *for the correct restricted internal state being in V is non-random.*

Proof. Note that in the pure random case, the list V should have a size of $2^{10} \cdot 0.896456 = 917.971$ with the probability 0.896456; now in Grain v1, we get a list V of size 848 with the same probability. In the pure random case, we have

$$E[|V|] = \mu = 2^{10} \cdot 0.896456 = 917.971,\ \sigma = \sqrt{2^{10} \cdot \frac{1}{4} \cdot \frac{3}{4}} = 13.8564.$$

Further, $\frac{\mu - 848}{\sigma} = \frac{917.971 - 848}{13.8564} = 5.0497$; from Chebyshev's inequality, the configuration $(848, 0.896456)$ is far from random with the probability around 0.99. □

Now we are ready to describe the attack in details based on the above attack parameter configuration.

5.2 Concrete Attack: Strategy and Profile

First note that if we just run Algorithm 1 along a randomly known keystream segment to retrieve the overlapping restricted internal states one-by-one without considering the concrete internal structure of Grain v1, then we will probably meet the complexity overflow problem in the process when the restored internal state $\mathbf{x}_{\text{merge}}$ does not cover a large enough internal state, and at the same time, the number of candidates and the complexity needed to check these candidates will exceed the security bound already. Instead, we proceed as follows to have a more efficient attack. First observe that if we target the keystream vector $\mathbf{z} = (z_i, z_{i+1})$ through rewriting variables in Table 5 and restore the variables therein by our method, then for such a 2-bit keystream vector, we can obtain 8 LFSR variables involved in the h function and 2 NFSR bits n_{i+63}, n_{i+64}, together with 2 linear equations $x_i = \bigoplus_{k \in \mathcal{A}} n_{i+k}$ and $x_{i+1} = \bigoplus_{k \in \mathcal{A}} n_{i+k+1}$ on the NFSR variables. If we repeat this procedure for the time instants from 0 to 19, then from $z_i = x_i \oplus h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63})$, we will have l_{i+3+j}, l_{i+25+j}, l_{i+46+j}, l_{i+64+j} and n_{i+63+j} for $0 \le j \le 19$ involved in Table 5.

Table 5. The target keystream equations first exploited in our attack

	output	output
eqns.	$1: x_0 \oplus h(l_3, l_{25}, l_{46}, l_{64}, n_{63}) = z_0$	$2: x_1 \oplus h(l_4, l_{26}, l_{47}, l_{65}, n_{64}) = z_1$
	$3: x_2 \oplus h(l_5, l_{27}, l_{48}, l_{66}, n_{65}) = z_2$	$4: x_3 \oplus h(l_6, l_{28}, l_{49}, l_{67}, n_{66}) = z_3$
	$5: x_4 \oplus h(l_7, l_{29}, l_{50}, l_{68}, n_{67}) = z_4$	$6: x_5 \oplus h(l_8, l_{30}, l_{51}, l_{69}, n_{68}) = z_5$
	$7: x_6 \oplus h(l_9, l_{31}, l_{52}, l_{70}, n_{69}) = z_6$	$8: x_7 \oplus h(l_{10}, l_{32}, l_{53}, l_{71}, n_{70}) = z_7$
	$9: x_8 \oplus h(l_{11}, l_{33}, l_{54}, l_{72}, n_{71}) = z_8$	$10: x_9 \oplus h(l_{12}, l_{34}, l_{55}, l_{73}, n_{72}) = z_9$
	$11: x_{10} \oplus h(l_{13}, l_{35}, l_{56}, l_{74}, n_{73}) = z_{10}$	$12: x_{11} \oplus h(l_{14}, l_{36}, l_{57}, l_{75}, n_{74}) = z_{11}$
	$13: x_{12} \oplus h(l_{15}, l_{37}, l_{58}, l_{76}, n_{75}) = z_{12}$	$14: x_{13} \oplus h(l_{16}, l_{38}, l_{59}, l_{77}, n_{76}) = z_{13}$
	$15: x_{14} \oplus h(l_{17}, l_{39}, l_{60}, l_{78}, n_{77}) = z_{14}$	$16: x_{15} \oplus h(l_{18}, l_{40}, l_{61}, l_{79}, n_{78}) = z_{15}$
	$17: x_{16} \oplus h(l_{19}, l_{41}, l_{62}, l_{80}, n_{79}) = z_{16}$	$18: x_{17} \oplus h(l_{20}, l_{42}, l_{63}, l_{81}, n_{80}) = z_{17}$
	$19: x_{18} \oplus h(l_{21}, l_{43}, l_{64}, l_{82}, n_{81}) = z_{18}$	$20: x_{19} \oplus h(l_{22}, l_{44}, l_{65}, l_{83}, n_{82}) = z_{19}$

Let $\mathbf{x}^*$ be the restricted internal state consisting of the input variables involved in Table 5, the details of how to restore the restricted internal state $\mathbf{x}^*$ is presented in the following Tables 6 and 7. We first use FNCA to restore $\mathbf{x}^*$, then we know nearly 80 bits of the LFSR internal state with the corresponding positions, from which we can easily recover the initial internal state of the LFSR with a quite low complexity. Algorithm 7 presents the sketch of our online attack against Grain v1.

Algorithm 7. The online attack on the full Grain v1

```
1: Apply FNCA to x* to restore the input variables
2: for each candidate of x* do
3:     use the statistical test in Section 5.4 to check the candidate
4: for the passed ones do
5:     recover the remaining NFSR state, shown in Section 5.4
6: for each candidate of x_full do
7:     check the consistency with the available keystream
```

After knowing the LFSR part and more than half of the NFSR, we could first identify the correct LFSR state by the Walsh distinguisher, then the remaining NFSR state could easily be retrieved with an algebraic attack, both shown in the following Sect. 5.4. Note that in Tables 6 and 7, the list size for each merging operation is listed in the middle column, based on Theorem 4 and Corollary 3. For example, let us look at the 1st step. The reason that 2^5 is used instead of 2^6 in denominator is that the x_i variables are not freely generated random variables, for we have made them linearly dependent on the 5 variables in h function and the corresponding keystream bits to fulfill our criterion on the pre-computed tables. $2^{14.4558}$ is the list size when merging the two restricted internal states corresponding to (z_0, z_1) and (z_1, z_2), respectively. After merging (z_0, z_1) and (z_1, z_2), we get a list for the restricted internal state of the 3-bit keystream vector (z_0, z_1, z_2). Now we further invoke the self-contained method for the keystream vector (z_0, z_2), which consists of only z_0 and the non-consecutive z_2. Since there are now 10 free common variables between the restricted

internal state of (z_0, z_1, z_2) and that of (z_0, z_2), thus the denominator becomes 2^{10}. During this merging procedure, we use 3 keystream vectors (z_0, z_1), (z_1, z_2) and (z_0, z_2), and 3 times of the union result to form 3 independent candidate lists of size 848 with the probability 0.896456 that the corresponding correct partial state is indeed therein. Thus we have the probability 0.896456^3. The subsequent procedures in Tables 6 and 7 are deduced in a similar way as the above. The key point here is to the count the number of freely chosen variables between the corresponding internal state subsets, not including the linearly dependent variables. This process is repeated until merging the 20th equation in Table 5.

