Diversity Algorithms for Laser Fault Injection

Marina Krček[1(✉)] and Thomas Ordas[2]

[1] Delft University of Technology, Delft, The Netherlands
m.krcek@tudelft.nl
[2] STMicroelectronics, Grenoble, France

Abstract. Before third-party evaluation and certification, manufacturers often conduct internal security evaluations on secure hardware devices, including fault injection (FI). Within this process, FI aims to identify parameter combinations that reveal device vulnerabilities. The impracticality of conducting an exhaustive search over FI parameters has prompted the development of advanced and guided algorithms. However, these proposed methods often focus on a specific, critical region, which is beneficial for attack scenarios requiring a single optimal FI parameter combination.

In this work, we introduce two novel metrics that align better with the goal of identifying multiple optima. These metrics consider the number of unique vulnerable locations and clusters (regions). Furthermore, we present two methods promoting diversity in tested parameter combinations - Grid Memetic Algorithm (GridMA) and Evolution Strategy (ES). Our findings reveal that these diversity methods, though identifying fewer vulnerabilities overall than the Memetic Algorithm (MA), still outperform Random Search (RS), identifying at least ≈8× more vulnerabilities. Using our novel metrics, we observe that the number of distinct vulnerable locations is similar across all three evolutionary algorithms, with ≈30% increase over RS. Importantly, ES and GridMA prove superior in discovering multiple vulnerable regions, with ES identifying ≈55% more clusters than the worst-performing MA.

Keywords: Laser Fault Injection · Parameter Search · Evolutionary Algorithms · Diversity Algorithms · Multiple Optima

1 Introduction

Small embedded devices frequently employ cryptographic algorithms to provide security. According to Kerckhoff's principle, it is expected that security is intact when the secret key is unknown, even if all the other information about the cryptographic system is public. Consequently, these algorithms are often mathematically secure, rendering brute-force attacks impractical. Regardless, implementation attacks, such as side-channel attacks (SCA) and fault injection (FI) attacks, can potentially lead to a successful security breach of such cryptographic systems. Side-channel attacks are passive, with the attacker measuring

M. Andreoni (Ed.): ACNS 2024 Workshops, LNCS 14586, pp. 121–138, 2024.
https://doi.org/10.1007/978-3-031-61486-6_8

the time [12], power consumption [11], or other side-channel data emanating from the target device. Given a correlation between the processed data and measured side-channel information, the attacker can obtain secret information. On the other hand, fault injection attacks are active, where the attacker purposely interacts with the device, inducing errors during the execution of the underlying algorithm. Specifically, the attack can use external sources, such as electromagnetic radiation [18], lasers [26], temperature [9], and voltage glitching [10], to manipulate data in memory, skip instructions, or alter instructions themselves. These implementation attacks are commonly used in security evaluations and are consequently extensively investigated [1,23]. The objective is to establish an enhanced and automated evaluation process that surpasses the current standard in terms of efficiency. The new algorithms should excel in uncovering more potential vulnerabilities while making more efficient use of available resources or possibly even reducing the required resources.

We focus on laser fault injections (LFI), as introduced by Skorobogatov et al. [26]. The issue with laser injection (and other types of fault injections) comes from the injection parameters determined by equipment. With the laser, we have to define the location of the laser shot on the targeted hardware device (x and y coordinates), the distance from the microscope lens, which is commonly used with lasers, and, lastly, we also have the laser settings, such as laser *intensity*, *delay*, and *pulse width*. Additionally, lasers can have pulses that demand several more parameters to define. Another critical component of successful injections is the trigger on when to perform the injection. In security evaluation, the worst scenario is often considered, where it is assumed that we have open access to the targeted device, and the trigger can be placed at any point in the execution. Obviously, there are *many parameters* we should consider. Additionally, the *possible values and combinations* of those parameters increase to the extent that exhaustive search is not feasible for security evaluation or attack.

