Deep Learning-Assisted Improved Differential Fault Attacks on Lightweight Stream Ciphers

Kok Ping Lim, Dongyang Jia and Iftekhar Salam

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

Abstract. Lightweight cryptographic primitives are widely deployed in resource-constrained environments, particularly in Internet of Things (IoT) devices. Due to their public accessibility, these devices are vulnerable to physical attacks, especially fault attacks. Recently, deep learning–based cryptanalytic techniques have demonstrated promising results; however, their application to fault attacks remains limited, particularly for stream ciphers. In this work, we investigate the feasibility of deep learning assisted differential fault attacks on three lightweight stream ciphers, namely ACORNv3, MORUSv2, and ATOM, under a relaxed fault model in which a single-bit bit-flipping fault is injected at an unknown location. We develop and train multilayer perceptron (MLP) models to identify the fault locations. Experimental results show that the trained models achieve high identification accuracies of 0.999880, 0.999231, and 0.823568 for ACORNv3, MORUSv2 and ATOM, respectively, and outperform traditional signature-based methods. For the secret recovery process, we introduce a threshold-based method to optimize the number of fault injections required to recover the secret information. The results show that the initial state of ACORN can be recovered with 21 to 34 faults, while MORUS requires 213 to 248 faults, with at most 6 bits of guessing. Both attacks reduce the attack complexity compared to existing works. For ATOM, the results show that it possesses a higher security margin, as the majority of state bits in the Nonlinear Feedback Shift Register (NFSR) can only be recovered under a precise control model. To the best of our knowledge, this work provides the first experimental results of differential fault attacks on ATOM.

Keywords: Differential Fault Attack · MLP · ACORN · MORUS · ATOM

1 Introduction

Over the past two decades, various cryptographic competitions, such as eSTREAM, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), and the National Institute of Standards and Technology (NIST) Lightweight Cryptography Project, were initiated for the design and standardization of lightweight cryptographic primitives. These primitives ensure secure communication for resource-constrained devices, e.g., Internet of Things (IoT) devices, as traditional ciphers are intended to operate in resource-rich environments. Many cryptanalysis techniques, such as cube attacks, side channel attacks, and fault attacks are applied to the lightweight ciphers. Among these attacks, fault attacks have gained significant attention in recent years due to their effectiveness against physical hardware and the ongoing advancement of fault injection techniques. Since IoT devices are commonly deployed in publicly accessible areas, adversaries often have physical access to them, making these devices viable targets for fault attacks [STB25].

E-mail: cys2209667@xmu.edu.my (Kok Ping Lim), cme2209122@xmu.edu.my (Dongyang Jia), iftekhar.salam@xmu.edu.my (Iftekhar Salam)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-05-19.

Fault attack was first introduced by Boneh et al. [BDL97] and targets physical hardware rather than the cryptographic algorithm itself. They applied the attack to RSA and Rabin signature schemes by introducing hardware faults to factor the large prime modulus N. In the same year, Biham and Shamir [BS97] implemented a variant of fault attack, differential fault attack (DFA), and applied it against the Data Encryption Standard (DES) to recover the 56-bit secret key. The attack works by inducing an error (fault) during the operation of the cipher to obtain the faulty outputs. By analyzing the differences between fault-free and faulty outputs, some secret information of the cipher, such as the internal state or the secret key can be recovered. Hoch and Shamir [HS04] further extended DFA to stream ciphers such as RC4, LILI-128, and SOBER-t32.

Following these foundational works, differential fault attacks were further extended and refined, and subsequently applied to evaluate the security of a wide range of cryptographic primitives. Orumiehchiha et al. [ORSP20] applied the first DFA against WG-8 stream cipher to recover the initial state and secret key of the cipher using 6 faults with a time complexity of 2^{24}. Ma, Tian and Qi [MTQ21] applied DFA to the LIZARD stream cipher and demonstrated that its initial state can be recovered using only six faults. Salam et al. [SOX+21] used bit-flipping and random fault attacks, under different control models, to recover the initial state of Grain128AEAD. In a later work, Salam et al. [SYPP23] recovered the initial state and secret key of CLX-128 by applying bit-flipping and random fault attacks with 54 and 134 faults, respectively. Rostami et al. [ROSA25] applied a random fault attack against Enocoro-80 and Enocoro-128v2. The results show that the attack only require 2 faults to recover the internal state of Enocoro-80 and Enocoro-128v2 with the time complexity of 2^{16} and 2^{29}, respectively. Radheshwar and Roy [RR25] applied DFA against ChaosForge stream cipher to recover its internal state and the secret key. Prajasantosa and Salam [PS25] recovered 58 state bits and 15 key bits of TinyJAMBU with bit-flipping and random fault attacks, under different control models.

Recent advancements in neural cryptanalysis have leveraged deep learning (DL) to construct high-performance neural distinguishers for various lightweight block ciphers [Goh19, HRC21, ZL22]. The deep learning approach has also been used to predict the secret key of lightweight block ciphers, such as S-DES and S-AES [KLK+23]. In terms of deep learning assisted fault attack, Baksi et al. [BSS+21] developed multilayer perceptron (MLP) models to identify the fault location for lightweight stream ciphers, Grain128a and Lizard, which achieve better performance than the traditional signature-based method. Cheng et al. [COZ+23] proposed a deep learning-based fault analysis, using MLP and Convolutional Neural Network (CNN) to recover the key of AES with 1 fault and an average of 1488 ciphertexts. Saha et al. [SAB+23] implemented a fault attack leakage assessment test with deep learning to detect fault-induced leakage in the ciphertext distributions. Mondal et al. [MDR+24] developed a deep neural network for identifying the fault location of Subterranean 2.0 and recovered the embedded secret key with no more than 5 faults.

The existing literature shows that differential fault attacks remain a popular cryptanalytic technique for state recovery or key recovery. In addition, the integration of deep learning in fault attacks offers a new path to enhance the performance of the attack. Given the limited research on deep learning–assisted DFA, the feasibility of such fault attacks remains largely underexplored. To address this, we propose a deep learning–assisted DFA framework for enhancing differential fault attacks and apply it to ACORNv3, MORUSv2, and ATOM.

1.1 Existing Differential Fault Attacks on ACORN and MORUS

Dalai and Roy [DR17] implemented differential fault attacks on the first and second version of ACORN. By precisely injecting 326 faults and guessing 120 state bits, the full internal state of the cipher at the 39th clock of the encryption phase can be recovered, with an attack complexity of 2^{120}. Zhang et al. [ZFL17] applied DFA against ACORNv2 under

the fault model of no control over the fault location. By analyzing the occurrence of 1s in the differential keystream, the fault locations were identified with a high probability of 97.08%. They used linearization and guess-and-determine strategies to solve the equations and recover the initial state. The linearization strategy required 31 to 88 faults to recover the initial state with an attack complexity of $c \cdot 2^{179.19-1.76n}$, where n is the number of fault injections, and c is the time complexity of solving the equations. For the guess-and-determine method, guessing 67 state bits could recover the initial state of the cipher with 41 fault experiments.

Siddhanti et al. [SSMC17] extended DFA to ACORNv3, optimizing the classical differential framework by incorporating both forward and backward state-update equations and fixing selected bits before SAT solving. In their attack, the fault locations were identified using the signature-based method. Due to the large state size of ACORN, the attack was optimized by guessing 20 state bits. Their work showed that the initial state of ACORNv3 can be recovered using 9 faults and a 1200-bit keystream, with a time complexity of $2^{25.40}$. Zhang et al. [ZFL18] also performed DFA on ACORNv3, following the same attack process as in [ZFL17]. To identify the fault location, they introduced the "unique" versus "non-unique" differential sets method. By incorporating the keystream extension and high probability priority strategies, the accuracy of their proposed method was 99.998%, using 163-bit differential keystream. For equation solving, a guess-and-determine method was used to recover the full initial state of the cipher. The attack complexity is $c \cdot 2^{146.5-3.52n}$, where n is the number of faults such that $26 < n < 43$ faults and c is the time complexity of solving the equations.

In summary, most existing work on DFAs against ACORN relies on the guess-and-determine method to recover the internal state. Furthermore, some of the works focus on retrieving only linear equations, which requires more fault injections to recover the state of the cipher. Additionally, attacks based on the fault model with no control over the fault location rely solely on traditional signature-based methods to identify the fault location. This presents a gap for investigating the use of deep learning to identify the fault locations and improve the attack complexity. Furthermore, incorporating higher-degree equations with a threshold-based technique could help optimize the number of required faults.

Wong et al. [WBSD20] performed a theoretical analysis of DFA on MORUS, targeting the bit-wise AND operation of the keystream generation function, where a random fault is injected into one of the two involved registers. By exploiting these faults, an adversary can reveal the non-zero state bits in the targeted register whenever the differential keystream bits are 1. Repeated fault injections allow these non-zero bits to be recovered with high probability, while the remaining bits are assumed to be 0 with high confidence. However, the attack only achieved partial state recovery and its reliance on precise control over the fault location is a relatively strong assumption, which motivated us to investigate attaining full state recovery through DL-based models in more relaxed settings.