Table 6. The attack process for recovering $\mathbf{x}^*$ (1)

	$\mathbf{z}$	*List merging*	*Probability*
1.	(z_0, z_1)	$\frac{848 \cdot 848}{2^5} = 2^{14.4558}$	$0.896456^3 = 2^{-0.473086}$
	(z_1, z_2)		
	(z_0, z_2)	$\frac{848 \cdot 2^{14.4558}}{2^{10}} = 2^{14.1837}$	
2.	(z_1, z_2)	$\frac{2^{14.1837} \cdot 2^{14.1837}}{2^{10}} = 2^{18.3674}$	$0.896456^{2 \cdot 3+1} = 2^{-1.10387}$
	(z_2, z_3)		
	(z_1, z_3)	$\frac{2^{18.3674} \cdot 848}{2^{10}} = 2^{18.0953}$	
3.	$(z_0, \cdots, z_3)$	$\frac{2^{18.0953} \cdot 2^{18.0953}}{2^{15}} = 2^{21.1906}$	$0.896456^{2 \cdot 7+1} = 2^{-2.36543}$
	$(z_1, \cdots, z_4)$		
	(z_0, z_4)	$\frac{2^{21.1906} \cdot 848}{2^{10}} = 2^{20.9185}$	
4.	$(z_0, \cdots, z_4)$	$\frac{2^{20.9185} \cdot 2^{20.9185}}{2^{20}} = 2^{21.837}$	$0.896456^{2 \cdot 15+1} = 2^{-4.88855}$
	$(z_1, \cdots, z_5)$		
	(z_0, z_5)	$\frac{2^{21.837} \cdot 848}{2^{10}} = 2^{21.5649}$	
5.	$(z_0, \cdots, z_5)$	$\frac{2^{21.5649} \cdot 2^{21.5649}}{2^{25}} = 2^{18.1298}$	$0.896456^{2 \cdot 31+1} = 2^{-9.93481}$
	$(z_1, \cdots, z_6)$		
	(z_0, z_6)	$\frac{2^{18.1298} \cdot 848}{2^{10}} = 2^{17.8577}$	
6.	$(z_0, \cdots, z_6)$	$\frac{2^{17.8577} \cdot 2^{17.8577}}{2^{30}} = 2^{5.7154}$	$0.896456^{2 \cdot 63+1} = 2^{-20.0273}$
	$(z_1, \cdots, z_7)$		
	(z_0, z_7)	$\frac{2^{5.7154} \cdot 848}{2^{10}} = 2^{5.44332}$	
7.	$(z_0, \cdots, z_7)$	$\frac{2^{5.44332} \cdot 2^{21.5649}}{2^{25}} = 2^{2.00822}$	$0.896456^{127+31+1} = 2^{-25.0736}$
	$(z_3, \cdots, z_8)$		
	(z_0, z_8)	$\frac{2^{2.00822} \cdot 848}{2^{10}} = 2^{1.73614}$	
8.	$(z_0, \cdots, z_8)$	$\frac{2^{1.73614} \cdot 2^{21.5649}}{2^{25}} = 2^{-1.69896}$	$0.896456^{159+31+1} = 2^{-30.1198}$
	$(z_4, \cdots, z_9)$		
	(z_0, z_9)	$\frac{2^{-1.69896} \cdot 848}{2^{10}} = 2^{-1.97104}$	
9.	$(z_0, \cdots, z_9)$	$\frac{2^{-1.97104} \cdot 2^{21.5649}}{2^{25}} = 2^{-5.40614}$	$0.896456^{191+31+1} = 2^{-35.1661}$
	$(z_5, \cdots, z_{10})$		
	(z_0, z_{10})	$\frac{2^{-5.40614} \cdot 848}{2^{10}} = 2^{-5.67822}$	

Table 7. The attack process for recovering $\mathbf{x}^*$ (2)

	$\mathbf{z}$	*List merging*	*Probability*
10.	$(z_0, \cdots, z_{10})$ $(z_6, \cdots, z_{11})$ (z_0, z_{11})	$\frac{2^{-5.67262} \cdot 2^{21.5649}}{2^{25}} = 2^{-9.11332}$ $\frac{2^{-9.11332} \cdot 848}{2^{10}} = 2^{-9.3854}$	$0.896456^{223+31+1} = 2^{-40.2123}$
11.	$(z_0, \cdots, z_{11})$ $(z_8, \cdots, z_{12})$ (z_0, z_{12})	$\frac{2^{-9.3854} \cdot 2^{20.9185}}{2^{20}} = 2^{-8.4669}$ $\frac{2^{-8.4669} \cdot 848}{2^{10}} = 2^{-8.73898}$	$0.896456^{255+15+1} = 2^{-42.7354}$
12.	$(z_0, \cdots, z_{12})$ $(z_9, \cdots, z_{13})$ (z_0, z_2)	$\frac{2^{-8.73898} \cdot 2^{20.9185}}{2^{20}} = 2^{-7.82048}$ $\frac{2^{-7.82048} \cdot 848}{2^{10}} = 2^{-8.09256}$	$0.896456^{271+15+1} = 2^{-45.2586}$
13.	$(z_0, \cdots, z_{13})$ $(z_{10}, \cdots, z_{14})$ (z_0, z_{14})	$\frac{2^{-8.09256} \cdot 2^{20.9185}}{2^{20}} = 2^{-7.17406}$ $\frac{2^{-7.17406} \cdot 848}{2^{10}} = 2^{-7.44614}$	$0.896456^{287+15+1} = 2^{-47.7817}$
14.	$(z_0, \cdots, z_{14})$ $(z_{12}, \cdots, z_{15})$ (z_0, z_{15})	$\frac{2^{-7.44614} \cdot 2^{18.0953}}{2^{15}} = 2^{-4.35084}$ $\frac{2^{-4.35084} \cdot 848}{2^{10}} = 2^{-4.62292}$	$0.896456^{303+7+1} = 2^{-49.0432}$
15.	$(z_0, \cdots, z_{15})$ $(z_{13}, \cdots, z_{16})$ (z_0, z_{16})	$\frac{2^{-4.62292} \cdot 2^{18.0953}}{2^{15}} = 2^{-1.52762}$ $\frac{2^{-1.52762} \cdot 848}{2^{10}} = 2^{-1.7997}$	$0.896456^{311+7+1} = 2^{-50.3048}$
16.	$(z_0, \cdots, z_{16})$ $(z_{14}, \cdots, z_{17})$ (z_0, z_{17})	$\frac{2^{-1.7997} \cdot 2^{18.0953}}{2^{15}} = 2^{1.2956}$ $\frac{2^{1.2956} \cdot 848}{2^{10}} = 2^{1.02352}$	$0.896456^{319+7+1} = 2^{-51.5664}$
17.	$(z_0, \cdots, z_{17})$ $(z_{15}, \cdots, z_{18})$ (z_0, z_{18})	$\frac{2^{1.02352} \cdot 2^{18.0953}}{2^{15}} = 2^{4.11882}$ $\frac{2^{4.11882} \cdot 848}{2^{10}} = 2^{3.84674}$	$0.896456^{327+7+1} = 2^{-52.8279}$
18.	$(z_0, \cdots, z_{18})$ $(z_{16}, \cdots, z_{19})$ (z_0, z_{19})	$\frac{2^{3.84674} \cdot 2^{18.0953}}{2^{15}} = 2^{6.94204}$ $\frac{2^{6.94204} \cdot 848}{2^{10}} = 2^{6.66996}$	$0.896456^{335+7+1} = 2^{-54.0895}$