In the attack, the adversary aims to find the parameters that lead to exploitable fault injection effects. These desired effects also depend on the method for the attack, where some of the popular attacks are differential fault analysis (DFA) [3], statistical fault attack (SFA) [7], and statistical ineffective fault attacks (SIFA) [6]. Each attack can require different characteristics of the FI effects. Still, some commonly desired and possibly exploitable faults include causing the device to skip instructions or change values in memory [1]. This work does not address identifying exploitable faults but focuses on a scenario within the internal security evaluation. During the security evaluation, a target characterization is performed, striving to uncover all vulnerabilities that can later be categorized based on their level of critical exploitability. While executing an exhaustive search would ensure that all possible vulnerabilities are observed, this process is not feasible as there are many products to be evaluated, and the search is impractical even for a single target. The aim is then adjusted to find as many vulnerabilities as possible within reasonable time and resources. Therefore, the FI parameter search is a process that should observe many vulnerabilities and provide high confidence that little to no vulnerabilities are overlooked. Instead

of an exhaustive search, the location on the target is often searched in a grid-like manner using the same laser settings. Defined laser settings could come from previous experience, which might be misleading if the target or the bench is entirely new [28]. If more options for laser settings are tested over the whole target area, this process becomes time-consuming, so alternatively, a random search is applied. However, both methods could omit parameter sets that lead to faults. In grid search, while the location is relatively thoroughly inspected, fixing the laser settings can contribute to overlooking many vulnerabilities. By including different laser settings, the search converges to an exhaustive search where the security analysts aim to reduce the search space based on previous knowledge, but the execution time for these algorithms could still be measured in weeks. On the other hand, random search is unreliable as different runs can lead to very different observations, which causes misrepresentation of the target's security level. Therefore, there is an incentive to improve the process of exploring the FI parameter search space more efficiently in an automated way.

Evolutionary algorithms (EAs) were explored for laser fault injection [14], voltage glitching [4,21,22], and electromagnetic fault injection [16,24] since laser fault injections are not the only type of injections suffering from the previously described issues. From the machine learning domain, hyperparameter optimization techniques [27], reinforcement learning [17] and Generative Adversarial Networks (GANs) [25] were also investigated. Additionally, the prediction ability of machine learning methods was explored for portability issues in the FI parameter search [13] and estimating the full target characterization [28].

The issue with aforementioned search algorithms, like evolutionary algorithms and tuning techniques, lies in their tendency to converge on a single vulnerable area as they are designed to obtain a single optimal solution. Previous works show a significant increase in the observed faults clustered in one sensitive region [16,21]. While that can be highly effective for attackers, we focus on security evaluation, where identifying multiple vulnerable regions is deemed a more favorable outcome. To better assess algorithm success in parameter search concerning the security evaluation goals, we propose to use the number of unique locations (x-y) and clusters with faults as additional metrics. We investigate the performance of several algorithms: random search, memetic algorithm, and two novel algorithms not explored before in the FI context - Grid Memetic Algorithm (GridMA) and Evolution Strategy (ES). The new algorithms are introduced as they promote the diversity of the parameter combinations. This diversity aims to achieve a more diverse search, uncovering distant vulnerabilities and identifying multiple optima instead of a single sensitive region. Experiments are performed with laser fault injections but should be suitable for other fault injection types.

Our main contributions are:

- We propose two methods that promote diversity among the tested FI parameter combinations. Promoted diversity ensures fewer vulnerabilities are overlooked, and multiple optima are uncovered during the search.
- We investigate other aspects of the algorithm performance for the FI parameter search, such as unique locations and clusters.

– The results show that the evolutionary algorithms find ≈30% more unique vulnerable locations than random search.
– The GridMA and ES algorithms found around 41% and 55% more vulnerable clusters than the worst-performing MA in this aspect, respectively. Thus, the diversity algorithms help determine more vulnerable regions.

2 Preliminaries

2.1 Random Search (RS)

Random Search (RS) is a widely used optimization method when exhaustive search is impractical. In the context of FI parameter search, it explores a predefined search space by randomly selecting parameter values and assessing their performance, with each value having an equal probability of selection. We ensure that only unique parameter combinations are considered, eliminating duplicates.

