Hindawi
Security and Communication Networks
Volume 2017, Article ID 3834685, 16 pages
https://doi.org/10.1155/2017/3834685

Research Article

Fault Attack on the Authenticated Cipher ACORN v2

Xiaojuan Zhang,[1,2] Xiutao Feng,[1,3] and Dongdai Lin[1]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China*
[3]*Key Laboratory of Mathematics Mechanization, Academy of Mathematics and System Science, Chinese Academy of Sciences, Beijing, China*

Correspondence should be addressed to Xiaojuan Zhang; zhangxiaojuan@iie.ac.cn

Received 9 May 2017; Revised 24 July 2017; Accepted 23 August 2017; Published 2 October 2017

Academic Editor: Angelos Antonopoulos

Fault attack is an efficient cryptanalysis method against cipher implementations and has attracted a lot of attention in recent public cryptographic literatures. In this work we introduce a fault attack on the CAESAR candidate ACORN v2. Our attack is done under the assumption of random fault injection into an initial state of ACORN v2 and contains two main steps: fault locating and equation solving. At the first step, we first present a fundamental fault locating method, which uses 99-bit output keystream to determine the fault injected location with probability 97.08%. And then several improvements are provided, which can further increase the probability of fault locating to almost 1. As for the system of equations retrieved at the first step, we give two solving methods at the second step, that is, linearization and guess-and-determine. The time complexity of our attack is not larger than $c \cdot 2^{179.19-1.76N}$ at worst, where N is the number of fault injections such that $31 \leq N \leq 88$ and c is the time complexity of solving linear equations. Our attack provides some insights into the diffusion ability of such compact stream ciphers.

1. Introduction

CAESAR [1] is a new competition calling for authenticated encryption schemes. Its purpose is to find authenticated ciphers that offer advantages over AES-GCM and are suitable for widespread adoption. In total, 57 candidates were submitted to the CAESAR competition, and after the challenge of two rounds, 15 submissions have been selected for the third round. As one of them, ACORN is a lightweight stream cipher based authenticated encryption cipher submitted by Hongjun [2–4]. The cipher consists of a simple binary feedback shift register (FSR, for short) of length 293 and aims to protect up to 2^{64} bits of associated data (AD) and up to 2^{64} bits of plaintext and to generate up to a 128-bit authentication tag by using a 128-bit secret key and a 128-bit initial value (IV).

There are some attacks against ACORN. Meicheng et al. showed the slid properties of ACORN v1 and used it to recover the internal state of ACORN v1 by means of guess-and-determine and differential-algebraic technique [5]. But the attack was worse than a brute force attack. Chaigneau et al. described an attack that allowed an instant key recovery when the nonce was reused to encrypt a small amount of chosen plaintexts [6]. Johymalyo and Sarkar kept the key and IV unchanged, then modified the associated data, and then found that the associated data did not affect any keystream bits if they had a small size [7]. Salam et al. investigated cube attacks against both ACORN v1 and v2 up to 477 initialization rounds which was far from threatening the real-life usage of the cipher [8]. Salam et al. developed an attack to find a collision of internal states when the key was known [9]. Frédéric et al. claimed that they developed practical attacks to recover the internal state and secret key, which were much more expensive than the brute force attack [10]. Dibyendu and Mukhopadhyay gave some results on ACORN [11]; one of them was that they found a probabilistic linear relation between plaintext bits and ciphertext bits, which held with probability $1/2 + 1/2^{350}$. The bias was too small to be tested. The other result was that they could recover the initial state of the cipher with complexity approximately equalling 2^{40}, which was done under an impractical assumption. The designer gave the comments on the analysis of ACORN in (https://groups.google.com/forum/#!topic/crypto-competitions/dzzNcybqFP4), which show that some of the attacks are not really attacks. Since fault differential attack is one of

side channel attacks working on physical implementations, it is interesting to apply side channel cryptanalysis to a cryptographic algorithm that is being used or will be used in reality. In [12], the authors shows that with 9 faults experiments, they can recover the initial state. However, the length of keystream bits they use is 1200, which mean that the optimizing SAT solver they used can solve the equations with very high degrees, as the equations they used are output functions and the feedback functions. So far, there are not any results of fault differential attacks on ACORN. In this paper we introduce a fault attack on ACORN v2.

Fault attack is one of the most powerful tools to retrieve the secret key of many cryptographic primitives due to the work of [13]. In [14], Hoch and Shamir first introduced the fault attack on stream ciphers. They showed that a typical fault attack allows an attacker to inject faults by means of laser shots/clock glitches [15, 16] into a device initialized by a secret key and change one or more bits of its internal state. Then he or she could deduce some information about the internal state or secret key by analyzing the difference between the faulty device and the right device. A number of recent works have shown that stream ciphers are vulnerable against fault attacks. In 2008, Michal and Bohuslav showed a differential fault attack on Trivium in [17]. In 2011, Mohamed et al. improved Michal and Bohuslav's attack by a SAT solver in [18]. In 2009, Castagnos et al. gave a fault analysis of Grain-128 by targeting the LFSR in [19]. Karmakar and Chowdhury also showed an attack of Grain-128 but by targeting the NFSR in [20]. Later on, Banik et al. presented a differential fault attack on the Grain family [21, 22]. In 2013, Banik and Maitra evaluated the security of MICKEY 2.0 against fault attacks in [23], and in 2015, Banik et al. gave its improvement in [24].

In this work we present a differential fault attack on ACORN v2. As there are not any practical attacks against the security of the second version of ACORN so far, the attack present in our paper is still of interest. Our basic idea is coming from the signature based model proposed in [19]. The main difference is that we use a new method to compute the signature vectors which are differential strings in our paper. Omitting the 0 components, we represent the differential string only as the sequence of positions where their corresponding components are either 1 or nonconstant functions on the initial state. We have added these statements in our paper. Our attack is based on a general fault model where a fault is injected into the initial state of ACORN v2 randomly, and our main idea is based on the observation that the first 99-bit keystream of ACORN v2 can be expressed as linear or quadratic functions of the initial state, which helps us retrieve enough linear equations to recover the initial state. Our attack consists of two main steps: fault locating and equation solving. At the first step, after a fault is injected into the initial state randomly, we can locate it with probability 97.08% by a 99-bit differential string between the error and correct keystream bits. If the string cannot determine the fault location uniquely, then it can determine at most 20 optional fault locations. Subsequently, some improvements are provided to increase the probability of fault locating and reduce the number of optional fault locations, including keystream extension, high probability priority, and making-the-most-use-of-things. At the second step, we give two methods of solving the equation system retrieved at the first step: linearization and guess-and-determine. The time complexity of our attack is not larger than $c \cdot 2^{179.19-1.76N}$ at worst, where N is the number of fault injections such that $31 \le N \le 88$ and c is the time complexity of solving linear equations.

The rest of this paper is organized as follows. In Section 2 a brief description of ACORN v2 is provided. In Section 3 we present a fault attack on ACORN v2 and further give a forgery attack on it. Finally, Section 4 concludes the paper.

2. Description of ACORN v2

We will recall ACORN v2 briefly in this section; for more details one can refer to [3]. Since our attack does not involve the procedures of the initialization, the process of associated data, and the finalization, here we do not intend to introduce them and just restate the encryption procedure briefly.

