IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 30, 2019, accepted December 4, 2019, date of publication December 13, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2959334

Optimizing Fast Near Collision Attack on Grain Using Linear Programming

SENSHAN PAN, YUEPING WU, AND LIANGMIN WANG, (Member, IEEE)
School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China

Corresponding author: Yueping Wu (wuyp1996@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61702230, Grant U1736216, Grant 61472001, Grant 61802154, and Grant 61902156, in part by the National Key Research and Development Program of China under Grant 2017YFB1400703, and in part by the Key Research and Development Program of Jiangsu Province under Grant BE2015136.

ABSTRACT In 2018, an attack named fast-near-collision attack (FNCA) was proposed, which is an improved version of near-collision attack (NCA) on Grain-v1, one of the three hardware-oriented finalists of the eSTREAM project. FNCA is designed as a key recovery attack and takes a divide-and-conquer strategy that needs a merging phase. We propose an improved FNCA where the merging phase is optimized by a linear programming based strategy. It decreases the candidates of the internal state vectors (ISVs) in each step of merging and has a reduction in the overall time complexity. Since the merging phase is vital for a divide-and-conquer strategy, where the most of bits of the full internal state are recovered, other analyses on Grain family with FNCA can get optimized by our method in varying degrees. This paper offers an experiment on a reduced Grain and a theoretical analysis on Grain-v1 to confirm the results. In the case of the reduced Grain of an 80-bit internal state, the time complexity is $2^{37.1096}$, which has a 27.8% reduction. For Grain-v1, its theoretical time complexity is around $2^{73.4}$, which is reduced by 79.4% compared with the original one.

INDEX TERMS Cryptanalysis, grain, near collision, stream ciphers, time-memory-data tradeoff attacks.

I. INTRODUCTION

The Grain-v1 stream cipher is proposed by Hell *et al.* [9]. It is selected as a finalist in the eSTREAM project after withstanding the cryptanalysis [2]. Grain is designed to satisfy the needs of constrained hardware environments, like the implementation of an RFID tag, where the amount of memory and power is limited. Consisting of two combined shift registers, which are simple but ingenious, one with linear feedback and the other with non-linear feedback, Grain-v1 generates secure keystream at a rapid speed.

A. RELATED WORKS

Based on the birthday paradox, [1] proposes that the design of stream cipher requires the internal state size to be twice as large as the key size. As a general rule, many time-memory-data attacks [3], [4], [8] take this theory into account. This idea prevents many stream ciphers, e.g., Grain family from a traditional collision attack. However, the proposal of NCA [17] makes the Grain-like stream cipher exposed in unsafe. Aiming at the drawbacks of pre-computation complexity and success ratio in NCA, an improved version of NCA named FNCA [16] is raised, which realizes a great fall of time complexity in pre-computation phase, along with an overall lower time complexity and a higher success ratio.

The main difference lies in FNCA and NCA is whether to recover the full internal state in one time. NCA returns full internal states after once implementation. However, FNCA recovers a set of internal state vectors (ISVs) that is composed of several bits of a full internal state and then merges the ISVs into full internal states. The bits of ISVs are chosen by the corresponding tap positions of the output function. Thus a well-designed output function can withstand a pure FNCA. Only a part of the full internal state can be recovered by an output function based collision attack, which is researched in [4]. To recover the rest of the internal state bits, other techniques like algebraic attack, correlation attack should be implemented. [16] finds that Grain-v1 cannot be attacked successfully by a pure FNCA, only LFSR state bits and a few state bits of NFSR can be recovered.

In FNCA, the merging phase recovers a skeleton of the full internal state, remaining nearly half of the full internal state bits to be recovered. In order to recover the rest bits, extra cryptanalysis will be performed on each candidate of the

The associate editor coordinating the review of this manuscript and approving it for publication was Luis Javier Garcia Villalba.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

TABLE 1. Comparison with the previous attack.

Attack	Complexities			
	Pre-comp	Data	Memory	Time
FNCA for Grain-v1 [16]	$2^{8.1}$	2^{19}	2^{28}	$2^{75.7}$
Optimized FNCA for Grain-v1	$2^{8.1}$	2^{19}	2^{28}	$2^{73.4}$
FNCA for reduced Grain [16]	$2^{2.61}$	$2^{4.3}$	2^{18}	$2^{37.58}$
Optimized FNCA for reduced Grain	$2^{2.61}$	$2^{4.3}$	2^{18}	$2^{37.11}$

The unit of data/memory complexity is 1 bit. In order to compare our method to former ones, here we use 1 cipher tick in [16] and [17] as the time complexity unit, where a tick of Grain is $\Omega = 2^{10.4}$ CPU cycles. In Section V and VI we analyze the time complexities of the improved attacks.

full internal state. Therefore the number of candidates of the internal states after merging dominates the time complexity of the attack. At the same time, the time complexity of the FNCA [16] on Grain-v1 will be higher when it comes to software implementation where fast correlation attack [12] will have a better performance than FNCA under the background of Grain-v1, in some situation.

B. CONTRIBUTIONS

This paper proposes an optimized FNCA by using linear programming to adjust the merging strategy. We track the candidates' increased times and the probability of the right candidate in each step, to get the final number of the candidates of the full-size ISV. Then we analyze the candidates increased times in each case of overlapping bits of the two keystream segments and construct a linear programming model for the whole merging phase. In this paper, we give a strategy to reduce the invalid candidates for the internal state further. Thus the final candidates (the candidates of the full internal state) decrease around 27.8% compared with the original attack on a reduced Grain of 80-bit internal state. In the case of Grain-v1, there is a theoretical optimization where the final candidates get a reduction of 79.4%.

As a method focusing on the merging, linear programming does not affect the data and pre-computation complexity. It is found that the memory usage of candidates of the ISVs in some merging step is unaffordable on a single PC when the attack is implemented on Grain-v1 or higher versions [16]. The solution is having the memory of the generated candidates freed after the next batch of the candidates is created in each merging step. Therefore the reduction effect of the number of final candidates is dispersed into each step of merging, the memory complexity between the original FNCA and our method is close.

This paper is organized as follows. The description of Grain-v1 and reduced Grain are introduced in Section II. In Section III, some preliminaries are presented, which includes the concept of the time-memory tradeoff in NCA. Section IV provides a description of the procedure of FNCA in [16] and illustrates our improved strategy of merging ISVs. The experiment results and the specific merging procedure

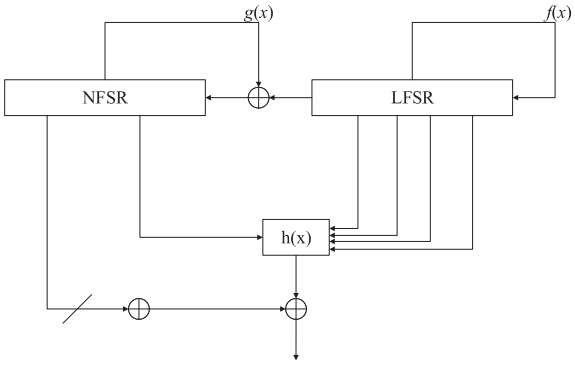

FIGURE 1. Structure of Grain-v1.

is given in Section V. In Section VI, we make a theoretical analysis of the optimization effect of our method on Grain-v1. At last, Section VII concludes this paper.

II. GRAIN-V1 AND REDUCED GRAIN

Grain-like stream ciphers are made of two connected shift registers and an output function. One is a non-linear feedback shift register (NFSR) which uses a non-linear polynomial function as its update function. Another one using a linear polynomial function as its update function is called linear feedback shift register (LFSR). To describe the internal states of the shift registers, we denote NFSR and LFSR internal states as $(n_i, n_{i+1}, \ldots, n_{i+s-1})$ and $(l_i, l_{i+1}, \ldots, l_{i+s-1})$ respectively. s represents the size of a shift register, e.g., $s = 40$ in Grain-v1, $s = 80$ in Grain-v1 and $s = 128$ in Grain-128. It is noticed that n_i (or l_i) is the most significant bit of the hexadecimal string stored in the NFSR (or LFSR).

