Fang *et al. J Surveill Secur Saf* 2024;5:62-79
DOI: 10.20517/jsss.2023.42

Journal of Surveillance, Security and Safety

Original Article

Open Access

Improved differential fault analysis of Grain-128AEAD

Tianyu Fang, Iftekhar Salam, Wei-Chuen Yau

School of Computing and Data Science, Xiamen University Malaysia, Sepang 43900, Selangor, Malaysia.

Correspondence to: Iftekhar Salam, School of Computing and Data Science, Xiamen University Malaysia, Sepang 43900, Selangor, Malaysia. E-mail: iftekhar.salam@xmu.edu.my; ORCID: 0000-0003-1395-4623

How to cite this article: Fang T, Salam I, Yau WC. Improved differential fault analysis of Grain-128AEAD. *J Surveill Secur Saf* 2024;5:62-79. http://dx.doi.org/10.20517/jsss.2023.42

Received: 15 Nov 2023 **First Decision:** 24 Jan 2024 **Revised:** 21 Feb 2024 **Accepted:** 13 Mar 2024 **Published:** 30 Mar 2024

Academic Editor: Moti Yung **Copy Editor:** Yanbin Bai **Production Editor:** Yanbin Bai

Abstract

The number of smart devices connected to the Internet has been constantly increasing, and as a result, lightweight cryptography (LWC) has become more important in the past decade. The Lightweight Cryptography (LWC) Project is an initiative taken by the National Institute of Standards and Technology (NIST) to standardize such LWC algorithms. Grain-128AEAD, which was submitted to the NIST LWC project, is an encryption algorithm that provides both confidentiality and integrity assurance. Third-party security analysis of the submitted ciphers is an important aspect of the evaluation of the submission to the NIST LWC project. Although several pieces of existing research, such as the bit-flipping attack, random fault attack, and deterministic random fault attack, have examined the security of Grain-128AEAD, there is still room for improvement in the fault attack models of these studies. This work aims to fill this research gap by analyzing the security margin of Grain-128AEAD against a series of improved differential fault attacks. In this study, we developed a probabilistic random fault attack and applied it to Grain-128AEAD. As an improvement of the existing research, a probabilistic approach can be applied to a more relaxed moderate control attack model. The existing moderate control model assumes the fault to be injected within any bit of a given byte, whereas the faults in our improved approach can be injected within any bits of a two-byte/four-byte segment, thereby relaxing the fault precision. The results indicate that the improved moderate control requires 388 keystreams for the two-byte model and 279 for the four-byte model to identify the target fault locations for implementing a state recovery attack. The relaxed fault attack models presented in this work are more practical to implement; hence, the findings of this research have improved the existing studies and narrowed the current research gap on the fault attack models of Grain-128AEAD.

Keywords: Stream cipher, lightweight cryptography, differential fault attack, Grain-128AEAD

© The Author(s) 2024. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, sharing, adaptation, distribution and reproduction in any medium or format, for any purpose, even commercially, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

INTRODUCTION

The National Institute of Standards and Technology (NIST) initiated the Lightweight Cryptography (LWC) Project to solicit, test, and standardize lightweight cryptographic algorithms for use in restricted environments[1]. This project aims to standardize the ciphers feasible for resource-constrained applications. After two rounds of evaluation, NIST announced the winner, ASCON, from the ten finalists. Among the finalists, Grain-128AEAD is one of the stream cipher-based algorithms. Several finalists are not fully explored by the third party against various fault attacks. This paper investigates improving the differential fault attacks on Grain-128AEAD.

This work presents a set of fault attacks that successfully recovers the majority of the internal state bits of Grain-128AEAD[2,3]. As an improvement of the research by Salam *et al.*[4], we have investigated two more relaxed fault attack models–a two-byte moderate control model assuming the injection of a random fault into two consecutive bytes and a four-byte moderate control model assuming the injection of a random fault into four consecutive bytes. This paper shows that the improved attack, a combined probabilistic-deterministic fault attack of more relaxed moderate control models, is feasible to identify all the required target fault registers in the linear feedback shift register (LFSR). In the moderate control models with two or four bytes, we employ a probabilistic approach to recover some of the fault targets when a deterministic approach is not feasible. Table 1 compares the results of this study with those obtained in the work conducted by Salam *et al.*[4]. Compared to their research, the findings reported in this paper require access to more keystreams and inject more faults; however, the fault attack models of this work are more practical to implement in terms of fault precision.

We use the term probabilistic-deterministic to refer to the fact that some of the fault target locations are identified using a deterministic signature, while some others are identified with a probabilistic signature. Table 2 shows that with two-byte moderate control precision, 100 target LFSR register locations can be identified using the deterministic method, and the remaining 28 need to be recovered with the probabilistic method. On the other hand, 96 target LFSR register locations can be identified using the deterministic method, and the remaining 32 need to be recovered with the probabilistic method. Comparing these two moderate control models, with the two-byte precision, more target registers have a deterministic signature but require more keystream and have a slightly higher data complexity. The four-byte precision requires using the probabilistic signature for a slightly larger number of target registers and, therefore, requires less keystream and lesser data complexity. The attacks presented in this paper are feasible in identifying the majority of the target registers. The fault precision with moderate control is practical as recent works have shown the practicality of fault injection using laser beams[5,6] and focused flashlights[7]. For a random byte fault model, depending on the target device, the fault may be induced with an optical flashgun or using a voltage glitch. The cost of such attacks ranges from low to 500 EUR, where a low cost refers to only a standard desktop PC (and in some cases, connection wires) to apply the attack[8]. Therefore, we conclude that the attacks presented in this work are practically feasible.

GRAIN-128AEAD SPECIFICATION

Grain-128AEAD[2,3] is a stream cipher-based design suitable for applications requiring authenticated encryption with associated data (AEAD). It is based on the Grain family of stream ciphers that consists of Grain-v1[9], Grain-128[10], Grain-128a[11] and Grain-128AEAD, which are known for their high security and efficiency. Grain-128AEAD uses a 128-bit key and operates on blocks of 128 bits[12]. It offers three different operation modes:

1) Grain-Authenticate: This mode provides integrity protection for the associated data and confidentiality for the message.
2) Grain-Encrypt: This mode provides confidentiality for both the associated data and the message.
3) Grain-Seal: This mode combines the features of the previous two modes, providing both confidentiality

Table 1. Summary of required keystreams to determine the faulty register

Ref.	Fault type	Fault precision	Required keystream	Data complexity
[4]	Bit-flipping	Precise	223	$2^{7.80}$
	Probabilistic random	Precise	223	$2^{11.60}$
	Deterministic random	Precise	200	$2^{8.80}$
		Moderate	223	$2^{12.98}$
This work	Probabilistic-Deterministic random	Moderate (Two-byte)	388	$2^{8.60}$
		Moderate (Four-byte)	279	$2^{8.12}$

Table 2. Comparison between Two-byte and Four-byte Precision Methods

Precision		Two-byte	Four-byte
Method	Deterministic	100	96
	Probabilistic	28	32
Identified target registers		128	128
Total Required Keystream		388	297

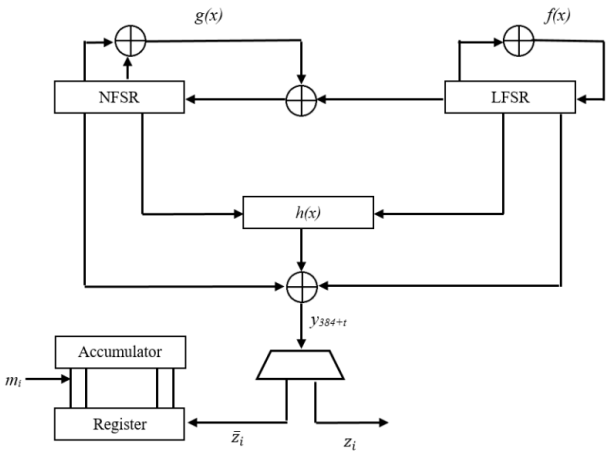

Figure 1. General structure of Grain-128AEAD [12].

and integrity protection for the associated data and the message.

Grain-128AEAD is designed from the idea of a nonlinear filter generator. It consists of the n-bit LFSR and the n-bit non-linear feedback shift register (NFSR). LFSR is used to produce a sequence of a large period, and a nonlinear function is used for the LFSR as input to ensure nonlinearity in the keystream.