5.3 Restoring the Internal State of the LFSR

From Table 5, $\mathbf{x}^*$ involves 78 LFSR bits in total, it seems that we need to guess 2 more LFSR bits to have a linear system covering the 80 initial LFSR variables. To have an efficient attack, first note that both l_{64} and l_{65} are used 2 times in these equations, thus the candidate values should be consistent on l_{64} and l_{65}, which will provide a reduction factor of $\frac{1}{2^2} = \frac{1}{4}$ on the total number of candidates. Further, from $l_{83} = l_{65} \oplus l_{54} \oplus l_{41} \oplus l_{26} \oplus l_{16} \oplus l_3$, we have a third linear consistency check on the candidates. Hence, the number of candidates after going through Tables 6 and 7 is $\frac{1}{2^{-54.0895}} \cdot 2^{6.66996} \cdot 2^{-3} = 2^{57.7595}$. By guessing 2 more bits l_0, l_1, we can get l_{23}, l_{24} from the recursion $l_{80+j} = l_{62+j} \oplus l_{51+j} \oplus l_{38+j} \oplus l_{23+j} \oplus l_{13+j} \oplus l_j$ for $j = 0, 1$. In addition, we can derive l_2 from $l_{82} = l_{64} \oplus l_{53} \oplus l_{40} \oplus l_{25} \oplus l_{15} \oplus l_2$.

Note that the LFSR updates independently in the keystream generation phase and we also know the positions of the restored LFSR bits either from FNCA or from guessing, thus we could make a pre-computation to store the inverse of the corresponding linear systems with an off-line complexity of $\frac{80^{2.8}}{\Omega}$ cipher ticks and a online complexity of $\frac{80^2}{\Omega}$ to find the corresponding unique solution, where 2.8 is the exponent for Gauss reduction. This complexity is negligible compared to those of the other procedures. The total number of candidates for the LFSR part and the accompanying partial NFSR state, $2^2 \cdot 2^{57.7595} = 2^{59.7595}$, will dominate the complexity.

Remarks. Note that the gain in our attack mainly comes from the following two aspects. First, we exploit the first 20-bit keystream information in this procedure in a *probabilistic* way, not in a deterministic way, which is depicted later in Theorem 7. Now we target $78 + 20 + 20 = 118$ variables, not 160 variables, in a tradeoff-like manner. Here only 98 variables can be freely chosen. This cannot be interpreted in a straightforward information-theoretical way, which is usually evaluated in a deterministic way. Second, we use the pre-computed tables which also contain quite some information on the internal structure of Grain v1 in an *implicit* way in the attack.

5.4 Restoring the Internal State of the NFSR

After obtaining the candidate list for the LFSR part, the adversary could run the LFSR individually forwards and backwards to get all the necessary values and thus peel off the non-linearity of the h function. Now there are 2 choices in front of us, one is to efficiently restore the 80 NFSR variables with a low complexity that allows to be invoked many times, for there are probably many candidates of the restricted internal state $\mathbf{x}^*$ to be checked; the other is to check the correctness of the LFSR candidate first, then the NFSR could be restored afterwards independently. Fortunately, we find the latter way feasible in this scenario, which is shown below.

From the above step, the FNCA method has provided the adversary with the NFSR bits n_{63+i} and $x_i = n_{i+1} \oplus n_{i+2} \oplus n_{i+4} \oplus n_{i+10} \oplus n_{i+31} \oplus n_{i+43} \oplus n_{i+56}$ for $0 \leq i \leq 19$, i.e., now there are $20 + 20 = 40$-bit information available on the NFSR initial state. We proceed as follows to get more information.

Collecting More Linear Equations on NFSR. First note that if we go back 1 step, we get $x_{-1} = n_0 \oplus n_1 \oplus n_3 \oplus n_9 \oplus n_{30} \oplus n_{42} \oplus n_{55}$, i.e., we get 1 more linear equation for free. If we go back further, we could get a series of variables that can be expressed as the linear combination of the known values and the target initial NFSR state variables. On the other side, if we go forwards and take a look at the coefficient polynomial of x_4 in the h function, i.e., $1 \oplus x_3 \oplus x_0x_2 \oplus x_1x_2 \oplus x_2x_3$, we find it is a balanced Boolean function. Thus, the n_{82+i} variables have a probability of 0.5 to vanish in the resultant keystream bit and the adversary could directly collect a linear equation through the corresponding x_{20+i} variable at the beginning time instants from 20.

To get more linear equations on the NFSR initial state, we can use the following Z-technique, which is based on the index difference of the involving variables in the keystream bit. Precisely, if n_{82+i} appears at the z_{19+i} position, let us look at the end of the keystream equation z_{26+i} to see whether n_{82+i} exists there or not. If it is not there, then this will probably give us one more linear equation on the NFSR initial variables due to the index difference $56 - 43 = 13 > 7$; if it is there, we could just xor the two keystream equations to cancel out the n_{82+i} variable to get a linear equation on the NFSR initial variables. Then increase i by 1 and repeat the above process for the new i. Since the trace of the equations looks like the capital letter 'Z', we call this technique Z-technique. An illustrative example is provided in Appendix A.

It can be proved by induction that the Z-technique can also be used to express the newly generated NFSR variables as linear combinations of the keystream bits and of the initial state variables in forward direction. In backward direction, it is trivial to do the same task. We have run extensive experiments to see the average number of linear equations that the adversary could collect using the Z-technique, it turns out that the average number is 8, i.e., we could reduce the number of unknown variables in the initial NFSR state to around $80 - 40 - (8 - 3) = 35$, which facilitates the following linear distinguisher.