Denote by $s = (s_0, s_1, \ldots, s_{292})$ the initial state of ACORN v2, that is, the state of the FSR after initialization and immediately before the keystream bits are outputted, and p the plaintext. There are three functions used in the encryption procedure of ACORN v2: the feedback function $f(s, p)$, the state update function $F(s, p)$, and the filter function $g(s)$. As is implied by its name, the feedback function $f(s, p)$ mainly involves in the feedback computation of the FSR and is defined as

$$\begin{aligned} f(s,p) = {} & 1 \oplus s_0 \oplus s_{61} \oplus s_{107} \oplus s_{196} \oplus s_{23}s_{160} \oplus s_{23}s_{244} \\ & \oplus s_{160}s_{244} \oplus s_{66}(s_{230} \oplus s_{193} \oplus s_{196}) \\ & \oplus s_{111}(s_{230} \oplus s_{193} \oplus s_{196}) \oplus p. \end{aligned} \tag{1}$$

Introduce intermediate variables y_i $(1 \le i \le 293)$:

$$\begin{aligned} y_{293} &= f(s,p), \\ y_{289} &= s_{289} \oplus s_{235} \oplus s_{230}, \\ y_{230} &= s_{230} \oplus s_{196} \oplus s_{193}, \\ y_{193} &= s_{193} \oplus s_{160} \oplus s_{154}, \\ y_{154} &= s_{154} \oplus s_{111} \oplus s_{107}, \\ y_{107} &= s_{107} \oplus s_{66} \oplus s_{61}, \\ y_{61} &= s_{61} \oplus s_{23} \oplus s_0, \\ y_i &= s_i \\ &\quad \text{for } 1 \le i \le 292,\ i \notin \{61, 107, 154, 193, 230, 289\}. \end{aligned} \tag{2}$$

Then the state update function $F(s, p)$ can be described as

$$s_i = y_{i+1} \quad \text{for } 0 \le i \le 292. \tag{3}$$

It is easy to check that $F(s, p)$ is invertible on s when p is fixed. The filter function $g(s)$ is used to derive a keystream z and defined as

$$\begin{aligned} g(s) &= s_{12} \oplus s_{154} \oplus s_{111} \oplus s_{107} \\ &\quad \oplus (s_{61} \oplus s_{23} \oplus s_0)(s_{193} \oplus s_{160} \oplus s_{154}) \\ &\quad \oplus (s_{61} \oplus s_{23} \oplus s_0) s_{235} \\ &\quad \oplus (s_{193} \oplus s_{160} \oplus s_{154}) s_{235}. \end{aligned} \tag{4}$$

At each step of the encryption procedure, one plaintext bit p is injected into the state of the FSR, and the ciphertext c is got by p XOR z. The pseudocode of the encryption procedure is given as follows:

$$\begin{aligned} &l \leftarrow \text{the bit length of the plaintext;} \\ &\text{for } i \text{ from 0 to } l-1 \text{ do} \\ &\qquad z_i = g(s); \\ &\qquad c_i = z_i \oplus p_i; \\ &\qquad s = F(s, p_i); \\ &\text{end for} \end{aligned} \tag{5}$$

3. Fault Attack on ACORN v2

Before introducing our fault attack on ACORN v2, we first give an outline of the fault attack model described in [19].

We assume that an attacker can access the physical device of a stream cipher and knows the IV and the keystream z. The goal of the attacker is to recover the key or forge a valid tag for plaintext. In our fault attack, the following privileges are required.

(1) The attacker has the ability to reset the physical device with the original Key-IV and restart cipher operations multiple times with the same plaintext.

(2) The attacker can inject a fault into the initial state randomly before the encryption procedure but not choose the location of fault injection.

Our attack contains two main steps: fault locating and equation solving. At the first step, we will demonstrate how to determine the fault location and retrieve a system of equations on the initial state, and at the second step, we will exploit how to recover the initial state from this system of equations. Once the initial state is recovered, the forgery attack can be executed easily.

3.1. Fault Locating. In this section we will discuss how to locate a fault after it is injected into the initial state of the FSR. We first introduce a fundamental fault locating method and then provide several improvements.

3.1.1. Fundamental Fault Locating Method. Let $s = (s_0, s_1, \ldots, s_{292})$ be the initial state of the FSR and $p = (p_0, p_1, \ldots, p_{98})$ the plaintext. Denote by $[a, b]$ the closed integer interval from a to b for two integers a and b, where $a \le b$. Let $z = (z_0, z_1, \ldots, z_{98})$ and $z^i = (z_0^i, z_1^i, \ldots, z_{98}^i)$ be the correct keystream and the error keystream generated by a faulty initial state at location i, respectively, where $i \in [0, 292]$. We define a 99-bit differential string Δz^i whose jth element satisfies $\Delta z_j^i = z_j \oplus z_j^i$, where $j \in [0, 98]$. Here we just consider 99-bit differential keystream since they all can be represented as linear or quadratic functions of s. When $t = 99$, the first feedback bit of degree 2 will come to 193rd position; the degree of z_t will be 4 and the degree of the differential keystream bit may be 3. So when $0 \le t < 99$, the degrees of the differential keystream bits will not be larger than 2. There are three steps to determine the fault location.

Firstly, we get all possible Δz^i for $i \in [0, 292]$. Let

$$\begin{aligned} &A \\ &= \{0, 12, 23, 61, 66, 107, 111, 154, 160, 193, 196, 230, 235, 244\}, \end{aligned} \tag{6}$$

which is the set of all locations that can be involved in $f(s, p)$ or $g(s)$ directly. For any $i \in A$, we can get Δz^i by changing one bit s_i, whose component Δz_j^i is 0, 1, or a function on s, $j \in [0, 292]$. When $i \in [0, 292]$ and $i \notin A$, the new differences that are not the differences caused by shifting are introduced when Δs_i shifts to the locations in A. So for any $i \notin A$, Δz^i can be got directly from some $\Delta z^{i'}$ by shifting or performing a linear transformation on $\Delta z^{i'}$, where $i' \in A$. Omitting the 0 components, we represent Δz^i only as the sequence of positions where their corresponding components are either 1 or nonconstant functions on s. To better understand the method, an example is given.

Example 1. When s_0 is changed, we can get

$$\begin{aligned} \Delta z^0 = \big(&\Delta z_0^0, 0^{37}, \Delta z_{38}^0, 0^{10}, 1, 0^8, \Delta z_{58}^0, 0^2, \Delta z_{61}^0, 0^{14}, \Delta z_{76}^0, \\ &0^{10}, 1, 0^8, \Delta z_{96}^0, 0^2\big), \end{aligned} \tag{7}$$

where 0^k means k consecutive 0s, and

$$\begin{aligned} \Delta z_0^0 &= s_{154} \oplus s_{160} \oplus s_{193} \oplus s_{235}, \\ \Delta z_{38}^0 &= s_{159} \oplus s_{165} \oplus s_{192} \oplus s_{194} \oplus s_{197} \oplus s_{198} \oplus s_{231} \\ &\quad \oplus s_{273}, \\ \Delta z_{58}^0 &= s_{119} \oplus s_{173} \oplus s_{185} \oplus s_{20} \oplus s_{212} \oplus s_{214} \oplus s_{217} \oplus s_{218} \\ &\quad \oplus s_{251} \oplus s_{43} \oplus s_{58} \oplus s_{73} \oplus s_{78} \oplus s_{81}, \\ \Delta z_{61}^0 &= 1 \oplus p_3 \oplus s_{110} \oplus s_{176} \oplus s_{188} \oplus s_{199} \oplus s_{215} \oplus s_{217} \\ &\quad \oplus s_{220} \oplus s_{221} \oplus s_{114}s_{233} \oplus s_{114}s_{199} \oplus s_{114}s_{196} \oplus s_{196}s_{69} \\ &\quad \oplus s_{199}s_{69} \oplus s_{237} \oplus s_{242} \oplus s_{163}s_{247} \oplus s_{254} \oplus s_{163}s_{26} \\ &\quad \oplus s_{247}s_{26} \oplus s_3 \oplus s_{64} \oplus s_{233}s_{69}, \end{aligned}$$

Require: fault location $i \in [a, b]$, where $a-1, b+1 \in A$ and there is not any $c \in A$ satisfying $a < c < b$; the components of Δz^{a-1}
Ensure: Δz^i
(1) **for** each component $\Delta z_j^{a-1} \neq 0$, where $j \in [0, 98]$ **do**
(2) $\quad \Delta z_{j+a-1-i}^i \leftarrow \Delta z_j^{a-1}$
(3) $\quad$ **if** $\Delta z_j^{a-1} \neq 1$ **then**
(4) $\quad\quad$ **for** each variable s_k in $\Delta z_{j+a-1-i}^i$, where $k \in [0, 292]$ **do**
(5) $\quad\quad\quad s_k \leftarrow s_k^{a-1-i}$
(6) $\quad\quad\quad s_{k+a-1-i}^0 \leftarrow \underbrace{L^{-1}(L^{-1}(\cdots(L^{-1}(L^{-1}}_{a-1-i}(s_k^{a-1-i})))\cdots)$
(7) $\quad\quad\quad s_k^{a-1-i} \leftarrow s_{k+a-1-i}^0$
(8) $\quad\quad$ **end for**
(9) $\quad$ **end if**
(10) **end for**
(11) **return** Δz^i is $\Delta z_{j+a-1-i}^i$

Algorithm 1: Obtain Δz^i, $i \in [0, 292]$ and $i \notin A$.