1.2 Our Contributions

We propose a deep learning-assisted differential fault attack (DL-DFA) framework under the single-bit bit-flipping fault model with no control over the fault location. The contributions of this work are summarized as follows:

MLP Models for Fault Location Identification. We generate differential keystream data sets for ACORNv3, MORUSv2 and ATOM, and design and train MLP models to identify the fault locations for these ciphers. The models achieve accuracies of 0.999880, 0.999231 and 0.823568 for ACORNv3, MORUSv2 and ATOM, respectively–significantly improving upon the traditional signature-based method. Table 1 summarizes the results of fault location identification in this work.

Threshold-based Attack Framework. We introduce a threshold-based method to minimize the number of fault injections. Meanwhile, we include all equations into the

Table 1: Summary of fault location identification accuracy across different ciphers

Cipher	Method	Keystream length	Accuracy	Ref.
ACORNv3	MLP (this work)	152 bits	0.999880	Sec. 4.1
	Set-based	149 bits	0.999774	[ZFL18]
	Signature-based	152 bits	0.999747	Sec. 4.1
MORUSv2	MLP (this work)	384 bits	0.999231	Sec. 4.2
	Signature-based	384 bits	0.952606	Sec. 4.2
ATOM	MLP (this work)	56 bits	0.823568	Sec. 4.3
	Signature-based	56 bits	0.588976	Sec. 4.3

Table 2: Summary of differential fault attacks on ACORNv3, MORUSv2, and ATOM

Cipher	Fault type	# Faults	# Recovered bits	Complexity	Ref.
ACORNv3	Bit-flipping	9	Full initial state	$2^{25.40}$	[SSMC17]
	Bit-flipping	27–42	Full initial state	$c \cdot 2^{146.5-3.52n}$	[ZFL18]
	Bit-flipping	21–34	Full initial state	Negligible	Sec. 4.1
MORUSv2	Random	–	256 state bits	–	[WBSD20]
	Bit-flipping	213–248	Full initial state	2^6 (At most)	Sec. 4.2
ATOM	Bit-flipping	46	Majority of NFSR state	$\geq 2^{60}$	Sec. 4.3

equation system for state recovery, regardless of the degree. We achieve a full state recovery on ACORNv3 and MORUSv2, improving the existing results in terms of attack complexity and number of recovered bits, respectively. We also implemented the first DFA against ATOM; the results show that ATOM possesses a higher security margin compared to ACORNv3 and MORUSv2. Table 2 summarizes our results of DFA.

Organization of the Paper. We provide a brief background on fault location identification and an overview of ACORNv3, MORUSv2 and ATOM in Section 2. In Section 3, we discuss the attack procedure, including the implementation of the MLP model and the proposed threshold-based method. The implementation of the attacks and the results for the three ciphers are presented in Section 4. Finally, we conclude the paper in Section 5.

2 Preliminaries and Background

2.1 Fault Location Identification

Our proposed deep learning-assisted fault attack framework works under the fault model of no control over the fault location. In this model, fault location identification is a critical process, as it can affect the secret recovery process. Consider a toy cipher with 4 bits state, $s_0\|s_1\|s_2\|s_3 = 0\|0\|0\|1$, where the first keystream bit is defined as:

$$z_0 = s_0 \oplus s_1 \oplus s_1 s_2 \oplus s_2 s_3. \tag{1}$$

During the attack, let us assume a fault is injected at the location s_2, resulting in the value of s_2 being complemented. The faulty keystream equation can be rewritten as:

$$z'_0 = s_0 \oplus s_1 \oplus s_1 s'_2 \oplus s'_2 s_3, \tag{2}$$

where $s'_2 = s_2 \oplus 1$. By XOR-ing Equation 1 and Equation 2, we can obtain the differential:

$$\begin{aligned} z_0 \oplus z'_0 &= (s_0 \oplus s_1 \oplus s_1 s_2 \oplus s_2 s_3) \oplus (s_0 \oplus s_1 \oplus s_1 s'_2 \oplus s'_2 s_3) \\ &= s_1 s_2 \oplus s_2 s_3 \oplus s_1 s'_2 \oplus s'_2 s_3 \\ &= s_1(s_2 \oplus s'_2) \oplus s_3(s_2 \oplus s'_2) \\ &= s_1 \oplus s_3. \end{aligned} \tag{3}$$

Substituting the values into Equation (3), we can find that the expression holds true. Note that in the no-control model, the adversary has no knowledge about the fault location. Suppose the fault is injected at s_2, but the adversary misidentifies the fault location as s_3, resulting in the following differential equation instead:

$$\begin{aligned} z_0 \oplus z'_0 &= (s_0 \oplus s_1 \oplus s_1 s_2 \oplus s_2 s_3) \oplus (s_0 \oplus s_1 \oplus s_1 s_2 \oplus s_2 s'_3) \\ &= s_2 s_3 \oplus s_2 s'_3 \\ &= s_2(s_3 \oplus s'_3) \\ &= s_2. \end{aligned} \tag{4}$$

Substituting the values into the equation, it shows that the wrong value is revealed for s_2 due to the misidentification. Thus, it is important to identify the correct fault location, under the no control model, to ensure that the valid differential can be derived. There are several methods to identify the fault location, with the signature-based method being one of the most popular. One can refer to [MSBD15, MSS17, SSMC17] for the details of signature-based method.

2.2 Specification of Ciphers

ACORNv3 is one of the algorithms in the final portfolio of the CAESAR competition for the use case of lightweight applications, while MORUSv2 is one of the additional finalists for high-performance application in the same competition. ATOM, on the other hand, is a stream cipher proposed by Banik et al. [BCI+21] which utilizes a double key filter. In our fault attacks, we mainly focus on the encryption phase of the ciphers, and thus only the encryption process of each cipher is described in the following sections. One can refer to [Wu16, WH16, BCI+21] for the detailed specification of each cipher.

2.2.1 ACORNv3

Let $S = \{s_0, s_1, \cdots, s_{292}\}$ denote the initial state of ACORNv3. In the encryption phase, the cipher processes the plaintext $P = \{p_0, p_1, \cdots, p_{mlen-1}\}$ and outputs the ciphertext $C = \{c_0, c_1, \cdots, c_{mlen-1}\}$. The cipher utilizes the output function $ks(S)$ to generate a keystream bit z_i, where $i = 0, 1, \cdots, mlen - 1$. The output function is defined as:

$$ks(S) = s_{12} \oplus s_{154} \oplus s_{61} s_{235} \oplus s_{93} s_{235} \oplus s_{61} s_{93} \oplus s_{111} s_{230} \oplus s_{66} \overline{s_{230}}, \tag{5}$$

where $\overline{s_{230}}$ denotes the complement of s_{230}. Additionally, a feedback function $fb(S)$ is used to compute a feedback bit y, where the function is defined as

$$fb(S) = s_0 \oplus \overline{s_{107}} \oplus s_{23} s_{244} \oplus s_{160} s_{244} \oplus s_{23} s_{160} \oplus s_{196}, \tag{6}$$

where $\overline{s_{107}}$ denotes the complement of the state bit s_{107}. The encryption process of ACORNv3 is described in Algorithm 1. The state bits s_{61}, s_{107}, s_{154}, s_{193}, s_{230} and s_{289} first undergo an intermediate update. Next, the keystream and feedback bits are computed based on this intermediate state. Finally, the cipher updates the internal state by shifting, the last state bit s_{292} is updated by XOR-ing the feedback bit y with the plaintext bit p_i.

Algorithm 1 Encryption phase of ACORNv3

Initialize list Z
for $i = 0$ **to** $mlen - 1$ **do**
  $s_{289} = s_{289} \oplus s_{235} \oplus s_{230}$
  $s_{230} = s_{230} \oplus s_{196} \oplus s_{193}$
  $s_{193} = s_{193} \oplus s_{160} \oplus s_{154}$
  $s_{154} = s_{154} \oplus s_{111} \oplus s_{107}$
  $s_{107} = s_{107} \oplus s_{66} \oplus s_{61}$
  $s_{61} = s_{61} \oplus s_{23} \oplus s_0$
  $z_i = ks(S)$
  Append z_i to Z
  $y = fb(S)$
  for $j = 0$ **to** 291 **do**
    $s_j = s_{j+1}$
  end for
  $s_{292} = y \oplus p_i$
end for
return $Z = \{z_0, z_1, \cdots, z_{mlen-1}\}$

2.2.2 MORUSv2

We apply the attack on MORUS-640-128, which is one of the three variants of MORUSv2. It takes a 128-bit secret key, with an internal state of 640 bits. The state is constructed of 5 registers, denoted as $S = S_0 \| S_1 \| S_2 \| S_3 \| S_4$, where each register holds 128-bit data. In each step of the encryption phase, the cipher processes a 128-bit plaintext block. Let the plaintext P consist of y blocks, where the plaintext block is denoted as P_i, $i \in \{0, 1, \cdots, y-1\}$. If the last plaintext block P_{y-1} is not a full block, a 0 valued string is padded to that block. The keystream block Z_i is computed by $S_0 \oplus (S_1 \ll 96) \oplus S_2 S_3$, where S_1 is rotated to the left by 96 bits, as defined in Algorithm 2.