The Walsh Distinguisher. First note that the NFSR updating function in Grain v1 has a linear approximation with bias $\frac{41}{512}$, shown below.

$$n_{80+i} = n_{62+i} \oplus n_{60+i} \oplus n_{52+i} \oplus n_{45+i} \oplus n_{37+i} \oplus n_{28+i} \oplus n_{21+i} \oplus n_{14+i} \oplus n_i \oplus e,$$

where e is the binary noise variable satisfying $\Pr(e = 0) = \frac{1}{2} + \frac{41}{512}$. Since now there are only around 35 unknown variables left, we could collect a system of probabilistic linear equations on the left 35 NFSR variables by iteratively expressing the NFSR variables with indices larger than 80 by the corresponding linear combinations of keystream bits and the known information from the LFSR part and the partial NFSR state. If there are δ NFSR variables represented in this process, the complexity is just $35 \cdot \delta$. As a result, we have a system of the following form

$$\begin{cases} c_0^0 n_{i_0} \oplus c_1^0 n_{i_1} \oplus \cdots \oplus c_{34}^0 n_{i_{34}} = kz_0 \oplus e_0 \\ c_0^1 n_{i_0} \oplus c_1^1 n_{i_1} \oplus \cdots \oplus c_{34}^1 n_{i_{34}} = kz_1 \oplus e_1 \\ \quad \vdots \qquad\quad \vdots \qquad\qquad \vdots \\ c_0^{\delta-1} n_{i_0} \oplus c_1^{\delta-1} n_{i_1} \oplus \cdots \oplus c_{34}^{\delta-1} n_{i_{34}} = kz_{\delta-1} \oplus e_{\delta-1}, \end{cases} \tag{5}$$

where $c_j^i \in \mathbb{F}_2$ for $0 \le i \le \delta - 1$ and $0 \le j \le 34$ is the coefficient of the remaining NFSR variable n_{i_j} $(0 \le j \le 34)$, kz_i $(0 \le i \le \delta - 1)$ is the accumulated linear combination of the keystream bits and the known information from the LFSR part and the partial NFSR state derived before and e_i $(0 \le i \le \delta - 1)$ is the binary noise variable with the distribution $\Pr(e_i = 0) = \frac{1}{2} + \frac{41}{512}$.

To further reduce the number of unknown NFSR variables, we construct the parity checks of weight 2 from the above system as follows. First note that the bias of the parity checks is $2 \cdot (\frac{41}{512})^2 = 2^{-6.2849}$ from the Piling-up lemma in [14].

Second, this problem is equivalent to the LF2 reduction in LPN solving problems [12], which can be solved in a sort-and-merge manner with a complexity of at most δ using pre-computed small tables. We have tuned the attack parameters in this procedure and found that if $\delta = 2^{19}$ and $y = 15$, we could collect $\binom{2^{19-15}}{2} \cdot 2^{15} = 2^{21.9069}$ parity-checks on $35 - y = 20$ NFSR variables of the bias $2^{-6.2849}$. Note that we could further cancel out 4 more NFSR variables in these parity-checks by only selecting those equations that the corresponding coefficient of the assigned variable is 0, in this way we could easily get $\frac{2^{21.9069}}{2^4} = 2^{17.9069}$ parity-checks on $20 - 4 = 16$ NFSR variables. On the other side, from the unique solution distance in correlation attacks [6,13], we have

$$\frac{8 \cdot 16 \cdot \ln 2}{1 - h(p)} = 2^{17.5121} < 2^{17.9069},$$

where $p = \frac{1}{2} + 2^{-6.2849}$ and $h(p) = -p \cdot \log p - (1-p) \cdot \log(1-p)$ is the binary entropy function. Thus, we can have the success probability very close to 1 given $2^{17.9069}$ parity-checks to identify the correct value of the 16 NFSR variables under consideration. That is, we reach the following theorem.

Theorem 6. *If both the LFSR candidate and the partial NFSR state are correct, we can distinguish the correct value of the remaining 16 NFSR variables from the wrong ones with a success probability very close to* 1.

Proof. It suffices to note that if either the LFSR or the partial NFSR state is wrong, there exists no bias in the system (5), thus following the classical reasoning from correlation attacks in [6,13], we have the claim. □

Precisely, for each parity-check of weight 2 for the system (3), we have

$$(c_0^{j_1} \oplus c_0^{j_2}) n_{i_0} \oplus (c_1^{j_1} \oplus c_1^{j_2}) n_{i_1} \oplus \cdots \oplus (c_{34-y}^{j_1} \oplus c_{34-y}^{j_2}) n_{i_0} = \bigoplus_{t=1}^{2} kz_{j_t} \oplus \bigoplus_{t=1}^{2} e_{j_t}. \quad (6)$$

Let $(n'_{i_0}, n'_{i_1}, \cdots, n'_{i_{34-y}})$ be the guessed value of $(n_{i_0}, n_{i_1}, \cdots, n_{i_{34-y}})$, we rewrite Eq.(6) as follows.

$$\bigoplus_{t=1}^{2} kz_{j_t} \oplus \bigoplus_{t=1}^{34-y} (c_t^{j_1} \oplus c_t^{j_2}) n'_{i_t} = \bigoplus_{t=1}^{34-y} (c_t^{j_1} \oplus c_t^{j_2})(n'_{i_t} \oplus n_{i_t}) \oplus \bigoplus_{t=1}^{2} e_{j_t}. \quad (7)$$

From (7), let $\Delta(j_1, j_2) = \bigoplus_{t=1}^{34-y} (c_t^{j_1} \oplus c_t^{j_2})(n'_{i_t} \oplus n_{i_t}) \oplus \bigoplus_{t=1}^{2} e_{j_t}$, it is obvious if $(n'_{i_0}, n'_{i_1}, \cdots, n'_{i_{34-y}})$ coincides with the correct value, we get $\Delta(j_1, j_2) = \bigoplus_{t=1}^{2} e_{j_t}$; otherwise, we have $\Delta(j_1, j_2) = \bigoplus_{t: n'_{i_t} \oplus n_{i_t} = 1} (c_t^{j_1} \oplus c_t^{j_2}) \oplus \bigoplus_{t=1}^{2} e_{j_t}$. Since $c_t^{j_1} \oplus c_t^{j_2}$ is the xor of 2 independent uniformly distributed variables, we have $\Pr(c_t^{j_1} \oplus c_t^{j_2} = 0) = \frac{1}{2}$. Hence, when $(n'_{i_0}, n'_{i_1}, \cdots, n'_{i_{34-y}})$ is wrongly guessed, $\Delta(j_1, j_2)$ has the distribution $\Pr(\Delta(j_1, j_2) = 0) = \frac{1}{2}$, which is quite different from the correct case, i.e., $\Pr(\Delta(j_1, j_2) = 0) = \frac{1}{2} + 2^{-6.2849}$. For $2^{17.9069}$ such