2.2 Memetic Algorithm (MA)

The memetic algorithm (MA) enhances the genetic algorithm (GA) by incorporating local search [19]. We apply the local search at the end of each GA iteration, constituting the first generation of memetic algorithms. This specific method has been successfully utilized in previous research [14,16]. The flow of the MA is depicted in Fig. 1. MA is a population-based optimization technique, operating on a set of individuals, each representing a potential solution to a specific optimization problem. The algorithm begins by generating an initial population using an *initialization method*, where a random sampling approach is often used. The algorithm then uses a problem-specific *fitness function* to evaluate each solution's performance. After evaluation, the genetic operators, including selection, crossover, and mutation, that drive the learning process are performed. The *selector operator* identifies solutions from the current population for reproduction. Usually, the best-performing solutions are favored as they are more likely to yield improved solutions. The selected solutions, called parent solutions, undergo the *crossover operator*, which combines their traits to create one or more offspring solutions. The new solutions (offspring) undergo the *mutation operator*, which introduces random variations into the new solutions. The mutation probability is commonly kept low, preventing the algorithm from acting like random sampling. The process generates a new population that continues into another algorithm iteration. To ensure the best-performing solutions are not lost, *elitism* is employed. Elitism explicitly preserves one or more of the best solutions from the current population for the next generation. Lastly, some solutions are selected for further improvement using the *local search*. In this work, we use the memetic algorithm introduced in [14], where the algorithm incorporates the Hooke-Jeeves as local search [8]. The algorithm runs until a predefined *termination condition* is satisfied. These termination conditions commonly consider the number of iterations or evaluations for ending the execution.

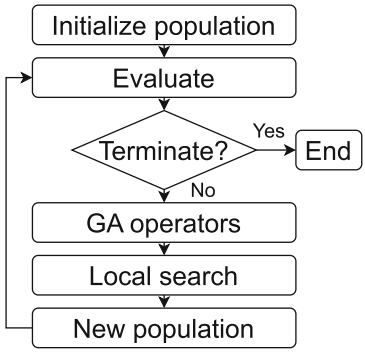

Fig. 1. Flow of the Memetic Algorithm.

2.3 Clustering Method

In the analysis, we use a clustering method called Mean Shift [5], an unsupervised clustering algorithm designed to identify clusters in a continuous distribution of data points. It is a centroid-based algorithm that updates candidates for centroids by computing the mean of points within a specific region (referred to as the *bandwidth*). Subsequently, these candidates are filtered in a post-processing stage to eliminate nearly identical centroids, forming the final set of centroids. We opted for the Mean Shift clustering algorithm because, unlike some other popular methods such as K-Means [15], it does not require users to predefine the number of clusters. While several different algorithms share this characteristic, we chose Mean Shift due to its simplicity as a centroid-based algorithm with only one hyperparameter. We believe it will provide satisfactory results for our analysis. However, we do not claim Mean Shift as the superior clustering algorithm. Note that the considered algorithms were those provided by a Python package called *scikit-learn* [20] to enable quick implementation and usage, as clustering is not the main topic of this work.

3 Related Work

Carpi et al. [4] investigated various search strategies for voltage glitch parameters, specifically the glitch shape and timing, for a successful FI attack. They explored glitch voltage and length with Monte Carlo (random), FastBoxing, Adaptive Zoom&Bound, and a Genetic Algorithm (GA). The genetic algorithm, without fine-tuning, required more measurements than the superior Adaptive Zoom& Bound algorithm. Picek et al. [22] extended the GA for the same voltage glitch parameters by employing a specialized crossover operator and selection mechanism, finding more faults than random search. Later, Picek et al. [21] introduced a memetic algorithm considering three voltage glitching parameters, namely glitch length, voltage, and offset. The authors mentioned the impracticality of specific algorithms used in previous work [4] due to increased dimensionality, excluding

