Differential Fault Attack on Atom: Bypassing the Double Key Filter using Filtered Faults

Vaibhav Dixit, Satyam Kumar, and Santanu Sarkar

***Abstract*—In this paper, we present a Differential Fault Attack (DFA) on the lightweight stream cipher Atom, proposed by Banik et al. in IACR Transactions on Symmetric Cryptography (TOSC)-2021. It employs two key filters simultaneously during the pseudo-random generation algorithm phase, one of which depends on LFSR state bits. Due to this LFSR-dependent key filter, the authors claim that forming algebraic equations relating key and state bits as variables to the keystream bits is difficult unless the entire LFSR state is known. In contrast, we propose a method to formulate such algebraic equations without guessing any LFSR bits. This enables us to implement a successful DFA on Atom. To the best of our knowledge, this is the first reported DFA on Atom. In the proposed DFA, we identify the location of injected faults using a weighted ensemble of trained MLP and XGBoost models. To further improve accuracy, we filter out ML predictions with confidence below a predefined threshold. We found that this strategy significantly reduces the number of SAT solver invocations and improves the overall time complexity of the attack. Based on our experiments, we demonstrate a successful DFA on Atom within a practical time by injecting 18 faults, provided all are correctly identified. Obtaining a set of 18 correctly identified faults requires, on average, 52 fault injections. The attack requires a total of 70 keystream bits (normal and faulty combined) just after a fault injection and guessing two random key bits.**

***Index Terms*—DFA, Stream cipher, Atom, SAT solver, Kissat Solver, Fault Identification, Machine learning, ANN, MLP, XGBoost**

I. Introduction

The rapid advancement of pervasive computing and the Internet of Things (IoT) has intensified the demand for cryptographic solutions that are both secure and resource-efficient, with minimal power consumption and hardware overhead. In such environments, stream ciphers are often the preferred choice for encryption due to their ability to generate a keystream bit at each round, achieving both speed and simplicity in constrained hardware settings. Stream ciphers are symmetric key ciphers in which both the sender and the receiver use the same key for encryption and decryption of messages. The security of stream ciphers is evaluated using both classical and hardware-based attack models. In classical cryptanalytic techniques, such as algebraic cryptanalysis, correlation attack, etc., the attacker uses the algorithmic structure of encryption functions to extract information about the internal state or the secret key. On the other hand, in hardware-based attack models, the attacker actively or passively interacts with the physical device and analyzes its output. Differential Fault Attack (DFA) is one of them, i.e., in DFA, the attacker actively interferes with the operation of the device by injecting faults and analyzes the resulting differences in output. Typically, faults are induced using techniques like laser shots or clock glitches [1].

Fault analysis was initially introduced by Boneh et al. to break number theoretic public key cryptosystems [2] and later extended to stream ciphers by Hoch and Shamir [3]. In recent times, many small state stream ciphers have been cryptanalyzed using DFA [4]–[8]. In this article, we will analyze the vulnerability of a lightweight stream cipher, Atom, against DFA.

Atom is a stream cipher proposed by Banik et al. [9] in FSE 2022 (published in IACR Transactions on Symmetric Cryptography, (TOSC)-2021). According to the authors, it is one of the smallest stream ciphers with a 159-bit state size that achieves a security level of 128 bits. It is built on the basic structure of the Grain-like stream cipher [10], [11] while incorporating design modifications to resist known attacks on small state stream ciphers. For stream ciphers, like Grain v1 [10] or Lizard [11], recovery of the internal state directly leads to the recovery of the secret key, since the state update functions are reversible. In contrast, Atom avoids this by incorporating secret key bits in the state update function. A similar design principle is adopted in Kreyvium [12] and Plantlet [13]. Atom distinguishes itself by using two secret key filters in the state update function: one based on the round and the index of other key is determined by the value of the last 7 bits of the LFSR. This design choice, i.e., making the key bit index LFSR-dependent, was explicitly intended to make equation-based cryptanalysis difficult. In their security discussion, the authors have remarked that

> *If the LFSR is variable or unknown, the attacker would find it difficult to enumerate any equation bank, as he wouldn't know which key bit was used in the state update.*

In this article, we demonstrate that a system of equations can be formulated without guessing any LFSR state bits. We present an algorithm to form a system of equations involving secret key and state variables, thereby enabling practical key recovery via DFA.

In recent work, Anand et al. [14] demonstrate that the Atom achieves quantum security equivalent to AES-128, meeting the NIST recommendation for 128-bit key ciphers, i.e., the cost of implementing Grover's attack is $\approx 2^{170}$. More recently, the work in [15] proposed a cube attack on a weakened variant of Atom. However, this weakened variant is obtained by reducing

Vaibhav Dixit, Satyam Kumar, and Santanu Sarkar are with the Department of Mathematics, Indian Institute of Technology Madras, Chennai 600036, India.
E-mail: vaibhavrdrk@gmail.com, mishrasatyam447@gmail.com, sarkar.santanu.bir1@gmail.com.

the number of initialization rounds and removing one key filter (which depends on the LFSR bits) through guessing the entire LFSR state. They concluded that these attacks do not extend to the original Atom cipher with both key filters present. To the best of our knowledge, no further analysis has been reported on Atom. This paper presents the first successful DFA on Atom.

A. DFA Model and Assumptions

In our DFA model, we make the following assumptions:

- The attacker can flip precisely any one random bit of LFSR at a time. However, the location of the flipped bit is unknown to the attacker.
- The attacker can restart the cipher multiple times using the same key-IV pair.
- The attacker can precisely control the timing of the fault injection, i.e., the injection round is known to the attacker.

Note that the same DFA assumptions have been used in [4], [16]. Moreover, the last two assumptions have been widely adopted in several recent works [5], [17], [18].

B. Our Contributions

Our main contribution in this paper is to demonstrate the first practical DFA on the stream cipher Atom. As per our literature survey, no prior DFA on Atom has been reported in the literature. However, the success of our DFA relies on the following key contributions:

- As per our attack model, the fault location is unknown to the attacker. Therefore, it is crucial to identify the fault location precisely to implement DFA. We employ two machine learning models: Multi-Layer Perceptron (MLP) and the Extreme Gradient Boosting (XGBoost) for fault identification (presented in Section III-B). We observe that the XGBoost achieves slightly higher accuracy than the MLP and a weighted ensemble of both models outperforms both models individually.
- Existing DFA works rely on either correlation-based or ML-based techniques for fault identification, with ML-based approaches demonstrating better performance than correlation-based techniques [8], [18]. In these approaches, the ML prediction with the highest confidence is taken as the output. Building upon this, we introduce a threshold-based filter on ML predictions, referred to as a Confidence-Threshold Strategy (CTS) (see Section III-B1). Under this strategy, only fault predictions with confidence exceeding a predefined threshold are accepted for the key recovery procedure. By discarding low-confidence and potentially incorrect predictions, the number of SAT solver invocations is significantly reduced, thereby improving the overall attack time complexity.
- Atom was designed with an explicit claim that forming algebraic equations involving the secret key and the internal state as variables, and relating them to the keystream bits, is difficult. More precisely, this difficulty arises due to the second key filter, k_{cnt}. In this work, we show that k_{cnt} can be expressed as a non-linear Boolean equation in terms of key and LFSR state bits (see Algorithm 1). As a result, we can construct a system of equations suitable for the SAT solver.

TABLE I
A SUMMARY FOR OUR PROPOSED DFA

Req. no. of correct faults	Avg. time for 1 SAT solver invocation* (Hrs:Mins)	# SAT solver invocations	Expected no. of fault injections
18	40 : 25	8	52
20	23 : 38	8	58
25	12 : 01	8	76

*All SAT experiments are performed using 70 keystream bits with two randomly guessed key variables.

Let us briefly explain our DFA framework. In the offline phase, the ensemble model is trained for fault location prediction. In the online phase, fault locations corresponding to injected faults are predicted using the ensemble model under the CTS. These predicted fault locations are then used for key recovery via the SAT solver.