Grain-128AEAD consists of two building blocks: the pre-output generator and the authenticator generator. The former consists of a 128-bit LFSR, 128-bit NFSR, and the output function $h(x)$. The latter comprises a 64-bit accumulator and a 64-bit shift register. Figure 1 illustrates the basic structure of Grain-128AEAD. In addition, the cipher consists of a 128-bit key, a 96-bit initial vector (IV), and a 64-bit tag, T, to ensure integrity assurance.

The 256-bit internal state is made up of the contents of LFSR $S = \{s_0, s_1, \ldots, s_{127}\}$ and NFSR $B = \{b_0, b_1, \ldots, b_{127}\}$ [12].

The primitive feedback polynomial of the LFSR is defined as

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128} \tag{1}$$

The LFSR update function is denoted as

$$\begin{aligned} s_{127}^{(t+1)} &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t \\ &= L(S_t) \end{aligned} \tag{2}$$

The NFSR function is given by

$$\begin{aligned} g(x) = 1 \quad &+ \quad x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} \\ &+ \quad x^{110}x^{111} + x^{115}x^{117} + x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40}. \end{aligned} \tag{3}$$

The update function for NFSR is defined by

$$b_{127}^{(t+1)} = s_0^t + F(B_t). \tag{4}$$

The Boolean function $h(x)$ takes nine state variables as input; seven are taken from the LFSR, and the other two are from the NFSR. The function is expressed as

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8. \tag{5}$$

The output generated by the output generator is formulated as

$$y_t = h(x) + s_{93}^t + \sum_{j \in A} b_j^t \tag{6}$$

where $A = \{2, 15, 36, 45, 64, 79, 89\}$.

During the initialization phase, the 128-bit key fills the NFSR, $b_i^0 = k_i, 0 \le i \le 127$, and the IV is loaded into the first 96 bits of the LFSR, $s_i^0 = IV_i, 0 \le i \le 95$. Another 32 are then loaded with padding π, 31 1s with one 0 at the last bit. Then, in the first 256 clock cycles, the pre-output function is fed back to the LFSR and the NFSR and then XOR-ed with the input. The state update functions are defined as

$$s_{127}^{t+1} = L(S_t) + y_t, \quad 0 \le t \le 255 \tag{7}$$

$$b_{127}^{t+1} = s_0^t + F(B_t) + y_t, \quad 0 \le t \le 255 \tag{8}$$

During the last 128 rounds of initialization, the states are updated as follows:

$$s_{127}^{t+1} = L(S_t) + k_{t-256}, 256 \le t \le 383 \tag{9}$$

$$b_{127}^{t+1} = s_0^t + F(B_t), 256 \le t \le 383 \tag{10}$$

During encryption, every even bit of the pre-output generator is used as the keystream bits z_i determined by

$$z_i = y_{384+2i} \tag{11}$$

FAULT ATTACK

A differential fault analysis (DFA) is a type of side-channel attack. In this attack, an adversary induces faults or errors in the execution of a cryptographic algorithm and observes the differences in the outputs caused by these faults. By analyzing these differences, the attacker aims to gain information about the secret key used in the cryptographic process[13]. Fault attacks can pose a significant threat to a wide range of industries, including banking, defense, and critical infrastructure. They have been shown to be a powerful technique against many modern ciphers; for examples of fault attacks on stream ciphers, refer to the provided references[14,15,15–23]. Fault attacks can be measured by four parameters: the fault type, duration, number, and precision[4,24].

1) The fault type describes how it affects a specific register. The target register bit may become 0 or 1, be flipped, or be a random value by the fault injection. To perform a bit-flipping attack, the attacker is capable of changing the value of the target bit(s) by complementing them. In the event of a random fault, the attacker has no control over its impact; the targeted register bit has an equal chance of flipping or remaining the same.
2) The fault duration determines the period it remains active. A fault can be divided into two types depending on the fault duration: temporary and permanent. For a temporary fault, the error remains active for a short period, such as a single clock cycle, while a permanent fault remains active for the entire duration of the operation.
3) The number of faults indicates the count of bits influenced by the fault injection process. The length of affected bits varies from a single bit to multiple bytes.
4) The precision of fault, indicating the ability to control the timing and the intended target of the fault, can be categorized into three types according to the degree of precision: precise, moderate, and no control. In the first type, the attacker can induce an error into a specific target at a specific time. In the second type, the attacker can moderately control the fault target and timing; e.g., it can inject fault in a specific byte but does not have control over the specific bits in the given byte. In the third type, the attacker cannot control the fault target and timing.

Existing fault attacks on Grain-128AEAD

Several recent studies have performed a series of fault analyses against Grain-128AEAD. In one of these works, a bit-flipping attack, a probabilistic random fault attack, and a deterministic random fault attack were implemented to recover the internal state of the cipher. We discuss these attacks briefly below.

Bit-flipping fault attack

A bit-flipping fault attack is a side-channel attack that complements the target register bit by injecting a fault. Salam *et al.*[4] applied a bit-flipping fault attack on Grain-128AEAD. This aspect of their work investigated the existence of unique quadratic terms in the keystream outputs. Consider the output function z_0, expressed as:

$$z_0 = s_{93} \oplus b_{12}s_8 \oplus s_{13}s_{20} \oplus b_{95}s_{43} \oplus s_{60}s_{79} \oplus b_{12}b_{95}s_{94} \oplus b_2 \oplus b_{15} \oplus b_{36} \oplus b_{45} \oplus b_{64} \oplus b_{73} \oplus b_{89}, \quad (12)$$

where the monomial $s_{13}s_{20}$ is a unique quadratic term in the fault-free pre-output z_0. This enables the recovery of s_{13} by complementing the content of s_{20}, and vice versa. Applying this approach to the different keystream polynomials of Grain-128AEAD, this work managed to recover 128 LFSR bits and 95 NFSR bits with a data complexity of $2^{7.88}$.

Probabilistic random fault attack

A probabilistic random fault attack is applied when the attacker cannot conclusively determine whether the value of the faulty register has been complimented, i.e., the effect of the fault is random. In the research by Salam *et al.*[4], several distinct cases are considered when a register s_i is injected by a fault for which the impact of the fault is unknown. Consider the fault target register appears as a linear term in the fault-free keystream z_i. By XOR-ing the fault-free keystream z_i and the faulty keystream z'_i, the output differential δ_i can be calculated by

$$\delta_i = z_i \oplus z'_i \quad (13)$$

By observing the output differential $\delta_i = 1$, the impact of the fault is conclusive; i.e., certain that the target register contents are complemented. However, the fault impact cannot be confirmed when $\delta_i = 0$. This finding suggests that if the fault is injected into the same target register several times, the impact of the injected fault can be concluded with a high probability. The probability of conclusively determining the impact of the fault is higher when more faults are injected into the target register. This work[4] indicates that the probability approaches 99.99% when injecting more than six faults. The data complexity of the applied probabilistic random fault model is $2^{11.60}$.

Deterministic random fault attack

The deterministic random fault attack[4] is carried out on attack models with three different levels of control precision. A simple example is used to illustrate the idea. Notice that s_{93} in Equation (12) is a unique linear term. Let z'_0 represent the faulty keystream resulting from a fault e at time $t = 0$, as given in

$$z'_0 = (s_{93} \oplus e) \oplus b_{12}s_8 \oplus s_{13}s_{20} \oplus b_{95}s_{43} \oplus s_{60}s_{79} \oplus b_{12}b_{95}s_{94} \oplus b_2 \oplus b_{15} \oplus b_{36} \oplus b_{45} \oplus b_{64} \oplus b_{73} \oplus b_{89} \quad (14)$$

Then, the value of the fault e, which can be determined by XOR-ing Equations (12) and (14), is calculated by

$$\begin{aligned} \delta_0 &= z_0 \oplus z'_0 \\ &= s_{93} \oplus (s_{93} \oplus e) \\ &= e. \end{aligned} \quad (15)$$

The unique linear term in the equation is targeted by the fault model and can be determined by calculating the first-order derivative of the keystream equations with respect to the LFSR register s_i or the NFSR register b_i, with $0 \leq i \leq 127$, as given in

$$\frac{\partial z_j}{\partial s_i} = 1 \quad or \quad \frac{\partial z_j}{\partial b_i} = 1 \quad (16)$$

With this approach, a list of the output indices can be generated and used to compute the value of the random fault for Grain-128AEAD. Accordingly, once the fault is determined to complement the target register, the corresponding equations with the unique quadratic terms (bit-flipping) are used to recover specific state bit(s).