In the case of Grain-v1 [9], the update function of LFSR is defined as

$$l_{i+s} = l_{i+62} + l_{i+51} + l_{i+38} + l_{i+23} + l_{i+13} + l_i.$$

The update function of NFSR is defined as

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

As a bit-oriented stream cipher, Grain denotes its keystream according to the order of clock t: $(z_0, z_1, \ldots, z_t)$, where z_0 means the first keystream bit generated after initialization. The output function contains a filter function $h(x)$ and a polynomial for masking. The filter function is defined as

$$\begin{aligned} h(x) = {} & x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 \\ & + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4. \end{aligned}$$

The variables x_0, x_1, x_2, x_3, x_4 represent the tap positions, and the internal state bits that show on these positions are called tap bits. These variables are defined as $l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63}$ respectively in the case of Grain-v1. FNCA is an attack based on the output function, which means the attack should be adjusted according to the specific output function and a well-designed combination of tap positions can effectively prevent it from a pure FNCA. It is found that Grain-v1 can withstand a pure FNCA [16], where around half of NFSR state bits remain unknown unless further efforts are made by other techniques, e.g., algebraic attack, Walsh distinguisher.

Since the outputs of filter function balanced [9], the correlation of the output of filter function to sums of inputs can be reduced, which prevents some attacks e.g., correlation attack, distinguish attack. The masking bits are defined as $\sum_{k\in\mathcal{A}} n_{i+k}$, where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$. Hence the complete output function is

$$z_i = \sum_{k\in\mathcal{A}} n_{i+k} + h(l_{i+3}, l_{i+25}, l_{i+46}, l_{i+64}, n_{i+63}).$$

In this paper, we mainly talk about the improvement of pure FNCA, which is supposed to be implemented in every FNCA on Grain-like ciphers. For the above purpose, we take a reduced Grain whose bit length of the full internal state is 80 as the target in our method. The reduced Grain can be recovered totally by guessing two internal state bits after a pure FNCA, according to Theorem IV-B.

In the case of the reduced Grain, the update function of LFSR is defined as

$$l_{i+s} = l_{i+33} + l_{i+18} + l_{i+9} + l_i.$$

The update function of NFSR is defined as

$$\begin{aligned} n_{i+s} = {} & l_i + n_{i+33} + n_{i+29} + n_{i+23} + n_{i+17} \\ & + n_{i+11} + n_{i+9} + n_{i+33}n_{i+29} + n_{i+23}n_{i+17} \\ & + n_{i+33}n_{i+9} + n_{i+33}n_{i+29}n_{i+23} \\ & + n_{i+29}n_{i+23}n_{i+17} + n_{i+33}n_{i+29}n_{i+23}n_{i+17} \\ & + n_{i+29}n_{i+23}n_{i+17}n_{i+11}n_{i+9}. \end{aligned}$$

The output function is defined as

$$z_i = \sum_{k\in\mathcal{A}} n_{i+k} + h(l_{i+1}, l_{i+21}, n_{i+22}),$$

where $\mathcal{A} = \{1, 7, 15\}$ and the filter function $h(\cdot)$ is defined as $h(x) = x_1 + x_0x_2 + x_1x_2 + x_0x_1x_2$.

III. PRELIMINARIES

This section consists of three parts, which give a general background of the attack. A list of notations is presented in the first part. The second part introduces the birthday paradox and time-memory-data tradeoff. The last part explains the near-collision theory and its combination with time/memory tradeoff cryptanalysis.

A. NOTATIONS

- n: the bit length of the internal state.
- $\mathbb{F}$: binary field.
- $\mathbb{F}_2^n$: n-bit internal state over $\mathbb{F}_2$.
- Δx: the internal state difference (ISD).
- Δz: the keystream segment difference (KSD).
- $w_H(\cdot)$: the Hamming weight of input (a binary string), whose output equals to the amount of ''1''s in the input.
- d: the Hamming weight of internal state difference (ISD).
- A, B: two sets of elements drawn uniformly and independently at random from $\mathbb{F}_2^n$.
- a, b: two random elements taken form A, B respectively, where $a \in A, b \in B$.
- $|\cdot|$: the cardinality of a set.

B. BIRTHDAY PARADOX AND TIME-MEMORY-DATA TRADEOFF

Birthday paradox gives a conclusion of finding a collision for a hash function. In [10], an equation of collision ratio (p), space of key (N) and the size of pre-computed table (s) is presented:

$$p = 1 - e^{-s^2/(2N)}. \tag{1}$$

It is found that when p is fixed at 0.5, the value of s is close to $\sqrt{N}$. This is why the size of a key needs to be doubled up to an internal state before keystream bits generation. Similarly, the birthday paradox can be considered in a set theory way. Given two sets A, B, a collision occurs when a pair (a, b) satisfies $a = b$.

A set theory version of birthday problem is introduced in [13], where illustrates that birthday problem finds a solution (a, b) exists with reasonable probability once $|A| \cdot |B| > 2^n$ holds, and if the list sizes are favorably chosen, the complexity of the optimal algorithm is $\Theta(2^{n/2})$.

Before a collision attack starts, it needs to have a mapping table (pre-computed table), which includes randomly generated keys (k_i) and their hash values (v_i). Let A stores v_i and B stores the target hash value (v^*). As the preimages of v_i are already known, once there occurs that $v_i = v^*$, the key for v^* will turn out to be k_i. In the above case, there is $|B| = 1$ holds for each hash value is independent. Nevertheless, when it comes to stream ciphers, the internal state is always correlated to the last clock's internal state. Reference [3] proposes to collect the real-time data of D bits produced by the generator, then all the later keystream can figure out as long as there is a collision between A and the D bits keystream for the attacker can run the generator forwards any steps he wants. Thus it can be more possible to crack a stream cipher by providing more keystream bits. In other words, there are more alternative targets for the same pre-computed table. According to the inequality $|A| \cdot |B| > 2^n$, by adding the elements in B, the cardinality of A can be decreased to reduce the space that the pre-computed table occupies.

Time-memory-data tradeoff attack is a kind of cryptanalysis recovering the internal state (or keys) by making tradeoff

among the overhead of time (T), memory (M) and real-time data (D). The simplest stream cipher based time-memory tradeoff attack is **BG attack**, which is raised by Babbage [1] and Golic [7] independently. **BG attack** gives a time/memory tradeoff curve $T \cdot M = N$ for any $1 \leq T \leq D$, where $T = D$, N denotes the space of internal state (or keys), and the overhead of pre-computation equals to M. When $T = M$ holds, the tradeoff curve will be transformed into $T = M = N^{1/2}$, which supports the birthday paradox theory.

C. TIME-MEMORY-DATA TRADEOFF IN NEAR COLLISION

When (a, b) satisfies $a = b$, a "complete collision" happens. We call it a "near collision" when $a+b \neq 0$. Let $\Delta x = a+b$ denotes ISD between a and b, it can be found that the degree of near collision of Δx equals to the hamming weight of Δx, which means $d = w_H(\Delta x)$.

Definition 1 [17]: For any a and b, if $w_H(a+b) \leq d$ holds, A will have d-near-collision with B.

For example, binary string 1011 has 1-near-collision with 0011, 1111, 1001, 1010, and 1011. According to the previous example, it can be found that, given any internal state x, we can generate all possible internal states that are d-near-collision with x, by adjusting the ISD and XORing ISD to x. All x's variants will be figured out if we enumerate all ISDs satisfying $w_H(\Delta x) \leq d$.

According to Definition III-C, it takes $(d + 1)$ stages to enumerate all elements in D. In brief, in stage i, we flip i bits of a binary string whose bit length is n and value equals to 0, hence there exists $\binom{i}{n}$ ways to flip. After adding up all ways of flipping from stage 0 to stage d we can compute the number of the ISDs will be used in a d-near-collision. Given a fixed d, let set D includes all possible Δx that satisfies $d \geq w_H(\Delta x)$, then $|D| = \sum_{i=0}^{d} \binom{i}{n}$.

Therefore D can be denoted as $\{\Delta x'_0, \Delta x'_1, \ldots, \Delta x'_{|D|-1}\}$. For each internal state stored in the pre-computed table, it has $|D|$ different states that get near collisions with it. In this way, we can infer $|D|$ times internal states, which means the pre-computed table covers more internal states without increasing its physical memory.

Theorem 1 [16]: Let D be a set that contains the chosen d-near-collision ISDs of A. Then there will be a pair (a, b) that satisfies d-near-collision if near-collision inequality

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

holds. Note that c is a constant to raise the possibility of the existence of the pair (a, b).

Both NCA [17] and FNCA [16] substitute near collision to complete collision, but the collision objects of these two attacks are different. NCA in [17] recovers a full internal state by once collision, while the original FNCA in [16] recovers a part of a full internal state per collision. It takes a few times of collisions before a full internal state is recovered by fast near collision. In the next section, we analyze the original FNCA based on reduced Grain, and then the cut-in point of an improved attack method is put forth.

IV. OPTIMIZED FNCA ON REDUCED GRAIN

FNCA contains two phases, an offline phase and an online phase. The offline phase describes the pre-computation algorithm and gives the pre-computed table. The online phase introduces internal state bits on tap positions, or tap bits in brief, recovery algorithm, and the strategy of merging the tap bits, where the linear programming optimization will be covered.