Our experimental results are summarized in Table I. A *fault injection* refers to a single injection attempt, whereas a *correct fault* denotes a fault location that is correctly identified. The column *Expected no. of fault injections* denotes the average number of fault injections required to obtain a set containing *required number of correct faults*. Since the correctness of such a set can only be validated through SAT solving, multiple sets are required. Experimentally, we observe that, on average, two such sets are sufficient for key recovery. For SAT solving, we use the Kissat solver [19]. On average, only 8 SAT solver invocations are required, making the attack practically feasible; with parallel processing, the total runtime can be reduced to approximately 1-2 days.

Outline: The paper is organized as follows. Section II provides the design of Atom, the background of machine learning algorithms and the notations and definitions used in this paper. Section III presents the proposed DFA, including our fault identification model, key recovery steps, and the experimental results. Finally, Section IV concludes the paper and discusses possible directions for future research.

II. PRELIMINARIES

A. Description of Atom

In this section, we briefly describe the design specification of Atom [9]. It comprises two registers, i.e., an NFSR of 90 bits and an LFSR of 69 bits. For a given round t, let us denote the NFSR state by $N^t = (n_0^t, n_1^t, \ldots, n_{89}^t)$ and the LFSR state by $L^t = (l_0^t, l_1^t, \ldots, l_{68}^t)$. The size of the secret key of Atom is 128, and let us denote it by $k = (k_0, k_1, \ldots, k_{127})$. Since our attack does not require an initialization phase, we do not intend to introduce it here; instead, we restate the PRGA in brief.

The output keystream bit z_t is given as

$$\begin{aligned} z_t = {} & n_1^t \oplus n_5^t \oplus n_{11}^t \oplus n_{22}^t \oplus n_{36}^t \oplus n_{53}^t \oplus n_{72}^t \oplus n_{80}^t \oplus n_{84}^t \\ & \oplus l_5^t l_{16}^t \oplus l_{13}^t l_{15}^t \oplus l_{30}^t l_{42}^t \oplus l_{22}^t l_{67}^t \\ & \oplus h(l_7^t, l_{33}^t, l_{38}^t, l_{50}^t, l_{59}^t, l_{62}^t, n_{85}^t, n_{41}^t, n_9^t), \end{aligned}$$

where $h(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ is a 5-degree polynomial. For a detailed definition of h, one can refer to [9].

After each key generation, LFSR and NFSR states are updated as follows:

$$\begin{aligned} cnt &= l_{62}^t||l_{63}^t||l_{64}^t||l_{65}^t||l_{66}^t||l_{67}^t||l_{68}^t, \\ n_{89}^{t+1} &= G(N^t) \oplus l_0^t \oplus k_{cnt} \oplus k_{t\%128}, \\ n_i^{t+1} &= n_{i+1}^t \ (0 \le i \le 88), \\ l_{68}^{t+1} &= F(L^t), \\ l_i^{t+1} &= l_{i+1}^t \ (0 \le i \le 67). \end{aligned}$$

where the update functions $G(N^t)$ and $F(L^t)$, used to update the last bit of NFSR and LFSR, respectively, are given as

$$\begin{aligned} &G(N^t) = n_0^t \oplus n_{24}^t \oplus n_{49}^t \oplus n_{79}^t \oplus n_{84}^t \oplus n_3^t n_{59}^t \oplus n_{10}^t n_{12}^t \\ &\oplus n_{15}^t n_{16}^t \oplus n_{25}^t n_{53}^t \oplus n_{35}^t n_{42}^t \oplus n_{55}^t n_{58}^t \oplus n_{60}^t n_{74}^t \oplus \\ &n_{20}^t n_{22}^t n_{23}^t \oplus n_{62}^t n_{68}^t n_{72}^t \oplus n_{77}^t n_{80}^t n_{81}^t n_{83}^t, \\ &F(L^t) = l_0^t \oplus l_5^t \oplus l_{12}^t \oplus l_{22}^t \oplus l_{28}^t \oplus l_{37}^t \oplus l_{45}^t \oplus l_{58}^t. \end{aligned}$$

Note that the terms k_{cnt} and $k_{t\%128}$ serve as a double key filter in Atom during the PRGA phase.

B. Machine Learning

Machine Learning (ML) refers to computational methods that enable systems to learn patterns from data and make predictions without being explicitly programmed. We employ ML techniques for fault location prediction. The fault prediction problem can be seen as a supervised multi-class classification problem, where each input sample (i.e., KSD) is associated with an output label (i.e., fault location). As an initial step, the given dataset is divided into three subsets: training, validation, and testing. The training set is used to learn the model parameters[1], the validation set is used to monitor the performance of the model during training, and the testing set is used to evaluate the final performance of the trained model. Once data preparation is complete, the processed data are then provided to the model for training and testing. Throughout this paper, *KSD* and *input feature vector* are used interchangeably to denote the input to the learning models. The predicted output is primarily referred to as the *predicted fault location* and is also termed the *output label* in an ML context.

In this work, we consider both a neural network-based learning model and an ensemble-based boosting model for fault location prediction. More specifically, a multi-layer perceptron (MLP) and eXtreme Gradient Boosting (XGBoost) are employed. The architectural details and training procedures of both models are discussed in the following subsections.

1) Multi-Layer Perceptron (MLP): A Multi-Layer Perceptron [20] is a type of Artificial Neural Network architecture (ANN). An ANN consists of multiple computational units, called neurons, interconnected through adaptive parameters known as weights. The network is organized into multiple layers, where the first and last layers are referred to as the input and output layers, respectively, and the intermediate layers are termed as *hidden layers*. These hidden layers enable the model to learn abstract features and internal representations of the input features. The number of neurons present in a layer defines its *size*. In MLP, the neural network is composed of fully connected layers, i.e., each neuron in a layer is connected to all neurons in the preceding layer.

The input layer receives the input feature vector, whose dimensionality determines the layer size, and propagates it through the hidden layers. At each layer, each neuron computes a weighted sum of its inputs with a bias and passes the result through a non-linear *activation function* such as Swish, ReLU, etc, to produce its output. The resulting outputs of one layer then serve as inputs to each neuron in the subsequent layer. The *loss function* is used to measure the difference between the predicted output and the true output. Based upon those values, *optimizer* updates the values of weights and biases for generalization of the model. The output layer produces a probability distribution over the output labels using a suitable activation function such as Softmax.

The MLP is trained using a standard supervised learning procedure; for a detailed description of the learning algorithm, one can refer to the article [20]. Model training is controlled using commonly used hyperparameters such as learning rate, batch size, number of epochs, and dropout rate. These implementation details follow conventional practices in modern neural network training.

2) Extreme Gradient Boosting Method (XGBoost): The eXtreme Gradient Boosting Algorithm [21] is a high performance implementation of a gradient-boosted decision tree. It constructs an ensemble of decision trees through a sequential boosting process, where each newly added decision tree (often referred to as a weak learner) is trained to correct the prediction errors made by the previous learners. Each decision tree may be trained on a (possibly) subsampled set of training samples (controlled by *subsample ratio*) and a subsampled set of input features (controlled by *column sampling ratio*) to introduce randomness and prevent overfitting. During the testing phase, a set of new unlabeled data samples is passed through all of the weak learners and the final prediction is computed by aggregating the outputs of each tree scaled by the *learning rate*. In this work, we use the multi-class soft probability objective (*multi:softprob*), which outputs a probability distribution over all possible fault locations. Training details and hyperparameter settings, including maximum tree depth, minimum child weight, Gamma, L1 & L2 regularization parameters, number of boosting rounds, etc., follow standard XGBoost practices [21].

C. Notations and Definitions

In this section, we introduce some notations and definitions used throughout the paper.

- As per our attack assumption, let us assume the fault is injected at a known round τ.
- *Single-bit fault injection* means only one bit of internal state is flipped during a fault injection.
- The terms z_i and z_i^f denote the normal and faulty keystream bits, respectively, obtained at the i-th round, where z_i^f corresponds to a fault injection at the f-th location of the internal state.

[1]These are adjustable and can be modified during training. While hyperparameters are external configuration settings set before the training starts.