$$\begin{aligned}
\Delta z_{76}^0 &= 1 \oplus p_{18} \oplus s_{125} \oplus s_{164} \oplus s_{170} \oplus s_{18} \oplus s_{191} \oplus s_{193} \\
&\oplus s_{195} \oplus s_{196} \oplus s_{199} \oplus s_{201} \oplus s_{202} \oplus s_{203} \oplus s_{214} \oplus s_{230} \\
&\oplus s_{232} \oplus s_{235} \oplus s_{236} \oplus s_{129}s_{248} \oplus s_{252} \oplus s_{257} \oplus s_{178}s_{262} \\
&\oplus s_{269} \oplus s_{178}s_{41} \oplus s_{262}s_{41} \oplus s_{248}s_{84} \oplus s_{129}s_{211} \\
&\oplus s_{129}s_{214} \oplus s_{211}s_{84} \oplus s_{214}s_{84} \oplus s_{79}, \\
\Delta z_{96}^0 &= (s_{159} \oplus s_{165} \oplus s_{198} \oplus s_{282})(s_{110} \oplus s_{111} \oplus s_{114} \\
&\oplus s_{116} \oplus s_{119} \oplus s_{157} \oplus s_{172} \oplus s_{178} \oplus s_{184} \oplus s_{190} \oplus s_{20} \\
&\oplus s_{211} \oplus s_{213} \oplus s_{215} \oplus s_{216} \oplus s_{219} \oplus s_{221} \oplus s_{222} \oplus s_{223} \\
&\oplus s_{230} \oplus s_{235} \oplus s_{250} \oplus s_{252} \oplus s_{255} \oplus s_{256} \oplus s_{289} \oplus s_{35} \\
&\oplus s_{43} \oplus s_{65} \oplus s_{73} \oplus s_{75} \oplus s_{78} \oplus s_{81} \oplus s_{96}).
\end{aligned} \tag{8}$$

Then omitting the 0 components, we rewrite Δz^0 as

$$\Delta z^0 = (0, 38, \underline{49}, 58, 61, 76, \underline{87}, 96), \tag{9}$$

where $\underline{i}$ $(i = 49, 87)$ means that the ith position is always 1.

For any $i \in [1, 11]$, it is easy to obtain Δz^i by shifting Δz^0. For example,

$$\begin{aligned}
\Delta z^1 &= (1, 39, \underline{50}, 59, 62, 77, \underline{88}, 97) \\
\Delta z^1 &= (1, 39, \underline{50}, 59, 62, 77, \underline{88}, 97), \\
\Delta z_1^1 &= s_{155} \oplus s_{161} \oplus s_{194} \oplus s_{236}, \\
\Delta z_{39}^1 &= s_{160} \oplus s_{166} \oplus (s_{193} \oplus s_{160} \oplus s_{154}) \oplus s_{195} \oplus s_{198} \\
&\oplus s_{199} \oplus s_{232} \oplus s_{274} = s_{154} \oplus s_{166} \oplus s_{193} \oplus s_{195} \oplus s_{198} \\
&\oplus s_{199} \oplus s_{232} \oplus s_{274}, \\
&\cdots.
\end{aligned} \tag{10}$$

Repeating the above process (see Algorithm 1),we can obtain all Δz^i $(i \in [0, 292])$, which are listed in Table 1.

Secondly, we divide Δz^i $(i \in [0, 292])$ into 99 categories denoted by B_t $(t \in [0, 98])$ according to the subscript t satisfying $\Delta z_t^i = 1$ $(t \in [0, 98])$ and $\Delta z_j^i = 0$ $(0 \leq j < t)$. For example, B_0 contains Δz^i whose first component Δz_0^i is 1. It is noticed that, for $\Delta z^0 = (0, 38, \underline{49}, 58, 61, 76, \underline{87}, 96)$, it may occur in B_0, B_{38}, and B_{49} since its first 1 may occur at position 0, 38, and 49 ($\Delta z_{49}^1 = 1$ always holds).

Finally, for a given Δz, we first determine which category it belongs to according to the position of its first 1. Then by comparing other locations of 1 appearing in Δz, we can determine all possible locations of a fault. In a very small number of cases, a single differential string can correspond to more than one fault location. Because of this, we cannot always determine the fault location uniquely.

Running through all possible Δz, we find that the proportion of strings that cannot determine the fault location uniquely is about 2.92%, and for each nonzero string, the number of optional fault locations is at most 20. So for a given string Δz, on average, we can determine the fault location uniquely with probability 97.08% (Table 2).

3.1.2. Several Improvement Strategies. In order to decrease the proportion of strings that cannot determine the fault location uniquely and reduce the number of optional fault locations, here we provide several improvement strategies.

(i) Keystream Extension Strategy. Extending keystream is a very valid method of increasing the proportion of strings determining the fault location uniquely. The longer the keystream available to us, the higher the probability of determining the unique fault location. We want to guarantee that the number of fault location candidates is less than or equal to 3. Running through the lengths of the keystream from 99 bits to 167 bits, the result shows that it is enough to choose 163 bits. We find that the proportion of strings

Table 1: Δz^i, $i \in [0, 121]$.

i	Δz^i											
0	**0**	**38**	**49**	**58**	**61**	**76**	**87**	**96**				
1	1	39	50	59	62	77	88	97				
2	2	40	51	60	63	78	89	98				
3	3	41	52	61	64	79	90					
4	4	42	53	62	65	80	91					
5	5	43	54	63	66	81	92					
6	6	44	55	64	67	82	93					
7	7	45	56	65	68	83	94					
8	8	46	57	66	69	84	95					
9	9	47	58	67	70	85	96					
10	10	48	59	68	71	86	97					
11	11	49	60	69	72	87	98					
12	**0**	**12**	**50**	**61**	**70**	**73**	**88**					
13	1	13	51	62	71	74	89					
14	2	14	52	63	72	75	90					
15	3	15	53	64	73	76	91					
16	4	16	54	65	74	77	92					
17	5	17	55	66	75	78	93					
18	6	18	56	67	76	79	94					
19	7	19	57	68	77	80	95					
20	8	20	58	69	78	81	96					
21	9	21	59	70	79	82	97					
22	10	22	60	71	80	83	98					
23	**0**	**11**	**23**	**38**	**49**	**58**	**72**	**76**	**81**	**84**	**87**	**96**
24	1	12	24	39	50	59	73	77	82	85	88	97
25	2	13	25	40	51	60	74	78	83	86	89	98
26	3	14	26	41	52	61	75	79	84	87	90	
27	4	15	27	42	53	62	76	80	85	88	91	
28	5	16	28	43	54	63	77	81	86	89	92	
29	6	17	29	44	55	64	78	82	87	90	93	
30	7	18	30	45	56	65	79	83	88	91	94	
31	8	19	31	46	57	66	80	84	89	92	95	
32	9	20	32	47	58	67	81	85	90	93	96	
33	10	21	33	48	59	68	82	86	91	94	97	
34	11	22	34	49	60	69	83	87	92	95	98	
35	12	23	35	50	61	70	84	88	93	96		
36	13	24	36	51	62	71	85	89	94	97		
37	14	25	37	52	63	72	86	90	95	98		
38	15	26	38	53	64	73	87	91	96			
39	16	27	39	54	65	74	88	92	97			
40	17	28	40	55	66	75	89	93	98			
41	18	29	41	56	67	76	90	94				
42	19	30	42	57	68	77	91	95				
43	20	31	43	58	69	78	92	96				
44	21	32	44	59	70	79	93	97				
45	22	33	45	60	71	80	94	98				
46	23	34	46	61	72	81	95					
47	24	35	47	62	73	82	96					
48	25	36	48	63	74	83	97					
49	26	37	49	64	75	84	98					
50	27	38	50	65	76	85						

Table 1: Continued.