As previously stated, three varying degrees of precision control are considered in these attacks: precise, moderate, and no control. From the results of precise control, it was concluded that an average of two faults is needed to complement a specific target register. The average number of required faults for this approach is 200, with a data complexity of $2^{8.88}$. In the moderate control, the fault was injected in a 1-byte array, assuming that the injected fault could affect any registers within that byte array. Applying this approach to all the 128-bit LFSR and 128-bit NFSR, the entire internal state of LFSR and 24 NFSR register bits can be identified with a data complexity of $2^{11.69}$. For the no-control approach, the fault is injected at a random location within the LFSR and at any time. In their study, the no-control model was shown to be infeasible in recovering the LFSR and required further investigation.

IMPROVED FAULT ATTACKS ON GRAIN-128AEAD

We consider possible extensions and improvements for the differential fault attacks based on the research by Salam *et al.*[4]. We investigated and implemented a moderate control model to a more relaxed degree. Moderate control refers to the assumption in which an attacker can introduce the error in a particular byte array, where the error can affect any of the bits in that byte array. We considered two scenarios: i) injecting faults within a two-byte array and ii) injecting faults within a four-byte array.

Inject a fault within two consecutive bytes

Instead of focusing on a single byte, this study first assumes the injection of a random fault to two consecutive bytes. The 128-bit LFSR and NFSR are grouped into eight arrays, each consisting of sixteen register bits. For example, $s_0, \ldots, s_{15}$ are grouped into $S_{byte0}\&S_{byte1}$, and $b_0, \ldots, b_{15}$ are grouped into $B_{byte0}\&B_{byte1}$. In this moderate control case, this study will have to observe multiple output differentials to confirm whether all the target registers within the two consecutive bytes array have been affected by the injected fault. The study by Salam *et al.* indicated that it is required to observe 425 keystreams in order to confirm that all the LFSR registers are affected by the random fault e, and in Table 3, the total number of keystream indices required is 361 for all the two-byte arrays.

Table 3. List of keystream indices required and the number of output indices for each two-byte array

Two-byte	Output indices	Quantity
$S_{byte0}\&S_{byte1}$	34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100	34
$S_{byte2}\&S_{byte3}$	44, 46, 48, 52, 54, 56, 58, 60, 66, 68, 70, 72, 74, 78, 80, 82, 86, 90, 94, 96, 98, 100, 102, 104, 106, 108, 114, 116	28
$S_{byte4}\&S_{byte5}$	34, 36, 38, 40, 42, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 130, 132, 134, 136, 138	42
$S_{byte6}\&S_{byte7}$	44, 46, 48, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154	41
$S_{byte8}\&S_{byte9}$	36, 40, 64, 82, 84, 86, 88, 90, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 152, 154, 156, 158, 160, 162	40
$S_{byte10}\&S_{byte11}$	0, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178	68
$S_{byte12}\&S_{byte13}$	2, 6, 10, 14, 34, 38, 42, 46, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 78, 92, 96, 98, 100, 102, 104, 106, 108, 110, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 136, 154, 158, 160, 162, 164, 166, 168, 170, 172, 174, 182, 186, 190, 194	56
$S_{byte14}\&S_{byte15}$	18, 20, 22, 24, 26, 28, 30, 32, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 84, 86, 88, 92, 94, 96, 108, 110, 128, 130, 132, 138, 146, 150, 152, 154, 170, 172, 174, 192, 198, 200, 202, 204, 206, 208, 210, 212	52

Register	Output Indices					
0	34	38	54	66	70	86
1	64	76	82			
2	36	40	56	68	72	88
3	66	78	84			
4	38	42	58	70	74	90
5	68	80	86			
6	40	44	60	72	76	92
7	34	66	70	82	88	
8	42	46	62	74	78	94
9	36	68	72	84	90	
10	44	48	64	76	80	96
11	38	70	74	86	92	
12	46	50	66	78	82	98
13	40	72	76	88	94	
14	48	52	68	80	84	100
15	42	74	78	90	96	

Figure 2. Output indices to determine the target registers in $S_{byte0}\&S_{byte1}$ (duplicated values are presented in matching colors).

Determining required single output index to confirm the effect of random fault

To further reduce the data complexity, this study obtains single or multiple output keystream bits that may be used to conclusively determine the impact on the fault for any target register. The first step is eliminating the duplicated output indices in two bytes because observing output indices with duplicated values can only conclude an ambiguous result. In Figure 2, all the output indices that output a differential of 1 for each register in S_{byte0} and S_{byte1} are listed. The duplicated output indices are in matching colors, and the unique output indices are not colored. Suppose that a fault is injected into a random register in S_{byte0} and S_{byte1}, and it can be observed from Figure 2 that when $\delta_{34} = 1$, the fault may be injected at s_0 or s_7. After removing the duplicated values, every output index of s_1, s_3, s_5, s_7, s_9, s_{10}, s_{11}, s_{13} and s_{15} is removed, leaving no keystream indices to identify the impact of the fault. After the same approach is performed on all eight LFSR two-byte arrays, it was observed that in every array, a similar situation occurs where some output indices of a target register are duplicated and eliminated. The next step is to determine whether it is possible to recover the fault location using a single output differential, δ_j. To do this, we experimented on each LFSR two-byte array, as given in

	Unique Output Differentials δ_j								
Registers	δ_{50}	δ_{52}	δ_{54}	δ_{56}	δ_{58}	δ_{60}	δ_{62}	δ_{98}	δ_{100}
0	0	0	100	0	0	0	0	52	74
1	0	0	0	0	0	0	0	36	51
2	42	0	0	100	0	0	0	47	49
3	0	0	0	0	0	0	0	41	31
4	0	45	0	0	100	0	0	11	56
5	0	0	0	0	0	0	0	24	48
6	0	0	46	0	0	100	0	73	16
7	0	0	0	0	0	0	0	71	26
8	0	0	0	57	0	0	100	35	48
9	53	0	0	0	0	0	0	11	80
10	0	0	0	0	56	0	0	44	39
11	0	46	0	0	0	0	0	13	7
12	100	0	0	0	0	46	0	100	56
13	0	0	50	0	0	0	0	43	12
14	0	100	0	0	0	0	41	33	100
15	0	0	0	53	0	0	0	0	45

Figure 3. Number of times $\delta_j = 1$ for registers in $S_{byte0}\&S_{byte1}$ in 100 tests (values of the number of times = 100 are colored in green and $0 < values < 100$ are colored in red).

Algorithm 1. For each register in each two-byte array, the output polynomials are tested for 100 random initial key-IV pairs. The output differentials between the faulty and fault-free keystreams for each keystream round ($0 \leq j \leq 199$) are then calculated, and the number of times when $\delta_j = 1$ in 100 tests is recorded.

Algorithm 1: Determining the number of times when $\delta_j = 1$ for each target register

Require: Target register R_t in two consecutive bytes and the corresponding output indices z_j of the two bytes

For R_t **do**
  Initialized an array of the size of the number of total required keystream bits, $result = [0, ..., 0]$;
  For $i = 0$ *to* 100 **do**
    Initialize Grain-128AEAD with a random initial state;
    Reinitialize with random initial states where faulty register = R_t;
    For j **do**
      Generate the fault-free keystream bit, z_j;
      Generate the faulty keystream bit, z'_j;
      $\delta_j = z_j \oplus z'_j$;
      If $\delta_j = 1$ **then**
        $result[i] += 1$;
      end
    end
  end
end

As an example, Figure 3 shows the result obtained from Algorithm 1 using registers in $S_{byte0}\&S_{byte1}$, i.e., the number of times for each register in the two-byte array where $\delta_j = 1$ in 100 random experiments. For better observation of the data representation, the cells with values equal to 100 are highlighted in green, while cells with values between 0 and 100 are highlighted in red. This experiment shows that we cannot determine the fault target by observing a single output differential, as the signature is not always unique in the given byte array. For instance, when $\delta_{50} = 1$, the fault could be applied at s_2, s_9, and s_{12}. This is because, in 100 tests, a