Algorithm 2 Encryption phase of MORUSv2

Initialize list Z
for $i = 0$ **to** $y - 1$ **do**
  $Z_i = S_0 \oplus (S_1 \ll 96) \oplus S_2 S_3$
  Append Z_i to Z
  Update the state with state update function
end for
return $Z = \{Z_0, Z_1, \cdots, Z_{y-1}\}$

2.2.3 ATOM

ATOM consists of a linear feedback shift register (LFSR) and a non-linear feedback shift register (NFSR). Let the initial states of the LFSR and NFSR be $L = \{l_0, l_1, \cdots, l_{68}\}$ and $B = \{b_0, b_1, \cdots, b_{89}\}$, respectively. During encryption, the keystream bit is computed as:

$$\begin{aligned} O(B, L) = {} & b_1 \oplus b_5 \oplus b_{11} \oplus b_{22} \oplus b_{36} \oplus b_{53} \oplus b_{72} \oplus b_{80} \oplus b_{84} \oplus l_5 l_{16} \oplus l_{13} l_{15} \\ & \oplus l_{30} l_{42} \oplus l_{22} l_{67} \oplus h(l_7, l_{33}, l_{38}, l_{50}, l_{59}, l_{62}, b_{85}, b_{41}, b_9). \end{aligned} \tag{7}$$

Note that $h(l_7, l_{33}, l_{38}, l_{50}, l_{59}, l_{62}, b_{85}, b_{41}, b_9)$ is a degree 5 function. Both the LFSR and NFSR possess the corresponding feedback function to update the state. The feedback functions for the LFSR and NFSR are denoted by $F(L)$ and $G(B)$, respectively, and are

Algorithm 3 Encryption phase of ATOM

Initialize list Z
for $i = 0$ **to** $mlen - 1$ **do**
  $z_i = O(B, L)$
  Append z_i to Z
  $cnt = l_{62} \| l_{63} \| l_{64} \| l_{65} \| l_{66} \| l_{67} \| l_{68}$
  $fb_B = G(B) \oplus l_0 \oplus k_{cnt} \oplus k_{511+i \ mod \ 128}$
  for $j = 0$ **to** 88 **do**
    $b_j = b_{j+1}$
  end for
  $b_{89} = fb_B$
  $fb_L = F(L)$
  for $j = 0$ **to** 67 **do**
    $l_j = l_{j+1}$
  end for
  $l_{68} = fb_L$
end for
return $Z = \{z_0, z_1, \cdots, z_{mlen-1}\}$

defined as:

$$F(L) = l_0 \oplus l_5 \oplus l_{12} \oplus l_{22} \oplus l_{28} \oplus l_{37} \oplus l_{45} \oplus l_{58}, \tag{8}$$

$$\begin{aligned} G(B) = {} & b_0 \oplus b_{24} \oplus b_{49} \oplus b_{79} \oplus b_{84} \oplus b_3 b_{59} \oplus b_{10} b_{12} \oplus b_{15} b_{16} \oplus b_{25} b_{53} \\ & \oplus b_{35} b_{42} \oplus b_{55} b_{58} \oplus b_{60} b_{74} \oplus b_{20} b_{22} b_{23} \oplus b_{62} b_{68} b_{72} \oplus b_{77} b_{80} b_{81} b_{83}. \end{aligned} \tag{9}$$

Algorithm 3 defines the encryption phase of ATOM. Note that the last seven bits in the LFSR are interpreted as a decimal number, *cnt*, which determines the index of the key bit k involved in the computation of the feedback bit for the NFSR. Specifically, the NFSR feedback bit is computed by XOR-ing the output of update function with l_0 and two key bits chosen through the *cnt* and modulo of current clock. The value of *cnt* is non-deterministic, as it depends on the evolving state of the LFSR.

3 Attack Description

Our attacks are conducted in a simulation-based environment, where the fault injection is simulated via software rather than on a physical device. The attack assumes a known-plaintext attack (KPA) model, where the adversary has access to specific keystream bits. Furthermore, the following assumptions are made:

- The adversary can inject a single-bit bit-flipping fault into the initial state
- The adversary has no control over the fault location.
- The adversary can reset the cipher and run the cipher with the embedded key and initialization vector (IV) to observe the keystream.

The attack consists of two phases, the fault location identification and secret recovery. The details of these two phases are discussed in the following section.

3.1 Fault Location Identification

The fault location identification is the critical process in our attack, as the fault is injected at an unknown location. Without knowledge of the fault location, we cannot derive valid differential keystream equations to recover the secret of the cipher. To address this, we trained MLP models to identify the fault location using differential keystream datasets.

3.1.1 Dataset Generation

Since no public datasets are available for fault location identification for these ciphers, we generated the datasets ourselves. The procedure of dataset generation for each cipher is defined in Algorithm 4. For each possible fault location f of the cipher, 1536 samples are

Algorithm 4 Dataset Generation

```
for each possible fault location f of the cipher do
  for j = 1 to 1536 do
    Generate random key K and IV V
    Run the cipher and obtain fault-free keystream Z
    Reset the cipher to initial state
    Inject bit-flipping fault at location f
    Run the cipher and obtain faulty keystream Z'
    Obtain differential keystream ΔZ = Z ⊕ Z'
    if j ≤ 1024 then
      Write K, V, Z, Z', ΔZ, f to training.csv
    else if j ≤ 1280 then
      Write K, V, Z, Z', ΔZ, f to testing.csv
    else
      Write K, V, Z, Z', ΔZ, f to validation.csv
    end if
  end for
end for
```

generated, each using a random key, K and IV, V. The key-IV pair is used to run the cipher for producing the fault-free keystream Z. Subsequently, the cipher is reset and run with the same key-IV pair, but a bit-flipping fault is injected to location f in the initial state to produce the faulty keystream Z'. The differential keystream is then computed by XOR-ing Z with Z'. Finally, the key-IV pair, fault-free keystream, faulty keystream, differential keystream and the fault location are recorded into the csv files.

For each cipher, three datasets are generated: training, testing, and validation. To enhance the neural network's ability to capture fault propagation patterns, the train-test ratio is set as 80:20. The size of training dataset is set as $n \cdot 2^{10}$, while testing and validation datasets are $n \cdot 2^8$, where n is the number of possible fault locations. Hence, each fault location has 2^{10} samples for training, 2^8 samples for testing. The validation dataset is used to evaluate the performance of the model during the training. The length of the differential keystream is set to the range between 50% and 60% of the state size of the cipher, as empirical studies suggest that this range is sufficient to capture the unique fault propagation signatures associated with different fault locations.

3.1.2 Implementation of Multilayer Perceptron Model

We employ multilayer perceptron (MLP) models to identify the fault locations. The models were implemented with the Keras library provided in the TensorFlow framework on the Google Colab platform. The number of neurons in the input layer is the length of the differential keystream, while the number of neurons in the output layer is the size of the internal state of the cipher, i.e., the number of possible fault locations.

The model is compiled with the Adam optimizer and the sparse categorical cross-entropy loss function. As this is a multi-class classification problem, accuracy is used as the primary evaluation metric. The model is trained for a maximum of 50 epochs, with an early-stopping criterion based on validation accuracy. If no improvement in validation accuracy is observed for seven consecutive epochs, the training process is terminated, and the weights associated

with the best validation accuracy are restored. Various hyperparameters, including the number of hidden layers, number of neurons per layer, dropout rate, and initial learning rate, are chosen through a systematic search over selected configurations.

3.2 Secret Recovery

Once the MLP models are trained, we proceed to apply the differential fault attack to the targeted ciphers. Figure 1 illustrates the process of the state recovery method. The attack begins by obtaining the fault-free keystream of the cipher. The cipher is then reset to its initial state, and a single bit-flipping fault is injected at an unknown location. We then run the cipher with the faulty initial state to obtain the faulty keystream. The differential keystream is then fed into the trained MLP model to identify the fault location. If the fault location is correctly identified, we proceed to the equation construction stage. Conversely, if the fault location is misidentified, the cipher is reset and the process is repeated.

With the correct fault location, we generate the corresponding differential keystream equations. A threshold is introduced to determine the minimum number of differential equations required for the state recovery while minimizing the number of fault injections. If the number of differential equations is below the defined threshold, the fault injection process is repeated to gather more information. Once the number of equations reaches or exceeds the threshold, we attempt to solve the system using the Gröbner basis method provided by the SageMath library. In our experiments, we set a one-minute time limit for the solver. If the Gröbner basis returns the result within this time frame, we then verify the recovered secret bits. Otherwise, the current computation is terminated, the threshold is increased by 5 and the process is repeated. Increasing the threshold adds more differential equations to the system, providing more information to the solver and increasing the likelihood of a successful recovery, but also increases the number of faults.