i	Δz^i														
51	28	39	51	66	77	86									
52	29	40	52	67	78	87									
53	30	41	53	68	79	88									
54	31	42	54	69	80	89									
55	32	43	55	70	81	90									
56	33	44	56	71	82	91									
57	34	45	57	72	83	92									
58	35	46	58	73	84	93									
59	36	47	59	74	85	94									
60	37	48	60	75	86	95									
61	**0**	**38**	**46**	**49**	**58**	**61**	**76**	**84**	**87**	**92**	**95**	**96**			
62	1	39	47	50	59	62	77	85	88	93	96	97			
63	2	40	48	51	60	63	78	86	89	94	97	98			
64	3	41	49	52	61	64	79	87	90	95	98				
65	4	42	50	53	62	65	80	88	91	96					
66	**5**	**43**	**46**	**51**	**54**	**58**	**63**	**66**	**81**	**84**	**87**	**89**	**92**	**95**	**97**
67	6	44	47	52	55	59	64	67	82	85	88	90	93	96	98
68	7	45	48	53	56	60	65	68	83	86	89	91	94	97	
69	8	46	49	54	57	61	66	69	84	87	90	92	95	98	
70	9	47	50	55	58	62	67	70	85	88	91	93	96		
71	10	48	51	56	59	63	68	71	86	89	92	94	97		
72	11	49	52	57	60	64	69	72	87	90	93	95	98		
73	12	50	53	58	61	65	70	73	88	91	94	96			
74	13	51	54	59	62	66	71	74	89	92	95	97			
75	14	52	55	60	63	67	72	75	90	93	96	98			
76	15	53	56	61	64	68	73	76	91	94	97				
77	16	54	57	62	65	69	74	77	92	95	98				
78	17	55	58	63	66	70	75	78	93	96					
79	18	56	59	64	67	71	76	79	94	97					
80	19	57	60	65	68	72	77	80	95	98					
81	20	58	61	66	69	73	78	81	96						
82	21	59	62	67	70	74	79	82	97						
83	22	60	63	68	71	75	80	83	98						
84	23	61	64	69	72	76	81	84							
85	24	62	65	70	73	77	82	85							
86	25	63	66	71	74	78	83	86							
87	26	64	67	72	75	79	84	87							
88	27	65	68	73	76	80	85	88							
89	28	66	69	74	77	81	86	89							
90	29	67	70	75	78	82	87	90							
91	30	68	71	76	79	83	88	91							
92	31	69	72	77	80	84	89	92							
93	32	70	73	78	81	85	90	93							
94	33	71	74	79	82	86	91	94							
95	34	72	75	80	83	87	92	95							
96	35	73	76	81	84	88	93	96							
97	36	74	77	82	85	89	94	97							
98	37	75	78	83	86	90	95	98							
99	38	76	79	84	87	91	96								
100	39	77	80	85	88	92	97								
101	40	78	81	86	89	93	98								
102	41	79	82	87	90	94									

Table 1: Continued.

i	Δz^i															
103	42	80	83	88	91	95										
104	43	81	84	89	92	96										
105	44	82	85	90	93	97										
106	45	83	86	91	94	98										
107	**0**	**43**	**46**	**47**	**58**	**84**	**86**	**87**	**92**	**93**	**94**	**95**				
108	1	44	47	48	59	85	87	88	93	94	95	96				
109	2	45	48	49	60	86	88	89	94	95	96	97				
110	3	46	49	50	61	87	89	90	95	96	97	98				
111	**0**	**4**	**43**	**50**	**51**	**58**	**62**	**86**	**88**	**90**	**91**	**93**	**94**	**96**	**97**	**98**
112	1	5	44	51	52	59	63	87	89	91	92	94	95	97	98	
113	2	6	45	52	53	60	64	88	90	92	93	95	96	98		
114	3	7	46	53	54	61	65	89	91	93	94	96	97			
115	4	8	47	54	55	62	49	90	92	94	95	97	98			
116	5	9	48	55	56	63	67	91	93	95	96	98				
117	6	10	49	56	57	64	68	92	94	96	97					
118	7	11	50	57	58	65	69	93	95	97	98					
119	8	12	51	58	59	66	70	94	96	98						
120	9	13	52	59	60	67	71	95	97							
121	10	14	53	60	61	68	72	96	98							
122	11	15	54	61	62	69	73	97								
123	12	16	55	62	63	70	74	98								
124	13	17	56	63	64	71	75									
125	14	18	57	64	65	72	76									
126	15	19	58	65	66	73	77									
127	16	20	59	66	67	74	78									
128	17	21	60	67	68	75	79									
129	18	22	61	68	69	76	80									
130	19	23	62	69	70	77	81									
131	20	24	63	70	71	78	82									
132	21	25	64	71	72	79	83									
133	22	26	65	72	73	80	84									
134	23	27	66	73	74	81	85									
135	24	28	67	74	75	82	86									
136	25	29	68	75	76	83	87									
137	26	30	69	76	77	84	88									
138	27	31	70	77	78	85	89									
139	28	32	71	78	79	86	90									
140	29	33	72	79	80	87	91									
141	30	34	73	80	81	88	92									
142	31	35	74	81	82	89	93									
143	32	36	75	82	83	90	94									
144	33	37	76	83	84	91	95									
145	34	38	77	84	85	92	96									
146	35	39	78	85	86	93	97									
147	36	40	79	86	87	94	98									
148	37	41	80	87	88	95										
149	38	42	81	88	89	96										
150	39	43	82	89	90	97										
151	40	44	83	90	91	98										
152	41	45	84	91	92											
153	42	46	85	92	93											
154	**0**	**33**	**39**	**43**	**47**	**66**	**72**	**78**	**82**	**91**	**93**	**94**				

Table 1: Continued.

i	Δz^i																					
155	1	34	40	44	48	67	73	79	83	92	94	95										
156	2	35	41	45	49	68	74	80	84	93	95	96										
157	3	36	42	46	50	69	75	81	85	94	96	97										
158	4	37	43	47	51	70	76	82	86	95	97	98										
159	5	38	44	48	52	71	77	83	87	96	98											
160	**0**	**6**	**33**	**39**	**45**	**49**	**53**	**58**	**66**	**72**	**78**	**82**	**84**	**86**	**88**	**91**	**97**					
161	1	7	34	40	46	50	54	59	67	73	79	83	85	87	89	92	98					
162	2	8	35	41	47	51	55	60	68	74	80	84	86	88	90	93						
163	3	9	36	42	48	52	56	61	69	75	81	85	87	89	91	94						
164	4	10	37	43	49	53	57	62	70	76	82	86	88	90	92	95						
165	5	11	38	44	50	54	58	63	71	77	83	87	89	91	93	96						
166	6	12	39	45	51	55	59	64	72	78	84	88	90	92	94	97						
167	7	13	40	46	52	56	60	65	73	79	85	89	91	93	95	98						
168	8	14	41	47	53	57	61	66	74	80	86	90	92	94	96							
169	9	15	42	48	54	58	62	67	75	81	87	91	93	95	97							
170	10	16	43	49	55	59	63	68	76	82	88	92	94	96	98							
171	11	17	44	50	56	60	64	69	77	83	89	93	95	97								
172	12	18	45	51	57	61	65	70	78	84	90	94	96	98								
173	13	19	46	52	58	62	66	71	79	85	91	95	97									
174	14	20	47	53	59	63	67	72	80	86	92	96	98									
175	15	21	48	54	60	64	68	73	81	87	93	97										
176	16	22	49	55	61	65	69	74	82	88	94	98										
177	17	23	50	56	62	66	70	75	83	89	95											
178	18	24	51	57	63	67	71	76	84	90	96											
179	19	25	52	58	64	68	72	77	85	91	97											
180	20	26	53	59	65	69	73	78	86	92	98											
181	21	27	54	60	66	70	74	79	87	93												
182	22	28	55	61	67	71	75	80	88	94												
183	23	29	56	62	68	72	76	81	89	95												
184	24	30	57	63	69	73	77	82	90	96												
185	25	31	58	64	70	74	78	83	91	97												
186	26	32	59	65	71	75	79	84	92	98												
187	27	33	60	66	72	76	80	85	93													
188	28	34	61	67	73	77	81	86	94													
189	29	35	62	68	74	78	82	87	95													
190	30	36	63	69	75	79	83	88	96													
191	31	37	64	70	76	80	84	89	97													
192	32	38	65	71	77	81	85	90	98													
193	**0**	**33**	**37**	**39**	**66**	**70**	**71**	**72**	**74**	**76**	**78**	**82**	**86**	**91**	**92**	**95**						
194	1	34	38	40	67	71	72	73	75	77	79	83	87	92	93	96						
195	2	35	39	41	68	72	73	74	76	78	80	84	88	93	94	97						
196	**3**	**36**	**37**	**40**	**42**	**58**	**61**	**69**	**70**	**71**	**73**	**75**	**76**	**77**	**79**	**81**	**85**	**89**	**92**	**94**	**95**	**98**
197	4	37	38	41	43	59	62	70	71	72	74	76	77	78	80	82	86	90	93	95	96	
198	5	38	39	42	44	60	63	71	72	73	75	77	78	79	81	83	87	91	94	96	97	
199	6	39	40	43	45	61	64	72	73	74	76	78	79	80	82	84	88	92	95	97	98	
200	7	40	41	44	46	62	65	73	74	75	77	79	80	81	83	85	89	93	96	98		
201	8	41	42	45	47	63	66	74	75	76	78	80	81	82	84	86	90	94	97			
202	9	42	43	46	48	64	67	75	76	77	79	81	82	83	85	87	91	95	98			
203	10	43	44	47	49	65	68	76	77	78	80	82	83	84	86	88	92	96				
204	11	44	45	48	50	66	69	77	78	79	81	83	84	85	87	89	93	97				
205	12	45	46	49	51	67	70	78	79	80	82	84	85	86	88	90	94	98				
206	13	46	47	50	52	68	71	79	80	81	83	85	86	87	89	91	95					