	Output Differentials δ_j																									
Registers	δ_{36}	δ_{38}	δ_{40}	δ_{42}	δ_{44}	δ_{50}	δ_{52}	δ_{54}	δ_{56}	δ_{58}	δ_{60}	δ_{62}	δ_{64}	δ_{66}	δ_{70}	δ_{74}	δ_{78}	δ_{80}	δ_{82}	δ_{84}	δ_{86}	δ_{88}	δ_{92}	δ_{98}	δ_{100}
0	0	100	0	0	0	0	0	100	0	0	0	0	52	100	100	6	20	53	57	25	100	27	74	40	70
1	0	0	0	0	0	0	0	0	0	0	0	0	100	28	44	14	0	0	100	0	57	12	27	36	54
2	100	0	100	0	0	51	0	0	100	0	0	0	0	55	7	8	34	23	56	45	18	100	23	54	56
3	26	0	0	0	0	0	0	0	0	0	0	0	0	100	49	12	100	0	0	100	0	56	10	38	35
4	44	100	0	100	0	0	54	0	0	100	0	0	0	0	100	100	15	35	26	53	42	25	26	15	56
5	0	27	0	0	0	0	0	0	0	0	0	0	0	0	27	55	15	100	0	0	100	0	6	24	41
6	0	57	100	0	100	0	0	47	0	0	100	0	0	0	51	8	8	12	31	24	48	54	100	73	12
7	0	0	32	0	0	0	0	0	0	0	0	0	0	100	100	48	16	51	100	0	0	100	54	79	23
8	0	0	55	100	0	0	0	0	44	0	0	100	0	17	0	100	100	9	90	29	52	53	31	35	60
9	100	0	0	20	0	42	0	0	0	0	0	0	0	0	0	31	56	11	54	100	0	0	0	10	78
10	26	0	0	53	100	0	0	0	0	50	0	0	100	0	45	38	7	100	10	88	27	55	46	45	36
11	0	100	0	0	23	0	42	0	0	0	0	0	0	0	100	100	56	49	9	48	100	0	100	9	10
12	0	25	0	0	51	100	0	0	0	0	49	0	0	100	23	0	100	4	100	9	89	31	49	100	47
13	0	0	100	0	0	0	0	52	0	0	0	0	0	0	0	0	30	47	47	15	45	100	0	52	15
14	0	0	20	0	0	0	100	0	0	0	0	41	0	0	0	62	53	100	7	100	6	88	53	35	100
15	0	0	0	100	0	0	0	0	49	0	0	0	0	0	0	100	100	27	50	44	16	54	0	0	41

Figure 4. Number of times $\delta_j = 1$ for registers in $S_{byte0}\&S_{byte1}$ in 100 tests with duplicated output differentials (values of the number of times = 100 are colored in green, and $0 < values < 100$ are colored in red).

fault injected at s_{12} leads to $\delta_{50} = 1$ for 100 times, while 42 times at s_2 and 53 times at s_9 for $\delta_{50} = 1$. The result of this experiment clearly indicates that it is infeasible to conclusively determine the target location in LFSR two-byte arrays using a single-bit keystream differential.

Then, attempting to gain a clearer conclusion, the experiment is extended by not removing the duplicated values in Figure 2 and using Algorithm 1. The results obtained are shown in Figure 4. We observe that although the output differential injected with a fault may be applied to only two registers, it is still impossible to indicate the fault location conclusively. For example, when $\delta_{58} = 1$, the fault could be applied at s_4 and s_{10}. This is because, in 100 tests, a fault injected at s_4 leads to $\delta_{58} = 1$ for 100 times, while 50 times at s_{10}. Hence, the fault location in LFSR two-byte arrays cannot be determined by observing a single output index only.

Determining required pairs of keystream indices to confirm the effect of random fault

Another experiment was conducted to determine the unique pairs of output keystream bits that can conclusively establish if the injected fault affects a target register. For this, in each register of the two-byte array, all the possible pairs of keystream indices were generated first. For example, pairs of differentials can be formed from the output indices array of register s_1 as $(\delta_{64}, \delta_{76})$, $(\delta_{64}, \delta_{82})$, $(\delta_{76}, \delta_{82})$.

The pair combination of output indices for each register in $S_{byte0}\&S_{byte1}$ is shown in Figure 5. The duplicated values are highlighted in matching colors. It can be seen that when $\delta_{34} = \delta_{70} = 1$, the potential faulty registers are s_0 and s_7. To further locate the injected fault, the duplicated pairs should be removed. Then, an experiment is conducted to determine the possibility of identifying the fault location using the differentials of the unique pairs of output keystream indices. This experiment aims to identify the conditions under which the faulty register can be decisively determined. Similar to the previous experiment, 100 tests are run with a random initial state for each register in the two-byte array and the number of times where the output differential $\delta_j = 1$ is recorded. The algorithm applied is similar to Algorithm 1. The difference is that the output is grouped into output indices pairs formed from the array of output indices according to the register.

Figure 6 presents the results obtained from the experiment with differential pairs of output indices for the target register s_0. The experiment shows that if a fault is injected into a random register within $S_{byte0}\&S_{byte1}$ and $\delta_{34} = \delta_{38} = \delta_{54} = \delta_{66} = 1$ is observed, it can be concluded that the register s_0 is affected by the fault. This is because no other faulty registers in $S_{byte0} \cap S_{byte1}$ produce $(\delta_{34} = \delta_{38}) = (1, 1)$ or $(\delta_{34} = \delta_{54}) = (1, 1)$ or $(\delta_{54} = \delta_{66}) = (1, 1)$.

Registers	Pairs of Output Indices (δ_i, δ_j)														
0	34,38	34,54	34,66	34,70	34,86	38,54	38,66	38,70	38,86	54,66	54,70	54,86	66,70	66,86	70,86
1	64,76	64,82	76,82												
2	36,40	36,56	36,68	36,72	36,88	40,56	40,68	40,72	40,88	56,68	56,72	56,88	68,72	68,88	72,88
3	66,78	66,84	78,84												
4	38,42	38,58	38,70	38,74	38,90	42,58	42,70	42,74	42,90	58,70	58,74	58,90	70,74	70,90	74,90
5	68, 80	68,86	80,86												
6	40,44	40,60	40,72	40,76	40,92	44,60	44,72	44,76	44,92	60,72	60,76	60,92	72,76	72,92	76,92
7	34,66	34,70	34,82	34,88	66,70	66,82	66,88	70,82	70,88	82,88					
8	42, 46	42, 62	42, 74	42, 78	42, 94	46, 62	46, 74	46, 78	46, 94	62, 74	62, 78	62, 94	74, 78	74, 94	78, 94
9	36, 68	36, 72	36, 84	36, 90	68, 72	68, 84	68, 90	72, 84	72, 90	84, 90					
10	44, 48	44, 64	44, 76	44, 80	44, 96	48, 64	48, 76	48, 80	48, 96	64, 76	64, 80	64, 96	76, 80	76, 96	80, 96
11	38, 70	38, 74	38, 86	38, 92	70, 74	70, 86	70, 92	74, 86	74, 92	86, 92					
12	46, 50	46, 66	46, 78	46, 82	46, 98	50, 66	50, 78	50, 82	50, 98	66, 78	66, 82	66, 98	78, 82	78, 98	82, 98
13	40, 72	40, 76	40, 88	40, 94	72, 76	72, 88	72, 94	76, 88	76, 94	88, 94					
14	48, 52	48, 68	48, 80	48, 84	48,100	52, 68	52, 80	52, 84	52, 100	68, 80	68, 84	68, 100	80, 84	80, 100	84, 100
15	42, 74	42, 78	42, 90	42, 96	74, 78	74, 90	74, 96	78, 90	78, 96	90, 96					

Figure 5. Pairs of output keystream indices to determine the target registers in S_{byte0}&S_{byte1} (Duplicated pairs are presented in matching colors).