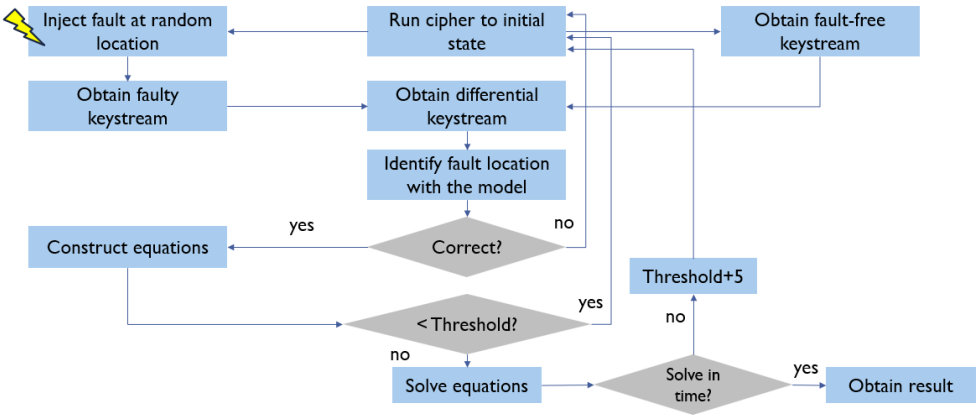


Figure 1: Threshold-based secret recovery process.

4 Experimental Evaluation and Analysis of the Attacks

The implementation[1] of the attack framework and the results for each cipher are discussed in this section. Note that the attacks on ACORN and MORUS follow the attack assumption defined in Section 3. For the attack on ATOM, there is a minor change in attack assumptions, which we will restate in the corresponding section. Due to the varying

[1]Source codes and datasets are available via GitHub: https://github.com/kokping0605/Deep_Learning_Assisted_Fault_Attack_Framework.git

structure of these ciphers, the model architectures and state recovery processes utilize different settings; these are discussed in detail in the following sections.

4.1 Differential Fault Attack on ACORNv3

For the attack on ACORN, the adversary is assumed to have knowledge of a 152-bit keystream. In this attack, the adversary obtains the differential keystream and feeds it into the trained MLP model to identify the fault location and generate the corresponding differential equations with respect to the initial state of the cipher. Once a sufficient number of equations is collected, they are solved to recover the initial state of ACORNv3.

4.1.1 Implementation of MLP Model for Fault Location Identification

Before conducting the attack, three datasets, namely training, testing and validation datasets are generated using Algorithm 4 to train the MLP model for fault location identification in ACORNv3. Training and hyperparameter tuning follow the method defined in Section 3.1.2. The architecture of the neural network is described as follows:

- **Input layer:** The number of neurons in the input layer is 152.
- **Hidden layer:** Five hidden layers, with 152, 512, 512, 512, and 512 neurons, respectively, are used. Each is a dense layer applying a fully connected transformation with L_2 weight decay $\left(\lambda = 1 \times 10^{-6}\right)$ to prevent overfitting, followed immediately by batch normalization to normalize intermediate activations and mitigate internal covariate shift. The ELU activation function is used to combine fast convergence with robustness to vanishing gradients and a dropout rate of 0.3 is set after each activation to prevent co-adaptation of neurons and reduce overfitting.
- **Output layer:** The number of neurons in this layer is 293, and they are activated with the softmax activation function.

4.1.2 Implementation of Differential Fault Attack on ACORN

The detailed attack process of ACORN is defined in Algorithm 5. Initially, two lists $\mathcal{E}$ and $\mathcal{F}$, are defined to store the differential equations and identified fault locations, respectively. The threshold $\mathcal{T}$ determines the minimum number of linear equations required in the system. Through our observation of different threshold settings, we found that $\mathcal{T} = 150$ provides the best results for the state recovery attack. Prior to the attack, the adversary obtains the 152-bit fault-free keystream, $Z = \{z_0, z_1, \cdots, z_{151}\}$ and generates fault-free Algebraic Normal Form (ANF) with respect to the initial state.

In the online phase, a bit-flipping fault is injected at a random location f in the initial state, such that s_f is complemented. The adversary utilizes the resulting faulty initial state to obtain the faulty keystream, $Z' = \{z'_0,\ z'_1,\ \cdots,\ z'_{151}\}$, and computes the differential keystream, $\Delta Z^f = \{\Delta z_0^f, \Delta z_1^f, \cdots, \Delta z_{151}^f\}$ by $\Delta z_i^f = z_i \oplus z'_i$, where $i = 0, 1, \cdots, 151$. The differential ΔZ^f is fed to the MLP model to identify the fault location. If the fault location f is correctly identified and is not already present in $\mathcal{F}$, it is added to the list. If the location has already been identified, the adversary re-injects a fault at a different random location. This ensures that only unique fault locations contribute to the equation system.

Once a unique location is identified, the faulty ANFs with respect to the initial state of the cipher, $FKS = \{fks_0, fks_1, \cdots, fks_{151}\}$ are generated. The adversary then derives the differential equations, $dks = nks_i \oplus fks_i \oplus \Delta z_i^f$, where $i = \{0, 1, \cdots, 151\}$. Each time a unique linear differential equation is obtained, the counter num_deg1 is increased by 1 and the equation is added into the equations list $\mathcal{E}$. The obtained unique quadratic equation is also added to the same list $\mathcal{E}$. This process is repeated iteratively until num_deg1

Algorithm 5 Differential Fault Attack on ACORNv3

Initialize list $\mathcal{E}, \mathcal{F}$
$solve \leftarrow$**false**, $\mathcal{T} \leftarrow 150$, $num_deg1 \leftarrow 0$
$Z = \{z_0, z_1, \cdots, z_{151}\} \leftarrow encACORN(S, P)$
Generate fault-free ANF, $NKS = \{nks_0, \cdots, nks_{151}\}$
while $solve$ **is false do**
  while $num_deg1 < \mathcal{T}$ **do**
    Reset cipher to initial state S'
    Inject fault at location f, where $s_f = s_f \oplus 1$
    $Z' = z'_0, z'_1, \cdots, z'_{151} \leftarrow encACORN(S\prime, P)$
    for $i = 0$ **to** 151 **do**
      $\Delta z_i^f = z_i \oplus z'_i$
    end for
    $f' \leftarrow modelACORN(\Delta Z^f = \{\Delta z_0^f, \cdots, \Delta z_{151}^f\})$
    if f **not in** $\mathcal{F}$ **and** $f = f'$ **then**
      Append f to $\mathcal{F}$
    else
      continue
    end if
    Generate faulty ANF, $FKS = \{fks_0, \cdots, fks_{151}\}$
    for $i = 0$ **to** 151 **do**
      $dks = nks_i \oplus fks_i \oplus \Delta z_i^f$
      if $dks.deg = 1$ **and** dks **not in** $\mathcal{T}$ **then**
        Append dks to $\mathcal{E}$
        $num_deg1 = num_deg1 + 1$
      else if $dks.deg = 2$ **and** dks **not in** $\mathcal{T}$ **then**
        Append dks to $\mathcal{E}$
      end if
    end for
  end while
  Solve $\mathcal{E}$ with Gröbner basis
  if Gröbner basis can solve in time **then**
    $solve \leftarrow$**true**
  else
    $\mathcal{T} = \mathcal{T} + 5$
  end if
end while

is greater than or equal to the threshold, $\mathcal{T}$. Subsequently, the system of differential equations in $\mathcal{E}$ are solved using Gröbner basis to recover the initial state S.

4.1.3 Results and Discussions of the Attack on ACORNv3

The performance of the trained model is shown in Table 3. Utilizing the proposed MLP architecture, the model achieves a high accuracy of 99.9880%, showing the model has successfully learned the fault propagation patterns to identify the fault locations with 152-bit differential keystream. The precision score of 99.9880% shows that the model is robust and effective to identify the actual fault locations with a negligible false positive rate. In terms of recall of 99.9880%, it shows that the model is able to capture nearly all fault locations.The F1-score of 99.9880% demonstrated balanced precision-recall across all target locations, validating the architectural efficacy and hyperparameter configuration.

Table 3: Performance of trained model for identifying fault locations on ACORNv3

Accuracy	Precision	Recall	F1-score
0.999880	0.999880	0.999880	0.999880

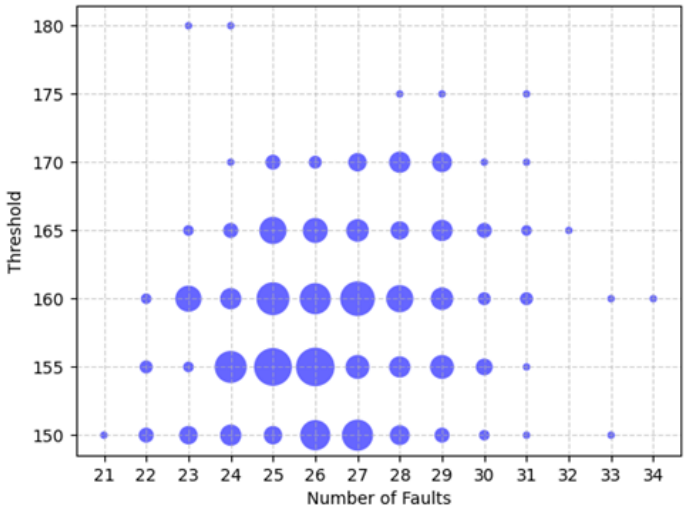


Figure 2: Correlation between fault injection counts and equation thresholds for ACORNv3

Furthermore, we benchmarked the performance of our model against the set-based fault location identification method by Zhang et al. [ZFL18]. Additionally, we implemented a traditional signature-based method using our generated dataset, where the training data were used to generate signatures and the testing data were used for evaluation. Because the differential keystream used by our model is longer than the 99-bit keystream used by Zhang et al. [ZFL18], we performed the comparison based on the keystream extension method described in their work. For the 149-bit keystream extension case, our model achieves an accuracy that is 0.0106% higher than that of the set-based method. Compared to the traditional signature-based method, which achieved an accuracy of 99.9747%, our proposed MLP model provides a 0.0133% improvement. These results confirm that the MLP-based approach offers superior and more robust fault location identification.