Table 1: Continued.

i	Δz^i																
207	14	47	48	51	53	69	72	80	81	82	84	86	87	88	90	92	96
208	15	48	49	52	54	70	73	81	82	83	85	87	88	89	91	93	97
207	16	49	50	53	55	71	74	82	83	84	86	88	89	90	92	94	98
210	17	50	51	54	56	72	75	83	84	85	87	89	90	91	93	95	
211	18	51	52	55	57	73	76	84	85	86	88	90	91	92	94	96	
212	19	52	53	56	58	74	77	85	86	87	89	91	92	93	95	97	
213	20	53	54	57	59	75	78	86	87	88	90	92	93	94	96	98	
214	21	54	55	58	60	76	79	87	88	89	91	93	94	95	97		
215	22	55	56	59	61	77	80	88	89	90	92	94	95	96	98		
216	23	56	57	60	62	78	81	89	90	91	93	95	96	97			
217	24	57	58	61	63	79	82	90	91	92	94	96	97	98			
218	25	58	59	62	64	80	83	91	92	93	95	97	98				
219	26	59	60	63	65	81	84	92	93	94	96	98					
220	27	60	61	64	66	82	85	93	94	95	97						
221	28	61	62	65	67	83	86	94	95	96	98						
222	29	62	63	66	68	84	87	95	96	97							
223	30	63	64	67	69	85	88	96	97	98							
224	31	64	65	68	70	86	89	97	98								
225	32	65	66	69	71	87	90	98									
226	33	66	67	70	72	88											
227	34	67	68	71	73	89											
228	35	68	69	72	74	90											
229	36	69	70	73	75	91											
230	**37**	**54**	**58**	**70**	**71**	**74**	**76**	**92**	**95**	**96**							
231	38	55	59	71	72	75	77	93	96	97							
232	39	56	60	72	73	76	78	94	97	98							
233	40	57	61	73	74	77	79	95	98								
234	41	58	62	74	75	78	80	96									
235	**0**	**42**	**54**	**59**	**63**	**75**	**76**	**79**	**81**	**96**	**97**						
236	1	43	55	60	64	76	77	80	82	97	98						
237	2	44	56	61	65	77	78	81	83	98							
238	3	45	57	62	66	78	79	82	84								
239	4	46	58	63	67	79	80	83	85								
240	5	47	59	64	68	80	81	84	86								
241	6	48	60	65	69	81	82	85	87								
242	7	49	61	66	70	82	83	86	88								
243	8	50	62	67	71	83	84	87	89								
244	**9**	**51**	**58**	**63**	**68**	**72**	**84**	**85**	**88**	**90**							
245	10	52	59	64	69	73	85	86	89	91							
246	11	53	60	65	70	74	86	87	90	92							
247	12	54	61	66	71	75	87	88	91	93							
248	13	55	62	67	72	76	88	89	92	94							
249	14	56	63	68	73	77	89	90	93	95							
250	15	57	64	69	74	78	90	91	94	96							
251	16	58	65	70	75	79	91	92	95	97							
252	17	59	66	71	76	80	92	93	96	98							
253	18	60	67	72	77	81	93	94	97								
254	19	61	68	73	78	82	94	95	98								
255	20	62	69	74	79	83	95	96									
256	21	63	70	75	80	84	96	97									
257	22	64	71	76	81	85	97	98									
258	23	65	72	77	82	86	98										

Table 1: Continued.

i	Δz^i					
259	24	66	73	78	83	87
260	25	67	74	79	84	88
261	26	68	75	80	85	89
262	27	69	76	81	86	90
263	28	70	77	82	87	91
264	29	71	78	83	88	92
265	30	72	79	84	89	93
266	31	73	80	85	90	94
267	32	74	81	86	91	95
268	33	75	82	87	92	96
269	34	76	83	88	93	97
270	35	77	84	89	94	98
271	36	78	85	90	95	
272	37	79	86	91	96	
273	38	80	87	92	97	
274	39	81	88	93	98	
275	40	82	89	94		
276	41	83	90	95		
277	42	84	91	96		
278	43	85	92	97		
279	44	86	93	98		
280	45	87	94			
281	46	88	95			
282	47	89	96			
283	48	90	97			
284	49	91	98			
285	50	92				
286	51	93				
287	52	94				
288	53	95				
289	54	96				
290	55	97				
291	56	98				
292	57					

Table 2: Δz^i in B_0.

Fault location i	Δz^i
0	0 38 49 58 61 76 87 96
12	0 12 50 61 70 73 88
23	0 11 23 38 49 58 72 76 81 84 87 96
61	0 38 46 49 58 61 76 84 87 92 95 96
107	0 43 46 47 58 84 86 87 92 93 94 95
111	0 4 43 50 51 58 62 86 88 90 91 93 94 96 97 98
154	0 33 39 43 47 66 72 78 82 91 93 94
160	0 6 33 39 45 49 53 58 66 72 78 82 84 86 88 91 97
193	0 33 37 39 66 70 71 72 74 76 78 82 86 91 92 95
235	0 42 54 59 63 75 76 79 81 96 97

that cannot determine the fault location uniquely depends mostly on the fault locations in [230, 292]. One of the main reasons is that there is not any components of the differential strings that can always be 1 when the fault locations belong to [230, 292]. This is because the diffusion ability of the last 63 register bits is stronger than that of the first 230 register bits.

Here we extend the keystream to at most 167 bits and divide all possible fault positions into two parts: [0, 229] and [230, 292]. When a fault is injected in s_i, where $i \in [0, 229]$, we can get an approximate distribution of differential strings on the numbers of optional fault locations by Algorithm 2, seen in Table 3. It is found that when the length of keystream is extended to 163 bits, the proportion of strings not locating a fault is decreased to 0.0650% and the number of optional fault locations is reduced to at most 3. We make a similar process for a fault location in [230, 292], seen in Table 4. It is seen that when the keystream length reaches 163 bits, the proportion of all zero strings can almost reduce to 0, but the proportion

Table 3: The distribution of the strings (fault injected in s_i, $i \in [0, 229]$).

ksl	nup (%)	2nup (%)	3nup (%)	4nup (%)	5nup (%)	6nup (%)	7nup (%)	Others (%)
99	4.1213	69.88	13.52	2.24	1.96	0.38	0.61	11.2
103	1.7420	53.91	20.29	7.93	2.40	1.30	1.93	12.24
107	1.2722	58.60	18.70	6.60	2.59	2.72	1.93	8.87
111	1.0206	61.06	20.36	2.95	3.33	3.48	1.33	7.49
115	0.8569	61.57	21.40	2.59	2.13	4.54	1.65	6.12
119	0.6744	67.69	18.33	2.59	1.74	4.02	1.36	4.28
123	0.4512	73.92	14.58	2.92	1.42	3.55	0.76	2.85
127	0.2991	74.33	16.55	2.33	1.75	2.26	0.70	2.07
131	0.2225	85.77	9.26	2.44	0.43	1.11	0.56	0.43
135	**0.1582**	**90.48**	**6.69**	**1.81**	**0.48**	**0.54**	**0**	**0**
139	0.1518	89.38	8.42	1.32	0.75	0	0.13	0
143	0.1320	91.98	6.72	1.08	0.07	0	0.14	0
145	0.1275	89.60	9.27	0.60	0.15	0.07	0.30	0
147	0.1174	89.93	7.80	1.79	0.08	0.16	0.24	0
151	**0.0770**	**90.83**	**8.80**	**0.37**	**0**	**0**	**0**	**0**
155	0.0773	92.73	6.04	0.86	0.25	0	0.12	0
159	0.0649	94.71	4.55	0.73	0	0	0	0
163	**0.0650**	**94.43**	**5.57**	**0**	**0**	**0**	**0**	**0**
167	0.0548	94.43	5.57	0	0	0	0	0

ksl: the length of keystream; nup: the proportion of the strings not locating a fault; $i$nup ($i = 2, \ldots, 7$): the proportion of the strings that can determine i optional fault locations among all strings not locating a fault; others: the proportion of the strings that can determine more than 7 optional fault locations among all strings not locating a fault.