Registers	Unique Pairs of Output Differentials (δ_i, δ_j)								
	(δ_{34}, δ_{38})	(δ_{34}, δ_{54})	(δ_{34}, δ_{86})	(δ_{38}, δ_{54})	(δ_{38}, δ_{66})	(δ_{54}, δ_{66})	(δ_{54}, δ_{70})	(δ_{54}, δ_{86})	(δ_{66}, δ_{86})
0	[100, 100]	[100, 100]	[100, 100]	[100, 100]	[100, 100]	[100, 100]	[100, 100]	[100, 100]	[100, 100]
1	[30, 0]	[30, 0]	[30, 55]	[0, 0]	[0, 25]	[0, 25]	[0, 59]	[0, 55]	[25, 55]
2	[45, 0]	[45, 0]	[45, 20]	[0, 0]	[0, 41]	[0, 41]	[0, 5]	[0, 20]	[41, 20]
3	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 100]	[0, 100]	[0, 45]	[0, 0]	[100, 0]
4	[0, 100]	[0, 0]	[0, 50]	[100, 0]	[100, 0]	[0, 0]	[0, 100]	[0, 50]	[0, 50]
5	[0, 29]	[0, 0]	[0, 100]	[29, 0]	[29, 0]	[0, 0]	[0, 34]	[0, 100]	[0, 100]
6	[0, 47]	[0, 55]	[0, 51]	[47, 55]	[47, 0]	[55, 0]	[55, 54]	[55, 51]	[0, 51]
7	[100, 0]	[100, 0]	[100, 0]	[0, 0]	[0, 100]	[0, 100]	[0, 100]	[0, 0]	[100, 0]
8	[25, 0]	[25, 0]	[25, 53]	[0, 0]	[0, 31]	[0, 31]	[0, 0]	[0, 53]	[31, 53]
9	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 0]
10	[0, 0]	[0, 0]	[0, 28]	[0, 0]	[0, 0]	[0, 0]	[0, 44]	[0, 28]	[0, 28]
11	[0, 100]	[0, 0]	[0, 100]	[100, 0]	[100, 0]	[0, 0]	[0, 100]	[0, 100]	[0, 100]
12	[0, 27]	[0, 0]	[0, 93]	[27, 0]	[27, 100]	[0, 100]	[0, 23]	[0, 93]	[100, 93]
13	[0, 0]	[0, 49]	[0, 50]	[0, 49]	[0, 0]	[49, 0]	[49, 0]	[49, 50]	[0, 50]
14	[0, 0]	[0, 0]	[0, 7]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 7]	[0, 7]
15	[0, 0]	[0, 0]	[0, 13]	[0, 0]	[0, 0]	[0, 0]	[0, 0]	[0, 13]	[0, 13]

Figure 6. Number of times where $\delta_i = \delta_j = 1$ for each register in S_{byte0}&S_{byte1}. The unique pairs of output indices listed here are for target register s_0. Pairs with both values equivalent to 100 are highlighted in green, while those greater than 0 and less than 100 are highlighted in red.

Generally, these conclusive pairs of output differentials can be obtained by examining the table columns where no other faulty target results in $\delta_i = \delta_j = 1$, i.e., by observing table columns with only green-colored cells. Carrying out the experiment for all the sixteen registers in S_{byte0} and S_{byte1} using the output indices, the conclusive pairs for each register are directly obtained except for s_1, s_3, s_5, s_7, s_9, s_{11}, s_{13} and s_{15}. Table 4 shows the conclusive pairs of output differentials for the sixteen registers in S_{byte0}&S_{byte1}.

Table 4 indicates that it is infeasible to obtain conclusive pairs of output differentials for registers s_1, s_3, s_5, s_7, s_9, s_{11}, s_{13} and s_{15} with the unique pair of output indices. This is because the unique pairs of these target registers do not produce similar results, as shown in Figure 6, where there is no single column with only green-colored cells. Take s_1 as an example; the result produced using the unique pairs of s_1 in S_{byte0}&S_{byte1} are displayed

Table 4. Required output differential pairs (δ_i, δ_j) for all registers in $S_{byte0}\&S_{byte1}$

Target register	Pairs of output differentials (δ_i, δ_j)
s_0	(34, 38), (34, 54), (54, 66)
s_1	No unique output differential pair
s_2	(36, 40), (36, 56), (56, 68)
s_3	No unique output differential pair
s_4	(38, 42), (38, 58)
s_5	No unique output differential pair
s_6	(40, 44), (40, 60)
s_7	No unique output differential pair
s_8	(42, 46), (42, 62)
s_9	No unique output differential pair
s_{10}	(44, 48), (44, 64)
s_{11}	No unique output differential pair
s_{12}	(46, 50)
s_{13}	No unique output differential pair
s_{14}	(48, 52)
s_{15}	No unique output differential pair

	Unique Pairs of Output Differentials (δ_i, δ_j)	
Registers	(δ_{64}, δ_{82})	(δ_{76}, δ_{82})
0	[53, 49]	[31, 49]
1	[100, 100]	[100, 100]
2	[0, 46]	[9, 46]
3	[0, 0]	[13, 0]
4	[0, 19]	[5, 19]
5	[0, 0]	[15, 0]
6	[0, 17]	[100, 17]
7	[0, 100]	[52, 100]
8	[0, 86]	[8, 86]
9	[0, 44]	[45, 44]
10	[100, 2]	[100, 2]
11	[0, 12]	[26, 12]
12	[0, 100]	[50, 100]
13	[0, 39]	[100, 39]
14	[0, 5]	[0, 5]
15	[0, 56]	[0, 56]

Figure 7. Number of times where $\delta_i = \delta_j = 1$ for each register in $S_{byte0}\&S_{byte1}$. The unique pairs of output keystream indices listed here are for target register s_1. Pairs with both values equivalent to 100 are highlighted in green, while those greater than 0 and less than 100 are highlighted in red.

in Figure 7. It can be observed that no clear indication of pairs can conclusively determine the location of the faulty register. For instance, when $\delta_{64} = \delta_{82} = 1$, the faulty register may be s_0, s_1, or s_{10}. The same situation appears in pair $(\delta_{76}, \delta_{82})$. Since no columns contain only the green-colored cell, there are no conclusive pairs of output indices for the register s_1. The same case happens to register s_1, s_3, s_5, s_7, s_9, s_{11}, s_{13} and s_{15}.

Determining required conditions to confirm the effect of random fault

After conducting the above-mentioned set of experiments for all the registers in $S_{byte0}\&S_{byte1}$, it can be observed that a particular register is affected by the fault if several conditions are satisfied. Figure 7 shows that when $\delta_{64} = \delta_{82} = 1$, there are three possible fault registers, s_0, s_1, or s_{10}. However, in Table 4, the conclusive pairs that can determine s_0 or s_{10} being injected with the fault can be found by observing if $\delta_{34} = \delta_{38} = \delta_{54} = \delta_{66} = 1$ and $\delta_{44} = \delta_{48} = \delta_{64} = 1$, respectively. Hence, ($\delta_{34} \neq 1$ or $\delta_{38} \neq 1$ or $\delta_{54} \neq 1$ or $\delta_{66} \neq 1$) and ($\delta_{44} \neq 1$ or $\delta_{48} \neq 1$ or $\delta_{64} \neq 1$) imply that s_0 and s_{10} are not injected with the fault. Furthermore, if $\delta_{64} = \delta_{82} = 1$, it can be concluded that s_1 is the register injected with the fault. Similarly, the conditions that should be satisfied to determine whether the registers s_3, s_5, s_7, s_9, s_{11}, s_{13} and s_{15} are the faulty register can be concluded. The summary of the conditions is provided in Table 5.

Table 5. Required output differential pairs (δ_i, δ_j) for all registers in $S_{byte0}\&S_{byte1}$