them from comparison. Their objective was to efficiently identify favorable parameters within minimal time, seeking both successful parameter combinations and regions with consistent behavioral outcomes. Maldini et al. [16] increased the parameter search space by optimizing five parameters for Electromagnetic Fault Injection (EMFI) using a memetic algorithm. Krček et al. [14] demonstrated the effectiveness of a similar memetic algorithm for laser fault injection (LFI). These studies showcased the efficacy of the memetic algorithm across various FI types. Werner et al. [27] employed two hyperparameter optimization techniques from machine learning to enhance the parameter search for voltage glitching. They proposed a two-stage optimization strategy to reduce the dimensionality of the parameter space, similar to Carpi et al. [4]. Rais-Ali et al. [24] compared three different methods for EMFI, with the GA consistently outperforming the others in identifying areas of interest. The authors emphasize that, from an attacker's perspective, the goal is to identify a single exploitable fault using a specific FI parameter set. However, in the evaluation context, the objective is to ensure device security without excessive time investment, requiring a high-dimensional search to avoid overlooking potential parameter combinations. In [16,21,24], the authors evaluated performance based on the number of observed vulnerabilities, considering the notion of distinct regions and faults. We introduce diversity methods within the evolutionary approach to improve the algorithms' ability to discover more distinct regions with vulnerabilities. Related work on voltage glitching parameter search typically involved optimization of two or three parameters, while EMFI and LFI examined five. We optimize the same five parameters for LFI, performing a high-dimensional parameter search.

Wu et al. [28] focused on laser settings' impact on a specific building block. The authors noted that the complete characterization took over a week to execute. This underscores the need for faster and more efficient algorithms in the field. However, their work differs from ours, as our research investigates fault injection considering five distinct parameters, intending to uncover vulnerabilities across various building blocks within an integrated circuit (IC). Krček et al. [13] explored the transferability of results to different samples of the same target using decision tree models, falling outside the scope of this work for comparison as this work focuses on improving the parameter search without prior knowledge. Lastly, Moradi et al. [17] and Sedaghatbaf et al. [25] applied reinforcement learning and Generative Adversarial Networks (GANs), respectively, for efficiently exploring the fault injection space in simulations for adaptive cruise control systems in autonomous vehicles domain. These techniques could potentially extend to fault injection on hardware devices, aligning with our work.

Comparing all methods from the mentioned related work is time-consuming and complex. Hence, we leave this task to future work, recognizing the importance of unifying and evaluating these advanced methods to determine the state-of-the-art approach for parameter search in the scope of security evaluation, target characterization, and FI attacks. In this study, we compare new algorithms with random search and memetic algorithm, previously employed for high-dimensional parameter search on EMFI and LFI.

4 Diversity Algorithms

This section explains the newly proposed diversity algorithms that should help identify multiple vulnerable regions within the FI parameter search.

4.1 Grid Memetic Algorithm (GridMA)

We propose a novel approach, named the Grid Memetic Algorithm, that involves partitioning the target area for exploration into a grid and running the previously explained memetic algorithm within each grid region. The primary objective of the GridMA approach is to ensure attention (time and evaluations) of the algorithm to all target regions, mitigating the risk of overlooking vulnerabilities in specific (x-y) locations. For instance, if the target area is divided into a 3×3 grid, resulting in nine distinct regions, GridMA executes the MA independently in each region during a single run. As the search space size within each grid region decreased, we reduced the MA hyperparameters, specifically the population and elite sizes. GridMA represents a minor adaptation to the established MA. Nevertheless, it is a valuable initial step in evaluating diversity algorithms, precisely when we aim to obtain multiple vulnerable target regions.

4.2 Evolution Strategy (ES)

Evolution Strategy, like genetic algorithms, belongs to the class of evolutionary algorithms inspired by the principles of natural evolution [2]. The initial version of ES consisted of a single-parent solution from which one offspring was produced through a mutation-like procedure. The superior solution between the parent and offspring is preserved, and it resumes the same iterative process until it fulfills specific termination criteria. These termination conditions align with those described for MA in Sect. 2.2, consisting of attributes such as the number of iterations, evaluations, or acquiring a specified fitness level. Over time, ES has evolved, and in its more general form, it adopts the notation of $(\mu^{+}_{,}\lambda)$-ES. For instance, the original version can be denoted as $(1 + 1)$-ES, suggesting the presence of only one parent and one offspring in the process. Thus, μ represents the number of parents, and λ indicates the number of offspring. Additionally, a μ/ρ notation can be used for parents, where μ denotes existing parents, and ρ indicates the number of parents selected for producing offspring. Typically, ρ is less than or equal to μ, meaning that a subset of the best individuals is chosen for reproduction. In the notation, we use symbols + or , to indicate whether the solutions selected for the following generation are derived from both parents and offspring $(\mu + \lambda)$ or the parents (μ) are discarded, and only the offspring (λ), regardless of their fitness, continue to the next generation.