- The term $\Delta Z^{f,\tau} = \left(z_{\tau+k} \oplus z^f_{\tau+k}\right)_{0 \le k < R}$ denotes the keystream bit differences (KSD) between normal and faulty keystream bits, where τ represents the fault injection round and R denotes the total number of keystream bits to be considered.
- For a given input feature vector $\Delta Z^{g,\tau}$, the ML model[2] outputs a probability distribution over the output labels, given by

$$\mathcal{P} = (p_0, p_1, \ldots, p_{n-1}), \quad \sum_{f=0}^{n-1} p_f = 1,$$

where n denotes the number of possible fault locations and p_f represents the probability that the fault injection occurred at the f-th location.
- The output label selected for the final prediction by the ML model is the label having the highest probability. It is referred to as the *predicted output*, and the corresponding probability $p_{\hat{f}}$ is referred to as the *prediction confidence*. The predicted output is obtained as

$$\hat{f} = \arg\max_f \ p_f.$$

We refer to this approach as the *Top-1 Selection* .
Unless stated otherwise, throughout this paper, the term *predicted output* denotes the output label selected using the *Top-1 Selection* .
- Let f denote the true fault location and $\hat{f}$ be the predicted output. The *success probability* of f is defined as the fraction of test sample (corresponding to the output label f) for which $\hat{f} = f$. The average of success probability over all fault locations defines the *accuracy* of the model.
- A fault location is *correctly identified* (and referred to as a *correct fault*) when the predicted output matches the true injection location, i.e., when $\hat{f} = f$.
- F denotes the required number of fault locations to form the equations for the SAT solver for key recovery.
- One *Key-recovery attempt* corresponds to executing the SAT solver on a given system of equations for all possible assignments of the guessed variables. In particular, if l Boolean variables are guessed, the SAT solver must be executed for each of 2^l possible assignments. This entire process counts as one *Key-recovery attempt*. In our case, $l = 2$ for $F = 18, 20, or\ 25$; thus, one key recovery attempt $= 4$ SAT solver invocations.

III. Differential Fault Attack: A Case Study on Atom

Differential Fault Attack (DFA) is one of the most common cryptanalytic attacks on stream ciphers. In DFA, faults are injected into the internal state during PRGA. Since the same Key-IV pair is used repeatedly while injecting the fault, it leads to immediate differences in the keystream bits. By analyzing these differences, the attacker can infer partial or complete information about the internal state or the secret key. A DFA attack is executed in two phases. In the offline phase, faults are injected at known locations in a reference implementation of the cipher, and the corresponding KSD is collected. Based on this information, a framework, often statistical or machine-learning-based, is developed to identify the fault locations with high accuracy from a given KSD. The online phase is then carried out in the following three major steps.

- *Fault injection:* Faults are injected at unknown locations on the target device, and the corresponding KSD is computed.
- *Fault identification:* The framework developed during the offline phase is then used to infer the fault locations from the computed KSD.
- *Key recovery:* A system of equations is then formed using the set of KSD computed in the first step and the fault locations identified in the second step. The equations are then solved using a SAT solver to recover the secret key.

In our attack model, we assume that a single LFSR bit is flipped during an injection and the offline phase is developed accordingly. We discuss the last two steps of the online phase one by one in the upcoming sections.

A. Fault Identification Techniques

Fault identification techniques such as correlation-based algorithm [5], [6] and ML-based algorithm [8], [18] are commonly used in the literature. Therefore, for a comparative analysis of our framework with these techniques, we briefly discuss both techniques in the following sections.

1) Correlation Based Fault Identification: This method relies on *fault signatures*, which capture the statistical pattern produced by fault injection at given locations. For each fault location, a corresponding signature is obtained by performing multiple experiments and estimating the probability that the faulty keystream bit differs from the original keystream bit. More precisely, for a given fault f, the signature is $\mathcal{Q}^f = (p^f_0, p^f_1, \ldots,)$, where $p^f_i = Pr(z^f_{i+\tau} \neq z_{i+\tau})$ and τ is the fault injection round. The signature of the Atom is illustrated as a heatmap in Fig. 1, where the x-axis represents rounds, the y-axis represents fault positions, and the signature Q^f is represented in the form of color-encoding. We performed the computation of the signature over 2^{12} experiments. From Fig. 1, it can be observed that up to approximately 70 rounds from the fault injection, the keystream bits display non-random behaviour, i.e., $p^f_i \neq 0.5$ for some f. Beyond this , $p^f_i \approx 0.5$ with deviation bounded by ± 0.001, for all f and i, stating that faulty and fault-free keystream bits are almost indistinguishable. Therefore, all subsequent experiments, including those for fault identification and for key recovery, are performed with 70 keystream bits, as additional keystream bits do not provide any useful information.

In the online phase, the fault injection step outputs a KSD, $\Delta Z^{g,\tau} = (v^g_\tau, v^g_{\tau+1}, \ldots,)$, where g is the fault location unknown to the attacker. Next, the correlation coefficients are computed between the collected KSD and each precomputed signature, i.e., $Corr(\Delta Z^{g,\tau}, \mathcal{Q}^f), \forall f$. In the Top-1 selection, the fault location corresponding to the maximum correlation value is selected as the predicted fault location. However, this prediction is not always correct, as there may exist other fault

[2]In this article, the term ML model refers to supervised models that output a probability distribution over output labels.

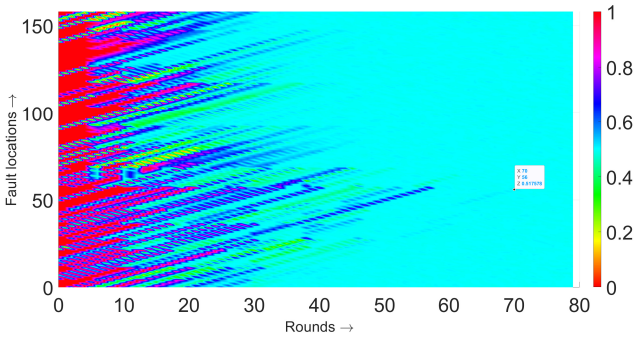


Fig. 1. Heatmap visualization of fault signatures for Atom, where $\mathcal{Q}^f$ (the signature vector) is encoded using color intensity.

locations whose correlation values are higher than that of the true fault location. To quantify this, for a given fault g and corresponding KSD $\Delta Z^{g,\tau}$, the rank $\alpha(\Delta Z^{g,\tau})$ is defined as,

$$\alpha(\Delta Z^{g,\tau}) = \#\{f \ : \ Corr(\Delta Z^{g,\tau}, \mathcal{Q}^f) \geq Corr(\Delta Z^{g,\tau}, \mathcal{Q}^g)\}.$$

Note that the computation of the $\alpha(\Delta Z^{g,\tau})$ is carried out in the offline phase. For a given fault position g, multiple experiments are performed to estimate the average rank α^g. Let α_{global} be defined as the average of α^g over all fault locations g. Using α_{global}, we can estimate the average number of attempts required to identify the correct fault location, i.e., a smaller value of α_{global} indicates better performance and $\alpha_{\text{global}} = 1$ means true fault location is ranked first.

In the online phase, after each fault injection, a list of correlation values $\text{Corr}(\Delta Z^{g,\tau}, \mathcal{Q}^f)$ for all fault locations f is computed and sorted in descending order. Since F distinct fault locations are required to break the cipher, we obtain F such lists. On average, among α^F_{global} combinations, one corresponds to the correct set of fault locations. Since the SAT solver is the only way to verify whether a given combination is correct, the average number of key-recovery attempts (refer to Section II-C) required is α^F_{global}. Let us refer to this method as *Rank-based Selection*.

A major drawback of rank-based selection is that if the number of required faults F is large ($F \geq 18$ in our case) and more than one candidate is tried (i.e., average rank ≈ 2) for each injection, then approximately 2^F key-recovery attempts would be needed. If the SAT solver time is high, then the total time complexity becomes huge and sometimes infeasible.

2) ML-based Fault Identification: Machine Learning offers an alternative approach to fault identification by converting it to a supervised classification problem. In this technique, an ML model (in particular, a multi-layer perceptron) is trained with KSD as the input feature vector and fault locations as output labels. Experimental results show that the ML technique outperforms the correlation-based technique [8], [18]. Since an ML outputs a probability distribution denoted by $(p_0, p_1, \ldots, p_{n-1})$ over the output labels, taking the reference of prediction probability p_i's, the Top-1 selection and rank-based selection can be defined in an ML context.