A. OFFLINE PHASE: PRE-COMPUTATION

Pre-computation aims to find the link between ISD and keystream segment difference (KSD). As we know the output function of the reduced grain is $z_i = \sum_{k\in\mathcal{A}} n_{i+k} + h(l_{i+1}, l_{i+21}, n_{i+22})$, where $\mathcal{A} = \{1, 7, 15\}$. Let $m_i = \sum_{k\in\mathcal{A}} n_{i+k}$, and l denotes the bit length of keystream segment. If the bit-length of a keystream segment is 2, we have a keystream segment vector $\vec{z}_i = \binom{z_i}{z_{i+1}} = (z_i, z_{i+1})^T$, where $z_i = m_i + h(l_{i+1}, l_{i+21}, n_{i+22})$ and $z_{i+1} = m_{i+1} + h(l_{i+2}, l_{i+22}, n_{i+23})$. Referring to the concept of **BSW sampling** [3], the keystream bit z_i is directly related to the internal state bits $(l_{i+1}, l_{i+21}, n_{i+22})$, showing in the output function as taps. The masking bits bit m_i can be solved after recovering the tap bits, then by solving the equations of $m_i = n_{i+1} + n_{i+7} + n_{i+15}$, where $0 \leq i < L$ and L denotes the bit length of real-time keystream, most internal state bits involved in the masking bits can be solved. We set $\vec{x}_i = (l_{i+1}, l_{i+2}, l_{i+21}, l_{i+22}, n_{i+22}, n_{i+23})^T$ as an initial ISV to represent a vector composed of tap bits, except the bits involved in the masking bits. By this mean, each keystream bit is bound to 4 variables, and for each keystream segment vector there is

$$\vec{z}_i = (m_i, m_{i+1})^T + h(\vec{x}_i). \tag{3}$$

In NCA, each keystream bit is bound to 80 variables, which results in more overheads. The general idea of FNCA is generating a set of random tap internal states vectors, then for $\vec{x}'_i$, we adjust a few bits of it, by XORing it to Δx, to hit a $\vec{z}'_i$ near the target $\vec{z}_i$. Let $\vec{z}'_i = \vec{z}_i + \Delta z$, then the correlation between ISD and KSD is established:

$$\begin{cases} (m_i, m_{i+1})^T = \vec{z}_i + h(\vec{x}_i) \\ \vec{x}'_i = \vec{x}_i + \Delta x \\ \vec{z}'_i = \vec{z}_i + \Delta z \end{cases} \Rightarrow \Delta z = h(\vec{x}_i) + h(\vec{x}_i + \Delta x) \tag{4}$$

Pre-computation discusses the correlation between ISD and KSD, by enumerating every element in $\mathbb{F}_2^6$, for each 2-bit-length keystream segment. This process is presented in Algorithm 1, which is modified from [16]. From [16] and TABLE 6, it is discovered that the ISDs in the table $T[\vec{z}_i, \Delta z_j]$ are not distributed uniformly. Hence it needed to figure out the probability of a specific ISD shows up in each $T[\vec{z}_i, \Delta z_j]$. Table $P[\vec{z}_i, \Delta z_j, \Delta x_k]$ expresses the occurring rate of each ISD existing in each $T[\vec{z}_i, \Delta z_j]$, and also the mathematical expectation of the amount of ISDs, after eliminating the duplicate ISDs in each $T[\vec{z}_i, \Delta z_j]$, can be figured out. Let $u = \vec{z}_i, v = \Delta z_j, w = \Delta x_k$, then T_{uv} and P_{uvw} denotes $T[\vec{z}_i, \Delta z_j]$ and $P[\vec{z}_i, \Delta z_j, \Delta x_k]$.

Algorithm 1 Pre-Computation

```
Initialize the pre-computed table T[z_i, Δz_j]
for x = 0 to |F_2^6| − 1 do
    for each Δx s.t. w_H(Δx) ≤ d do
        determine whether z_i, Δz_j, x, Δx satisfy Eq.(4)
        if yes then
            store the Δx in T_uv
        end if
    end for
end for
Initialize ISD ratio table P_uvw
for each Δx in T_uv do
    P_uvw = P_uvw + 1
end for
drop the Δx if P_uvw = 0
for each Δx in P_uvw do
    P_uvw = P_uvw/|F_2^6|
end for
set P_uvw as the occurring rate of Δx
```

After pre-computation, it is the online phase where the FNCA is mounted on a determined real-time keystream. The real-time keystream is divided into segments ($\vec{z}_i$) and the corresponding internal states ($\vec{x}_i$) will be recovered on the basis of table $T[\vec{z}_i, \Delta z_j]$ and $P[\vec{z}_i, \Delta z_j, \Delta x_k]$.

Depending on the results of the table P_{uvw}, the average probability for a specific T_{uv} that satisfies Eq.(4) can be computed, where [16] names it as diversified probability. Let $|T_{uv}|$ denotes the number of ISDs in T_{uv}, the diversified probability of this table T_{uv} is defined as $P_{divs} = \frac{\sum_{\Delta x \in T} P_{uvw}}{|T_{uv}|}$, where Δx traverses the whole table T. Diversified probability measures the reducing effect of a table T, where a smaller P_{divs} means a smaller $|T_{uv}|$.

B. ONLINE PHASE: TAP BITS RECOVERY

As it is known that only six internal state bits can be recovered from a single keystream segment, we need to figure out how many keystream segments it takes to recover a full internal state. Both [16] and [4] give a solution to estimate the minimum required keystream bits to make a time-memory-data attack on a Grain-like cipher.

After constructing a table of keystream bits from clock 0 to clock 19, it can be found that almost every bit of a full internal state is covered in the table. According to TABLE 2, nearly the whole LFSR state bits are covered, except l_0. However, it can be derived from LFSR's update function that $l_{40} = l_{33} + l_{18} + l_9 + l_0$. At the same time, around half of NFSR state bits $n_{22} \sim n_{41}$ are covered, and the rest of NFSR state bits obey the equations

$$\begin{cases} m_i = n_{i+1} + n_{i+7} + n_{i+15} \\ z_i + h(l_{i+1}, l_{i+22}, n_{i+23}) = m_i \end{cases}, \quad 0 \le i \le 19 \qquad (5)$$

By Eq.(5), the rest NFSR state bits are expressed by tap bits, which is helpful when we deduce the unknown bits from the known ones.

TABLE 2. The tap bits restricted by the keystream bits.

$z_0 = m_0 + h(l_1, l_{21}, n_{22})$	$z_{10} = m_{10} + h(l_{11}, l_{31}, n_{32})$
$z_1 = m_1 + h(l_2, l_{22}, n_{23})$	$z_{11} = m_{11} + h(l_{12}, l_{32}, n_{33})$
$z_2 = m_2 + h(l_3, l_{23}, n_{24})$	$z_{12} = m_{12} + h(l_{13}, l_{33}, n_{34})$
$z_3 = m_3 + h(l_4, l_{24}, n_{25})$	$z_{13} = m_{13} + h(l_{14}, l_{34}, n_{35})$
$z_4 = m_4 + h(l_5, l_{25}, n_{26})$	$z_{14} = m_{14} + h(l_{15}, l_{35}, n_{36})$
$z_5 = m_5 + h(l_6, l_{26}, n_{27})$	$z_{15} = m_{15} + h(l_{16}, l_{36}, n_{37})$
$z_6 = m_6 + h(l_7, l_{27}, n_{28})$	$z_{16} = m_{16} + h(l_{17}, l_{37}, n_{38})$
$z_7 = m_7 + h(l_8, l_{28}, n_{29})$	$z_{17} = m_{17} + h(l_{18}, l_{38}, n_{39})$
$z_8 = m_8 + h(l_9, l_{29}, n_{30})$	$z_{18} = m_{18} + h(l_{19}, l_{39}, n_{40})$
$z_9 = m_9 + h(l_{10}, l_{30}, n_{31})$	$z_{19} = m_{19} + h(l_{20}, l_{40}, n_{41})$

Algorithm 2 The Construction of Candidate $\vec{x}_i$ Set for $\vec{z}_i$

```
Initialize e = 0 and fetch all Δx stored in T_uv
while e ≤ c · 2^n/|D| do
    load x'_i with a random value, after that it generates
    (m_i, m_{i+1})^T = z_i + Δz + h(x'_i)
    for each Δx in T_uv do
        compute x_i = x'_i + Δx
        if z_i = (m_i, m_{i+1})^T + h(x_i) then
            put x_i into the candidate set S_(z_i.z_{i+1})
        end if
    end for
    e = e + 1
end while
```

Theorem 2: In the FNCA on a reduced Grain, after the 60 tap bits restricted by 20 consecutive keystream bits are recovered, all bits of a full internal state can be computed by implementing the update functions of LFSR and NFSR and guessing two internal state bits.