parity-checks of the system (3), $\sum_{t=1}^{2^{17.9069}}(\Delta(j_1,j_2)\oplus 1)$ should follow the binomial distribution $(2^{17.9069}, \frac{1}{2}+2^{-6.2849})$ if $(n'_{i_0}, n'_{i_1}, \cdots, n'_{i_{34-y}})$ is correctly guessed; otherwise this sum should have the binomial distribution $(2^{17.9069}, \frac{1}{2})$. Now the situation is the same as that in binary correlation attacks. Thus, we can use the FWT technique to speed up the whole process as follows. Denote the set of the constructed parity-checks by P_t. First regroup the $2^{17.9069}$ parity-checks according to the pattern of $x = (c_0^{j_1}\oplus c_0^{j_2}, c_1^{j_1}\oplus c_1^{j_2}, \cdots, c_{34-y}^{j_1}\oplus c_{34-y}^{j_2})$ and define $f_{\mathrm{NFSR}}(x) = \sum_{(c_0^{j_1}\oplus c_0^{j_2}, c_1^{j_1}\oplus c_1^{j_2}, \cdots, c_{34-y}^{j_1}\oplus c_{34-y}^{j_2})}(-1)^{kz_{j_1}\oplus kz_{j_2}}$ for all the values of the coefficient vector appearing in the $2^{17.9069}$ parity-checks; if some value of $(c_0^{j_1}\oplus c_0^{j_2}, c_1^{j_1}\oplus c_1^{j_2}, \cdots, c_{34-y}^{j_1}\oplus c_{34-y}^{j_2})$ is not hit in these equations, just let $f_{NFSR}=0$ at that point. For this well-defined function f_{NFSR}, consider the Walsh transform $F(\omega) = \sum_{x\in\mathbb{F}_2^{35-y}} f_{\mathrm{NFSR}}(x)\cdot(-1)^{\omega\cdot x} = \sum_{P_t}(-1)^{kz_{j_1}\oplus kz_{j_2}\oplus\bigoplus_{t=0}^{34-y}\omega_t(c_t^{j_1}\oplus c_t^{j_2})} = F_0 - F_1$, where $\omega = (\omega_0,\omega_1,\cdots,\omega_{34-y}) \in \mathbb{F}_2^{35-y}$, F_0 and F_1 are the number of 0s and 1s, respectively. It is easy to see that if $\omega = (n_{i_0}, n_{i_1}, \cdots, n_{i_{34-y}})$, we have $\sum_{t=1}^{2^{17.9069}}(\Delta(j_1,j_2)\oplus 1) = \frac{F(\omega)+F_1}{2}$. If we set a threshold value T and accept only those guesses of x satisfying $\frac{F(\omega)+F_1}{2}\geq T$, then the probability that the correct value of $(c_0^{j_1}\oplus c_0^{j_2}, c_1^{j_1}\oplus c_1^{j_2}, \cdots, c_{34-y}^{j_1}\oplus c_{34-y}^{j_2})$ will pass the test is $P_1 = \sum_{t=T}^{2^{17.9069}}\binom{2^{17.9069}}{t}(\frac{1}{2}+2^{-6.2849})^t(\frac{1}{2}-2^{-6.2849})^{2^{17.9069}-t}$ and the probability that a wrong guess will be accepted is $P_2 = \sum_{t=T}^{2^{17.9069}}\binom{2^{17.9069}}{t}(\frac{1}{2})^{2^{17.9069}}$. Set $T = 2^{16.9306}$, we find that $P_1 = 0.999996$ and $P_2 \approx 2^{-53}$, i.e., the correct LFSR candidate and the correct partial NFSR state will pass almost certainly; while about $2^{59.7595}\cdot 2^{-53} = 2^{6.7595}$ wrong cases will survive in the above statistical test.

Hence, the time complexity of this Walsh distinguisher for one invoking is $\frac{2^{19}\cdot 35+2^{19}+2^{17.9069}+2^{16}\cdot 16}{\Omega} = \frac{2^{24.2656}}{\Omega}$ cipher ticks. Hence, by observing the Walsh spectrum of the function, the adversary could identify the correct LFSR and the correct partial NFSR states if they survived through the first step.

Restoring the Remaining NFSR State. After the Walsh distinguisher step, we could use an algebraic attack as that in [3] to restore the remaining NFSR state, which has a complexity much lower than the previous step. Precisely, the adversary could exploit the non-linear feedback function, say g, of the NFSR in Grain v1 to establish algebraic equations. Note that the algebraic degree of g in Grain v1 is 6 and the multiple $(n_{28}\oplus 1)(n_{60}\oplus 1)\cdot g$ has the algebraic degree 4, thus if the linearization method is adopted for solving the algebraic system, there are now $\sum_{i=0}^{4}\binom{35}{i} \approx 2^{15.8615}$ monomials in the system. If we take the same complexity metric as that in [7] in complexity estimate and taking into account that we have to repeat the solving routine for restoring the remaining NFSR state for each candidate survived through the above statistical test, the time complexity of this step is $T_{solving} = \frac{2^{6.7595}\cdot\frac{7\cdot(2^{15.8615})^{\log_2 7}}{64}}{\Omega} \doteq \frac{2^{48.0957}}{\Omega}$ cipher ticks. Finally, the overall time complexity of all the procedures in Sect. 5.4 is $2^{59.7595}\cdot\frac{2^{24.2656}}{\Omega}+\frac{2^{48.0957}}{\Omega}$ cipher ticks.

5.5 Final Complexity Analysis

Now we analyze the final complexity of the above attack against Grain v1. First note that quite some complexity analysis have already been involved in the above sections, here we just focus on the total complexity, which is stated in the following theorem.

Theorem 7. *Let T_{Alg5} and λ_{Alg5} be the time complexity and the number of invoking times of Algorithm 5, then the time complexity of our attack is*

$$T_{Alg6} + \xi \cdot \left(\frac{1}{Pr_{\mathbf{x}}^{\lambda_{Alg5}}} \cdot \left(T_{Alg5} \cdot \lambda_{Alg5} + \sum_{i=1}^{18} T_{merg}^{i}\right) + \frac{|L_{18}|}{Pr_{\mathbf{x}}^{\lambda_{Alg5}}} \cdot T_{walsh} + T_{solving} + T_{cst}\right)$$

cipher ticks, where T_{Alg6} is the pre-computation complexity of Algorithm 6, T_{merg}^{i} $(1 \le i \le 18)$ is the list merging complexity at step i in Tables 6 and 7, T_{walsh} is the complexity for the Z-technique and Walsh distinguisher in Sect. 5.4, $T_{solving}$ is the complexity for restoring the remaining NFSR state in Sect. 5.4, T_{cst} is the complexity of the final consistency examination and we repeat the online attack ξ times to ensure a high success probability. The memory complexity of our attack is at most $2^{2l} \cdot V(n,d) \cdot (\lceil log_2 n \rceil \cdot d + 14) + max_{1 \le i \le 18}|L_i|$ bits, where L_i $(1 \le i \le 18)$ are the lists generated during the process in Tables 6 and 7, and the data complexity is $2^{19} + 20 + 160 = 2^{19.0005}$ keystream bits.

Proof. Note that in our attack, to assure the existence of each correct restricted internal state in the corresponding candidate list, we have to assure its existence in the generated list when invoking Algorithm 5. This contributes to the factor $\frac{1}{\mathrm{Pr}_{\mathbf{x}}^{\lambda_{Alg5}}}$ since our attack is a dynamically growing process with the assumption that the probability $\mathrm{Pr}_{\mathbf{x}}$ is stable. $\mathrm{T}_{Alg5} \cdot \lambda_{Alg5}$ stands for the complexity of invoking Algorithm 5 during the evolution process in Tables 6 and 7 and $\sum_{i=1}^{18} T_{merg}^{i}$ comes from the sorting and merging complexities when merging the candidate lists for the steps from 1 to 18 in Tables 6 and 7. When the adversary goes out of all the steps in Tables 6 and 7, there exists only a resultant list L_{18} for each time, all the other lists generated in the intermediate process have been erased and overwritten. Thus, the total number of candidates for the LFSR part and partial NFSR state is $\frac{|L_{18}|}{\mathrm{Pr}_{\mathbf{x}}^{\lambda_{Alg5}}}$, and we have to use the Z-technique and Walsh distinguisher in Sect. 5.4 to check the correctness of these candidates. Since we could identify the correct LFSR candidate and the partial NFSR state with high probability independent of the remaining unknown NFSR state, the factor $T_{solving}$ is added in, not multiplied by. Finally, we have to find out the real correct internal state by the consistency test with the available keystream.