Target register	Keystream condition
s_0	$\delta_{34} = \delta_{38} = 1$
s_1	$(\delta_{34}, \delta_{38}) \neq (1,1)$, and $(\delta_{44}, \delta_{48}) \neq (1,1)$, and $\delta_{64} = \delta_{82} = 1$
s_2	$\delta_{36} = \delta_{40} = 1$
s_3	$(\delta_{34}, \delta_{38}) \neq (1,1)$, and $(\delta_{36}, \delta_{40}) \neq (1,1)$, and $(\delta_{42}, \delta_{46}) \neq (1,1)$, and $(\delta_{46}, \delta_{50}) \neq (1,1)$, and $\delta_{64} = \delta_{68} = 1$
s_4	$\delta_{38} = \delta_{42} = 1$
s_5	$(\delta_{34}, \delta_{38}) \neq (1,1)$, and $(\delta_{64}, \delta_{82}) \neq (1,1)$, and $(\delta_{36}, \delta_{40}) \neq (1,1)$, and $(\delta_{38}, \delta_{42}) \neq (1,1)$, and $(\delta_{42}, \delta_{46}) \neq (1,1)$, and $(\delta_{44}, \delta_{48}) \neq (1,1)$, and $(\delta_{48}, \delta_{52}) \neq (1,1)$, and $\delta_{68} = \delta_{86} = 1$
s_6	$\delta_{40} = \delta_{44} = 1$
s_7	$(\delta_{34}, \delta_{38}) \neq (1,1)$, and $(\delta_{64}, \delta_{82}) \neq (1,1)$, and $(\delta_{36}, \delta_{40}) \neq (1,1)$, and $(\delta_{42}, \delta_{46}) \neq (1,1)$, and $\delta_{34} = \delta_{82} = 1$
s_8	$\delta_{42} = \delta_{46} = 1$
s_9	$(\delta_{36}, \delta_{40}) \neq (1,1)$, and $(\delta_{64}, \delta_{68}) \neq (1,1)$, and $(\delta_{38}, \delta_{42}) \neq (1,1)$, and $(\delta_{44}, \delta_{48}) \neq (1,1)$, and $\delta_{36} = \delta_{84} = 1$
s_{10}	$(\delta_{44}, \delta_{48}) = (1,1)$
s_{11}	$(\delta_{34}, \delta_{38}) \neq (1,1)$, and $(\delta_{38}, \delta_{42}) \neq (1,1)$, and $(\delta_{68}, \delta_{86}) \neq (1,1)$, and $(\delta_{40}, \delta_{44}) \neq (1,1)$, and $(\delta_{46}, \delta_{50}) \neq (1,1)$, and $\delta_{38} = \delta_{92} = 1$
s_{12}	$(\delta_{46}, \delta_{50}) = (1,1)$
s_{13}	$(\delta_{36}, \delta_{40}) \neq (1,1)$, and $(\delta_{40}, \delta_{44}) \neq (1,1)$, and $(\delta_{34}, \delta_{82}) \neq (1,1)$, and $(\delta_{42}, \delta_{46}) \neq (1,1)$, and $(\delta_{48}, \delta_{52}) \neq (1,1)$, and $\delta_{40} = \delta_{94} = 1$
s_{14}	$(\delta_{48}, \delta_{52}) = (1,1)$
s_{15}	$(\delta_{38}, \delta_{42}) \neq (1,1)$, and $(\delta_{42}, \delta_{46}) \neq (1,1)$, and $(\delta_{36}, \delta_{84}) \neq (1,1)$, and $(\delta_{44}, \delta_{48}) \neq (1,1)$, and $\delta_{42} = \delta_{96} = 1$

After implementing this experiment on all eight two-byte arrays, the conditions required to confirm the target registers are determined for $S_{byte0}\&S_{byte1}$ and $S_{byte2}\&S_{byte3}$, but for the remaining six, some target registers cannot be conclusively determined; one example is $S_{byte10}\&S_{byte11}$, where no target registers can be determined.

Determining required combinations of output indices to confirm the effect of random fault

To further investigate the conditions used to confirm all the registers in the remaining six two-byte arrays, instead of using pairs, a combination of all the unique output indices of the corresponding register is generated and used to further determine the faulty register. The process is similar to Algorithm 1, but the output pairs are replaced with a list of all unique output differentials of the register. As determined by Algorithm 2, arrays are declared to store the result of all the registers in the two-byte array of the 100-round tests. The experiment is carried out on every two consecutive bytes of LFSR.

For example, Figure 8 presents the results of the experiment conducted using all the combinations of output indices for register s_{82} in $S_{byte10}\&S_{byte11}$. The outcomes illustrate that if a random register in $S_{byte10}\&S_{byte11}$ is subjected to a fault, and $\delta_{46} = \delta_{82} = \delta_{104} = \delta_{116} = \delta_{136} = \delta_{148} = \delta_{168} = 1$ is observed, then it can be inferred that s_{82} is affected by the fault. This is due to the absence of any other faulty register that can produce $(\delta_{46}, \delta_{82}, \delta_{104}, \delta_{116}, \delta_{136}, \delta_{148}, \delta_{168}) = (1,1,1,1,1,1,1)$ in 100 out of 100 tests. However, we observed that the signatures generated through this experiment are insufficient to precisely determine the fault locations for all the sixteen target registers in $S_{byte10}\&S_{byte11}$. For example, Figure 9 indicates that when $\delta_{44} = \delta_{102} = \delta_{114} = \delta_{118} = \delta_{134} = \delta_{146} = \delta_{150} = \delta_{166} = 1$, the fault might have been injected either at s_{80} or s_{87}.

Figure 10 demonstrates that for the target register s_{81} in $S_{byte10}\&S_{byte11}$, although the combination of output indices equals one for s_{81} and s_{82}, it only appears to s_{81} where the combined output indices are equal to one in the same round test. Here, the same round test refers to all the corresponding differentials being simultaneously one for the unique combinations of s_{81} or s_{82}. Hence, the target register s_{81} can be determined using the result of the combined output indices.

Algorithm 2: Determine the number of times when combined unique output indices result in a differential of one for each register in Grain-128AEAD

Require: Register R_t in two consecutive bytes and the corresponding output indices z_j of the two bytes
For R_t **do**
  Initialize an array of the size of the number of total required keystream bits of the byte, $result = [0, ..., 0]$;
  Declare an empty array to store the result of 100 rounds of test results for all registers, $differential = []$;
  Declare an empty array as the sub-array of $differential$ to store the single round test results for all registers, $single\_test = differential$;
  For $i = 0$ *to* 100 **do**
    Initialize $single\_test$ of the size of the number of total required keystream bits of the two-byte array filled with zeros, $single\_test = [0, ..., 0]$;
    Initialize Grain-128AEAD with a random initial state;
    Reinitialize with random initial states where faulty register = R_t;
    For j **do**
      Generate the fault-free keystream bit, z_j;
      Generate the faulty keystream bit, z'_j;
      $\delta_j = z_j \oplus z'_j$;
      If $\delta_j = 1$ **then**
        $result[i] += 1$;
        $single\_test[i] += 1$;
      end
    end
  end
end

Registers	Combining All Unique Output Differentials ($\delta_0,...,\delta_j$) ($\delta_{46}, \delta_{82}, \delta_{104}, \delta_{116}, \delta_{136}, \delta_{148}, \delta_{168}$)
80	[0, 0, 0, 61, 0, 6, 51]
81	[0, 0, 0, 0, 56, 55, 15]
82	[100, 100, 100, 100, 100, 100, 100]
83	[25, 47, 32, 51, 0, 50, 44]
84	[0, 0, 0, 46, 0, 47, 25]
85	[0, 44, 0, 46, 42, 41, 55]
86	[0, 0, 53, 0, 55, 0, 42]
87	[0, 0, 100, 0, 51, 45, 100]
88	[0, 0, 21, 0, 47, 52, 44]
89	[45, 0, 49, 100, 100, 100, 40]
90	[0, 0, 47, 50, 23, 39, 50]
91	[0, 56, 0, 0, 0, 21, 0]
92	[0, 0, 0, 0, 48, 0, 25]
93	[100, 0, 0, 0, 0, 59, 100]
94	[20, 0, 0, 100, 0, 100, 15]
95	[0, 0, 0, 33, 48, 0, 50]

Figure 8. All unique output indices for register s_{82} are combined. The table illustrates the differentials of these combinations for each target register in S_{byte10}&S_{byte11} (all values equal to 100 are colored in green).

Figure 11 shows the number of times where the differentials $\delta_i = \delta_j = 1$ for each register in S_{byte10}&S_{byte11}. The unique combination of output indices listed here is for target register s_{80}. As Figure 11 shows, using only the combined output indices, it is impossible to determine whether s_{80} is the faulty register, as the combined output indices are equal to one for both s_{80} and s_{87}. However, applying this approach to all sixteen registers

Registers	Combining All Unique Output Differentials ($\delta_0,\ldots,\delta_j$) ($\delta_{44}, \delta_{102}, \delta_{114}, \delta_{118}, \delta_{134}, \delta_{146}, \delta_{150}, \delta_{166}$)
80	[100, 100, 100, 100, 100, 100, 100, 100]
81	[24, 31, 58, 45, 0, 56, 53, 43]
82	[0, 0, 47, 48, 0, 49, 30, 29]
83	[0, 0, 53, 0, 50, 51, 54, 53]
84	[0, 41, 0, 100, 46, 0, 100, 48]
85	[0, 100, 0, 64, 55, 56, 55, 100]
86	[0, 31, 0, 50, 47, 50, 40, 53]
87	[47, 57, 100, 52, 100, 100, 43, 55]
88	[0, 50, 51, 0, 23, 45, 0, 54]
89	[0, 0, 0, 0, 0, 28, 55, 0]
90	[0, 0, 0, 0, 49, 0, 46, 36]
91	[100, 0, 0, 100, 0, 59, 100, 100]
92	[22, 0, 100, 48, 0, 100, 48, 10]
93	[0, 0, 30, 0, 58, 0, 26, 50]
94	[0, 0, 0, 0, 0, 0, 0, 43]
95	[0, 0, 0, 0, 0, 50, 45, 11]

Figure 9. All unique output indices for register s_{80} are combined. The table illustrates the differentials of these combinations for each target register in $S_{byte10}\&S_{byte11}$ Differential combinations with all values equal to 100 are colored in green, and combinations with all values larger than 0 and less than 100 are colored in red.