In Top-1 selection, the output label f with the highest prediction probability p_f is selected for the final output. The rank-based method follows a similar approach as discussed in Section III-A1. Similar to the correlation approach, we define the rank for ML-based identification as,

$$\beta(\Delta Z^{g,\tau}) = \#\{f \ : \ p_f \geq p_g\}.$$

Similarly, fault-wise average rank β^g and overall rank β_{global} can be computed analogously to α^g and α_{global} from Section III-A1. For attacks requiring F distinct faults, the rank-based selection evaluates β^F_{global} candidate combinations for the key-recovery attempts.

To summarize, two fault identification techniques are commonly used in the literature: correlation-based and ML-based. For each technique, two evaluation rules are considered, namely Top-1 selection and rank-based selection. Under the Top-1 selection, performance is measured in terms of success probability, defined as the probability that the true fault location is ranked highest based on the correlation values or the prediction probabilities. In contrast, under the rank-based selection, performance is quantified using the average rank of the true fault location. These metrics determine the total number of key-recovery attempts and, consequently, the number of SAT solver invocations.

Based on our experimental results for Atom, the Top-1 selection outperforms the rank-based selection for both ML-based and correlation-based fault identification, as it significantly reduces the number of SAT solver invocations, albeit at the cost of requiring additional fault injections. A detailed experimental comparison is presented in Section III-B, where we further refine the Top-1 selection rule using a filter, referred to as the Confidence-Threshold Strategy for the fault selection.

B. Our ML-based Fault Identification Framework

In this section, we present our fault identification techniques for the Atom cipher and analyze their computational complexity. The training, validation, and testing data splits ($2^{19.524}$, $2^{17.523}$, and $2^{18.261}$, respectively) are used to train, validate, and test the models. First, we trained an MLP model with an input layer containing a KSD of length 70 and an output layer whose labels represent fault locations, i.e., 0–68 (according to the first assumption of our fault model). We used four hidden layers with sizes 512, 412, 332, and 268, where the layer size decreases successively by a factor of 0.8062 to improve generalization. The Swish activation function is used in the hidden layers. To obtain a normalized probability distribution over all fault locations, the softmax activation function is used in the output layer. The remaining hyperparameters used to train the model are listed in Table II(a). On the test dataset, the model achieved an accuracy of 0.84.

Since the input features are Boolean, we also employed decision-tree-based algorithms, which are well suited for discrete-valued inputs and can efficiently capture complex nonlinear feature interactions via hierarchical splits. Among boosting techniques such as XGBoost, LightGBM, and CatBoost, we found that XGBoost achieved a higher accuracy of 0.8431 using the hyperparameters listed in Table II(b).

We use *mlogloss* and *merror* as the evaluation metrics. Here, *mlogloss* denotes the multiclass cross-entropy loss, while *merror* denotes the fraction of misclassified samples. To determine appropriate hyperparameters for each model, we performed hyperparameter tuning using Optuna[3], with 60 trials for the MLP and 20 trials for XGBoost.

TABLE II
TRAINING CONFIGURATION AND PERFORMANCE OF OUR ML MODEL

Hyperparameter	Value
Number of hidden layers	4
Hidden layer sizes	[512, 412, 332, 268]
Dropout rate	0.1915
Batch size	128
Activation function	Swish, Softmax
Optimizer	Adam
Learning rate	3.38×10^{-4}
Loss function	Sparse Categorical Cross-Entropy
Epochs	150
Train Accuracy	0.860045
Validation Accuracy	0.837984
Test Accuracy	**0.840054**

(a) Multi-Layer Perceptron Model

Hyperparameter	Value
Objective function	Multi-class Soft Probability
Number of classes	69
Evaluation metrics	mlogloss, merror
Maximum tree depth	6
Minimum child weight	9
Gamma	0.35
Learning rate	0.035
Subsample ratio	0.8
Column sampling ratio	0.8
L1 regularization	5.5
L2 regularization	3.0
Tree construction method	Histogram-based
Number of boosting rounds	2500
Early stopping rounds	200
Best iteration	1624
Training accuracy	0.8897
Validation accuracy	0.8419
Test accuracy	**0.8431**

(b) eXtreme Gradient Boosting Model

Model	XGBoost	MLP
Weight	0.65	0.35
Test Accuracy	**0.845028**	

(c) Ensemble Model

Since fault identification is a classification problem, both MLP and XGBoost return a probability distribution over fault locations. We constructed a weighted ensemble model by combining the outputs of these two models. Specifically, for a given KSD, the output of the ensemble model is also a probability distribution over the fault locations and is given by $\mathcal{P}_{\text{ensemble}} = 0.65 \times \mathcal{P}_{\text{XGB}} + 0.35 \times \mathcal{P}_{\text{MLP}}$, where $\mathcal{P}_{\text{MLP}}$ and $\mathcal{P}_{\text{XGB}}$ denote the corresponding probability distribution for MLP and XGBoost model, respectively. As a result, the ensemble model further improves the accuracy to 0.845028 (see Table II(c)).

A comparative analysis of our ensemble model with the correlation-based technique is shown in Fig. 2. The comparison in terms of average rank under the rank-based selection method and the success probability under the Top-1 selection method is shown in Fig. 2a and Fig. 2b, respectively. From these figures, we observe a consistent improvement for all fault locations with respect to both metrics. Therefore, our ensemble model outperforms the correlation-based technique. In particular, the average rank over all fault locations for the correlation-based technique is 1.93, whereas it is reduced to 1.34 using our approach. Similarly, the success probability under Top-1 selection improves from 0.722 for the correlation-based technique to 0.845 for our ensemble model.

[3]Optuna is a Python module for automatic hyperparameter optimization (https://optuna.org)

TABLE III
PERFORMANCE EVALUATION OF OUR ENSEMBLE AND THE CORRELATION-BASED MODEL FOR THE CASE F = 18

Model	Avg Rank (α_{global})	# Key-recovery attempts	# SAT invocations	Total no. of fault injections
Correlation	1.93	138049	552196	18
Our Ensemble	1.35	222	888	18

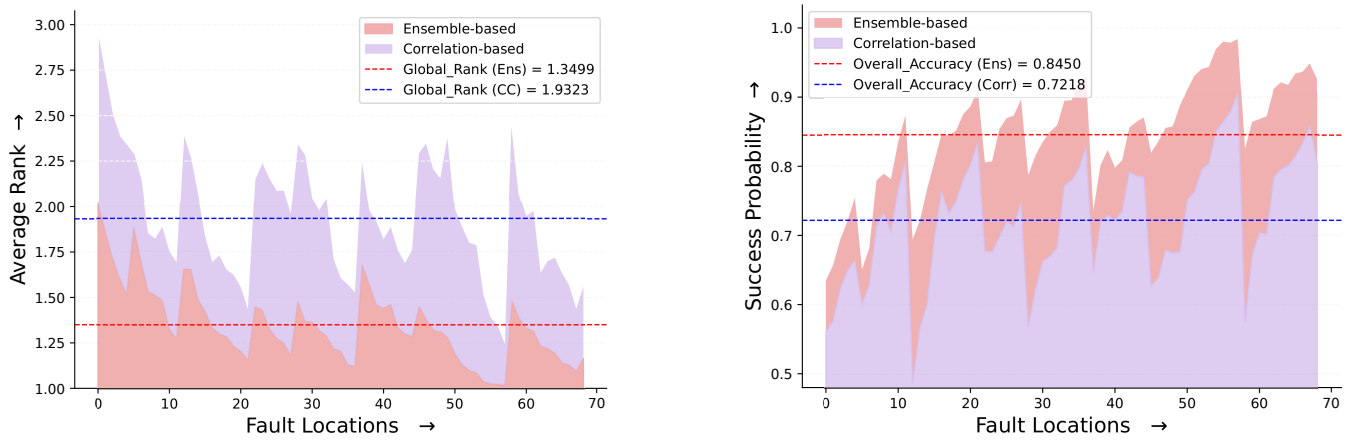
(a) Under rank-based selection

Model	Accuracy		# Key-recovery attempts	# SAT invocations	Expected no. of fault injections*
	1 fault	18 fault			
Correlation	0.722	0.0028	358	1432	6444
Our Ensemble	0.845	0.048	21	84	378