Proof: In this case, LFSR state bits $l_1 \sim l_{40}$ and NFSR state bits $n_{22} \sim n_{41}$ are the tap bits. After these 60 internal state bits are recovered, another 15 NFSR state bits can be solved in the order of $n_{16} \sim n_{20}, n_{10} \sim n_{14}, n_4 \sim n_6, n_8, n_2$ according to the equations in 2. With the update function of LFSR, we can compute $l_0 = l_{40} + l_{33} + l_{18} + l_9$. Similarly, the value of n_0 can be computed from the update function of NFSR.

So far, all LFSR state bits have been recovered while there still remain 7 state bits of NFSR to be solved. They are $n_1, n_3, n_7, n_9, n_{15}, n_{17}, n_{21}$. However, if the concrete values of n_3 and n_9 are known, $n_{15}, n_{21}, n_7, n_1, n_{17}$ will be solved successively. There exist 2^2 combinations of (n_3, n_9), which requires that all 2^2 different versions of the 8 NFSR state bits should be checked. □

Then Algorithm 2 [16] is implemented to recover the tap bits. We collect a 20-bit-keystream $(z_0, z_1, \ldots, z_{19})$ and convert it into 19 keystream segment vectors $\vec{z}_i = (z_i, z_{i+1})^T$, $0 \le i \le 18$. For each $\vec{z}_i$, a set of candidates $\vec{x}_i$ will be generated, denoting the set as $S_{(z_i, z_{i+1})}$. After all $S_{(z_i, z_{i+1})}$ are generated, by putting them together a new set of probable full internal states will be sorted out, which may exist the 80-bit-internal-state that generates the 20-bit-keystream. In brief, we denote the set of candidates for sub-state as candidate set.

A difference between the near-collision inequality of NCA and the one of FNCA is that the former uses $|T_{uv}|$ instead of $|D|$. Based on the next section, V. EXPERIMENT, it is found that $\vec{z}_i$ does not decide the number of kinds of Δx stored in $|T_{uv}|$ while Δz does. Therefore it is more efficient to generate $S_{(z_i,z_{i+1})}$ by choosing a reasonable Δz than by analyzing all situations by traversing its values. Meanwhile, the cardinality of $S_{(z_i,z_{i+1})}$ effects the final candidate internal states after the merging phase, which is also constrained by choice of Δz.

The mathematical expectation of the number of ISDs in T_{uv} can be computed through diversified probability of the table T_{uv}, which equals to $|T_{uv}| \cdot P_{divs}$. However, the cardinality of $S_{(z_i,z_{i+1})}$ cannot be reckoned as $e \cdot |T_{uv}| \cdot P_{divs}$. It is barely possible to generate i different $\vec{x}'_i$ during the invoking of Algorithm 2. Then the mathematical expectation of the cardinality of $S_{(z_i,z_{i+1})}$ can be computed.

Theorem 3 [16]: Let $a = c \cdot 2^n/|D| \cdot |T_{uv}| \cdot P_{divs}$, which denotes the number of the random $\vec{x}'_i$ generated in Algorithm 2, and $b = |\mathbb{F}_2^6|$, which denotes the number of all values that can be hit. Then the mathematical expectation of the cardinality ξ of $S_{(z_i,z_{i+1})}$ after one invoking of Algorithm 2 is

$$E[\xi] = \sum_{r=1}^{a} \frac{\binom{b}{r} \cdot r! \cdot \left\{ {a \atop r} \right\}}{b^a} \cdot r. \tag{6}$$

where $\left\{ {a \atop r} \right\}$ is the Stirling number of the second kind, $\binom{b}{r}$ is the binomial coefficient, noting that $\binom{b}{r} = 0$ if $b < r$,and $r!$ is the factorial.

However, in the practical situation, $\vec{x}'_i$ cannot be truly generated at random, which is known as pseudo-random. For a chosen random number generator, it should be experimented to figure out the actual number of the $\vec{x}'_i$ after eliminating the duplicate ones. According to Eq.(1) we can compute the probability p' that a right candidate exists in $S_{(z_i,z_{i+1})}$.

C. ONLINE PHASE: MERGING STRATEGY

So far, we have generated 19 sets. They are $S_{(z_0,z_1)}$, $S_{(z_1,z_2)}, \ldots, S_{(z_{18},z_{19})}$ in which contain the candidate $\vec{x}$ for $\vec{z}_0, \vec{z}_1, \ldots, \vec{z}_{18}$ respectively, and for each $S_{(z_i,z_{i+1})}$ the mathematical expectation of its cardinality is $E[\xi]$. The merging of $\vec{x}$ restricted by $\vec{z}_0 \sim \vec{z}_{18}$ tries to find the $\vec{x}$ which share the same clock when the target keystream was being generated. For example, (l_2, l_{22}, n_{23}) is the overlapping part of $\vec{x}_0$ and $\vec{x}_1$, it connects (l_1, l_{21}, n_{22}) to (l_3, l_{23}, n_{33}). Merging $\vec{x}_0$ and $\vec{x}_1$ into $(l_1, l_{21}, n_{22}, l_2, l_{22}, n_{23}, l_3, l_{23}, n_{24})$, the number of its candidates that generate (z_0, z_1, z_2) can be computed.

Theorem 4 [16]: Let $S_{\vec{x}_i}$ denotes the candidate set of $\vec{x}_i$, then when merging the candidate set $S_{\vec{x}_i}$ and $S_{\vec{x}_{i+1}}$ to cover a union state $\vec{x}_i \cup \vec{x}_{i+1}$, the expected number of the candidates for the union state $\vec{x}_i \cup \vec{x}_{i+1}$ is

$$E[S_{\vec{x}_i \cup \vec{x}_{i+1}}] = \frac{|S_{\vec{x}_i}| \cdot |S_{\vec{x}_{i+1}}|}{|S_{\vec{x}_i} \cap S_{\vec{x}_{i+1}}|}. \tag{7}$$

where $S_{\vec{x}_i \cup \vec{x}_{i+1}}$ denotes the candidate set for the union state $\vec{x}_i \cup \vec{x}_{i+1}$.

Therefore the mathematical expectation of $S_{(z_i,z_{i+1},z_{i+2})} = \frac{\xi^2}{2^3}$. Then we need to add constraints to make sure the clock of $\vec{x}_0$ is the same as the clock of $\vec{x}_1$. It is obvious that the tap bits of the candidates in $S_{(z_0,z_1)}$ or $S_{(z_1,z_2)}$ share the same clock, while the ones of $S_{(z_0,z_2)}$ are not certain. Let $\vec{x}_{0,1} = (l_1, l_{21}, n_{22}, l_3, l_{23}, n_{24})$ which restricted by (z_0, z_2). Like the generation of $S_{(z_i,z_{i+1})}$, we can get a $S_{(z_0,z_2)}$ in which includes the candidates of $\vec{x}_{0,1}$ for (z_0, z_2). By the state union $\vec{x}_0 \cup \vec{x}_1 \cup \vec{x}_{0,1}$, denoting it as $\vec{x}_{01}$ in brief, and computing $E[S_{(z_0,z_1,z_2)}] = \frac{\xi^3}{2^9}$, a set of union candidates of $(l_1, l_{21}, n_{22}, l_2, l_{22}, n_{23}, l_3, l_{23}, n_{24})$ that share one clock over (z_0, z_1, z_2) and mathematical expectation of its cardinality can be got respectively.