Rregisters	Combining All Unique Output Differentials ($\delta_0,\ldots,\delta_j$) ($\delta_{34}, \delta_{66}, \delta_{98}, \delta_{108}, \delta_{128}, \delta_{140}, \delta_{156}, \delta_{162}$)	All $\delta_j = 1$ in the Same Test
80	[0, 51, 0, 0, 48, 40, 52, 57]	0
81	[100, 100, 100, 100, 100, 100, 100, 100]	100
82	[26, 23, 19, 40, 23, 62, 13, 45]	0
83	[0, 0, 53, 0, 0, 28, 49, 55]	0
84	[0, 0, 45, 0, 49, 0, 55, 55]	0
85	[0, 0, 0, 0, 0, 48, 17, 0]	0
86	[0, 0, 0, 100, 0, 100, 44, 30]	0
87	[0, 47, 0, 26, 36, 0, 46, 100]	0
88	[0, 50, 0, 0, 0, 0, 20, 8]	0
89	[0, 0, 0, 0, 0, 46, 54, 52]	0
90	[0, 0, 0, 50, 46, 48, 100, 45]	0
91	[0, 0, 0, 100, 48, 44, 60, 9]	0
92	[0, 0, 0, 23, 48, 45, 59, 49]	0
93	[0, 0, 0, 55, 0, 100, 58, 50]	0
94	[0, 0, 48, 40, 100, 30, 0, 23]	0
95	[54, 0, 0, 0, 53, 0, 55, 47]	0

Figure 10. Number of counts for which all the combinations of unique differentials are one for each register in $S_{byte10}\&S_{byte11}$ in the 100 random tests. The unique combination of output indices listed here is for target register s_{81}. Differential combinations with all values equal to 100 are highlighted in green, and combinations with all values larger than 0 and less than 100 are highlighted in red.

in $S_{byte10}\&S_{byte11}$, we may infer the condition for identifying the target register s_{80} when all the unique output differentials of s_{80} are one while at least one of the unique output differentials of s_{87} is zero. Therefore, the condition for the injected fault to be located at s_{80} is given by $(\delta_{40}, \delta_{72}, \delta_{104}, \delta_{114}, \delta_{134}, \delta_{146}, \delta_{162}, \delta_{168}) \neq (1, 1, 1, 1, 1, 1, 1, 1)$, i.e., excluding the condition for s_{87}, and $(\delta_{44} = \delta_{102} = \delta_{114} = \delta_{118} = \delta_{134} = \delta_{146} = \delta_{150} = \delta_{166} = 1)$, where δ_i represents the i^{th} output index.

After conducting the above approach to all the registers in the last six two-byte arrays, the injected faulty target can be located for the majority of the registers. However, there still exists a situation where a few target registers cannot be determined due to insufficient conditions. Then, the probability of successfully determining the remaining faulty register is calculated based on the experiment results of the combined output indices, as

Registers	Combining All Unique Output Differentials ($\delta_0,\ldots,\delta_j$) ($\delta_{44}, \delta_{102}, \delta_{114}, \delta_{118}, \delta_{134}, \delta_{146}, \delta_{150}, \delta_{166}$)	All $\delta_j = 1$ in the Same Test
80	[100, 100, 100, 100, 100, 100, 100, 100]	100
81	[24, 31, 58, 45, 0, 56, 53, 43]	0
82	[0, 0, 47, 48, 0, 49, 30, 29]	0
83	[0, 0, 53, 0, 50, 51, 54, 53]	0
84	[0, 41, 0, 100, 46, 0, 100, 48]	0
85	[0, 100, 0, 64, 55, 56, 55, 100]	0
86	[0, 31, 0, 50, 47, 50, 40, 53]	0
87	[47, 57, 100, 52, 100, 100, 43, 55]	4
88	[0, 50, 51, 0, 23, 45, 0, 54]	0
89	[0, 0, 0, 0, 0, 28, 55, 0]	0
90	[0, 0, 0, 0, 49, 0, 46, 36]	0
91	[100, 0, 0, 100, 0, 59, 100, 100]	0
92	[22, 0, 100, 48, 0, 100, 48, 10]	0
93	[0, 0, 30, 0, 58, 0, 26, 50]	0
94	[0, 0, 0, 0, 0, 0, 0, 43]	0
95	[0, 0, 0, 0, 0, 50, 45, 11]	0

Figure 11. Number of counts for which all the combinations of unique differentials are one for each register in $S_{byte10}\&S_{byte11}$ in the 100 random tests. The unique combination of output indices listed here is for target register s_{80}. Differential combinations with all values equal to 100 are highlighted in green, and combinations with all values greater than 0 and less than 100 are highlighted in red.

given in

$$P(s_i) = \frac{m}{m+n} \times 100, \quad \text{for } 0 \le i \le 127, \tag{17}$$

where $P(s_i)$ is the probability of determining the faulty register s_i, and m and n stand for the number of times the faulty register and other registers equal one for the combined output indices in the same test round. For these experiments, we used the threshold of 100 random tests similar to the threshold value used in previous works. A larger threshold value enables us to estimate the probability more precisely; however, this also slows down the experimental process. To identify the fault target, we have included all the unique output signatures for a given target register. Hence, using the 100 tests to identify the target register is reasonable, given that all the respective output indices resulting in a differential of 1 in the same round test are negligible. In our experiments, if the probability of locating the fault is higher than 95%, then we consider that the fault target can be determined with a high probability. By implementing these approaches to all eight two-byte arrays in the LFSR, we can determine the required conditions to confirm which target registers in LFSR are injected with a fault. The details of the required keystreams to be observed and the conditions to be satisfied are listed in tables in Appendix A.

Summarizing required conditions to confirm the effect of random fault

The method employed in this work reduces the number of output indices that need to be observed to identify the target register in the LFSR. For example, based on earlier observations, the output differentials required to be observed for identifying the fault targets in $S_{byte0}\&S_{byte1}$ are 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 64, 68, 82, 84, 86, 92, 94, and 96. Applying this method to all the eight two-byte arrays, we get the output indices required for the entire LFSR. The total number of required keystream bits for every two-byte array is calculated based on the tables in Appendix A. Table 6 shows the number of keystream bits that need to be observed in each two-byte array to confirm whether the register is affected by the injected fault.

Inject a fault within four consecutive bytes

To further relax the moderate control precision, instead of focusing on two consecutive bytes, we also investigated the injection of a random fault that affects a randomly chosen single register from a collection of thirty-two registers, i.e., four consecutive bytes: $S_{byte0}\&S_{byte1}\&S_{byte2}\&S_{byte3}, \cdots, S_{byte12}\&S_{byte13}\&S_{byte14}\&S_{byte15}$. Similar to the experiments in the prior sections that have been carried out on the eight two-byte arrays, a series of experiments are implemented on the four four-byte arrays, including:

1. Confirming the required single output indices, to eliminate the duplicated output indices in four bytes to

Table 6. List of keystream indices required for the attack and the number of output indices for two-byte arrays

Two-byte	Output indices
S_{byte0}&S_{byte1} (18 indices)	34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 64, 68, 82, 84, 86, 92, 94, 96
S_{byte2}&S_{byte3} (21 indices)	44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 80, 82, 84, 98, 100, 108, 110, 112
S_{byte4}&S_{byte5} (38 indices)	38, 40, 42, 60, 62, 66, 67, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 130, 132, 134, 136, 138
S_{byte6}&S_{byte7} (44 indices)	48, 50, 52, 54, 56, 58, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154
S_{byte8}&S_{byte9} (45 indices)	34, 36, 38, 40, 42, 60, 62, 64, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 103, 104, 106, 108, 110, 112, 114, 116, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 152, 154, 156, 158, 160
S_{byte10}&S_{byte11} (71 indices)	0, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180
S_{byte12}&S_{byte13} (76 indices)	2, 4, 6, 8, 10, 12, 14, 16, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 182, 184, 186, 188, 190, 192, 194, 196
S_{byte14}&S_{byte15} (75 indices)	18, 20, 22, 24, 26, 28, 30, 32, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 198, 200, 202, 204, 206, 208, 210, 212

avoid ambiguous results;

2. Confirming the required pairs of output indices, where duplicated pairs will be removed to further determine the fault location;
3. Determining the required combinations of output indices to confirm the effect of the random injected fault. To further investigate the conditions used to confirm all the registers in the four-byte arrays, a combination of all the unique output indices is used instead of pairs. The probabilistic approach is used to identify the fault target registers that cannot be determined using deterministic signatures.