A total of 400 experiments were conducted to analyze the performance of the proposed method in recovering the initial state of ACORNv3. Excluding eight experiments exhibiting misidentified fault locations, the remaining trials demonstrated that the 293-bit initial state of ACORNv3 can be fully recovered. The attack was successful with a high probability of 0.98. While experiments were halted upon misidentification, the model still identified the majority of fault locations correctly in those cases. This suggests that even with minor identification errors, a substantial portion of the state could likely be recovered, as inconsistent equations can be filtered out through satisfiability checking.

Figure 2 illustrates the relationship between the thresholds and the number of faults required to recover the initial state of ACORNv3. Experimental results show that the number of faults ranges between 21 and 34, with an average of 27 injections required for full recovery. In Figure 2, the larger the circle, the more experiments utilized the same number of faults and threshold. The figure shows that the same number of faults may result in a varied range of thresholds, as different fault locations can generate different number of linear equations.

Figure 3 shows the distribution of linear differential equations produced by each fault location in ACORNv3. It is observed that some locations, such as 66 to 77, 230, 231 and

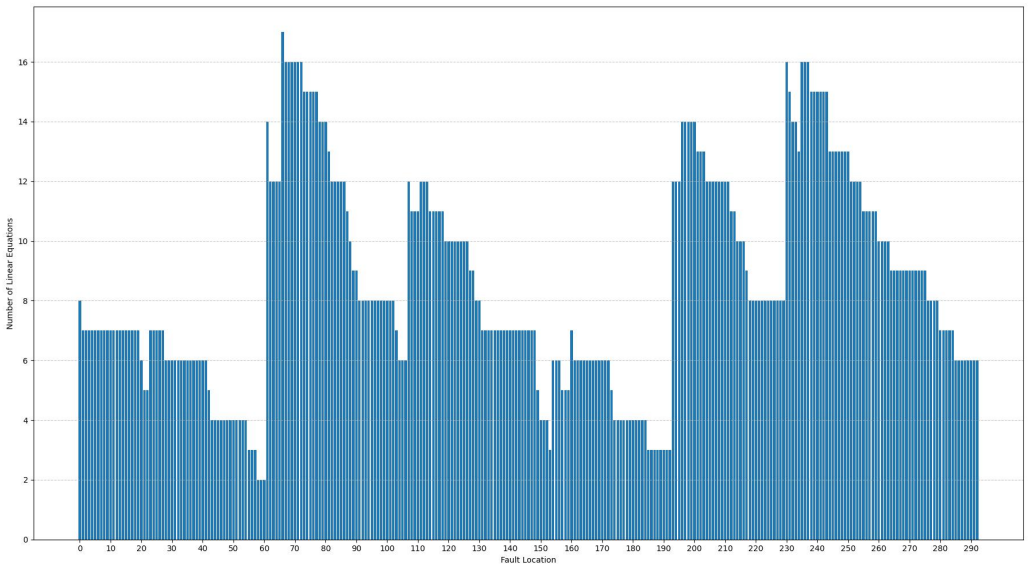


Figure 3: Number of linear equations produced by each fault location in ACORNv3.

235 to 243 generate 15 or above linear equations, which are the ideal fault locations. If the fault is injected at these locations, the number of faults required in the attack can be reduced. However, the required number of fault injections also depends on the algebraic complexity of the resulting equations. If the system is too complex for the Gröbner basis sovler to solve within the set time limit, additional fault injections are required.

For the study conducted by Siddhanti et al. [SSMC17], although only a small number (9) of faults are required to recover the state, the attacker must know the values of 20 state bits, which increases the complexity of the attack by 2^{20}. In the work by Zhang et al. [ZFL18], the authors focus on recovering the initial state of ACORNv3 by solving the system of linear equations. Their approach requires $26 < n < 43$ faults to recover the initial state using a guess-and-determine method, as the number of linear equations obtained is fewer than the number of state variables. Because their attack requires guessing certain state bits, the attack complexity is $c \cdot 2^{146.5-3.52n}$, where c is the time complexity of equation solving and n is the number of faults.

In contrast, we incorporate both linear and quadratic equations for the state recovery process, and achieve a full state recovery without necessitating a guess-and-determine approach. Our proposed method successfully recovers the full 293-bit state of ACORNv3 with 21 to 34 faults. This represents a 19.04% reduction in the maximum number of faults required compared to the results of Zhang et al. [ZFL18]. The time complexity of our attack is negligible and significantly lesser than the two existing works.

4.2 Differential Fault Attack on MORUSv2

For the attack on MORUS, the adversary is assumed to have the knowledge of three plaintext blocks $P = \{P_0, P_1, P_2\}$, i.e., a total of a 384-bit plaintext. During the attack, the adversary obtains the differential keystream and feeds it into the trained model to identify the fault location and generate the corresponding differential equations with respect to the initial state. The same process is repeated until sufficient number of differential equations are collected. The initial state of MORUS is recovered by solving the equations. In the following sections, the initial state is denoted as $S = \{s_0, s_1, \cdots, s_{639}\}$.

4.2.1 Implementation of MLP Model for Fault Location Identification

Using the datasets generated with Algorithm 4, an MLP model for fault location identification for MORUS is implemented. We train the model and tune the hyperparameters with the method defined in Section 3.1.2. The model architecture is described as follow:

- **Input layer:** This input layer consists of 384 neurons.
- **Hidden layer:** A total of three hidden layers are used, each containing 512 neurons. Each is a dense layer followed by batch normalization to mitigate internal covariate shift and increase training stability. For the first two hidden layers, the ReLU activation function and the dropout rate of 0.2 are applied after the batch normalization to reduce overfitting. The final hidden layer utilizes ReLU activation after batch normalization without dropout.
- **Output layer:** This layer consists of 640 neurons (the number of possible fault locations in MORUS). We utilize softmax as the activation function in this layer.

4.2.2 Implementation of Differential Fault Attack on MORUS

The process of differential fault attack on MORUS is described in Algorithm 6. The initial settings of the attack are identical to the attack done on ACORN, except for the setting of the threshold $\mathcal{T}$. The threshold $\mathcal{T}$ is obtained through experimental observation, as in the case of ACORNv3, and set to 896 for MORUS. To apply the attack to MORUS, we include all equations into the equation system, regardless the degree of equations. Prior to the attack, the adversary first runs the cipher with fault-free initial state $S = \{s_0, s_1, \cdots, s_{639}\}$ and obtains the 384-bit fault-free keystream, $Z = \{z_0, z_1, \cdots, z_{383}\}$. Due the constraint of SageMath, we can only generate the ANF for the first 256 iterations of the encryption phase; therefore we construct 256 fault-free keystream equations $NKS = \{nks_0, nks_1, \cdots, nks_{255}\} NKS = \{nks_0, nks_1, \cdots, nks_{255}\}$.

In the online phase of the attack, it follows the same procedure as in the attack on ACORNv3, where the adversary reset the cipher to the initial state, $S' = \{s_0, s_1, \cdots, s_{639}\}$ and injects a bit-flipping fault at a random location f, such that s_f is complemented. The faulty initial state is then used to run the cipher and output the faulty keystream, $Z' = z'_0,\ z'_1,\ \cdots,\ z'_{383}$. The resulting differential keystream, $\Delta Z^f = \Delta z_0^f, \Delta z_1^f, \cdots, \Delta z_{383}^f$, is fed into the MLP model to identify the fault location. Once the fault location is identified, the ANFs of the first 256-bit faulty keystream with respect to the initial state of the cipher are generated. Each retrieved differential equation is added to the equation system $\mathcal{E}$, regardless of its algebraic degree. The process is repeated until the number of equations in the equation system reaches or exceeds the threshold of 896.

Once a sufficient number of differential equations has been collected and stored in $\mathcal{E}$, a Gröbner basis is applied to solve the system and recover the initial state. This procedure follows a similar approach to that used for ACORN. Note that in this attack, we also include the first 256 bits of the original keystream equations into the equation system $\mathcal{E}$ to make it more constrained, due to the larger state size of MORUS.