For the memory complexity, we invoke Algorithm 6 in the pre-processing phase with the l-bit KSD/keystream prefix, thus the factor 2^{2l} comes. That is, the adversary constructed 2^{2l} relatively small pre-computed tables, each of which consists of at most $V(n,d)$ items. Among all these tables, the adversary chooses one to be used in the online phase. It is worth noting that we could pre-compute the most inner routine of the self-contained method by storing all the possible

xors between all the ISDs and each value of the restricted internal state. Taking into account on the concrete storage data structure in Sect. 4.2 of each item, we have the first item in the expression. During the process illustrated in Tables 6 and 7, we only have to allocate a memory space that fits the largest memory consumption among all the intermediate lists L_i for $1 \leq i \leq 18$. By checking the list sizes in Tables 6 and 7 and the corresponding number of variables in Table 5, we have $2 \cdot 2^{21.5649} \cdot 42 < 2^{28}$. For different iterations, the same memory is reused.

For the data complexity, only the first 20 keystream bits are exploited when recovering the LFSR and the partial NFSR state. Note that $2^{15.8615} < 2^{19}$ and 2^{19} keystream bits are used by the Walsh distinguisher in Sect. 5.4. The last 160 bits are needed in consistency test for the surviving candidates. □

Since our attack is dynamically executed, the above formula can also depict the time consuming in the intermediate process. Here the dominated factors are the complexity cost by the invoking of Algorithm 5 and that of checking the surviving candidates. When $l = 2$, $n = 10$, $c = 10$ and $d = 3$ with the chosen pre-computed Table of size $|\mathrm{T}[0x0, \texttt{prefix}]| = 176$, the time complexity can be computed as

$$(343 \cdot \frac{2^{20.3436}}{\Omega} \cdot 2^{54.0895} + 2^{59.7595} \cdot \frac{2^{24.2656}}{\Omega} + \frac{2^{48.0957}}{\Omega} + 2^{54.0895} \cdot \frac{\sum_{i=1}^{18} T_{merg}^{i}}{\Omega} + T_{cst}) \cdot 2^2 \approx 2^{75.7}$$

cipher ticks, where $\frac{21 \cdot 6 \cdot 60 \cdot 176}{\Omega} = \frac{2^{20.3436}}{\Omega}$ is the complexity for one invoking of Algorithm 5 and $\frac{\sum_{i=1}^{18} T_{merg}^{i}}{\Omega}$ is the sorting and merging complexity in Tables 6 and 7. If we adopt the same $\Omega = 2^{10.4}$ as that in [18] and let $\xi = 2^2$ to stabilize a success rate higher than 0.9, we have the above result. The memory complexity is around $2 \cdot 2^{21.5649} \cdot 42 \approx 2^{28}$ bits. The pre-computation complexity is $\frac{2^{10} \cdot 176 \cdot 2 + 2^4 \cdot 176 \cdot 2}{\Omega} = \frac{2^{18.4818}}{\Omega} = 2^{8.1}$ cipher ticks.

Remarks. First note that the time complexity actually depends on the parameter Ω, which may be different for different implementations. Second, in the existence of a faster hardware implementation of Grain v1 that could generate 16 keystream bits in one cipher tick, our attack still holds, for such a hardware will also speed up the attack 16 times.

6 Experimental Results

6.1 The Experiments on Grain v1

Note that a large proportion of practical experiments are already presented in the previous sections, here we just provide the remaining simulations. We have run extensive experiments on Grain v1 to check the correctness of our attack. Since the total complexity is too large to be implemented on a single PC, we have verified the beginning steps of Tables 6 and 7 practically. Note that the

remaining steps in the evolution process are just the repetition process of the first ones, we have enough confidence that the whole process for recovering the inner state of the LFSR part is correct.

The profile of our experiments is as follows. We first generate the inner states of the LFSR and NFSR in Grain v1 randomly. Here the RC4 stream cipher is adopted by discarding the first 8192 output bytes as the main random source. Then we run the cipher forwards from this random state and generate the corresponding keystream. After that, we apply the FNCA to generate the possible states following the steps in Tables 6 and 7. At each step, we use the concrete data from simulations to verify the complexity and probability predictions in the Tables. We have done a large number of experiments to recover the restricted internal state corresponding to the keystream segment $(z_0, z_1, \cdots, z_6)$, and almost all the experimental results conform to our theoretical predictions in Table 6. For example, if the states of the LFSR and NFSR in Grain v1 are `0xB038f07C133370269B6C` and `0xC7F5B36FF85C13249603`, respectively, then the first 20-bit keystream are $(z_0, z_1, \cdots, z_{19})$ = `11100000111000000100`. Set $l = 2$, $d \leq 3$ and $c = 10$, let $\beta = 21$ and $\gamma = 6$, run Algorithm 5 to generate the union list of size 848. In 10^6 repetitions, the average probability of the correct restricted internal state being in the final list is quite close to the theoretical value 0.896456, which confirmed the correctness of the theoretical prediction. We also verified the list merging procedure of FNCA in experiments in the beginning steps 1 to 5 of Table 6, though the complexity of this procedure does not dominate. In general, the list sizes got in simulations match well with the theoretical estimates when represented in terms of the power of 2. Further, we have implemented the Walsh distinguisher to check its validity and got the confirmed results as well.