In the context of the described notation, we employ the $(\mu + \lambda)$-ES. The original, $(1 + 1)$-ES, using one single parent and an offspring, still converges to one optimal solution. Thus, to achieve diversity and reduce the risk of focusing on a single optimal solution, we set $\mu > 1$, effectively creating a population of size μ as employed in MA. The initial set of solutions is distributed across

different locations, and in each iteration, new offspring are generated from each of these parent solutions as we set $\lambda > 1$. Consequently, ES maintains a population of diverse solutions that evolve through iterations. This iterative process may lead to finding distinct solution clusters representing local optima. Thus, by employing ES, we expect to decrease the chances of overlooking vulnerabilities within the target area and observe more distinct solutions with optimal fitness.

5 Experimental Setup

5.1 Target

In collaboration with STMicroelectronics, we utilize their products for our experiments. Due to confidentiality reasons, we cannot disclose the details of the targets and the utilized laser bench. The target for our experiments is an IC constructed with 40nm technology. Since we use lasers for fault injection, mechanical thinning, a standard procedure, was part of the preparation for the experiments. During security evaluation, test programs can be deployed on the targeted products. The program running on our target device is a test program where data words are loaded into a register from the non-volatile memory (NVM). This test program can commonly be a part of the functionalities occurring within different algorithms on these devices. The target has no security countermeasures as the purpose here is not an attack breaking the device's security and countermeasures. Additionally, this provides the worst-case scenario. The implementation is done in the C programming language, and the pseudocode is displayed in Pseudocode 1.1. The pseudocode shows calls to three functions, where the first function is the `trigger_event`. The trigger event is a monitored event used to trigger the laser shot to inject faults at the desired time. In this case, the injection is aimed during the execution of the following function. That function loads the data from the NVM into a register. Lastly, we read the register and compare the value with the expected data. There is a fault if the register value has changed (fault class *fail*). On the other hand, if the injection was unsuccessful and the data is unmodified, equal to the expected value, then we give this response a fault class *pass*. Lastly, if there is no response from the device due to a time-out error or reset, we categorize this as a fault class *mute*. Note that the IC was reset to the initial state after each injection to provide a clean condition for each injection.

Pseudocode 1.1. Pseudocode of the program running on the target device.

```
...
trigger_event()
load_register() // injection here
read_register()
...
```

The FI parameter search is done on the following five parameters - x, y, *delay*, *laser pulse width*, and *intensity*. These parameters are commonly used

in literature and practice during a security evaluation [13,16]. We use a subset of the available values for each of the five parameters, defined according to the known layout and target cartography. Step sizes are defined based on the minimum possible step according to the utilized bench equipment, and the target area size includes different building blocks of the IC. The intervals are kept the same for all experiments. While we cannot share the parameter intervals as they are specific to the product and laser bench, we note that there are 370 772 710 possible combinations of the parameter values. The exhaustive search with the defined subset of possible values will take around 643 days if we consider that one laser shot takes ≈0.15 s.

While we focus on a single target in this study, the parameter search algorithms we introduce are versatile and applicable across various targets, bench configurations, and FI types. On average, the relative performance of these algorithms is expected to remain similar across mentioned scenarios. The obtained target responses guide these algorithms. Therefore, regardless of the selected target and setup, they strive to identify optimal solutions within the current setup and measured responses. The extent of improvements is limited by the finite number of detectable vulnerabilities associated with a specific target and bench setup.

5.2 Algorithm Details

In all our experiments, we specified a maximum limit of 6 000 evaluations of unique FI parameter combinations as a termination condition. Since we perform injection five times with the same parameter combination, we allow 30 000 laser shots. The number of evaluations is a practical upper bound on the algorithm's execution time. Previous work [13] indicates that a similar evaluation count leads to successful convergence, thus further justifying its selection.