4.2.3 Results and Discussions of the Attack on MORUSv2

The performance of the trained MLP model in fault location identification is shown in Table 4. The results in Table 4 show that the trained model achieves a high accuracy of 99.9231%, i.e., the model is able to capture most of the pattern of fault propagation. With a precision score of 99.9235%, the model demonstrates a robust ability to correctly classify fault locations while maintaining a very low probability of false positive identifications. The recall score of 99.9231% is slightly lower than the precision score by 0.0004%, indicating that the probability of misidentification is slightly higher than the probability of false

Algorithm 6 Differential Fault Attack on MORUSv2

Initialize list $\mathcal{E}, \mathcal{F}$
$solve \leftarrow$**false**, $\mathcal{T} \leftarrow 896$, $num_eq \leftarrow 0$
$Z = \{z_0, z_1, \cdots, z_{383}\} \leftarrow encMORUS(S, P)$
Generate fault-free ANF, $NKS = \{nks_0, \cdots, nks_{255}\}$
for $i = 0$ **to** 255 **do**
  Append $nks_i \oplus z_i$ to $\mathcal{E}$
end for
while $solve$ **is false do**
  while $num_eq < \mathcal{T}$ **do**
    Reset cipher to initial state S'
    Inject fault at location f, where $s_f = s_f \oplus 1$
    $Z' = \{z'_0, z'_1, \cdots, z'_{383}\} \leftarrow encMORUS(S', P)$
    for $i = 0$ **to** 383 **do**
      $\Delta z_i^f = z_i \oplus z'_i$
    end for
    $f' \leftarrow modelMORUS(\Delta Z^f = \{\Delta z_0^f, \cdots, \Delta z_{383}^f\})$
    if f **not in** $\mathcal{F}$ **and** $f = f'$ **then**
      Append f to $\mathcal{F}$
    else
      continue
    end if
    Generate faulty ANF, $FKS = \{fks_0, \cdots, fks_{255}\}$
    for $i = 0$ **to** 255 **do**
      $dks = nks_i \oplus fks_i \oplus \Delta z_i^f$
      if $dks.deg \geq 1$ **and** dks **not in** $\mathcal{T}$ **then**
        Append dks to $\mathcal{E}$
        $num_eq = num_eq + 1$
      end if
    end for
  end while
  Solve $\mathcal{E}$ with Gröbner basis
  if Gröbner basis can solve in time **then**
    $solve \leftarrow$**true**
  else
    $\mathcal{T} = \mathcal{T} + 5$
  end if
end while

Table 4: Performance of trained model for identifying fault locations on MORUSv2

Accuracy	Precision	Recall	F1-score
0.999231	0.999235	0.999231	0.999231

positive identifications. However, the score of 99.9231% is still considered as a good score, as the model can identify most of the actual fault locations among all actual positives cases. The F1 score of 99.9231% shows a balanced trade-off between the precision and recall, indicating that the trained model has high reliability in identifying the fault locations.

To provide a comprehensive evaluation, we also implemented a traditional signature-based method to benchmark the performance of the trained MLP model. The signature-based method utilized the same training and testing datasets as the MLP model for signature generation and evaluation, respectively. The accuracy of fault location identifica-

tion for the MLP model is 99.9231%, while the signature-based method is 95.2606%. The result shows that the proposed MLP model outperforms the traditional signature-based method in fault location identification of MORUS. The state update function of MORUS utilizes AND, XOR, and rotation operations, which complicates the fault propagation through the state. The MLP model demonstrates a superior ability to learn these complex, non-linear propagation patterns. In contrast, the signature-based method relies on static, predefined signatures, and may struggle with the diffusion properties of the cipher. In certain locations, fault propagation becomes computationally indistinguishable, leading to identification errors that the neural network is better equipped to resolve.

We conducted a total of 400 experiments to analyze the performance of the proposed state recovery attack on MORUSv2. The results show that 76 out of 400 experiments encountered a misidentification of the fault location. While the trained model maintains a high accuracy in fault location identification, the large state size of MORUS requires a larger number of faults (213 to 248 faults) to recover the initial state. The increase in number of faults has degraded the performance of model, as the higher number of fault injections has also increased the probability of misidentification cases. Note that in these 76 misidentification cases, the model still correctly identified the majority of fault locations within each experiment. However, the experimental setup was designed to terminate prior to the equation-solving phase upon the first occurrence of a misidentification. This conservative approach ensures that the reported success rates reflect a "zero-tolerance" policy for identification errors, even though many of these "failed" experiments contained only a single erroneous location out of more than 200 injections. Because the number of correct identifications outweighs the erroneous ones, the majority of the derived equations remain valid. Consequently, it is expected that solving the system of equations with misidentification cases can recover the majority of the state bits.

In the remaining 324 cases, the results show that the full initial state of MORUSv2 can be successfully recovered. Among these 324 successful attacks, 86 cases directly recovered the state by solving equations, while 238 required the adversary to guess a few state bits in order to achieve full state recovery. For these 238 cases, the recovered state bits can be further categorized into three groups: directly recovered bits, indirectly recovered bits and guessing bits. The directly recovered bits refer to the state bits that can be directly recovered by solving the equations. The guessing bits are the state bits that cannot be recovered by solving the equations, and the adversary is required to guess the value to recover the state. The indirectly recovered bits are the state bits that can be recovered from the equations that involved one or more guessing bits. To recover the indirectly recovered bits correctly, the adversary needs to guess the value of the guessing bits correctly. In these attacks, the number of directly recovered bits ranged from 600 to 639, while the number of indirectly recovered bits and guessing bits ranged from 0 to 35 and 1 to 6, respectively. Considering the highest number of guessing bits of 6, the adversary only needs to try for 2^6 possible combinations at most. With the correct guess, the indirectly recovered bits are resolved, leading to a full recovery of the initial state. The average number of faults required to recover the initial state is 231. Figure 4 illustrates the distribution of the number of faults required for a successful state recovery of MORUS across the conducted experiments.

The attack proposed by Wong et al. [WBSD20] can recover only two register's contents under the assumption of multi-byte random fault with precise control over fault location. In contrast, our proposed attack utilizes a more relaxed assumption in terms of the fault precision (no control over the target register). Consequently, our proposed attack require a weaker fault model while achieving full initial state recovery of MORUSv2. Although Wong et al. [WBSD20] performed a theoretical analysis, they did not implement the attack on the cipher. Hence, the proposed method is considered as the first implementation of differential fault attack against MORUS. The total complexity of the attack is bounded by

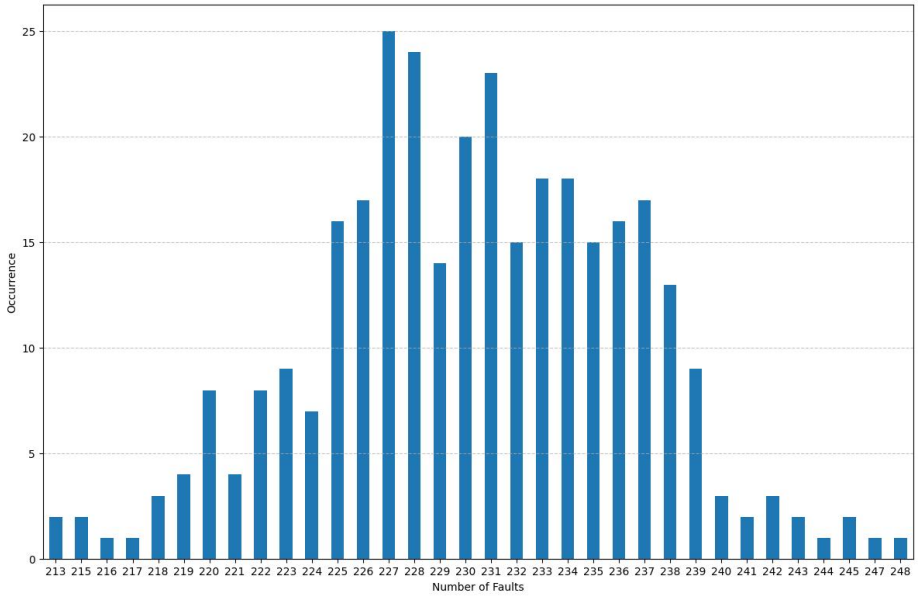


Figure 4: Frequency of the number of faults for each successful attack.

the Gröbner basis computation for solving the equation system and guessing the values of h guessing bits, where $h \in \{1, 2, 3, 4, 5, 6\}$. Since the time complexity of Gröbner basis solver to resolve this specific system is negligible, the overall complexity of the attack is at most 2^h.

4.3 Differential Fault Attack on ATOM

In the attack on ATOM, we assumed that the adversary knows 56 bits of plaintext $P = \{p_0, p_1, \cdots, p_{55}\}$. Due to the design of the cipher, the attack assumption made for this attack is more stringent than those in previous sections. Since the *cnt* in the keystream generation phase for the key filter depends on the state of LFSR, the adversary needs to possess the information of the LFSR state to determine which key bits are involved in the operation. Hence, it is assumed that the adversary knows the first 60 bits of the LFSR. The adversary does not need to guess the value of the last 9 bits of the LFSR, as they are initialized to a constant value of 1 at the beginning of encryption phase. The fault is assumed to be injected at the initial state of NFSR.