*18 × # Key-recovery attempts

(b) Under Top-1 selection

A comparative analysis of the performance of the correlation-based and our ensemble model under Top-1 selection and rank-based selection is presented in Table III. To recover the secret key within a feasible time, at least 18 correct faults are required (refer to Table I). Thus, under rank-based selection, the correlation-based technique requires, on average $1.93^{18} \approx 138049$ key-recovery attempts, which is equivalent to $4 \times 138049 = 552196$ SAT solver invocations (see Section II-C). However, our ensemble model requires $1.35^{18} \approx 222$ key-recovery attempts, equivalent to $4 \times 222 = 888$ SAT solver invocations, representing a significant improvement (presented in Table III(a)). Let us observe the performance under the Top-1 selection (refer to Table III(b)). The overall success probability of all 18 fault locations being correctly identified is $0.722^{18} \approx 0.0028$ for the correlation-based technique, whereas it increases to $0.845^{18} \approx 0.048$ for the proposed ensemble model. Therefore, the expected number of key-recovery attempts required to obtain a set of 18 correct fault locations is $1/0.0028 \approx 358$ for the correlation model and $1/0.048 \approx 21$ for the proposed ensemble model. Since each key-recovery attempt requires a set of 18 distinct fault locations. Under Top-1 selection, a failed attempt requires new fault injections into the cipher to obtain another set of 18 fault locations. Thereby, the expected number of fault injections required for one successful key recovery is $18\times$ the number of key-recovery attempts. Consequently, the required number of SAT solver invocations is approximately $4 \times 358 = 1432$ for the correlation-based model and $4 \times 21 = 84$ for the ensemble



(a) Fault-wise rank under the rank-based selection

(b) Fault-wise success probability under the Top-1 selection

Fig. 2. Fault-wise performance comparison of the ensemble-based and correlation-based fault identification model

model. As for fault injections, it requires 6444 fault injections for the correlation-based model and 378 fault injections for the ensemble model.

Among all technique-selection rule pairs, the ensemble model under Top-1 selection clearly outperforms other pairs in terms of SAT solver invocations. Although this represents a substantial improvement, the number of SAT solver invocations and fault injections is still high. For further improvement, we continue with the same technique and selection rule, i.e., the ensemble model under Top-1 selection, and introduce a threshold-based filter on the selection rule. We referred to this refined strategy as the Confidence-Threshold strategy as described next.

1) Confidence-Threshold Strategy (CTS): In this strategy, an ML prediction is accepted as the final fault location if and only if its prediction probability exceeds a predefined threshold, λ. In contrast to the Top-1 selection, where a prediction is always accepted regardless of its prediction probability, the CTS selectively accepts only high-confidence predictions. The procedure is described below.

Let F be the number of different faults required by the SAT solver to initiate the key recovery procedure. In the online phase, perform fault injection followed by fault identification through our ensemble model (refer to Section III), which outputs a predicted fault location along with its prediction confidence[4]. Accept the predicted fault location for the key recovery process if and only if prediction confidence is greater than a given threshold λ, for $\lambda \in [0, 1]$. We refer to such a fault location as an *accepted fault location*. This procedure is repeated until F different accepted fault locations are collected.

Note that the acceptance of each fault can be verified independently. These F fault locations are then used for the key recovery procedure. To further analyze the performance, let us define some relevant probabilistic notations:

- G denote a random variable representing the predicted fault location.
- P is a random variable denoting the prediction probability corresponding to G, i.e., the prediction confidence.
- C denotes a random variable indicating the correctness of G,

$$C = \begin{cases} 1, & \text{if } G = g^{\star} \text{ (correct fault location)}, \\ 0, & \text{otherwise.} \end{cases}$$

- $Pr(P > \lambda)$ denotes the probability that a predicted fault location is accepted.
- $Pr(C = 1 \mid P > \lambda)$ denotes the probability that the predicted fault location is correct given it is accepted.

We now analyze the performance of the proposed CTS. The expected number of trials required to get one accepted fault location is $\frac{1}{Pr(P>\lambda)}$. Since the SAT solver requires F accepted faults, the expected number of trials needed to collect F accepted faults is $\frac{F}{Pr(P>\lambda)}$, as the acceptance criteria for each fault can be verified independently. Once a set of F accepted faults is obtained, the correctness of all F predictions can only be verified by executing the SAT solver. If the solver returns a unique solution, this indicates that all fault locations are correctly predicted. Therefore, the probability that all F accepted fault locations in a set are correct is $Pr(C = 1 \mid P > \lambda)^F$. Consequently, the expected number of such sets, equivalently, the number of key-recovery attempts required to achieve one successful key recovery is $\frac{1}{Pr(C=1|P>\lambda)^F}$.

Thus, the expected number of fault injections needed for one successful key recovery is given by

$$\left\lceil \frac{F}{Pr(P > \lambda)} \right\rceil \times \left\lceil \frac{1}{Pr(C = 1 \mid P > \lambda)^F} \right\rceil. \quad (1)$$

Let us illustrate the above analysis with an example where 18 correct faults (i.e., F = 18) locations are required to recover the secret key. We set the threshold λ to 0.79. As per our ensemble model output statistics, for a given KSD in the online phase, the probability that the predicted fault location is accepted, i.e., $Pr(P > 0.79)$, is 0.713. Therefore, the expected

[4]In Top-1 Selection rule, we refer to prediction probability as prediction confidence.

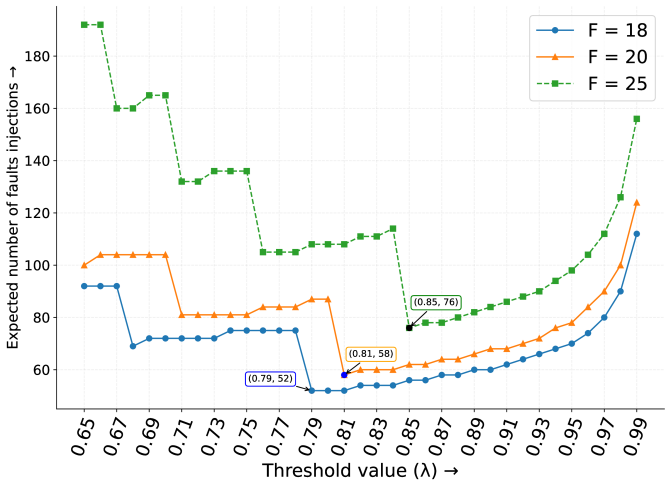


Fig. 3. Total number of fault injections required versus confidence threshold λ for different numbers of correct faults F.

number of fault injections required to get a set of 18 accepted fault locations is $\left\lceil \frac{18}{Pr(P>0.79)} \right\rceil = \lceil \frac{18}{0.713} \rceil = 26$. Once the fault prediction is accepted, the probability that it matches with the correct fault location, i.e., $Pr(C = 1 \mid P > 0.79)$, is 0.963. As a result, the expected number of sets containing 18 accepted fault locations required to achieve one successful key recovery is $\left\lceil \frac{1}{Pr(C=1|P>0.79)^{18}} \right\rceil = \lceil \frac{1}{0.506} \rceil = 2$. These analytical estimates are reported in Table IV. In other words, two key recovery attempts are required, on average, to break the cipher. Therefore, on average, only 8 SAT solver invocations are required for a successful key recovery with a set of 18 correct fault locations obtained by injecting a total of 52 faults into the cipher.

Fig. 3 shows the expected number of fault injections needed for a successful key recovery for different thresholds λ and F values (18, 20, and 25). The expected number of fault injections for a fixed F depends upon two terms: $Pr(P > \lambda)$ and $(Pr(C = 1 \mid P > \lambda))^F$, both of which vary with threshold λ (see Eq. 1). However, our model statistics indicate that these two terms show opposing behaviour as λ increases. Specifically, $Pr(P > \lambda)$ decreases (gradually at first, then sharply as $\lambda \to 1$), while $Pr(C = 1 \mid P > \lambda)$ increases steadily with roughly constant incremental improvements.