6.2 Simulations on the Reduced Version

For the reduced version of Grain v1 in Appendix B, we rewrite the keystream bit as $z'_i = x_i \oplus h(l'_{i+1}, l'_{i+21}, n'_{i+23})$, where $x_i = n'_{i+1} \oplus n'_{i+7} \oplus n'_{i+15}$. We have established a similar evolution process for restoring the restricted internal state of $(z_0, z_1, \cdots, z_{19})$, which consists of 40 LFSR state bits, 20 NFSR variables n_{i+23} and 20 x_i variables for $(0 \leq i \leq 19)$. In simulations, we first randomly loaded the internal states of the LFSR and NFSR as `0x9b97284782` and `0xb20027ea7d`, respectively. Then we got the first 20-bit keystream, `00010010010000 001100`. Let $l = 2$, $d \leq 2$ and $c = 8$, the probability that the correct inner state is in the candidate list is 0.923 after one call of Algorithm 3. Now we adopt a group of attack parameters similar to the case of Grain v1 in the distilling phase. We invoked the Algorithm 3 10 times to generate the corresponding 10 lists, and then intersect these lists to have a smaller list. Repeat this process 4 times to acquire 4 similar intersection lists. At last, combine these 4 lists to form the union list and the existence probability of the correct state in the list is around 0.91. For (z_0, z_1), we can recover an 8-bit restricted internal state, i.e., $(n_{22}, l_1, l_{21}, n_{23}, l_2, l_{22}, x_0, x_1)$. Note that there are 6 free variables in this case and 64 possibilities in total. After distilling, the candidate number of this partial

inner state is reduced to 50, while the value of expectation in theory is 53. Next, for (z_1, z_2), we got 51 candidates. Since the inner states of (z_0, z_1) and (z_1, z_2) have 3 common free variables, the expected number of inner states of (z_0, z_1, z_2) is $50 \cdot 51/2^3 \approx 319$. In the experiments, the practical number is 308. The same method can also be applied to (z_1, z_2, z_3) to recover the candidates list of size 312. There are 6 common free variables between the inner state of (z_0, z_1, z_2) and that of (z_1, z_2, z_3). We can reduce the number of inner states associated with (z_0, z_1, z_2, z_3) to $308 \cdot 319/2^6 \approx 2^{10.9}$, and the number got in experiments is $1921 \approx 2^{10.9}$. We continue this process to z_{19} until we have recovered the target inner state. In the experiments, we repeat the above whole process for $2^{24.26}$ times until the correct inner state is indeed in the candidate list of size $2^{12.01}$. In theory, we need about $2^{23.94}$ repetitions of the whole process and get a list with $2^{11.64}$ candidates. Therefore, on average we could restore the internal state with a complexity of about $2^{37.58}$ reduced version cipher ticks, and currently, it took several hours for our non-optimized C implementation to have the candidate list with the correct candidate in, which verified the theoretical analysis of FNCA. For the reduced version, there is no need to use the Z-technique and Walsh distinguisher to deal with the LFSR independently. The codes for the reduced version experiments are available via https://github.com/martinzhangbin/nca_reducedversion.

7 Conclusions

In this paper, we have tried to develop a new cryptanalytic method, called fast near collision attack, on modern stream ciphers with a large internal state. The new attack utilizes the basic, yet often ignored fact in the primitives that each keystream vector actually depends on only a subset of the internal state bits, not on the full internal state. Thus it is natural to combine the near collision property with the divide-and-conquer strategy to mount the new kind of state recovery attacks. In the process, a self-contained method is introduced and improved to derive the partial internal state from the partial state difference efficiently. After the recovery of certain subsets of the whole internal state, a careful merging and further retrieval step is conducted to restore the full large internal state. As an application of the new methodology, we demonstrated a key recovery attack against Grain v1, one of the 7 finalists in the European eSTREAM project. Combined with the rewriting variables technique, it is shown that the internal state of Grain v1, thus the secret key, can be reliably restored in $2^{75.7}$ cipher ticks after the pre-computation of $2^{8.1}$ cipher ticks, given 2^{28}-bit memory and around 2^{19} keystream bits in the single key model, which is the best key recovery attack against Grain v1 so far. It is suggested to strengthen Grain v1 with a new NFSR that eliminates the existence of good linear approximations for the feedback function. It is our future work to study fast near collision attacks against other NFSR-based primitives.

Acknowledgements. We would like to thank the anonymous reviewers for very helpful comments. This work is supported by the National Key R&D Research programm

(Grant No. 2017YFB0802504), the program of the National Natural Science Foundation of China (Grant No. 61572482), National Cryptography Development Fund (Grant No. MMJJ20170107) and National Grand Fundamental Research 973 Programs of China (Grant No. 2013CB338002).

A An Example to Illustrate the Z-technique

Example 1. Assume the adversary collects the following linear equations of z_{20+i} for $i \geq 0$. Now he could use the Z-technique as follows to derive more linear equations on the initial NFSR state.

$$
\begin{aligned}
1: &\ z_{20} = n_{21} \oplus n_{22} \oplus n_{24} \oplus n_{30} \oplus n_{51} \oplus n_{63} \oplus n_{76} \oplus n_{83} \\
2: &\ z_{21} = n_{22} \oplus n_{23} \oplus n_{25} \oplus n_{31} \oplus n_{52} \oplus n_{64} \oplus n_{77} \oplus n_{84} \\
3: &\ z_{22} = n_{23} \oplus n_{24} \oplus n_{26} \oplus n_{32} \oplus n_{53} \oplus n_{65} \oplus n_{78} \\
4: &\ z_{23} = n_{24} \oplus n_{25} \oplus n_{27} \oplus n_{33} \oplus n_{54} \oplus n_{66} \oplus n_{79} \\
5: &\ z_{24} = n_{25} \oplus n_{26} \oplus n_{28} \oplus n_{34} \oplus n_{55} \oplus n_{67} \oplus n_{80} \oplus n_{87} \\
6: &\ z_{25} = n_{26} \oplus n_{27} \oplus n_{29} \oplus n_{35} \oplus n_{56} \oplus n_{68} \oplus n_{81} \\
7: &\ z_{26} = n_{27} \oplus n_{28} \oplus n_{30} \oplus n_{36} \oplus n_{57} \oplus n_{69} \oplus n_{82} \\
8: &\ z_{27} = n_{28} \oplus n_{29} \oplus n_{31} \oplus n_{37} \oplus n_{58} \oplus n_{70} \oplus n_{83} \\
9: &\ z_{28} = n_{29} \oplus n_{30} \oplus n_{32} \oplus n_{38} \oplus n_{59} \oplus n_{71} \oplus n_{84} \oplus n_{91} \\
10: &\ z_{29} = n_{30} \oplus n_{31} \oplus n_{33} \oplus n_{39} \oplus n_{60} \oplus n_{72} \oplus n_{85} \oplus n_{92} \\
11: &\ z_{30} = n_{31} \oplus n_{32} \oplus n_{34} \oplus n_{40} \oplus n_{61} \oplus n_{73} \oplus n_{86} \oplus n_{93} \\
12: &\ z_{31} = n_{32} \oplus n_{33} \oplus n_{35} \oplus n_{41} \oplus n_{62} \oplus n_{74} \oplus n_{87} \oplus n_{94} \\
13: &\ z_{32} = n_{33} \oplus n_{34} \oplus n_{36} \oplus n_{42} \oplus n_{63} \oplus n_{75} \oplus n_{88} \oplus n_{95} \\
14: &\ z_{33} = n_{34} \oplus n_{35} \oplus n_{37} \oplus n_{43} \oplus n_{64} \oplus n_{76} \oplus n_{89} \\
15: &\ z_{34} = n_{35} \oplus n_{36} \oplus n_{38} \oplus n_{44} \oplus n_{65} \oplus n_{77} \oplus n_{90} \\
16: &\ z_{35} = n_{36} \oplus n_{37} \oplus n_{39} \oplus n_{45} \oplus n_{66} \oplus n_{78} \oplus n_{91} \\
17: &\ z_{36} = n_{37} \oplus n_{38} \oplus n_{40} \oplus n_{46} \oplus n_{67} \oplus n_{79} \oplus n_{92} \\
18: &\ z_{37} = n_{38} \oplus n_{39} \oplus n_{41} \oplus n_{47} \oplus n_{68} \oplus n_{80} \oplus n_{93} \oplus n_{100} \\
19: &\ z_{38} = n_{39} \oplus n_{40} \oplus n_{42} \oplus n_{48} \oplus n_{69} \oplus n_{81} \oplus n_{94} \oplus n_{101} \\
20: &\ z_{39} = n_{40} \oplus n_{41} \oplus n_{43} \oplus n_{49} \oplus n_{70} \oplus n_{82} \oplus n_{95} \oplus n_{102} \\
21: &\ z_{40} = n_{41} \oplus n_{42} \oplus n_{44} \oplus n_{50} \oplus n_{71} \oplus n_{83} \oplus n_{96} \\
22: &\ z_{41} = n_{42} \oplus n_{43} \oplus n_{45} \oplus n_{51} \oplus n_{72} \oplus n_{84} \oplus n_{97} \oplus n_{104} \\
23: &\ z_{42} = n_{43} \oplus n_{44} \oplus n_{46} \oplus n_{52} \oplus n_{73} \oplus n_{85} \oplus n_{98} \\
24: &\ z_{43} = n_{44} \oplus n_{45} \oplus n_{47} \oplus n_{53} \oplus n_{74} \oplus n_{86} \oplus n_{99} \\
25: &\ z_{44} = n_{45} \oplus n_{46} \oplus n_{48} \oplus n_{54} \oplus n_{75} \oplus n_{87} \oplus n_{100} \\
26: &\ z_{45} = n_{46} \oplus n_{47} \oplus n_{49} \oplus n_{55} \oplus n_{76} \oplus n_{88} \oplus n_{101} \\
27: &\ z_{46} = n_{47} \oplus n_{48} \oplus n_{50} \oplus n_{56} \oplus n_{77} \oplus n_{89} \oplus n_{102}
\end{aligned}
$$