After we get $S_{(z_0,z_1,z_2)}$, there are two methods to generate $S_{(z_0,z_1,z_2,z_3)}$. Method 1: merging $S_{(z_0,z_1,z_2)}$ and $S_{(z_2,z_3)}$, or method 2: merging $S_{(z_0,z_1,z_2)}$ and $S_{(z_1,z_2,z_3)}$. The difference between method 1 and method 2 is the number of overlapping bits of the two keystream segments. In method 1, we merge $S_{(z_0,z_1,z_2)}$ and $S_{(z_1,z_2,z_3)}$ together, after that we merge it with $S_{(z_0,z_3)}$ to get a candidate set $S_{(z_0,z_1,z_2,z_3)}$ where the tap bits share the same time clock. It can be computed that $E[S_{(z_0,z_1,z_2,z_3)}] = \frac{\xi^6}{2^{18}}$, while in the method 2 it is $E[S_{(z_0,z_1,z_2,z_3)}] = \frac{\xi^7}{2^{30}}$. Because $0 < \xi < 64$, method 2 has a better reducing effect than method 1 on the union candidate set after merging. Then we get $S_{(z_0,z_1,z_2,z_3)}$ whose expected cardinality is $E[S_{(z_0,z_1,z_2,z_3)}] = \frac{\xi^7}{2^{30}}$. Now the bit length of the keystream segment is 4 and there is one more merging method, because for a 4-bit keystream segment there are at most 3 bits to overlap (overlapping 4 bits cannot help this keystream segment grow into a 5-bit keystream segment). Therefore, the cases of the expected cardinality of the merged segment can be $E[S_{(z_0,z_1,z_2,z_3,z_4)}] = \frac{\xi^9}{2^{39}}, \frac{\xi^{11}}{2^{51}}, \frac{\xi^{15}}{2^{75}}$ respectively. Obviously when the number of overlapping bits is 3, the cardinality of the merged segment reaches a minimum value, compared with another 2 methods. It concludes that by increasing the overlapping bits of two candidate sets, a smaller union candidate set can be generated. However, it is undesirable to increasing the overlapping bits without halting, in which case the cardinality of the final candidate set $E[S_{fnl}] = E[S_{(z_0,z_1,\ldots,z_{19})}] = \frac{\xi^\alpha}{2^\beta}$ $(\alpha, \beta \in Z^+)$ can smaller than 1. The next steps are similar to the process of the generation of $S_{(z_0,z_1,z_2)}$ or $S_{(z_0,z_1,z_2,z_3)}$. The overall procedure of merging can be summarized as FIGURE 2.

Notice that when a merging method of q overlapping bits is going to be implemented, the bit length of the keystream segment to be merged should bigger than q. Before we start our linear programming with 5 merging methods, we first grow the candidate set form $S_{(z_0,z_1)}$ into $S_{(z_0,z_1,\ldots,z_5)}$, which takes 4 steps. As it is found that larger overlapping bits of two candidate sets leads to a smaller union candidate set, when we grow the $S_{(z_0,z_1)}$ into a candidate set $S_{(z_0,z_1,z_2,z_4,z_5)}$, the way to minimize the candidates before the linear programming is to make the number of overlapping bits in each step is exactly 1 bit smaller than the bit length of the keystream segment.

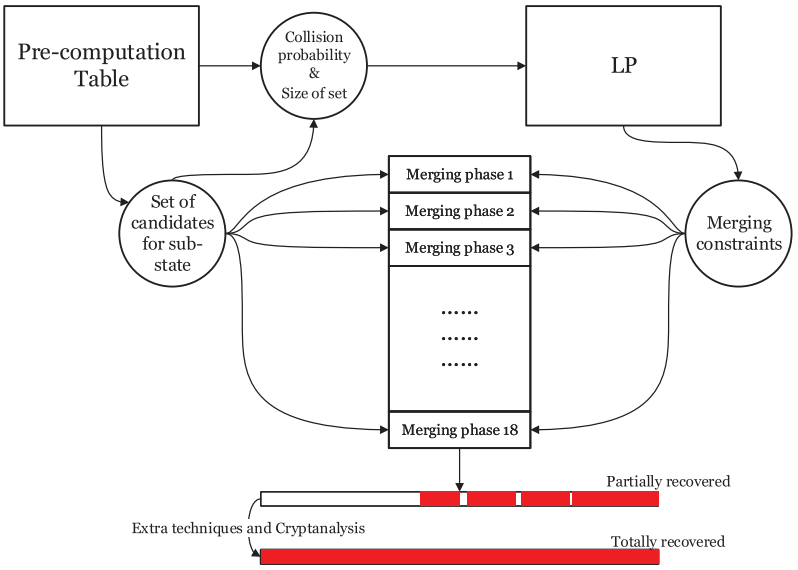

FIGURE 2. The overall procedure of merging process.

TABLE 3. Data increased times with the numbers of overlapping bits for the reduced Grain.

Number of overlapping bits	1	2	3	4	5
Data increased times	$\frac{\xi^2}{2^9}$	$\frac{\xi^4}{2^{21}}$	$\frac{\xi^8}{2^{45}}$	$\frac{\xi^{16}}{2^{93}}$	$\frac{\xi^{32}}{2^{189}}$

In step 1, we generate $S_{(z_0,z_1)}$, $S_{(z_1,z_2)}$ and $S_{(z_0,z_2)}$, after that, we merge them into $S_{(z_0,z_1,z_2)}$. In step 2, since all $S_{(z_i,z_{i+1})}$ $(0 \leq i \leq 18)$ are generated after implementing Algorithm 2, we use the same way in step 1 to generate $S_{(z_1,z_2,z_3)}$. Then we merge it with $S_{(z_0,z_1,z_2)}$ and $S_{(z_0,z_2)}$, finally we get $S_{(z_0,z_1,z_2,z_3)}$. Similarly, in step 3, $S_{(z_0,z_1,z_2,z_3)}$, $S_{(z_1,z_2,z_3,z_4)}$ and $S_{(z_0,z_4)}$ are merged into $S_{(z_0,z_1,z_2,z_3,z_4)}$. In step 4, $S_{(z_0,z_1,z_2,z_3,z_4)}$, $S_{(z_1,z_2,z_3,z_4,z_5)}$ and $S_{(z_0,z_5)}$ are merged into $S_{(z_0,z_1,z_2,z_3,z_4,z_5)}$.

Since from $S_{(z_0,z_1)}$ it takes 18 steps to get $S_{(z_0,z_1,\ldots,z_{18},z_{19})}$, the first 4 steps can be reckoned as an initialization for the linear programming. In the following 14 steps, we discuss the choices of the merging methods. By adjusting the overlapping bits of two candidate sets, the increased times of the cardinalities of union candidate sets can be computed. In TABLE 3, we list 5 different constraints. Each constraint calculates the increased times of a candidate set after it takes a merging process. It is found that when the constraint of 6 or more overlapping bits is taken into the merging, the reduction effects of final candidates is not better than the case of the previous five constraints. After Theorem IV-C is introduced, we will explain why five merging constraints are enough in the case of the reduced Grain.

Here the merging constraints are similar to the ones in [16]. However, we apply an optimized strategy of the choices of these methods, which turns out a better result. We exploit the results in TABLE 3 to convert the merging phase into a linear programming problem. Here a more explicit merging strategy is illustrated.

Theorem 5: Let non-negative variable w_q denote the number of times of the q^{th} constraint used in the whole merging phase, the number of the final candidates after merging can be expressed as

$$f(w_1, w_2, w_3, w_4, w_5) = \frac{(\frac{\xi}{p})^{2w_1+4w_2+8w_3+16w_4+32w_5+31}}{2^{9w_1+21w_2+45w_3+93w_4+189w_5+168}}, \quad (8)$$

which is the function to be minimized. The problem constraints are set as

$$\begin{cases} 1 \leq q \leq 5 \\ w_q \geq 1 \\ \sum w_q = 14 \\ E[S_{\boldsymbol{fnl}}] > 1, \end{cases} \quad (9)$$

where $E[S_{\boldsymbol{fnl}}]$ denotes the expected number of the final candidates for $(z_0, z_1, z_2, \ldots, z_{18}, z_{19})$.

Proof: From TABLE 3 we have 5 ways to generate $S_{(z_i,z_{i+1},z_{i+2})}$ from $S_{(z_i,z_{i+1})}$. Therefore j satisfies $1 \leq j \leq 5$. $w_j \geq 1$ makes sure that each constraint should be implemented at least once. The total step if merging is fixed, then after 18 steps we can get the final candidate set of $(z_0, z_1, z_2, \ldots, z_{18}, z_{19})$. To implement the 5 merging constraints, it is needed to make $S_{(z_0,z_1)}$ grow up to $S_{(z_0,z_1,\ldots,z_5)}$, which takes 4 steps. After that, the expected cardinality is $\frac{\xi^{31}}{2^{168}}$. In the following 14 steps, each of them requires comparisons that implementing which merging constraint can minimize the $E[S_{\boldsymbol{fnl}}]$ with the requirement that $E[S_{\boldsymbol{fnl}}] \geq 1$, and $\sum w_j = 14$ guarantees the 14 steps are covered in the linear programming. Depending on the above analysis, we set

the problem constraints as

$$\begin{cases} 1 \le q \le 5 \\ w_q \ge 1 \\ \sum w_q = 14 \\ E[S_{fnl}] > 1, \end{cases}$$

and the expected cardinality of the final candidate set

$$E[S_{fnl}] = \frac{\xi^{2w_1+4w_2+8w_3+16w_4+32w_5+31}}{2^{9w_1+21w_2+45w_3+93w_4+189w_5+168}}$$

can be computed.