4.3.1 Implementation of MLP Model for Fault Location Identification

The implementation of the MLP model for ATOM is focused on identifying the fault location in the NFSR. Thus, the generated datasets only include samples of faults injected in the NFSR. The training and hyperparameters tuning follow the same method as described earlier. The architecture of the developed MLP model is described as follows:

- **Input layer:** The number of neurons in the input layer is 56.
- **Hidden layer:** Three hidden layers, with 128, 256, 128 neurons respectively, are used. For the first two hidden layers, the ELU activation function and the dropout rate of 0.2 are applied after the batch normalization to reduce overfitting. The last hidden layer utilizes the ELU activation after batch normalization.
- **Output layer:** 90 neurons are used in this layer (number of possible fault locations in the NFSR of ATOM), using softmax as the activation function.

4.3.2 Implementation of Differential Fault Attack on ATOM

We describe the implementation of DFA on ATOM in Algorithm 7. In this attack, the threshold $\mathcal{T}$ is set to 234. This value accounts for the 90 differential equations required to determine the 90-bit NFSR state, 128 differential equations to determine the 128-bit secret key, and an additional 16 original fault-free equations included to further constrain the system. Note that only a partial recovery of the secret key may be possible, as symbolic generation in SageMath was limited to the first 16 keystream equations. These constraints are insufficient to uniquely resolve all 128 secret key bits.

In our experiments, we first generate and store the fault-free keystream equations into $\mathcal{E}$ by XOR-ing nks_i with the corresponding keystream bit z_i, where $i = \{0, 1, \cdots, 15\}$. Next, a bit-flipping fault is injected at a random location f in the initial state of NFSR B, such

Algorithm 7 Differential Fault Attack on ATOM

Initialize list $\mathcal{E}, \mathcal{F}$
$solve \leftarrow$**false**, $\mathcal{T} \leftarrow 234$, $num_eq \leftarrow 0$
$Z = \{z_0, z_1, \cdots, z_{55}\} \leftarrow encATOM(S, P)$
Generate fault-free ANF, $NKS = \{nks_0, \cdots, nks_{15}\}$
for $i = 0$ **to** 15 **do**
  Add $nks_i \oplus z_i$ to $\mathcal{E}$
  $num_eq = num_eq + 1$
end for
while *solve* **is false do**
  while $num_eq < \mathcal{T}$ **do**
    Reset cipher to initial state B', L
    Inject fault at location f, where $b_f = b_f \oplus 1$
    $Z' = \{z'_0, z'_1, \cdots, z'_{55}\} \leftarrow encATOM(B', L)$
    for $i = 0$ **to** 55 **do**
      $\Delta z_i^f = z_i \oplus z'_i$
    end for
    $f' \leftarrow modelATOM(\Delta Z^f = \{\Delta z_0^f, \cdots, \Delta z_{55}^f\})$
    if f **not in** $\mathcal{F}$ **and** $f = f'$ **then**
      Append f to $\mathcal{F}$
    else
      continue
    end if
    Generate faulty ANF, $FKS = \{fks_0, \cdots, fks_{15}\}$
    for $i = 0$ **to** 15 **do**
      $dks = nks_i \oplus fks_i \oplus \Delta z_i^f$
      if $dks.deg \geq 1$ **and** dks **not in** $\mathcal{T}$ **then**
        Append dks to $\mathcal{E}$
        $num_eq = num_eq + 1$
      end if
    end for
  end while
  Solve $\mathcal{E}$ with Gröbner basis
  if Gröbner basis can solve in time **then**
    solve $\leftarrow$**true**
  else
    $\mathcal{T} = \mathcal{T} + 5$
  end if
end while

that b_f is complemented. Once the differential keystream, $\Delta Z^f = \Delta z_0^f, \Delta z_1^f, \cdots, \Delta z_{55}^f$ is obtained, it is fed into the MLP model to identify the fault location. If the fault location is correctly identified, the differential equations dks are retrieved by $nks_i \oplus fks_i \oplus \Delta z_i^f$, where $i = \{0, 1, \cdots, 15\}$ and added to $\mathcal{E}$, regardless of their algebraic degree. The same process of generating and collecting differential equations is repeated until the number of equations in the equation system reaches or exceeds the threshold $\mathcal{T}$. The system of equations is solved using Gröbner basis when the number of equations stored in $\mathcal{E}$ exceeds the set threshold. The solving phase follows the same methodology used for ACORNv3 and MORUSv2. In instances where the solver does not return a solution within the specified time limit, the threshold is increased by 5 to include more equations in the equation system.

4.3.3 Result and Discussion of the Attack on ATOM

Table 5 shows the performance of the MLP model for ATOM. It shows that the performance of the trained model achieves an accuracy of 82.3568%. Compared to the trained models for ACORNv3 and MORUSv2, this model has a lower accuracy on ATOM. Despite achieving an accuracy exceeding 80%, the model remained susceptible to misidentification errors when applied to ATOM. This is primarily due to the cipher's design structure. ATOM utilizes a double key filter where the setting of *cnt* in the encryption phase depends on the current LFSR state. This dependency introduces a high degree of complexity in the fault propagation, making the resulting patterns less deterministic and harder for the MLP to learn. The model achieves a precision score of 82.1186%, indicating that there is a moderate chance that it will encounter false positive identifications. The recall score of 82.3568% is slightly higher than the precision score, showing that the probability of misidentification is slightly lower than the probability of false positive identifications. Finally, the F1 score of 81.6163% shows a balanced trade-off between the precision and recall.

Table 5: Performance of trained model for identifying fault locations on ATOM

Accuracy	Precision	Recall	F1-score
0.823568	0.821186	0.823568	0.816163

On the other hand, the traditional signature-based method is also implemented to benchmark the performance of the trained model. The same training and testing datasets are used to generate the fault signature and evaluate the signature-based method. The accuracy of the MLP model is 82.3568%, while the accuracy for the signature-based method is 58.8976%. The results show that the traditional signature-based method has a poor performance in fault location identification for ATOM. Overall, the MLP model demonstrates a significant performance improvement over the signature-based method in learning complex fault propagation patterns, exceeding its accuracy by a margin of approximately 23%. Although the performance of this model is not as significant as the previously trained models for ACORN and MORUS, it nonetheless demonstrates that deep learning model possesses a superior capability compared to traditional signature-based methods to learn the highly non-deterministic fault propagation patterns.

A total of 400 experiments were conducted to evaluate the performance of the proposed state recovery and key recovery attack on ATOM. Under the strict parameters of this study, the attack only proceeds to the equation-solving phase if all fault locations are identified with 100% accuracy. However, in all 400 trials, at least one misidentification occurred during the collection process. These results show that the utilization of a double key filter provides a relatively stronger security margin against the attacks presented in this paper, as it may prevent the adversary from constructing a system of equations where every equation is valid. If the adversary utilizes the equation system with the misidentification cases and attempts to solve the system, it is expected that some of the state bits or the

key bits will not be recovered correctly. In our experiments, a new fault is injected into the NFSR whenever the number of equations is below the threshold of 234. To reach this threshold, the attack requires an average of 46 faults.

Since the attack under the random fault location model (where the adversary has no control over fault locations) encountered misidentification errors in all 400 trials, a subsequent experiment was conducted assuming a precise fault model. In this experiment, the utilization of threshold is excluded. Based on the previous experiments, 46 faults are needed on average to construct the equation system with at least 234 equations. Suppose that the adversary repeatedly reset the cipher and injects each fault in $\mathcal{F}$, where $\mathcal{F}$ consists of 46 faults that are defined as follows:

$$\begin{aligned}\mathcal{F} = \{&b_3, b_7, b_9, b_{10}, b_{11}, b_{16}, b_{17}, b_{20}, b_{21}, b_{22}, b_{24}, b_{25}, b_{30}, b_{31}, b_{34}, b_{35}, b_{36}, b_{37}, b_{40}, b_{41},\\ &b_{42}, b_{44}, b_{46}, b_{48}, b_{49}, b_{52}, b_{53}, b_{54}, b_{56}, b_{58}, b_{62}, b_{63}, b_{65}, b_{67}, b_{69}, b_{72}, b_{73}, b_{75}, b_{76},\\ &b_{79}, b_{82}, b_{83}, b_{84}, b_{85}, b_{86}, b_{89}\}.\end{aligned} \tag{10}$$

The fault locations in $\mathcal{F}$ are chosen randomly, and subsequently the corresponding differential equations were generated, and included in the equation system together with the 16 fault-free equations. Because the secret key bits remain as unresolved variables within the differential equations, a brute-force guessing strategy was employed to facilitate the solving process. By iterating through the possible values of the involved key bits, the system becomes sufficiently constrained to allow the Gröbner basis solver to operate. Under these conditions, 74 out of the 90 NFSR initial state bits were successfully recovered. However, some state bits were recovered with incorrect values; this is likely due to the limited keystream length, which leads to an underdetermined system where multiple mathematical solutions satisfy the given equations

5 Conclusion

This work investigates the feasibility of deep learning-assisted differential fault attacks (DFA) against ACORNv3, MORUSv2, and ATOM, utilizing a single-bit bit-flipping fault with no control over the fault location. We implemented three Multilayer Perceptron (MLP) models to identify fault locations, each achieving higher accuracy than traditional signature-based methods. Additionally, we proposed a threshold-based strategy to optimize the number of fault injections. Our results demonstrate that the initial state of ACORN can be recovered with negligible complexity, and full state recovery can be achieved for MORUS—both results outperforming the existing literature. Furthermore, we implemented the first DFA on ATOM, recovering the majority of the NFSR state bits under a precise fault-location model. These findings suggest that the double key filter design increases the security margin against the described fault attacks. Future work may examine the feasibility of deep learning-assisted fault attacks under a more relaxed random fault model, where the attacker has no control over the fault location or impact.