Applying this method to all the four-byte arrays, we obtained the total output indices required to be observed for identifying the fault targets. The probability is calculated for the target registers that cannot be directly determined. If the probability is greater than 95%, we consider the inaccuracy to be negligible, and the target register can be determined in such a case with high probability. If the probability is less than 95%, the probability is used to represent the chances of targeting the register. Table 7 shows the output indices required to be observed and the number of output indices for each four-byte array. The complete tables of the required keystream bits to be observed and the conditions to be satisfied for fault injection for the four-byte precision model are listed in tables in Appendix B.

Based on the experimental results, 96 target registers can be determined using the deterministic method, and the rest 32 can be probabilistically represented. Hence, the faults injected into the LFSR can be determined, and we conclude that a fault attack on Grain-128AED can also be applied with the four-byte moderate control model.

CONCLUSION

In this work, we extended the DFA on Grain-128AEAD with two relaxed fault attack models: a two-byte moderate control model and a four-byte moderate control model. Unlike the previous work, instead of every single byte, the two-byte model focuses on every two consecutive bytes, and the four-byte model focuses on every four consecutive bytes. The results of these attacks are promising and suggest that moderate control models are feasible to identify the majority of the target registers with a high probability. The models used in this paper are more practical to implement as they have more relaxed assumptions. The previous experimental

Table 7. List of keystream indices required for the attack and the number of output indices for each four-byte array

Four-byte	Output indices
$S_{byte0}\&S_{byte1}\ \&S_{byte2}\&S_{byte3}$ (38 indices)	34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 104, 106, 110, 112
$S_{byte4}\&S_{byte5}\ \&S_{byte6}\&S_{byte7}$ (59 indices)	38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154
$S_{byte8}\&S_{byte9}\ \&S_{byte10}\&S_{byte11}$ (75 indices)	0, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180
$S_{byte12}\&S_{byte13}\ \&S_{byte14}\&S_{byte15}$ (107 indices)	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 123, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212

results show that Grain-128AEAD is vulnerable to a state recovery attack if an adversary can successfully inject fault in the LFSR registers–therefore, the results reported in this work can be used to perform a fault-based state recovery attack on Grain-128AEAD with a more relaxed fault model.

We note that the fault model used in this work may require more keystream bits compared to the single-byte moderate control; however, the two models used in this work have a more relaxed assumption and, hence, are more practical in implementation. It is worth noting that the designers of Grain-128AEAD did not claim security against fault attacks, so the results reported in this article do not violate their security claims. These findings highlight the importance of implementing proper physical protections to prevent fault attacks on Grain-128AEAD.

We also note that when the control is relaxed even further to no control, it appears infeasible to recover any bits, suggesting that additional investigation is needed. Therefore, future research could focus on applying moderate control models to explore the feasibility of differential fault attacks using no control models. These experiments could be conducted using a probabilistic approach, such as determining the likelihood that a register is affected by fault injection based on output differentials. Additionally, since our experiments could only recover the initial states of the cipher, future work could focus on investigating approaches for recovering the secret key.

DECLARATIONS

Authors' contributions

Made substantial contributions to the conception and design of the study and performed data analysis and interpretation: Fang T, Salam I, Yau WC

Availability of data and materials

Not applicable.

Financial support and sponsorship

This work is supported by the Xiamen University Malaysia Research Fund under Grant XMUMRF/2022-C9/IECE/0032.

Conflicts of interest

All authors declared that there are no conflicts of interest.

Ethical approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Copyright

© The Author(s) 2024.

REFERENCES

1. NIST Lightweight Cryptography Project. Available from: https://csrc.nist.gov/Projects/lightweight-cryptography.
2. Hell M, Johansson T, Meier W, Sönnerup J, Yoshida H. An AEAD variant of the grain stream cipher. In: Carlet C, Guilley S, Nitaj A, Souidi EM, editors. Codes, Cryptology and Information Security. Cham: Springer International Publishing; 2019. pp. 55–71. DOI
3. Hell M, Johansson T, Maximov A, Meier W, Yoshida H. Grain-128AEADv2: strengthening the initialization against key reconstruction. In: Conti M, Stevens M, Krenn S, editors. Cryptology and Network Security. Cham: Springer International Publishing; 2021. pp. 24–41. DOI
4. Salam I, Ooi TH, Xue L, Yau WC, Pieprzyk J, et al. Random differential fault attacks on the lightweight authenticated encryption stream cipher Grain-128AEAD. *IEEE Access* 2021;9:72568–86. DOI
5. Selmke B, Heyszl J, Sigl G. Attack on a DFA protected AES by simultaneous laser fault injections. In: 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). IEEE; 2016. pp. 36–46. DOI
6. Trichina E, Korkikyan R. Multi fault laser attacks on protected CRT-RSA. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography. IEEE; 2010. pp. 75–86. DOI
7. Skorobogatov S. Optical fault masking attacks. In: 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography. IEEE; 2010. pp. 23–29. DOI
8. Breier J, Hou X. How practical are fault injection attacks, really? *IEEE Access* 2022;10:113122–30. DOI
9. Hell M, Johansson T, Meier W. Grain: a stream cipher for constrained environments. *Int J Wirel Mob Comput* 2007;2:86–93. DOI
10. Hell M, Johansson T, Maximov A, Meier W. A stream cipher proposal: Grain-128. In: 2006 IEEE International Symposium on Information Theory. IEEE; 2006. pp. 1614–18. DOI
11. Ågren M, Hell M, Johansson T, Meier W. Grain-128a: a new version of Grain-128 with optional authentication. *Int J Wirel Mob Comput* 2011;5:48–59. DOI
12. Hell M, Johansson T, Maximov A, Meier W, Sönnerup J, et al. Grain-128AEADv2-A lightweight AEAD stream cipher. Available from: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf.
13. Biham E, Shamir A. Differential fault analysis of secret key cryptosystems. In: Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17. Springer; 1997. pp. 513–25. DOI
14. Dey P, Rohit RS, Sarkar S, Adhikari A. Differential fault analysis on Tiaoxin and AEGIS family of ciphers. In: International Symposium on Security in Computing and Communication. Springer; 2016. pp. 74–86. DOI
15. Salam I, Mahri HQA, Simpson L, Bartlett H, Dawson E, et al. Fault attacks on Tiaoxin-346. In: Proceedings of the Australasian Computer Science Week Multiconference. Association for Computing Machinery; 2018. pp. 1–9. DOI
16. Bartlett H, Dawson E, Qahur Al Mahri H, Salam MI, Simpson L, et al. Random fault attacks on a class of stream ciphers. *Secur Commun Netw* 2019;2019. DOI
17. Wong KKH, Bartlett H, Simpson L, Dawson E. Differential random fault attacks on certain CAESAR stream ciphers. In: International Conference on Information Security and Cryptology. Springer; 2019. pp. 297–315. DOI
18. Dey P, Rohit RS, Adhikari A. Full key recovery of ACORN with a single fault. *J Inf Secur Appl* 2016;29:57–64. DOI
19. Salam I, Law KY, Xue L, Yau WC. Differential fault based key recovery attacks on TRIAD. In: International Conference on Information Security and Cryptology. Springer; 2020. pp. 273–87. DOI
20. Karmakar S, Roy Chowdhury D. Fault analysis of Grain-128 by targeting NFSR. In: Progress in Cryptology–AFRICACRYPT 2011: 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings 4. Springer; 2011. pp. 298–315. DOI
21. Sarkar S, Banik S, Maitra S. Differential fault attack against grain family with very few faults and minimal assumptions. *IEEE Trans Comput* 2014;64:1647–57. DOI
22. Banik S, Maitra S, Sarkar S. A differential fault attack on the grain family under reasonable assumptions. In: Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13. Springer; 2012. pp. 191–208. DOI
23. Dey P, Chakraborty A, Adhikari A, Mukhopadhyay D. Improved practical differential fault analysis of Grain-128. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE; 2015. pp. 459–64. DOI
24. Baksi A, Bhasin S, Breier J, Jap D, Saha D. A Survey on Fault Attacks on Symmetric Key Cryptosystems. *ACM Comput Surv* 2023;55:1-34. DOI