Then the problem is converted into a scheduling problem, which can be thought as using linear programming to find a $(w_1, w_2, w_3, w_4, w_5)$ which makes $E[S_{fnl}]$ reach its minimum with a probability $P = p^{2w_1+4w_2+8w_3+16w_4+32w_5+31}$. In order to recover the tap bits steadily, we will repeat the above online phase P^{-1} times, therefore the objective function to be minimized is

$$f(w_1, w_2, w_3, w_4, w_5) = \frac{(\frac{\xi}{p})^{2w_1+4w_2+8w_3+16w_4+32w_5+31}}{2^{9w_1+21w_2+45w_3+93w_4+189w_5+168}},$$

□

When the number of overlapping bits is 6, the data increased times is $\frac{\xi^{64}}{2^{381}}$. We convert Theorem IV-C into a six overlapping bits version, which means before linear programming, we first grow the candidate set into $S_{(z_0,z_1,\ldots,z_6)}$.

$$f(w_1, w_2, w_3, w_4, w_5, w_6) = \frac{(\frac{\xi}{p})^{2w_1+4w_2+8w_3+16w_4+32w_5+64w_6+63}}{2^{9w_1+21w_2+45w_3+93w_4+189w_5+381w_6+357}},$$

which subjects to

$$\begin{cases} 1 \le q \le 6 \\ w_q \ge 1 \\ \sum w_q = 13 \\ E[S_{fnl}] > 1, \end{cases}$$

We figure out that when $(w_1, w_2, w_3, w_4, w_5, w_6) = (4, 3, 3, 1, 1, 1)$ the final time complexity is $2^{37.1096}$, which is the same with the result of the 5 overlapping bits version $2^{37.1096}$, and this phenomenon also occurs when the number of overlapping bits larger than 6 in the case of the reduced Grain. Since using the constraints of small overlapping bits can be more flexible in linear programming, we only consider 5 merging constraints in this case.

At the end of the merging phase, we collect S_{fnl} and each candidate of it includes most bits of the full internal state. Then we implement Theorem IV-B to restore the rest bits in the case of the reduced Grain. For Grain-v1, extra techniques in [16] is required. After all bits of final candidates are figured out, by checking whether they can generate the target keystream segment one by one, the target internal state can be determined. In the next section, we will solve the linear programming problem and give the merging process in TABLE 6, where the choice of the number of overlapping bits in each step is discussed.

TABLE 4. The cardinality of T_{uv}.

$\lvert T_{uv}\rvert$: $\vec{z}_i$ \ Δz_j	0x0	0x1	0x2	0x3
0x0	528	368	368	144
0x1	528	368	368	144
0x2	528	368	368	144
0x3	528	368	368	144
Total	5632			

TABLE 5. Probability distribution of ISDs in T_{uv}.

$(\vec{z}_i, \Delta z)$	(*,0x0)			(*,0x1)(*,0x2)			(*,0x3)
probability	1	0.5	0.25	1	0.5	0.25	0.25
number	1	10	9	1	5	9	9

V. EXPERIMENT

Based on the previous sections and the following results of the pre-computation, we can solve the values of the basic parameters in the linear programming problem of Eq.(8) and Eq.(9). After that, the concrete strategy of merging is determined, and we can mount our improved attack. This section is divided into three parts. In the first part, the construction of the pre-computed table generation will be shown. The second part recovers the tap bits. The third part gives the process of merging, which contains the number of the candidates and the probability of existing the right one by each step. We randomly generated the internal states of NFSR and LFSR as 0x6279665762 and 0xe2b9fc0729, and the first 20-bit keystream is $(z_0, z_1, \ldots, z_{19}) = 1001\ 1011\ 1000\ 0000\ 1100$.

A. PRE-COMPUTATION

Algorithm 1 describes the whole process of the pre-computation, from which we started our experiment. TABLE 6 and TABLE 4 depict the cardinality of T_{uv} and the ISD probability distribution respectively.

It has been found that the value of $\vec{z}_i$ does not affect the cardinality or the ISD probability distribution of a specific T_{uv}. From TABLE 4, the diversified probability of the T_{uv} can be computed.

$$P_{divs=}\begin{cases} 0.4125, & if\ \Delta z = 0x0 \\ 0.3833, & if\ \Delta z = 0x1 \\ 0.3833, & if\ \Delta z = 0x2 \\ 0.2500, & if\ \Delta z = 0x3. \end{cases}$$

If $\Delta z = 0x3$, its diversified probability turns out to be the smallest one. When it comes to the construction of the candidate set for $\vec{z}_i$, we only consider the situation that $\Delta z = 0x3$. Hence the time complexity of the pre-computation is $\frac{2^6 \cdot 22 \cdot 2 + 2^4 \cdot 22 \cdot 2}{\Omega} = \frac{3520}{2^{10.4}} \approx 2^{2.6051}$.

B. TAP BITS RECOVERY

Through Algorithm 2, let $c = 8$, then a candidate set for each $\vec{z}_i$ can be generated. After that, we got $E[\xi] = 53$ by Eq.(6) and computed the probability $p' \approx 0.9195$ that it contains a right candidate by Eq.(1). At the same time, we followed the algorithm in [16] to distill the candidates to get a smaller

TABLE 6. The process of merging for the reduced Grain.

Step	$\vec{z}$	$\lvert S_{union}\rvert$	P	Step	$\vec{z}$	$\lvert S_{union}\rvert$	P	Step	$\vec{z}$	$\lvert S_{union}\rvert$	P
1	(z_0, z_1) (z_1, z_2) (z_0, z_2)	$\frac{\xi^2}{2^3}$ $\frac{\xi^3}{2^9}$	p^3	7	$(z_0, \ldots, z_7)$ $(z_3, \ldots, z_8)$ (z_0, z_8)	$\frac{\xi^{126}}{2^{729}}$ $\frac{\xi^{127}}{2^{735}}$	p^{127}	13	$(z_0, \ldots, z_{13})$ $(z_{12}, \ldots, z_{14})$ (z_0, z_{14})	$\frac{\xi^{202}}{2^{1167}}$ $\frac{\xi^{203}}{2^{1173}}$	p^{203}
2	(z_0, z_1, z_2) (z_1, z_2, z_3) (z_0, z_3)	$\frac{\xi^6}{2^{24}}$ $\frac{\xi^7}{2^{30}}$	p^7	8	$(z_0, \ldots, z_8)$ $(z_4, \ldots, z_9)$ (z_0, z_9)	$\frac{\xi^{158}}{2^{918}}$ $\frac{\xi^{159}}{2^{924}}$	p^{159}	14	$(z_0, \ldots, z_{14})$ $(z_{13}, \ldots, z_{15})$ (z_0, z_{15})	$\frac{\xi^{206}}{2^{1188}}$ $\frac{\xi^{207}}{2^{1194}}$	p^{207}
3	$(z_0, \ldots, z_3)$ $(z_1, \ldots, z_4)$ (z_0, z_4)	$\frac{\xi^{14}}{2^{69}}$ $\frac{\xi^{15}}{2^{75}}$	p^{15}	9	$(z_0, \ldots, z_9)$ $(z_6, \ldots, z_{10})$ (z_0, z_{10})	$\frac{\xi^{174}}{2^{1011}}$ $\frac{\xi^{175}}{2^{1017}}$	p^{175}	15	$(z_0, \ldots, z_{15})$ $(z_{14}, \ldots, z_{16})$ (z_0, z_{16})	$\frac{\xi^{210}}{2^{1209}}$ $\frac{\xi^{211}}{2^{1215}}$	p^{211}
4	$(z_0, \ldots, z_4)$ $(z_1, \ldots, z_5)$ (z_0, z_5)	$\frac{\xi^{30}}{2^{162}}$ $\frac{\xi^{31}}{2^{168}}$	p^{31}	10	$(z_0, \ldots, z_{10})$ $(z_8, \ldots, z_{11})$ (z_0, z_{11})	$\frac{\xi^{182}}{2^{1056}}$ $\frac{\xi^{183}}{2^{1062}}$	p^{183}	16	$(z_0, \ldots, z_{16})$ $(z_{15}, \ldots, z_{17})$ (z_0, z_{17})	$\frac{\xi^{214}}{2^{1230}}$ $\frac{\xi^{215}}{2^{1236}}$	p^{215}
5	$(z_0, \ldots, z_5)$ $(z_1, \ldots, z_6)$ (z_0, z_6)	$\frac{\xi^{62}}{2^{351}}$ $\frac{\xi^{63}}{2^{357}}$	p^{63}	11	$(z_0, \ldots, z_{11})$ $(z_9, \ldots, z_{12})$ (z_0, z_{12})	$\frac{\xi^{190}}{2^{1101}}$ $\frac{\xi^{191}}{2^{1107}}$	p^{191}	17	$(z_0, \ldots, z_{17})$ $(z_{17}, \ldots, z_{18})$ (z_0, z_{18})	$\frac{\xi^{216}}{2^{1239}}$ $\frac{\xi^{217}}{2^{1245}}$	p^{217}
6	$(z_0, \ldots, z_6)$ $(z_2, \ldots, z_7)$ (z_0, z_7)	$\frac{\xi^{94}}{2^{540}}$ $\frac{\xi^{95}}{2^{546}}$	p^{95}	12	$(z_0, \ldots, z_{12})$ $(z_{10}, \ldots, z_{13})$ (z_0, z_{13})	$\frac{\xi^{198}}{2^{1146}}$ $\frac{\xi^{199}}{2^{1152}}$	p^{199}	18	$(z_0, \ldots, z_{18})$ $(z_{18}, \ldots, z_{19})$ (z_0, z_{19})	$\frac{\xi^{218}}{2^{1248}}$ $\frac{\xi^{219}}{2^{1254}}$	p^{219}