For the region $0 \leq \lambda \leq 0.79$ (when $F = 18$), the increase in $Pr(C = 1 \mid P > \lambda)$ dominates the drop in $Pr(P > \lambda)$, thus decreasing the total number of fault injections required. Beyond this region, the term $Pr(P > \lambda)$ declines rapidly, overpowering the improvement from $Pr(C = 1 \mid P > \lambda)$, which causes a sharp increase in the total number of fault injections. This behaviour leads to an optimal threshold region that minimizes the expected number of fault injections. As shown in Fig. 3, for $F = 18$, the optimal threshold lies in [0.79, 0.815], requiring just 52 fault injections. For $F = 20$ and $F = 25$, the optimal ranges are [0.81, 0.818] and [0.85, 0.854], requiring 58 and 76 injections, respectively.

Table IV summarizes the performance of our ensemble model with respect to the different number of correct faults required, i.e., $F = 18, 20, 25$. The data reported in this table are according to the optimal threshold value λ obtained through Fig. 3. The metrics discussed above, i.e., λ, $Pr(P > \lambda)$, $Pr\ (C = 1 |\ P > \lambda)$ are used as defined in the previous discussion. Pr (18 correct faults $|P > \lambda$) denotes the accuracy of the ensemble model to correctly predict 18 faults, provided that all are accepted, i.e., their confidence $> \lambda$. The term F denotes the required number of correct faults, whereas the *Expected no. of fault injections* denotes the expected number of fault injection attempts required to get a set of F correct faults. For all the cases, we need 8 SAT solver invocations on average. For the case where 18 different faults are needed for the SAT solver, the best data is observed in terms of the expected number of fault injections, i.e., with 52 single fault injections, we obtain, on average, two sets of 18 accepted fault predictions, out of which one corresponds to the set of correct fault locations needed for key recovery. The trained models and the corresponding CSV files containing rank and success probability results are publicly available at https://github.com/Satyam-Crypto/Atom_Fault_Verification_Files.git

To summarize, we demonstrate numerically that the proposed CTS significantly outperforms the other techniques under Top-1 selection. For $F = 18$, we compare three approaches: baseline correlation-based, our ensemble model, and our ensemble with confidence-threshold filtering. The results show dramatic reductions in both metrics: the total number of fault injections ($6444 \to 378 \to 52$) and the total number of SAT solver invocations ($1432 \to 84 \to 8$). In comparison with the rank-based selection, the CTS also achieves a significant reduction in the number of SAT solver invocations ($552{,}196 \to 888 \to 8$). The corresponding comparisons are reported in Tables III and IV.

2) Discussion on the Proposed Fault Identification Framework: ML-based fault identification captures the relationship among multiple positions of the KSD, which cannot be exploited by correlation-based methods. In particular, it learns how different KSD positions jointly characterize a fault location, resulting in higher fault identification accuracy than correlation-based techniques.

Building on this ML-based approach, we introduce two additional techniques that further improve the effectiveness of the attack: a weighted ensemble model, which improves the fault identification accuracy, and the CTS, which reduces the number of required fault injections even when the identification accuracy is not close to 1.

The weighted ensemble combines the predictions of MLP and XGBoost, thereby reducing the dependence on any single classifier. Since the two models have different learning mechanisms and often make partially different prediction errors, one model can compensate when the other is less confident or incorrect. A higher classification accuracy is achieved by assigning a larger weight to the stronger classifier, as shown in Table II. Although only two classifiers are considered here, the same framework naturally extends to multiple models by taking a weighted average of their prediction probabilities. Furthermore, the proposed ML and ensemble techniques can be extended for fault identification in other stream ciphers by properly tuning their hyperparameters.

Even with the ensemble model, the fault identification

TABLE IV
PERFORMANCE EVALUATION OF OUR ENSEMBLE MODEL W.R.T. CTS

	Req. no. of correct faults(F)	Threshold (λ)	$Pr(P > \lambda)$	$Pr(C = 1 \mid P > \lambda)$	$Pr(F$ correct faults $\mid P > \lambda)$	# Key recovery attempts	# SAT invocations	Expected no. of fault injections*
Confidence	18	0.79	0.713	0.963	0.506	2	8	52
Threshold	20	0.81	0.697	0.966	0.508	2	8	58
Strategy	25	0.85	0.662	0.974	0.515	2	8	76

* Expected number of faults needed to get a set of F accepted fault locations $\times$ # Key-recovery attempts

accuracy may still be imperfect, leading to unnecessary SAT solver invocations on incorrectly identified faults. The CTS addresses this issue by filtering out low-confidence predictions before the key recovery stage, thereby reducing the number of SAT solver invocations as well as the required fault injections (Section III-B1). Similar to the ensemble framework, this strategy can also be incorporated into fault attacks on other stream ciphers whenever the fault identification accuracy is not close to one. The threshold parameter λ can be selected to minimize the expected number of fault injections required for a successful attack (refer to Equation (1)).

Remark 1: The *expected number of fault injections* reported in Table III(b) and Table IV counts only *distinct* injections. Repeated injections with the same Key–IV pair and targeted round τ always generate identical KSD and are detected by KSD matching; such duplicates are excluded. If key recovery fails, another set of faults is collected using the same rule. Duplicate injections within a set are discarded, while repetitions across different sets are allowed and counted.
If distinct accepted faults become insufficient for a fixed targeted round, then the injection round can be changed for the next set. This does not violate our attack assumptions, since the fault injection round is assumed to be known to the attacker. All other steps remain unchanged as the ML model is trained using fault injections at random rounds.

C. State Recovery From the Differential Key-Streams

Our next objective is to extract the secret key using normal and faulty keystream bits. As per our literature survey, prior to the design of Atom, no stream cipher had ever used the state bits to select the secret key bits to update the internal state. However, as discussed earlier in Section II-A, an additional key filter, k_{cnt}, is incorporated during the PRGA phase of Atom. This filter selects the key bit at index cnt, where cnt is the decimal representation of the last 7 LFSR state bits. Since the SAT solver can only process linear or nonlinear Boolean equations, we need an algebraic Boolean expression for k_{cnt}. Therefore, we formulate an 8-degree polynomial, $K(k, L, cnt)$ representing k_{cnt} as follows (also shown in Algorithm 1):

Algorithm 1 Algebraic expression for the Key Filter k_{cnt}: $K(k, L, cnt)$

Input: LFSR state variables $L = (l_0, l_1, \ldots, l_{68})$, secret key variables $k = (k_0, \ldots, k_{127})$.

Output: Boolean algebraic expression of k_{cnt}.

$cnt \leftarrow l_{62} \| l_{63} \| l_{64} \| l_{65} \| l_{66} \| l_{67} \| l_{68}$
$K(k, L, cnt) \leftarrow 0$
for $i = 0$ to 127 **do**
  $a_0^i \| a_1^i \| \cdots \| a_6^i \leftarrow$ 7-bit binary representation of i.
  $temp \leftarrow 1$
  for $j = 62$ to 68 **do**
    $temp \leftarrow temp \ \& \ ((1 \oplus a_{j-62}^i) \oplus l_j)$
  end for
  $K(k, L, cnt) \leftarrow K(k, L, cnt) \oplus (temp \ \& \ k_i)$
end for
return $K(k, L, cnt)$

$$K(k, L, cnt) = \bigoplus_{i=0}^{127} (A_{62}^i A_{63}^i A_{64}^i A_{65}^i A_{66}^i A_{67}^i A_{68}^i) k_i$$

where

$$\begin{aligned} L &= (l_0, l_1, \ldots, l_{68}) \text{ represents the LFSR state,} \\ cnt &= l_{62} \| l_{63} \| l_{64} \| l_{65} \| l_{66} \| l_{67} \| l_{68}, \\ A_j^i &= (1 \oplus a_{j-62}^i) \oplus l_j, \quad \text{for } j \in \{62, 63, \ldots, 68\}, \end{aligned}$$

and $a_0^i \| a_1^i \| \cdots \| a_6^i$ is the 7-bit binary representation of i. (2)

To better understand, let $cnt = 0||0||0||0||0||1||1$. In this case, $k_{cnt} = k_3$ is used to update the NFSR. From (2), following holds: $A_{62}^i = 1 \oplus a_0^i$, $A_{63}^i = 1 \oplus a_1^i$, $A_{64}^i = 1 \oplus a_2^i$, $A_{65}^i = 1 \oplus a_3^i$, $A_{66}^i = 1 \oplus a_4^i$, $A_{67}^i = a_5^i$, $A_{68}^i = a_6^i$. Therefore, $\prod_{j=62}^{68} A_j^i = (1 \oplus a_0^i)\ (1 \oplus a_1^i)\ (1 \oplus a_2^i)\ (1 \oplus a_3^i)\ (1 \oplus a_4^i)\ a_5^i\ a_6^i = 1$ if and only if $a_0^i = a_1^i = a_2^i = a_3^i = a_4^i = 0$ and $a_5^i = a_6^i = 1$, *i.e.*, $i = 3$. As a result, from (2), $K(k, L, cnt) = k_3$ which matches k_{cnt}.