In our approach, as we perform five measurements with the same parameter combination, we can acquire distinct fault class responses given the same parameters. Thus, there is a slight variation in our fault classification compared to related work. In our results, we present classes so that if there is even a single *fail* response within the measurements, we consider it a critical outcome and label it under *fail comb.* notation, signifying a *fail combination*. This approach aggregates all *fail* occurrences, disregarding the specific combinations that led to them. A more fine-grained categorization might be beneficial if we consider a specific attack, as parameter combinations with consistent outcomes might be more suitable for attacks. However, since we are in a security evaluation scenario, any occurrence of a *fail* response is considered critical. Other classes we include are those with *mute* response, a combination of *mute* and *pass* referred to as *mute_pass*, and lastly, there is a *pass* class where only *pass* class occurred in five measurements. The fitness function for all algorithms is calculated as

$$fitness = \frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F},$$

where f_P, f_M, and f_F correspond to the fitness values assigned to the fault classes *pass*, *mute*, and *fail*, respectively. Similarly, N_P, N_M, and N_F represent the frequency of these classes occurrences within the number of measurements for a specific parameter combination. The sum of N_P, N_M, and N_F constitutes the total number of measurements per parameter combination. This fitness function definition follows the previous works [13,14]. In our case, the fitness values for f_P, f_M, and f_F are 1, 2, 10, respectively. These values differ slightly from prior works, as we choose to create a more pronounced distinction in fitness value between each fault class. This design decision emphasizes the significance of any *fail* combination by assigning it a significantly higher fitness value. Before evaluating the entire population, we conduct a sorting operation using a greedy approach that considers the Manhattan distance between different locations of the FI parameter combinations within the population as described in [14].

MA Hyperparameters. We employ a population of size 100, with an *elite_size* of 10. The initialization method employs a random sampling strategy while preventing duplicates. For selection, we implement the roulette wheel method. We use uniform crossover and a uniform mutation with a mutation probability of 0.05. Lastly, the Hooke-Jeeves algorithm is applied for local search. Note that the hyperparameters in our experiments remain the same and have been taken based on information from previous work [14].

GridMA Hyperparameters. We dedicated additional experiments to exploring the hyperparameters of the GridMA algorithm, as it is a newly proposed method. We performed a minor hyperparameter search focused on the grid size, the population size, and the elite size of the MA. This section outlines the hyperparameters for the final version of the GridMA, whose results are shown in Sect. 6. The MA instances running in each grid have the same hyperparameters as described in Sect. 5.2, except for the mentioned hyperparameters that we were able to decrease due to the reduced scope of exploration within each grid region. Accordingly, the population size is set at 30 individuals, with an *elite_size* of 5. We divide the area in 4×4 grid, effectively conducting a total of 16 MA algorithms during a single run of GridMA. Since we maintain the total number of evaluations at 6 000 parameter combinations, each grid region is limited to evaluating only 375 FI parameter combinations.

ES Hyperparameters. Evolution Strategy is a new approach, so we explored several hyperparameters. Specifically, we assessed the algorithm's performance concerning the number of parents and offspring and the mutation probability. While we quickly obtained reported results, further fine-tuning may improve performance. The reported results are derived from ES employing 40 parent solutions and 5 offspring with the initial generation of parents established through random sampling. This algorithm uses the mutation operator as the sole source of introducing solution modifications, so a higher mutation probability will be

necessary. We observed that the mutation probability of 0.4, much higher than used with MA, produces the best results without converging to a purely random search approach. The mutation probability applies to each specific dimension within the parameter combinations. For example, with 40% probability, mutation will occur from uniformly distributed values of the given parameter. Uniform mutation encourages more substantial modification, allowing more 'jumps', particularly beneficial in the context of FI parameter search, as there are more non-vulnerable areas than vulnerable ones. In Sect. 6.4, we explore several modifications to the ES algorithm, including the Gaussian mutation approach, which is more commonly utilized to ensure a higher probability of local changes.