candidate set by which we had $p = 1-[1-(p')^{\beta}]^{\gamma} \approx 0.8960$, where $\beta = 10$, $\gamma = 4$.

C. MERGING STRATEGY

First, just as the discussion in Section IV-C, we grew the candidate set into $S_{(z_0,z_1,\ldots,z_6)}$, after that, we started the linear programming. We used LINGO to solve the linear programming problem of Eq.(8) and Eq.(9). When $(w_1, w_2, w_3, w_4, w_5) = (2, 4, 3, 1, 4)$, we got the minimum value of Eq.(8), which is $2^{35.1096}$. Note that after a merging phase, the number of final candidates is $2^{0.4145}$. Thus we needed to repeat the merging process about $2^{34.6960}$ times to have the internal state restored at a high probability. According to Theorem IV-B, by guessing two more bits, the full internal state can be recovered. Therefore the time complexity of the online attack is $2^{37.1096}$, comparing to the one in [16], which is $2^{37.58}$, our method has an improvement by 27.8%.

TABLE 5 shows the cardinality of the candidate set for the union state after each merging step and the probability for it to have the right candidate. The maximum memory cost among the 18 steps is step 3, where it takes a memory usage of $2 \cdot \frac{\xi^{15}}{2^{75}} \cdot 61 \approx 2 \cdot 2^{10.9188} \cdot 2^{5.9307} \approx 2^{17.8495}$ bits. Also, the memory usage of the pre-computed table is $2^{2l} \cdot V(n, d) \cdot (\lceil log_2 n \rceil \cdot d + 14) \approx 2^{12.7814}$ bits, where l denotes the bit length of a keystream segment and the constant 14 denotes the memory to store the occurring rate of Δx. Therefore the overall memory complexity satisfies $2^{17.8495} + 2^{12.7814} = 2^{17.8919} < 2^{18}$.

VI. THE THEORETICAL ANALYSIS OF IMPROVED FNCA ON GRAIN-V1

The FNCA experiment of Grain-v1 can hardly be carried on a single PC, whereas [16] gives a credible proof of the feasibility of implementing FNCA on Grain-v1. Here we analyze the improvement effect on Grain-v1 with our method, and it turns out to be a reduction of 79.4%. In the case of the reduced Grain, we only discuss the number of overlapping bits which is not more than 5. For a better improvement effect when it comes to a larger scale cipher, we take the situation where the number of overlapping bits is six into consideration.

TABLE 7. Data increased times with the numbers of overlapping bits for Grain-v1.

Number of overlapping bits	1	2	3	4	5	6
Data increased times	$\frac{\xi^2}{2^{15}}$	$\frac{\xi^4}{2^{35}}$	$\frac{\xi^8}{2^{75}}$	$\frac{\xi^{16}}{2^{155}}$	$\frac{\xi^{32}}{2^{315}}$	$\frac{\xi^{64}}{2^{635}}$

We first grow the candidate set form $S_{(z_0,z_1)}$ into $S_{(z_0,z_1,\ldots,z_6)}$, which takes 5 steps. In the following 13 steps, we use linear programming to choose merging constraints. Similar to TABLE 3, we compute the data increased times of different numbers of overlapping bits for Grain-v1. The results are presented in TABLE 7. Let the non-negative vector $(w_1, w_2, w_3, w_4, w_5, w_6)$ denotes the number of times of the 6 merging constraints to be implemented respectively. After that, we get the linear function to be minimized is

$$f(w_1, w_2, w_3, w_4, w_5, w_6) = \frac{(\frac{\xi}{p})^{2w_1+4w_2+8w_3+16w_4+32w_5+64w_6+63}}{2^{15w_1+35w_2+75w_3+155w_4+315w_5+635w_6+595}},$$

which subjects to

$$\begin{cases} 1 \le q \le 6 \\ w_q \ge 1 \\ \sum w_q = 13 \\ E[S_{\mathit{fnl}}] > 1. \end{cases}$$

From [16], it is known that the (ξ, p) of the original FNCA on Grain-v1 is $(848, 0.896456)$. Building on the previous conclusion, the $(w_1, w_2, w_3, w_4, w_5, w_6)$ that minimizes the linear function is $(2, 2, 2, 1, 4, 2)$. Similar to the process of the merging for the reduced Grain, the expected number of the final candidates can be computed as $2^{58.4785}$. In Appendix we give a Grain-v1 version of the tap bits restricted by the

TABLE 8. The constraint of overlapping bits taken in each step of the merging phase for the optimized FNCA on Grain-v1.

Step	1	2	3	4	5	6
Merging constraint	w_1	w_2	w_3	w_4	w_5	w_6
Step	7	8	9	10	11	12
Merging constraint	w_6	w_5	w_5	w_5	w_5	w_4
Step	13	14	15	16	17	18
Merging constraint	w_3	w_3	w_2	w_2	w_1	w_1

TABLE 9. Comparisons of the cardinality of the final candidate set between original FNCA and optimized FNCA.

Number of final candidates	Original FNCA [16]	Optimized FNCA
Reduced Grain	$2^{35.58}$	$2^{35.1096}$
Grain-v1	$2^{57.7595}$	$2^{55.4785}$

keystream bits. It is found that both l_{64} and l_{65} appear 2 times. Besides, from the update function of LFSR we get $l_{83} = l_{65}+l_{54}+l_{41}+l_{26}+l_{16}+l_3$, which means there is a third linear consistency check on the candidates. Therefore the number of candidates to be checked is $2^{58.4785} \cdot 2^{-3} = 2^{55.4785}$.

In the case of the original FNCA, the number of final candidates is $2^{60.7595}$. After the issue of the reused LFSR bits and a third linear consistency check is discussed, the number of the rest candidates to be checked is $2^{60.7595} \cdot 2^{-3} = 2^{57.7595}$. Comparing to the original FNCA, our method has a reduction of 79.42%. To recover the full state, each one of these $2^{57.7595}$ candidates should be checked with extra techniques. According to [16], it can be found that the time complexity of implementing extra techniques is so large that it almost equals the time complexity of the overall attack procedure. Thus, the time complexity of the improved FNCA has a reduction of around 79.4% compared with the original one.

At step 4, $|S_{\boldsymbol{union}}|$ reaches its maximum value, which is $\frac{\xi^{31}}{2^{280}} = \frac{848^{31}}{2^{280}} \approx 2^{21.5655}$. Thus the memory complexity is $2 \cdot \frac{\xi^{31}}{2^{280}} \cdot 42 + 2^{2l} \cdot V(n,d) \cdot (\lceil log_2 n \rceil \cdot d + 14) \approx 2^{27.9579} + 2^{15.9189} \approx 2^{27.9582} < 2^{28}$, which is close to the memory complexity of the original FNCA.

VII. CONCLUSION

This paper proposes an improved FNCA, which optimizes the merging strategy and the final candidates. We run experiments on a reduced Grain whose internal state is 80 bits and make theoretical analysis on Grain-v1. The final online attack time complexity has a 27.8% reduction for the reduced Grain and a theoretical reduction of 79.4% for Grain-v1, compared with the original attack. As an optimization for the merging phase, our method can be implemented on the FNCA based cryptanalysis of some Grain-like stream ciphers. Recent years improved FCA like [15] and [14] promote the researches of Grain-like small state stream ciphers e.g., Plantlet [11], Fruit [6] and [5], while there are few papers for FNCA. Implementing FNCA on Grain-like small state stream ciphers can be a promising work in the future.