(1) Choose 2^{15} initial states randomly
(2) **for** each initial state **do**
(3) proceed the encryption phase of ACORN v2 to get a 180-bit keystream z
(4) Choose 32 fault locations i randomly, where $i \in [0, 229]$
(5) **for** each fault locations i **do**
(6) $s_i \leftarrow s_i \oplus 1$
(7) proceed the encryption phase of ACORN v2 to get a 180-bit keystream z^i
(8) **for** different length of keystream from 99 to 180 **do**
(9) determine the fault location i with Δz^i
(10) calculate the number of optional fault locations
(11) **end for**
(12) **end for**
(13) **end for**
(14) **return** the numbers of optional fault locations

Algorithm 2

of the strings not locating a fault only decreases to 14.45%. How to use the strategy in our fault locating method will be described in the improved fault locating method.

(ii) High Probability Priority Strategy. Here we assume that the initial state of the FSR is random and uniformly distributed. For a given string Δz, we find that different fault location candidates appear with different probabilities. For example, when we get

$$\Delta z = \left(\overbrace{0, \ldots, 0}^{85}, 1, \overbrace{0, \ldots, 0}^{13} \right) = (\underline{85}), \tag{11}$$

since each candidate i in B_{85} needs to satisfy $\Delta z_j^i = 0$, where $j \in [0, 98]$ and $j \neq 85$, by the expression of Δz^i, it is known that i takes 278 with probability 2^{-3}, but 239 with probability 2^{-8} (the probabilities of all candidates i in B_{85} are listed in Table 5). For each candidate i, we prefer to choose i with higher probability and call it high probability priority strategy.

(iii) Cross-Referencing Strategy. Cross-referencing is a common maximized way. Here we adopt it to decrease the proportion of strings not locating a fault. Indeed, there are some inherent relations among the strings got from faults at distinct locations. For a new string Δz, it is helpful to

Table 4: The distribution of the strings (fault injected in s_i, $i \in [230, 292]$).

ksl	nup	n0s	2nup	3nup	4nup	5nup	6nup	7nup	8nup	9nup	10nup	Others
99	37.15	18.93	25.72	14.42	8.86	0.90	1.69	0.54	0.99	0.62	8.72	18.60
103	33.82	14.37	29.09	14.29	9.80	1.61	1.84	0.30	0.69	0.41	11.23	16.39
107	31.65	10.98	29.88	15.63	10.64	3.35	1.75	0.56	1.44	0.69	8.74	16.33
111	29.11	7.90	31.26	17.01	9.77	4.41	2.94	1.33	1.40	0.80	7.18	16.00
115	27.62	5.31	33.38	17.93	8.71	6.50	2.25	3.02	1.29	0.72	5.32	15.57
119	26.71	4.02	38.44	15.61	8.23	6.74	3.04	3.71	1.06	0.95	3.73	14.47
123	24.88	2.76	37.87	15.75	10.30	6.82	3.09	3.79	2.27	1.63	3.13	12.59
127	22.49	1.74	36.96	17.01	10.12	7.33	3.99	3.84	1.78	1.37	2.29	13.57
131	19.94	1.38	37.27	19.45	10.84	6.86	3.63	4.55	1.93	1.32	2.17	10.62
135	18.18	0.60	37.18	20.66	12.49	7.32	5.02	3.04	1.96	2.33	1.43	7.96
139	17.60	0.29	39.59	21.04	11.19	7.04	4.60	3.36	2.72	1.77	1.53	6.87
143	17.06	0.14	37.66	21.83	11.15	7.55	4.80	3.76	2.06	1.81	1.77	7.48
145	17.13	0.23	39.48	21.47	11.22	6.68	4.51	3.19	2.28	2.21	1.64	7.09
147	16.67	0.24	39.84	22.62	11.99	6.33	4.48	2.71	2.12	1.83	1.72	6.11
151	15.85	0.04	42.32	21.74	11.78	6.37	4.16	3.54	1.89	1.58	1.04	5.54
155	15.24	0.02	41.93	22.51	10.63	6.59	3.64	3.60	2.06	1.64	1.58	5.79
159	15.27	0.12	43.73	23.18	10.25	7.27	3.68	2.54	2.28	1.38	0.86	4.72
163	14.45	0	45.24	24.01	9.57	5.72	4.56	2.37	2.13	1.54	1.27	3.59
167	14.20	0	45.14	24.16	10.43	6.19	3.74	2.71	1.83	1.42	1.10	3.29

In order to shorten the table, we omit the (%) in each column. The symbols are the same as in Table 3, except n0s: the proportion of the zero string among all strings not locating a fault; others: the proportion of the strings that can determine more than 10 optional fault locations among all strings not locating a fault.

Table 5: Optional fault locations of $\Delta z^i = (\underline{85})$.

Fault location	Δz^i										Probability
278	43	**85**	92	97							2^{-3}
271	36	78	**85**	90	95						2^{-4}
266	31	73	80	**85**	90	94					2^{-5}
261	26	68	75	80	**85**	89					2^{-5}
257	22	64	71	76	81	**85**	97	98			2^{-7}
245	10	52	59	64	69	73	**85**	86	89	91	2^{-9}
241	6	48	60	65	69	81	82	**85**	87		2^{-8}
244	9	51	58	63	68	72	84	**85**	88	90	2^{-9}
239	4	46	58	63	67	79	80	83	**85**		2^{-8}

make the most use of knowledge retrieved from old strings to locate a new fault. The following three observations on the filter function $g(s)$ will help us to execute the above strategy.

Observation 1. For any $j \in [0, 57]$, by the first nonconstant component of Δz^{235+j}, we have

$$\begin{aligned}
\Delta z_j^{235+j} &= \Delta z_j^{61+j} \oplus \Delta z_j^{193+j} = \Delta z_j^{61+j} \oplus \Delta z_j^{160+j} \\
&= \Delta z_j^{61+j} \oplus \Delta z_j^{154+j} = \Delta z_j^{23+j} \oplus \Delta z_j^{193+j} \\
&= \Delta z_j^{23+j} \oplus \Delta z_j^{160+j} = \Delta z_j^{23+j} \oplus \Delta z_j^{154+j} \\
&= \Delta z_j^{j} \oplus \Delta z_j^{193+j} = \Delta z_j^{j} \oplus \Delta z_j^{160+j} \\
&= \Delta z_j^{j} \oplus \Delta z_j^{154+j}.
\end{aligned} \tag{12}$$

Observation 2. For any $j \in [0, 19]$, by the second nonconstant component of Δz^{230+j}, we have $\Delta z_{37+j}^{230+j} = \Delta z_{37+j}^{160+37+j} = \Delta z_{37+j}^{154+37+j}$.

Observation 3. For any $j \in [0, 3]$, by the third nonconstant component of Δz^{230+j}, we have $\Delta z_{54+j}^{235+j} = \Delta z_{54+j}^{289+j}$.

For example, when we get $\Delta z = (\underline{85})$, candidates i are listed in Table 5. If we have located the fault at s_{65} and s_{197} which satisfy $\Delta z_4^{65} \oplus \Delta z_4^{197} = 1$, we can exclude the candidate $i = 239$. Because if the candidate $i = 239$ is the fault location, Δz_4^{239} should be 1.

3.1.3. Improved Fault Locating Method. Here we present an improvement of the fundamental fault locating method by means of the above optimized strategies. For a given 99-bit Δz, we first determine which category it belongs to according to the position of its first 1. Then by comparing other locations of 1 appearing in Δz, on average, we can locate the fault with probability 97.08%. If Δz cannot locate the fault, we adopt the keystream extension strategy and extend the keystream to at most 163 bits. After this step, the fault has been located with probability 99.95%. If Δz cannot still locate the fault, we will first use the making-the-most-use-of-things strategy to exclude some candidates and then use the high probability priority strategy to guess the right fault location. At last we can locate the fault with probability almost 1. For more detail, see Algorithm 3.

3.2. Recovering the Initial State. Once a fault is located, we will retrieve some equations on the initial state s. When

Require: A 99-bit differential string Δz
Ensure: the fault locations
(1) Determine which category Δz belongs to according to the position of its first 1
(2) Determine the candidates by comparing other locations of 1 appearing in Δz and using the making-the-most-use-of-things strategy
(3) **if** the number of candidates is 1 **then**
(4) **return** the unique candidate
(5) **else**
(6) **for** the keystream length extended to 99 + i bits, i from 1 to 64 **do**
(7) use the making-the-most-use-of-things strategy
(8) compare the extra i locations of 1 appearing in Δz
(9) **if** the number of candidates can be reduced to 1 **then**
(10) **return** the unique candidate
(11) **end if**
(12) **end for**
(13) **if** the number of candidates is still larger than 1 **then**
(14) use the high probability priority strategy to choose the location i' that the string appears in the $\Delta z^{i'}$ with the highest probability
(15) **return** the unique candidate i'
(16) **end if**
(17) **end if**

Algorithm 3

the number of equations is enough, we can recover s from them. Below we show how to retrieve equations and provide two equation solving methods: linearization and guess-and-determine.