6 Experimental Results

This section presents results from applying the described algorithms to the same IC and laser bench. We aim to identify more locations with a *fail* outcome and uncover multiple vulnerable regions. To achieve this, we avoid restricting our search to a $2D$ location exploration, as it could overlook numerous parameter combinations due to the need for fixed laser settings. To effectively assess and identify algorithms that perform well for our objective, we compare them not only based on the observed $5D$ parameter combinations with a *fail* outcome but also on the number of unique locations ($2D$) and clusters. We executed each algorithm five times and reported the average results to ensure statistically relevant observations.

6.1 Number of Unique Parameter Combinations ($5D$)

We initially assess the number of unique parameter combinations with *fail* outcomes, where we use percentages from total tested combinations as in previous work for a more straightforward comparison. The results, shown in Table 1, reveal a comparable increase in *fail* responses between random search (RS) and memetic algorithm (MA) to the reported results in [13,14,16]. The MA identified $\approx$55.6$\times$ more FI parameter combinations leading to *fail* response than RS. In contrast, the two new methods, which provide greater diversity in the population of the FI parameters, obtained a lower percentage of *fail* responses compared to the MA. Compared to RS, we still find $\approx$12.4$\times$ more *fails* with GridMA, and $\approx$7.8$\times$ more with the ES. This decrease in the percentage arises from GridMA's exploration of areas where vulnerabilities might not exist. The ES, which relies solely on mutation, introduces more randomness than the MA, leading to a decrease in the number of identified vulnerabilities. Moreover, each parent evolved independently, resulting in more dispersed parameters and less exploitation of sensitive locations. These features should enhance our current objective but are shown to impact this metric negatively. Considering only the number of unique $5D$ parameter combinations tested, MA outperforms the other tested algorithms. However, the evolutionary approaches with diversity still offer advantages and should be preferred over random search.

Table 1. The average percentage of observed fault classes from all tested parameter combinations (6 000) using four different algorithms on the same IC. The average is calculated over five runs.

	RS	MA	GridMA	ES
fail comb.	0.61%	33.84%	7.54%	4.77%
mute	1.23%	3.23%	4.44%	4.53%
mute_pass	0.79%	1.21%	2.24%	2.83%
pass	97.36%	61.72%	85.79%	87.88%

6.2 Number of Unique Locations ($2D$)

In this work, we explore other metrics that could be used to evaluate the performance of different parameter search algorithms employed for fault injection. As we explain, MA converges commonly to one region sensitive to the utilized FI type and exploits it, leading to many observed FI parameter combinations with *fail* outcome. These parameter combinations come from a cluster of close x-y locations that can be detected visually (see Fig. 1b in [16] and Fig. 2 in [21]). In security evaluation, there should be a certain confidence that not many vulnerabilities are missed during the assessment of the IC. Also, we aim to find multiple regions with vulnerabilities, so we explore algorithms that promote diversity as it should help produce vulnerabilities distant in the utilized $5D$ space. More importantly, we want distant solutions when looking at the observed vulnerabilities' location (x-y). Thus, in Table 2, we report the number of unique parameter combinations with different fault classes and the number of unique locations per fault class from those parameter combinations. The table has two columns per algorithm, with the first showing the numbers from all the tested parameter combinations and the second showing the number of unique x-y locations. We also calculate what we refer to as location coverage, dividing the number of unique locations ($2D$) by the number of total tested unique $5D$ parameter combinations. This number shows the ratio of covered area within the tested parameter combinations. The numbers are rather small if we look at the absolute possible locations instead of relative to the tested parameters. To put it into perspective, from all possible combinations (≈370 million), we only test 0.00162% with 6 000 combinations. Unique tested locations from all possible locations ($2D$) per algorithm are 4.27%, 1.67%, 2.02%, and 3.07% for RS, MA, GridMA, and ES, respectively. We see an increase in the absolute location coverage between different evolutionary approaches, but RS has the best result. In the table, we report the relative location coverage as it provides an easier comparison. The relation between the algorithms is the same when we compare the absolute and relative location coverage. The results show that RS has the best coverage with 97.95% as the algorithm has no guidance. The worst location coverage is with MA (38.24%), supporting the motivation for this work. GridMA and ES improve coverage with 46.36% for GridMA and 70.29% for ES. Location coverage can serve as a measure of the algorithm's confidence in not overlooking vulnerable areas. Comparing the

unique locations with *fail* response between MA, GridMA, and ES, we see that the algorithms find a similar number of unique locations with *fail* - around 48, which is around 30% more than with RS (36.4). While similar in the number of unique locations with *fail* outcome, GridMA found the most unique locations on average with higher location coverage than MA. This improvement over MA is not as significant as the difference in performance between the evolutionary approaches and RS. Still, it shows the potential of diversity algorithms for security evaluation as they provide better coverage and thus confidence in identified vulnerabilities while delivering a similar improvement over RS in the number of unique, vulnerable locations.