APPENDIX
TAP BITS OF GRAIN-V1

The tap bits restricted by the keystream bits, in the case of Grain-v1, are given in the below equations:

$$
\begin{aligned}
z_0 &= m_0 + h(l_3, l_{25}, l_{46}, l_{64}, n_{63}) \\
z_1 &= m_1 + h(l_4, l_{26}, l_{47}, l_{65}, n_{64}) \\
z_2 &= m_2 + h(l_5, l_{27}, l_{48}, l_{66}, n_{65}) \\
z_3 &= m_3 + h(l_6, l_{28}, l_{49}, l_{67}, n_{66}) \\
z_4 &= m_4 + h(l_7, l_{29}, l_{50}, l_{68}, n_{67}) \\
z_5 &= m_5 + h(l_8, l_{30}, l_{51}, l_{69}, n_{68}) \\
z_6 &= m_6 + h(l_9, l_{31}, l_{52}, l_{70}, n_{69}) \\
z_7 &= m_7 + h(l_{10}, l_{32}, l_{53}, l_{71}, n_{70}) \\
z_8 &= m_8 + h(l_{11}, l_{33}, l_{54}, l_{72}, n_{71}) \\
z_9 &= m_9 + h(l_{12}, l_{34}, l_{55}, l_{73}, n_{72}) \\
z_{10} &= m_{13} + h(l_{35}, l_{56}, l_{74}, l_{65}, n_{73}) \\
z_{11} &= m_{14} + h(l_{36}, l_{57}, l_{75}, l_{67}, n_{74}) \\
z_{12} &= m_{15} + h(l_{37}, l_{58}, l_{76}, l_{69}, n_{75}) \\
z_{13} &= m_{16} + h(l_{38}, l_{59}, l_{77}, l_{71}, n_{76}) \\
z_{14} &= m_{17} + h(l_{39}, l_{60}, l_{78}, l_{73}, n_{77}) \\
z_{15} &= m_{18} + h(l_{40}, l_{61}, l_{79}, l_{75}, n_{78}) \\
z_{16} &= m_{19} + h(l_{41}, l_{62}, l_{80}, l_{77}, n_{79}) \\
z_{17} &= m_{20} + h(l_{42}, l_{63}, l_{81}, l_{79}, n_{80}) \\
z_{18} &= m_{21} + h(l_{43}, l_{64}, l_{82}, l_{81}, n_{81}) \\
z_{19} &= m_{22} + h(l_{44}, l_{65}, l_{83}, l_{83}, n_{82}).
\end{aligned}
$$

Note that, the m_i denotes the sum of linear masking bits $m_i = \sum_{k \in \mathcal{A}} n_{i+k}$, where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$. Being different from the case of the reduced Grain, there leaves about one-fourth of the internal state bits unsolved after these tap bits get recovered.

REFERENCES

[1] S. Babbage, "A space/time trade-off in exhaustive search attacks on stream ciphers," in *Proc. EUROCRYPT*, Zaragoza, Spain, May 1996. [Online]. Available: https://www.iacr.org/conferences/ec96/rump/index.html
[2] C. Berbain, H. Gilbert, and A. Maximov, "Cryptanalysis of grain," in *Fast Software Encryption*, vol. 4047, M. Robshaw, Ed. Berlin, Germany: Springer, 2006, pp. 15–29. [Online]. Available: https://link.springer.com/chapter/10.1007%2F11799313_2
[3] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science), vol. 1976. Berlin, Germany: Springer, 2000, pp. 1–13, doi: 10.1007/3-540-44448-3_1.
[4] T. E. Bjørstad, "Cryptanalysis of grain using time/memory/data tradeoffs," in *Proc. Estream Phase*, 2013. [Online]. Available: https://pdfs.semanticscholar.org/a6f2/066e38e82e7c259797c518a8198ddccb4d85.pdf
[5] V. A. Ghafari, H. Hu, and Y. Chen, "Fruit-v2: Ultra-lightweight stream cipher with shorter internal state," Cryptol. ePrint Arch., Tech. Rep. 2016/355, 2016. [Online]. Available: https://eprint.iacr.org/2016/355
[6] V. A. Ghafari and H. Hu, "Fruit-80: A secure ultra-lightweight stream cipher for constrained environments," *Entropy*, vol. 20, no. 3, p. 180, Mar. 2018. [Online]. Available: https://www.mdpi.com/1099-4300/20/3/180
[7] J. D. Golić, "Cryptanalysis of alleged A5 stream cipher," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 1233. Berlin, Germany: Springer, 1997, pp. 239–255, doi: 10.1007/3-540-69053-0_17.

[8] M. Hamann, M. Krause, W. Meier, and B. Zhang, "Time-memory-data tradeoff attacks against small-state stream ciphers," Cryptol. ePrint Arch., Tech. Rep. 2017/384, 2017. [Online]. Available: https://eprint.iacr.org/2017/384.pdf
[9] M. Hell, T. Johansson, and W. Meier, "Grain: A stream cipher for constrained environments," *Int. J. Wireless Mobile Comput.*, vol. 2, no. 1, p. 86, 2007. [Online]. Available: https://www.ecrypt.eu.org/stream/e2-grain.html
[10] *Birthday Problem*. Accessed: Dec. 2, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Birthday_problem
[11] V. Mikhalev, F. Armknecht, and C. Müller, "On ciphers that continuously access the non-volatile key," *IACR Trans. Symmetric Cryptol.*, vol. 2016, no. 2, pp. 52–79, 2017. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/565
[12] Y. Todo, T. Isobe, W. Meier, K. Aoki, and B. Zhang, "Fast correlation attack revisited: Cryptanalysis on full Grain-128a, Grain-128, and Grain-v1," in *Advances in Cryptology—CRYPTO*, vol. 10992, H. Shacham and A. Boldyreva, Eds. Cham, Switzerland: Springer, 2018, pp. 129–159, doi: 10.1007/978-3-319-96881-0_5.
[13] D. Wagner, "A generalized birthday problem," in *Advances in Cryptology—CRYPTO*, 2002, pp. 288–304, doi: 10.1007/3-540-45708-9_19.
[14] S. Wang, M. Liu, D. Lin, and L. Ma, "Fast correlation attacks on grain-like small state stream ciphers and cryptanalysis of plantlet, fruit-v2 and fruit-80," Cryptol. ePrint Arch., Tech. Rep. 2019/763, 2019. [Online]. Available: https://eprint.iacr.org/2019/763
[15] B. Zhang, X. Gong, and W. Meier, "Fast correlation attacks on Grain-like small state stream ciphers," *IACR Trans. Symmetric Cryptol.*, vol. 2017, pp. 58–81, Dec. 2017. [Online]. Available: https://tosc.iacr.org/index.php/ToSC/article/view/803
[16] B. Zhang, C. Xu, and W. Meier, "Fast near collision attack on the Grain v1 stream cipher," in *Advances in Cryptology—EUROCRYPT*, vol. 10821, J. B. Nielsen and V. Rijmen, Eds. Cham, Switzerland: Springer, 2018, pp. 771–802, doi: 10.1007/978-3-319-78375-8_25.
[17] B. Zhang, Z. Li, D. Feng, and D. Lin, "Near collision attack on the Grain v1 stream cipher," in *Fast Software Encryption*, vol. 8424, S. Moriai, Ed. Berlin, Germany: Springer, 2014, pp. 518–538, doi: 10.1007/978-3-662-43933-3_27.

SENSHAN PAN received the B.S. degree in information and computing sciences from Nanjing Normal University, in 2009, and the Ph.D. degree in cryptology from Xidian University, Xi'an, China, in 2015. He is currently a Lecturer with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. His research interests include data security and privacy.

YUEPING WU received the B.Eng. degree in computer science and technology from the Nanjing University of Posts and Telecommunications, China, in 2018. He is currently pursuing the master's degree in computer science with Jiangsu University, China. His research interests include data security and privacy.

LIANGMIN WANG (M'12) received the B.S. degree in computational mathematics from Jilin University, Changchun, China, in 1999, and the Ph.D. degree in cryptology from Xidian University, Xi'an, China, in 2007.

He is currently a Full Professor with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, China. He has published over 60 technical articles at premium international journals and conferences, such as the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE Global Communications Conference, and the IEEE Wireless Communications and Networking Conference. His current research interests include data security and privacy. Dr. Wang is a member of ACM and a Senior Member of Chinese Computer Federation. He has severed as a TPC Member for many IEEE conferences, such as IEEE ICC, IEEE HPCC, and IEEE TrustCOM. He has been honored as a Wan-Jiang Scholar of Anhui Province, since November 2013. He is an Associate Editor of *Security and Communication Networks*.