References

[BCI+21] Subhadeep Banik, Andrea Caforio, Takanori Isobe, Fukang Liu, Willi Meier, Kosei Sakamoto, and Santanu Sarkar. Atom: A stream cipher with double key filter. *IACR Transactions on Symmetric Cryptology*, 2021(1):5–36, Mar 2021. doi: 10.46586/tosc.v2021.i1.5-36.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233, pages 37–51, Berlin, Heidelberg, 1997. Springer. doi: 10.1007/3-540-69053-0_4.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294, pages 513–525, Berlin, Heidelberg, 1997. Springer. doi: 10.1007/BFb0052259.

[BSS+21] Anubhab Baksi, Santanu Sarkar, Akhilesh Siddhanti, Ravi Anand, and Anupam Chattopadhyay. Differential fault location identification by machine learning. *CAAI Transactions on Intelligence Technology*, 6(1):17–24, Mar 2021. doi: 10.1049/cit2.12027.

[COZ+23] Yukun Cheng, Changhai Ou, Fan Zhang, Shihui Zheng, Shengmin Xu, and Jiangshan Long. DLFA: Deep learning based fault analysis against block ciphers. Cryptology ePrint Archive, Paper 2023/021, 2023. URL: https://eprint.iacr.org/2023/021.

[DR17] Deepak Kumar Dalai and Dibyendu Roy. A state recovery attack on ACORN-v1 and ACORN-v2. In Zheng Yan, Refik Molva, Wojciech Mazurczyk, and Raimo Kantola, editors, *Network and System Security*, pages 332–345, Cham, 2017. Springer. doi: 10.1007/978-3-319-64701-2_24.

[Goh19] Aron Gohr. Improving attacks on round-reduced Speck32/64 using deep learning. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology — CRYPTO 2019*, volume 11693, pages 150–179, Cham, 2019. Springer. doi: 10.1007/978-3-030-26951-7_6.

[HRC21] ZeZhou Hou, JiongJiong Ren, and ShaoZhen Chen. Improve neural distinguishers of SIMON and SPECK. *Security and Communication Networks*, 2021(1), Dec 2021. doi: 10.1155/2021/9288229.

[HS04] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156, pages 240–253, Berlin, Heidelberg, 2004. Springer. doi: 10.1007/978-3-540-28632-5_18.

[KLK+23] Hyunji Kim, Sejin Lim, Yeajun Kang, Wonwoong Kim, Dukyoung Kim, Seyoung Yoon, and Hwajeong Seo. Deep-learning-based cryptanalysis of lightweight block ciphers revisited. *Entropy*, 25(7), Jun 2023. doi: 10.3390/e25070986.

[MDR+24] Sandip Kumar Mondal, Prakash Dey, Himadry Sekhar Roy, Avishek Adhikari, and Subhamoy Maitra. Improved fault analysis on Subterranean 2.0. *IEEE Transactions on Computers*, 73(6):1631–1639, Jun 2024. doi: 10.1109/TC.2024.3371784.

[MSBD15] Subhamoy Maitra, Santanu Sarkar, Anubhab Baksi, and Pramit Dey. Key recovery from state information of Sprout: application to cryptanalysis and fault attack. Cryptology ePrint Archive, Paper 2015/236, 2015. URL: https://eprint.iacr.org/2015/236.

[MSS17] Subhamoy Maitra, Akhilesh Siddhanti, and Santanu Sarkar. A differential fault attack on Plantlet. *IEEE Transactions on Computers*, 66(10):1804–1808, Oct 2017. doi: 10.1109/TC.2017.2700469.

[MTQ21] Zhen Ma, Tian Tian, and Wenfeng Qi. Differential fault attack on the stream cipher LIZARD. *Chinese Journal of Electronics*, 30(3):534–541, 2021. doi: 10.1049/cje.2021.04.007.

[ORSP20] Mohammad Ali Orumiehchiha, Saeed Rostami, Elham Shakour, and Josef Pieprzyk. A differential fault attack on the WG family of stream ciphers. *Journal of Cryptographic Engineering*, 10(2):189–195, Jun 2020. doi: 10.1007/s13389-020-00222-x.

[PS25] Samuel Russell Prajasantosa and Iftekhar Salam. Differential fault analysis of TinyJAMBU. In Colin Boyd, Reihaneh Safavi-Naini, and Leonie Simpson, editors, *Information Security in a Connected World: Celebrating the Life and Work of Ed Dawson*, pages 68–88, Cham, 2025. Springer. doi: 10.1007/978-3-031-83490-5_4.

[ROSA25] Saeed Rostami, Mohammad Ali Orumiehchiha, Elham Shakour, and Sadegh Alizadeh. Fault attack on enocoro stream cipher family. *Journal of Cryptographic Engineering*, 15, Jan 2025. doi: 10.1007/s13389-024-00367-z.

[RR25] R Radheshwar and Dibyendu Roy. Differential fault attack on ChaosForge. In *Proc. 6th International Conference on Recent Advances in Information Technology (RAIT)*, pages 1–6. IEEE, 2025. doi: 10.1109/RAIT65068.2025.11089261.

[SAB+23] Sayandeep Saha, Manaar Alam, Arnab Bag, Debdeep Mukhopadhyay, and Pallab Dasgupta. Learn from your faults: Leakage assessment in fault attacks using deep learning. *Journal of Cryptology*, 36(3), May 2023. doi: 10.1007/s00145-023-09462-6.

[SOX+21] Iftekhar Salam, Thian Hooi Ooi, Luxin Xue, Wei-Chuen Yau, Josef Pieprzyk, and Raphaël C.-W. Phan. Random differential fault attacks on the lightweight authenticated encryption stream cipher Grain-128AEAD. *IEEE Access*, 9:72568–72586, 2021. doi: 10.1109/ACCESS.2021.3078845.

[SSMC17] Akhilesh Siddhanti, Santanu Sarkar, Subhamoy Maitra, and Anupam Chattopadhyay. Differential fault attack on Grainv1, ACORNv3 and Lizard. In Sk Subidh Ali, Jean-Luc Danger, and Thomas Eisenbarth, editors, *Security, Privacy, and Applied Cryptography Engineering – SPACE 2017*, pages 247–263, Cham, 2017. Springer. doi: 10.1007/978-3-319-71501-8_14.

[STB25] Cláudio Silva, Nelson Tenório, and Jorge Bernardino. Lightweight encryption algorithms for IoT. *Computers*, 14(12), Nov 2025. doi: 10.3390/computers14120505.

[SYPP23] Iftekhar Salam, Wei-Chuen Yau, Raphaël C.-W. Phan, and Josef Pieprzyk. Differential fault attacks on the lightweight authenticated encryption algorithm CLX-128. *Journal of Cryptographic Engineering*, 13:265–281, Sep 2023. doi: 10.1007/s13389-023-00326-0.

[WBSD20] Kenneth Koon-Ho Wong, Harry Bartlett, Leonie Simpson, and Ed Dawson. Differential random fault attacks on certain CAESAR stream ciphers (supplementary material). Cryptology ePrint Archive, Paper 2020/022, 2020. URL: https://eprint.iacr.org/2020/022.

[WH16] Hongjun Wu and Tao Huang. The authenticated cipher MORUS (v2). Submission to CAESAR Competition, 2016. URL: https://competitions.cr.yp.to/round3/morusv2.pdf.

[Wu16] Hongjun Wu. Acorn: A lightweight authenticated cipher (v3). Submission to CAESAR Competition, 2016. URL: https://competitions.cr.yp.to/round3/acornv3.pdf.

[ZFL17] Xiaojuan Zhang, Xiutao Feng, and Dongdai Lin. Fault attack on the authenticated cipher ACORNv2. *Security and Communication Networks*, 2017(1), Oct 2017. doi: 10.1155/2017/3834685.

[ZFL18] Xiaojuan Zhang, Xiutao Feng, and Dongdai Lin. Fault attack on ACORN v3. *The Computer Journal*, 61(8):1166–1179, Aug 2018. doi: 10.1093/comjnl/bxy044.

[ZL22] Behnam Zahednejad and Lijun Lyu. An improved integral distinguisher scheme based on neural networks. *International Journal of Intelligent Systems*, 37(10):7584–7613, May 2022. doi: 10.1002/int.22895.