3.2.1. Equation Retrieving. As shown in fundamental fault locating method, we just consider 99-bit differential keystream since they all can be represented as linear or quadratic functions of s. We first get differential equations when fault is injected in s_i, where $i \in A$,

$$A = \{0, 12, 23, 61, 66, 107, 111, 154, 160, 193, 196, 230, 235, 244\}. \tag{13}$$

When $i \in [0, 292]$ and $i \notin A$, the main idea to retrieve differential equations is to shift or perform the inversion of the linear transformation on $\Delta z^{i'}$, where $i' \in A$. For more detail, one can see Example 1. Note that the inversion of the linear transformation will not lead to the transformation of a linear function to a nonlinear function but increase the number of terms in the function (ignoring possible cancellations due to the exclusive OR operation).

For each fault location i, where $i \in [0, 292]$, we have stored the corresponding equations containing both linear and quadratic equations. When one fault experiment is executed, we first judge the fault location and then find the corresponding equations according to the differential string. In order to recover the initial state, next, we will show two methods to solve the equations.

3.2.2. Linearization Method. Our basic idea is to retrieve as many linear equations as possible and then solve the system of linear equations to get s. At first, one observation of the functions used in ACORN v2 is given.

Observation 4. Let

$$y = x_i x_j \oplus x_i x_k \oplus x_j x_k, \tag{14}$$

where x_i, x_j, and x_k are linear functions of the initial state. Then we have

$$\Pr\left[y = x_i\right] = \frac{3}{4}, \qquad \Pr\left[y = x_j = x_k \mid y \neq x_i\right] = 1. \tag{15}$$

If we have n_1 equations of the forms (14), we can get n_1 linear equations with probability $(3/4)^{n_1}$.

According to the expressions of Δz_j^i and the functions used in ACORN v2, where $i \in [0, 292]$ and $j \in [0, 98]$, we get the following propositions:

(P1) The first 58-bit keystream without fault injection are quadratic functions of the initial state and the quadratic terms are of the forms (14). So we can get 58 linear equations with probability

$$\left(\frac{3}{4}\right)^{58} \approx 2^{-24}. \tag{16}$$

(P2) Consider Δz_j^i that can be expressed as quadratic functions of s. There are two forms of quadratic functions Δz_j^i which are

$$x_{k_1} \oplus x_{i_1} x_{j_1} \oplus x_i x_j \oplus x_i x_k \oplus x_j x_k \tag{17}$$

and $x_i x_j$, where x_{i_1}, x_{j_1}, x_{k_1}, x_j, x_j, and x_k are linear functions of s. According to Observation 4, the term

$x_ix_j \oplus x_ix_k \oplus x_jx_k$ can be linearized as x_i with probability 3/4 and $x_{i_1}x_{j_1}$ can be linearized as 0 or x_{j_1} by guessing the value of x_{i_1} with probability 1/2. So (17) can be linearized as $x_{k_1} \oplus x_i$ or $x_{k_1} \oplus x_{j_1} \oplus x_i$ with probability $3/4 \cdot 1/2$ and provide two linear equations. For quadratic function of form x_ix_j, if $x_ix_j = 1$, we know that $x_i = 1$ and $x_j = 1$. If $x_ix_j = 0$, we guess the values of x_i and x_j with probability 1/3. So by guessing the value of x_i and x_j, we can get 2 linear equations with probability 1/2.

(P3) For all Δz^i ($i \in [0, 292]$), we calculate the numbers of the quadratic functions of form (17) and x_ix_j. On average, the numbers of linear equations, quadratic equations of form (17), and quadratic equations of form x_ix_j are 2.7, 3.3, and 1.2 for each Δz^i, respectively. According to (P2), we can get $11.7 = 2.7+3.3\times2+1.2\times$ 2 linear equations and 3.3 simple quadratic equations with probability

$$\left(\frac{1}{2}\right)^{4.5} \cdot \left(\frac{3}{4}\right)^{3.3} \approx 2^{-5.87}. \tag{18}$$

Based on the above observations, we can retrieve enough linear equations to recover s. By (P1), about 58 linear equations can be retrieved with probability 2^{-24}, and by (P3), about 11.7 linear equations with probability $2^{-5.87}$ for each fault. Let n be the number of fault experiments. In order to guarantee the probability of recovering s is larger than 2^{-128}, n should satisfy

$$2^{-5.87n-24} > 2^{-128}; \tag{19}$$

that is, $n \leq 17$. The remaining $235 - 11.7n$ linear equations will be given by $\lceil (235 - 11.7n)/2.7 \rceil$ new fault experiments. Thus the total number N of fault experiments is

$$\left\lceil \frac{235 - 11.7n}{2.7} \right\rceil + n = \lceil 87.04 - 3.33n \rceil, \quad n \leq 17. \tag{20}$$

Replace n by N in $2^{-5.87n-24}$; the probability of recovering s is $2^{-179.19+1.76N}$. In particular, when $n = 0$, 88 fault experiments are needed and the probability is 2^{-24}. When $n = 17$, 31 fault experiments are needed and the probability is $2^{-124.63}$. As fault injection is hard work and each fault experiment would damage the device, we hope the number of fault experiments required should be as small as possible. 31 fault experiments is the smallest number in our attack.

Below we roughly estimate the time complexity of recovering s with probability 1. When N fault experiments are carried out, denote by X the random event of recovering s. Then X follows a binomial distribution with parameters $n' \in \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers) and $p = 2^{-179.19+1.76N}$, denoted by $X \sim B(n', p)$. If the expected value of X is 1, the expected value of n' is about $2^{179.19-1.76N}$, where $31 \leq N \leq 88$. Actually, the value of n' is smaller than $2^{179.19-1.76N}$. As shown in Observation 4, if the first experiment is failed with probability $1 - p$, the success probability of the next experiment becomes $(4/3)p$. So, the time complexity of recovering s with probability 1 is smaller than $c \cdot 2^{179.19-1.76N}$, where c is the time complexity of solving linear equations and N is the number of fault experiments such that $31 \leq N \leq 88$. By the birthday paradox, there is a high chance of randomly chosen locations being repeated by the time $\sqrt{293} \approx 17$ experiments are performed, so the number of actual experiments required to obtain N distinct fault locations will be rather higher than N.

3.2.3. Guess-and-Determine Method. Here we discuss the complexity of solving the above equation system by guess-and-determine method. For one fault experiment, on average, we can get 4.5 quadratic equations including 1.2 quadratic equations of form x_ix_j and 2.7 linear equations as shown in (P2). The quadratic function of form x_ix_j can be regarded as one linear equation. For quadratic function of form x_ix_j, it is expected to obtain 1 linear equation. If $x_ix_j = 1$, we know that $x_i = 1$ and $x_j = 1$. If $x_ix_j = 0$, we guess the values of x_i and x_j with probability 1/3. So by guessing the value of x_i and x_j, we can get 2 linear equations with probability 1/2. So for one fault experiment, on average, we can get 3.3 linear equations and 3.3 quadratic equations. So we can get 295 equations with 160 linear equations with 41 fault experiments. By guessing 67-bit value, the initial state s can be recovered. The time complexity of recovering s is $c \cdot 2^{67}$, where c is the time complexity of solving linear equations.

3.2.4. Implementation and Verification. To prove the validity of our guess-and-determine method, we experimentally test it on a shrunk cipher with similar structure and properties. More specifically, we built a small stream cipher according to the design principles used for ACORN but with a small state of 31 bits. We then implemented our attack to recover the initial state.