This form of representation increases the degree of the NFSR update function from 3 to 8, making it more complex. However, it allows us to formulate the LFSR-dependent subscript term into a nonlinear Boolean polynomial, enabling us

to generate equations for the SAT solver.

Now, we discuss the formulation of equations using state and key bits as variables, and the known keystream bits. According to the design of Atom, each round uses two nonlinear components: a 5-degree Boolean function h and an 8-degree NFSR update function (refer to Section II-A and (2)) to output one keystream bit. As the number of rounds increases, both the number of monomials and the algebraic degree of the equations grow rapidly. Consequently, constructing explicit algebraic equations for each keystream bit involving the state bit and key bit variables becomes infeasible beyond a certain number of rounds. During the PRGA phase, the algebraic degree increases from 5 to 12 by round 6, while the number of monomials grows dramatically from 127 at round 0 to 210,358 at round 6, 5,788,380 at round 8, and 184,842,838 at round 10.

In the literature, this problem is countered by introducing dummy variables [6], [7]. Following the same approach, we add two dummy Boolean variables at each round representing the output of the LFSR and NFSR update functions. Consequently, two supplementary equations are formed. The detailed algorithm for equation generation is given in Algorithms 2 and 3. First, in Algorithm 2, we define a function Gen_Equ for the generation of equations at a given round r, with input as LFSR variables L_{var}, NFSR variables N_{var}, key variables K_{var}, fault location f and corresponding keystream bit z_r. The keystream bit equation is equated to the corresponding known keystream bit, z_r (Step 1). Next, as discussed above, we introduce two new variables $\mathcal{L}_r^f$ and $\mathcal{N}_r^f$ (Step 2) and equate them to LFSR and NFSR update functions respectively (Steps 3, 4). Lastly, the LFSR and NFSR, i.e., L_{var} and N_{var} are updated with the new dummy variables (Steps 5, 6). Algorithm 2 is also used when no fault is injected. To accommodate the fault-free instances, we represent f by $None$. Now we discuss Algorithm 3 in which Algorithm 2 is repeatedly called at each round.

Algorithm 3 demonstrates the steps for key recovery using the SAT solver. After obtaining the fault locations from the fault identification step (stated in Section III-B), store them in a set $\mathcal{F}$. Let R be the total number of rounds to be considered. Create an array, Equ_list, to store the equations for the SAT solver. Initialize L_{var}, N_{var} and K_{var} with Boolean variables to represent the targeted state and the secret key (Step 2). Form the equations corresponding to the instances where no fault has been injected (i.e., $f = None$) in Step 3 - 8, by calling the function Gen_Equ at each round $(\tau + i)$ in Step 5. Append the keystream bit equations to the list Z_{equ} for further usage (Step 6). Note that Algorithm 2 generates three equations for each round. As a result, a system of $3R$ equations is obtained for a sequence of R keystream bits, involving 69+90+128+$2R$ variables. Out of these $3R$ equations, only $2R$ equations are provided to the SAT solver, since only l_{equ} and n_{equ} are stored in Equ_list, other R equations, which are stored in Z_{equ} are combined later with some new equations to form a single equation.

Next, for each fault location f, restart the targeted device with the same (Key, IV) pair (Step 10) and also re-initialize L_{var} and N_{var} to represent the targeted state at round τ (Step 11). We inject the fault at f-th LFSR position of the targeted device and the corresponding position in L_{var} is flipped (Step 13). Extract the keystream bit $z_{\tau+i}^f$ from the targeted device and call the function Gen_Equ to form the equation at the given round $\tau + i$ (Step 16). Here, instead of passing the faulty keystream bit equations (z_{equ}^f) and the normal keystream bit equations (z_{equ}) stored in Z_{equ} separately to the SAT solver, we consider the XOR of z_{equ}^f with the corresponding normal keystream bit equations (Step 17) for the SAT solver. Store all the equations in Equ_list. Note that the total number of equations stored in Equ_list is $2R + 3R|\mathcal{F}|$ involving $69+90+128+2R(|\mathcal{F}|+1)$ variables. Finally, all the equations in Equ_list are fed into the SAT solver (Step 20). At last, we extract the secret key from the SAT solution and verify it. If the verification fails, this indicates the presence of multiple solutions. In that case, repeat the Algorithm 3 with an increased number of rounds, i.e., with more information. However, in our experiments with 70 rounds, we did not observe any such instances. If the solver returns *unsat*, then the entire process, i.e., from the fault injection and fault identification steps to equation formation, followed by solving, is repeated.

Experimental Results: For Atom, we form a system of equations as mentioned previously and feed it to the SAT solver. We tried Kissat[5] solver [19] and found it to be more efficient for our system of equations. As a result, for 18 correctly predicted faults, we need 40 hours and 25 minutes on

---

Algorithm 2 Generate Internal State Equation: $Gen_Equ(L_{var}, N_{var}, K_{var}, f, r, z_r)$

Input:

- $L_{var}, N_{var}, K_{var}$: LFSR state, NFSR state and the Secret Key in terms of Boolean variables.
- f: Injected fault location.
- r : Current round.
- z_r : Keystream bit at round r.

Functions:

- O, F, G : Keystream, LFSR and NFSR update function, respectively.

1: $z_{equ} \leftarrow z_r \oplus O(L_{var}, N_{var})$;
2: Define dummy variables $\mathcal{L}_r^f$ and $\mathcal{N}_r^f$.
3: $l_{equ} \leftarrow \mathcal{L}_r^f \oplus F(L_{var})$;
4: $n_{equ} \leftarrow L_{var}[0] \oplus \mathcal{N}_r^f \oplus G(N_{var}) \oplus K_{var}[r\%128] \oplus K(K_{var}, L_{var}, cnt)$; ▷ $K(\cdot)$ defined in Algorithm 1
5: $L_{var} = L_{var}[1:] + [\mathcal{L}_r^f]$;
6: $N_{var} = N_{var}[1:] + [\mathcal{N}_r^f]$;

Output: $z_{equ}, l_{equ}, n_{equ}, L_{var}, N_{var}$.

---

[5]https://github.com/arminbiere/kissat

Algorithm 3 Key Recovery Algorithm using SAT solver

Input: $\mathcal{F}$ (set of fault locations), R (rounds), τ (target round)

Equ_list=$\{\phi\}$; $Z_{equ} = \{\phi\}$
Initialize L_{var}, N_{var}, K_{var} with Boolean variables
for $i \leftarrow \{0, 1, \ldots, R-1\}$ **do**
    $z_{\tau+i} \leftarrow$ keystream bit at round $\tau + i$
    $z_{equ}, l_{equ}, n_{equ}, L_{var}, N_{var}$ = $Gen_Equ(L_{var}, N_{var}, K_{var}, none, \tau + i, z_{\tau+i})$ ▷ Algorithm 2
    $Z_{equ} = Z_{equ} \cup z_{equ}$
    $Equ_list = Equ_list \cup (l_{equ}, n_{equ})$
end for
for $f \in \mathcal{F}$ **do**
    Restart the device with the original IV and KEY
    Reset L_{var} and N_{var}
    Inject fault at fth LFSR position at round t
    $L_{var}[f] = L_{var}[f] \oplus 1$
    for $i \leftarrow \{0, 1, \ldots, R-1\}$ **do**
        $z^f_{\tau+i} \leftarrow$ keystream bit at round $\tau + i$
        $z^f_{equ}, l^f_{equ}, n^f_{equ}, L_{var}, N_{var}$ = $Gen_Equ(L_{var}, N_{var}, K_{var}, f, \tau + i, z^f_{\tau+i})$ ▷ Algorithm 2
        $Equ_list = Equ_list \cup (l^f_{equ}, n^f_{equ}, z^f_{equ} \oplus Z_{equ}[i])$
    end for
end for
Feed the Equ_list to the SAT solver
Extract the secret key from the SAT solution if found.

average by guessing two key bits. Similarly, for another set of correct faults, i.e., 20 or 25, the results are reported in Table V. Each Kissat run is executed with a randomly chosen seed and with randomized decision and tumble options enabled (–seed, –randec, –tumble).