From Eqs.1 $\rightharpoonup$ 8, $z_{20} \oplus z_{27}$ is a linear equation on the initial NFSR state. From Eqs.2 $\rightharpoonup$ 9 $\rightharpoonup$ 16, $z_{21} \oplus z_{28} \oplus z_{35}$ is also a linear equation on the initial NFSR state. Eqs.3 and 4 provide 2 more such equations directly. From Eqs.5 $\rightharpoonup$ 12 $\rightharpoonup$ 19 $\rightharpoonup$ 26 $\rightharpoonup$ 13 $\rightharpoonup$ 20 $\rightharpoonup$ 27 $\rightharpoonup$ 14, another linear equation on the initial NFSR state is established. Finally, as n_{80}, n_{81} and n_{82} are known from the previous step, Eqs.6 and 7 are also linear equations on the initial NFSR state. Thus, the adversary could collect 7 linear equations in forwards direction and 1 linear equation in backwards direction. In total, he could collect 8 linear equations on the initial NFSR state with a negligible complexity. □

B The Reduced Version of Grain v1

This reduced cipher generates the keystream from a 40-bit key and a 32-bit IV. Precisely, let $f'(x) = 1 + x^7 + x^{22} + x^{31} + x^{40}$ be the primitive polynomial of degree 40, then the updating function of the LFSR is defined as $l'_{i+40} = l'_{i+33} + l'_{i+18} + l'_{i+9} + l'_i$ for $i \geq 0$. Similar to the original Grain v1, the updating function of the NFSR is

$$\begin{aligned} n'_{i+40} = \; & l'_i \oplus n'_{i+33} \oplus n'_{i+29} \oplus n'_{i+23} \oplus n'_{i+17} \oplus n'_{i+11} \oplus n'_{i+9} \oplus n'_{i+33}n'_{i+29} \\ & \oplus n'_{i+23}n'_{i+17} \oplus n'_{i+33}n'_{i+9} \oplus n'_{i+33}n'_{i+29}n'_{i+23} \oplus n'_{i+29}n'_{i+23}n'_{i+17} \\ & \oplus n'_{i+33}n'_{i+29}n'_{i+23}n'_{i+17} \oplus n'_{i+29}n'_{i+23}n'_{i+17}n'_{i+11}n'_{i+9}. \end{aligned}$$

The filter function $h'(x)$ is defined as $h'(x) = x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_0x_1x_2$ with the different tap positions, which are provided below. The output function is $z'_i = \sum_{k \in \mathcal{A}'} n'_{i+k} \oplus h'(l'_{i+1}, l'_{i+21}, n'_{i+22})$, where $\mathcal{A}' = \{1, 7, 15\}$. Its key/IV initialization is similar to that of Grain v1 with 80 initialization rounds. The actual complexity of the brute force attack for a fixed IV on the reduced version is $(2^{40} - 1) \cdot (80 + \sum_{i=1}^{40} i \cdot \frac{1}{2^{i-1}}) \approx 2^{46}$ cipher ticks.

References

1. Anderson, R.: Searching for the optimum correlation attack. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 137–143. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60590-8_11
2. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of grain. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 15–29. Springer, Heidelberg (2006). https://doi.org/10.1007/11799313_2
3. Berbain, C., Gilbert, H., Joux, A.: Algebraic and correlation attacks against linearly filtered non linear feedback shift registers. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 184–198. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_12
4. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_1
5. De Cannière, C., Küçük, Ö., Preneel, B.: Analysis of grain's initialization algorithm. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 276–289. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68164-9_19

6. Chepyzhov, V.V., Johansson, T., Smeets, B.: A simple algorithm for fast correlation attacks on stream ciphers. In: Goos, G., Hartmanis, J., van Leeuwen, J., Schneier, B. (eds.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44706-7_13
7. Courtois, N.T., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_21
8. http://www.ecrypt.eu.org/stream/e2-grain.html
9. Hawkes, P., Rose, G.G.: Rewriting variables: the complexity of fast algebraic attacks on stream ciphers. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 390–406. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_24
10. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. Int. J. Wirel. Mob. Comput. (IJWMC) **2**(1), 86–93 (2007)
11. Fischer, S., Meier, W.: Algebraic immunity of S-boxes and augmented functions. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 366–381. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74619-5_23
12. Guo, Q., Johansson, T., Löndahl, C.: Solving LPN using covering codes. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 1–20. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45611-8_1
13. Lu, Y., Vaudenay, S.: Faster correlation attack on bluetooth keystream generator E0. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 407–425. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28628-8_25
14. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48285-7_33
15. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional differential cryptanalysis of NLFSR-based cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_8
16. Koch, P.C.: Cryptanalysis of stream ciphers-analysis and application of the near collision attack for stream ciphers, Technical University of Denmark, Master Thesis-Supervisor: Christian Rechberger, November 2013
17. http://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kind
18. Zhang, B., Li, Z., Feng, D., Lin, D.: Near collision attack on the grain v1 stream cipher. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 518–538. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_27