Denote by $s = (s_0, s_1, \ldots, s_{30})$ the initial state of the toy cipher and p the plaintext. The feedback function $f(s, p)$ is defined as

$$\begin{aligned} f(s,p) = {} & 1 \oplus s_0 \oplus s_8 \oplus s_{12} \oplus s_{21} \oplus s_3s_{18} \oplus s_3s_{27} \\ & \oplus s_{18}s_{27} \oplus s_{14}(s_{23} \oplus s_{21} \oplus s_{20}) \\ & \oplus s_9(s_{23} \oplus s_{21} \oplus s_{20}) \oplus p. \end{aligned} \tag{21}$$

Introduce intermediate variables y_i ($1 \leq i \leq 31$):

$$\begin{aligned} y_{31} &= f(s,p), \\ y_{29} &= s_{29} \oplus s_{26} \oplus s_{23}, \\ y_{23} &= s_{23} \oplus s_{21} \oplus s_{20}, \\ y_{20} &= s_{20} \oplus s_{18} \oplus s_{17}, \\ y_{17} &= s_{17} \oplus s_{14} \oplus s_{12}, \\ y_{12} &= s_{12} \oplus s_9 \oplus s_8, \\ y_8 &= s_8 \oplus s_3 \oplus s_0, \\ y_i &= s_i \quad \text{for } 1 \leq i \leq 30,\ i \notin \{8, 12, 17, 20, 23, 29\}. \end{aligned} \tag{22}$$

Then the state update function $F(s, p)$ can be described as

$$s_i = y_{i+1} \quad \text{for } 0 \le i \le 30. \tag{23}$$

The filter function $g(s)$ is used to derive a keystream z and defined as

$$\begin{aligned} g(s) = {} & s_1 \oplus s_{17} \oplus s_{14} \oplus s_{12} \\ & \oplus (s_8 \oplus s_3 \oplus s_0)(s_{20} \oplus s_{18} \oplus s_{17}) \\ & \oplus (s_8 \oplus s_3 \oplus s_0) s_{26} \oplus (s_{20} \oplus s_{18} \oplus s_{17}) s_{26}. \end{aligned} \tag{24}$$

The encryption procedure of the toy cipher is the same as that of ACORN v2.

Here we just consider the first 9-bit keystream, since the first 9-bit differential keystream can be represented as linear or quadratic functions of s. Statistic shows that for one fault experiment, on average, we can get 2.3 linear equations and 1.5 quadratic equations. So with 9 fault experiments, we can get 34 equations where there are 21 linear equations. By guessing 5-bit value, the initial state s can be recovered. Based on heuristic, the time complexity of recovering s is $c \cdot 2^5$, where c is the time complexity of solving linear equations. Next, we will provide some experimental results.

Assume that the initial state is

$$\begin{aligned} s = {} & (0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, \\ & 1, 1, 1, 0, 0, 1, 0) \end{aligned} \tag{25}$$

and the 9 fault locations have been located which are

$$s_1, s_7, s_8, s_{12}, s_{14}, s_{17}, s_{21}, s_{25}, s_{29}. \tag{26}$$

Totally, we can get 20 linearly independent linear equations and 6 quadratic equations with respect to the initial state. By guessing the values of s_{31}, s_{32}, s_{33}, and s_{34}, the 6 quadratic equations can be simplified as linear equations and provide 4 new quadratic equations. Using Gaussian elimination method and guessing one bit more, the 26 linear equations and 4 quadratic equations can be solved easily. The time complexity c is the sum of the Gaussian elimination about 26 linear equations and solving the 4 quadratic equations with 4 variables. There are some differences in the value of c comparing to our estimation. So the time complexity in the realistic attack may be higher than that of our estimation. We also try several other fault locations and the result shows that if the linearly independent equations are enough, we can always recover the initial state. Of course, if the linearly independent equations are not enough, we need to carry out more fault experiments.

3.3. Forgery Attack. Once the initial state of ACORN v2 is recovered, we can encrypt any message to get the ciphertext and generate a valid tag for it. In other words, we can forge tags for any plaintext. It should be pointed out that our attack is suitable to ACORN v1 as well. Due to the invertibility of the initial process in ACORN v1, we can further recover its secret key.

4. Conclusion

In this work we present a fault attack on ACORN v2 which is one of the second round candidates of CAESAR. Our results show that we can locate almost all faults and recover the initial state with at least 41 fault experiments, whose time complexity is $c \cdot 2^{67}$, where c is the time complexity of solving linear equations.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant no. 61379139 and Grant no. 61572491) and the "Strategic Priority Research Program" of the Chinese Academy of Sciences (Grant no. XDA06010701).

References

[1] CAESAR, "Cryptographic competitions," http://competitions.cr.yp.to/index.html.

[2] W. Hongjun, *ACORN: A Lightweight Authenticated Cipher (v1)*, CAESAR, 2014.

[3] W. Hongjun, *ACORN: A Lightweight Authenticated Cipher (v2)*, CAESAR, 2015.

[4] W. Hongjun, *ACORN, A Lightweight Authenticated Cipher (v3)*, CAESAR, 2016.

[5] L. Meicheng and L. Dongdai, "Cryptanalysis of Lightweight Authenticated Cipher ACORN," *Posed on the crypto-competition mailing list*, 2014.

[6] C. Chaigneau, F. Thomas, and H. Gilbert, "Full Key-recovery on ACORN in Nonce-reuse and Decryption-misuse settings," *Posed on the crypto-competition mailing list*, 2015.

[7] J. Johymalyo and S. Sarkar, "Some observations on ACORN v1 and Trivia-SC," in *Proceedings of the Lightweight Cryptography Workshop 2015*, National Institute of Standards and Technology, Gaithersburg, Maryland, Md, USA, 2015.

[8] M. I. Salam, H. Bartlett, E. Dawson, J. Pieprzyk, L. Simpson, and K. K.-H. Wong, "Investigating cube attacks on the authenticated encryption stream cipher ACORN," *Communications in Computer and Information Science*, vol. 651, pp. 15–26, 2016.

[9] M. I. Salam, L. Simpson, K. K.-H. Wong, E. Dawson, H. Bartlett, and J. Pieprzyk, "Finding state collisions in the authenticated encryption stream cipher ACORN," in *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2016*, aus, February 2016.

[10] L. Frédéric, L. Lerman, M. Olivier, and V. H. Dirk, *SAT-based cryptanalysis of ACORN*, 2016.

[11] R. Dibyendu and S. Mukhopadhyay, "Some results on ACORN," *IACR Cryptology ePrint Archive, 1132, 2016*.

[12] A. A. Siddhanti, S. Sarkar, S. Maitra, and A. Chattopadhyay, "Differential Fault Attack on Grain v1, ACORN v3 and Lizard," *Cryptology ePrint Archive: Report 2017/678*, 2017.

[13] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97*, vol.

1294 of *Lecture Notes in Computer Science*, pp. 513–525, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[14] J. J. Hoch and A. Shamir, "Fault analysis of stream ciphers," in *Cryptographic Hardware and Embedded Systems—CHES 2004*, M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*, pp. 240–253, Springer, Berlin, Germany, 2004.

[15] S. Skorobogatov, "Optically Enhanced Position-Locked Power Analysis," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, vol. 4249 of *Lecture Notes in Computer Science*, pp. 61–75, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[16] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, vol. 2523 of *Lecture Notes in Computer Science*, pp. 2–12, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[17] H. Michal and R. Bohuslav, "Differential Fault Analysis of Trivium," in *Proceedings of the Fast Software Encryption, 15th International Workshop, FSE 2008*, Lausanne, Switzerland, February 2008.

[18] S. Mohamed, S. Bulygin, and J. A. Buchmann, "Using SAT Solving to Improve Differential Fault Analysis of Trivium," in *Proceedings of the Information Security and Assurance - International Conference, ISA 2011*, Brno, Czech Republic, August 2011.

[19] G. Castagnos, B. Alexandre, C. Cécile et al., "Fault Analysis of Grain-128," in *Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, HOST 2009*, San Francisco, CA, USA, July 2009.

[20] S. Karmakar and D. R. Chowdhury, "Fault Analysis of Grain-128 by Targeting NFSR," in *Progress in Cryptology – AFRICACRYPT 2011*, vol. 6737 of *Lecture Notes in Computer Science*, pp. 298–315, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[21] S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on the grain family of stream ciphers," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7428, pp. 122–139, 2012.

[22] S. Sarkar, S. Banik, and S. Maitra, "Differential fault attack against Grain family with very few faults and minimal assumptions," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 64, no. 6, pp. 1647–1657, 2015.

[23] S. Banik and S. Maitra, "A Differential Fault Attack on MICKEY 2.0," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, vol. 8086 of *Lecture Notes in Computer Science*, pp. 215–232, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[24] S. Banik, S. Maitra, and S. Sarkar, "Improved differential fault attack on MICKEY 2.0," *Journal of Cryptographic Engineering*, vol. 5, no. 1, pp. 13–29, 2015.