D. Relaxing Assumption: Injection Round Unknown to the Attacker

Following the approach of [8], we also consider the case where the fault injection round is unknown but fixed across all fault injections used for a key-recovery attempt. To estimate the unknown injection round, the attacker uses the earliest non-zero-difference round among the obtained KSDs, denoted by τ^*, as the initial guess. The attacker then performs key recovery attempts with guesses $\tau^*, \tau^* - 1, \ldots$ until it succeeds. Note that no additional fault injections are required for these retries. Recall that, for $F = 18$, an average of 26 fault injections is required for a key recovery attempt. Based on 10,000 independent experiments, each with 26 fault injections, we observed that the actual injection round equals τ^* with probability 97.47% and belongs to $\{\tau^*, \tau^* - 1\}$ with probability 99.94%. Therefore, with overwhelming probability, the assumption of a known injection round can be relaxed at the cost of only two additional key recovery attempts, while the expected number of fault injections remains 52 for successful recovery.

TABLE V
SAT SOLVER RUNTIME FOR KEY RECOVERY

# Rounds (R)	# Faults (F)	# Guessed Key Bits	# Experiments	SAT Time (in Hrs:Mins)		
				Min.	Max.	Avg.
70	18	2	10	11:32	72:17	40:25
70	20	2	20	4:14	94:08	23:38
70	25	2	20	6:14	38:18	12:01

E. Machine Configuration

All experiments were conducted on the Aqua cluster (IIT Madras), utilizing dual Intel® Xeon® Gold 6248 processors (40 physical cores/80 logical CPUs) running on Linux Server 7.6 (Maipo). Kissat Solver version 4.0.4 is used as a SAT solver. All machine learning models were implemented using TensorFlow 2.20 (Keras 3.12) for MLP and XGBoost 3.1.2 for gradient-boosted decision trees. Hyperparameter optimization was performed using Optuna 4.6.0.

IV. CONCLUSION

This paper presents the first differential fault attack on the stream cipher Atom. By guessing two secret key bits and injecting 18/20/25 single-bit faults into the LFSR state at a known round, the proposed attack successfully recovers the secret key within 1–2 days. Fault locations are identified using a weighted ensemble of MLP and XGBoost. The proposed confidence-threshold strategy is then applied to filter out low-confidence predictions, significantly reducing the number of SAT solver invocations. As a result, the expected numbers of fault injections required to obtain 18, 20, and 25 correctly identified faults are 52, 58, and 76, respectively. In addition, we propose a method for formulating algebraic equations for the LFSR-dependent key filter that bypasses the barrier imposed by the key filter k_{cnt} in Atom. We believe that our findings advance our knowledge of Atom's security and demonstrate the application of ML-assisted techniques in fault-based cryptanalysis.

Among the Grain-family ciphers analyzed under DFA, Atom requires comparatively more fault injections for successful key recovery. This is primarily due to the rapid convergence of the signature values p^f_i toward 0.5 beyond approximately 20 rounds (see Fig. 1), which reduces the distinguishability between fault locations and consequently lowers the fault identification accuracy. As a result, obtaining 18 correctly identified faults requires, on average, 52 fault injections. To

conclude, we list the following follow-up works that can be pursued in future:

- Improving the fault identification accuracy and further exploiting Atom's algebraic structure are promising directions for reducing the number of fault injections.
- In our attack model, we make one relatively strong assumption, namely that fault injection is restricted to the LFSR state bits. A more practical model would allow faults to be injected into the entire state. In our experiment, we could not obtain feasible or sufficient results under the relaxed assumption. We therefore leave the development of an alternative method to address this more general setting as future work.

REFERENCES

[1] S. P. Skorobogatov, "Optically enhanced position-locked power analysis," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2006.

[2] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults (extended abstract)," in *International Conference on the Theory and Application of Cryptographic Techniques*, 1997.

[3] J. J. Hoch and A. Shamir, "Fault analysis of stream ciphers," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2004.

[4] S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on the grain family of stream ciphers," in *Workshop on Cryptographic Hardware and Embedded Systems*, 2012.

[5] D. Roy, B. N. Bathe, and S. Maitra, "Differential fault attack on kreyvium & flip," *IEEE Transactions on Computers*, vol. 70, pp. 2161–2167, 2021.

[6] S. Maitra, A. A. Siddhanti, and S. Sarkar, "A differential fault attack on plantlet," *IEEE Transactions on Computers*, vol. 66, pp. 1804–1808, 2017.

[7] A. Siddhanti, S. Sarkar, S. Maitra, and A. Chattopadhyay, "Differential fault attack on grain v1, acorn v3 and lizard," in *Security, Privacy, and Applied Cryptography Engineering*. Cham: Springer International Publishing, 2017, pp. 247–263.

[8] S. K. Mondal, P. Dey, H. S. Roy, A. Adhikari, and S. Maitra, "Improved fault analysis on subterranean 2.0," *IEEE Transactions on Computers*, vol. 73, no. 6, pp. 1631–1639, 2024.

[9] S. Banik, A. Caforio, T. Isobe, F. Liu, W. Meier, K. Sakamoto, and S. Sarkar, "Atom: A stream cipher with double key filter," *IACR Trans. Symmetric Cryptol.*, vol. 2021, pp. 5–36, 2021.

[10] M. Hell, T. Johansson, and W. Meier, "Grain: a stream cipher for constrained environments," *Int. J. Wirel. Mob. Comput.*, vol. 2, pp. 86–93, 2007.

[11] M. Hamann, M. Krause, and W. Meier, "Lizard – a lightweight stream cipher for power-constrained devices," *IACR Transactions on Symmetric Cryptology*, vol. 2017, no. 1, p. 45–79, Mar. 2017.

[12] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," *Journal of Cryptology*, vol. 31, no. 3, pp. 885–916, 2018.

[13] V. Mikhalev, F. Armknecht, and C. Müller, "On ciphers that continuously access the non-volatile key," *IACR Trans. Symmetric Cryptol.*, vol. 2016, pp. 52–79, 2017.

[14] R. Anand, R. Nandi, and T. Isobe, "Analysis of atom against quantum attacks," *IET Quantum Communication*, vol. 5, no. 1, pp. 88–102, 2024.

[15] Y. Makwana, A. Panigrahi, and S. Pal, "Analyzing security of small-state cipher atom for bitstream and image encryption," *Journal of Cyber Security Technology*, vol. 0, no. 0, pp. 1–15, 2025.

[16] S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on grain-128a using macs," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2012, pp. 111–125.

[17] W. Wang and D. Tang, " Differential Fault Attack on HE-Friendly Stream Ciphers: Masta, Pasta, and Elisabeth ," *IEEE Transactions on Computers*, vol. 74, no. 07, pp. 2267–2277, Jul. 2025.

[18] A. Baksi, S. Sarkar, A. Siddhanti, R. Anand, and A. Chattopadhyay, "Differential fault location identification by machine learning," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 17–24, 2021.

[19] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024," in *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, ser. Department of Computer Science Report Series B, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2024-1. University of Helsinki, 2024, pp. 8–10.

[20] F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, no. 5, pp. 183–197, 1991.

[21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds. ACM, 2016, pp. 785–794.