Table 2. The average number of unique parameter combinations and x-y locations per fault class, and in total for all four algorithms. The average is calculated over five runs.

	RS		MA		GridMA		ES	
Nb. comb. | Nb. loc.	6000	5877.2	6000	2294.6	6000	2781.4	6000	4217.4
Location coverage	0.9795		0.3824		0.4636		0.7029	
fail comb.	37	36.4	2030.2	48.4	452	**49.2**	285.6	47
mute	74	73	194	60.4	266.2	70.4	271.6	93.2
mute_pass	47.4	47	72.8	45.8	134.6	61.4	169.8	67.6
pass	5841.6	5723.4	3703	2218.8	5147.2	2704.8	5273	4088.6

6.3 Number of Location Clusters

Finding distant, vulnerable locations is considered more valuable as the smaller regions could further be explored with an exhaustive search on a significantly reduced search space [24]. Thus, we compare the algorithms based on the number of observed location clusters with a specific fault class. We calculate the number of clusters using the Mean Shift clustering algorithm. The bandwidth hyperparameter for the Mean Shift algorithm defines the window/region from which the mean is calculated. We executed the clustering with different bandwidth values, precisely $0.1, 0.2, 0.3, 0.4$. With a bandwidth of 0.3, the region was large enough to categorize all the x-y points as one cluster for all fault classes. Table 3 shows the number of clusters averaged over five runs with bandwidth set to 0.1. Using the same bandwidth ensures the number of clusters is comparable as the same window size is considered. Note that if the number of clusters is more significant, the vulnerabilities are observed in more distant and distinct locations on the target, which is the desired objective. The results show that GridMA finds the most clusters with *fail* outcome, implying that the observed locations are more distant than other algorithms. GridMA and ES obtain a similar number of clusters on average, closely followed by RS. MA, on the other hand, clearly shows a smaller number of clusters observed. These results further emphasize the benefits of diversity methods for security evaluation and finding multiple regions

Table 3. The number of clusters based on Mean Shift clustering algorithm over the unique *x-y* locations per fault class. The bandwidth size is 0.1. The number of clusters is averaged over five runs.

	RS	MA	GridMA	ES
fail comb.	7.8	5.8	**8.2**	8
mute	11.4	8.6	9.6	10.6
mute_pass	9.6	8.4	10.8	10.2
pass	41.8	37	34.2	33.8

sensitive to the utilized FI type. We checked the clustering model's predictions visually for several cases to ensure that classified clusters are meaningful. While some more distinct locations were still clustered together using this bandwidth, the predicted clusters seemed reasonable. Moreover, we use the same bandwidth to ensure comparable results, as relative correlation is essential.

6.4 Further Exploring the Evolution Strategy Algorithm

The results show that evolutionary algorithms perform better than random search when considering the number of unique FI parameter combinations and unique locations with *fail* response. Considering the number of clusters, RS was better than MA, but the diversity algorithms were better overall. Thus, while the performance was not significantly improved using the diversity algorithms considering these metrics, the observed minor improvements show promising results. Therefore, we deem it necessary to explore these algorithms more within the scope of future work. In this section, we explore several ES versions to obtain enhanced performance.

We test the ES algorithm with a more common Gaussian mutation, which uses the Gaussian distribution to set the probabilities of each of the parameter values getting selected. The mutation probability will then be used as a standard deviation σ parameter, while the parameter's current value will be the mean μ. This mutation makes local changes more likely, while the more distant significant changes have a low probability of occurring, but not zero. We refer to this version of ES as *ES gauss*. Another modification we test is the initialization method, where we use a grid approach to set the parents of ES in distinct regions over the target area, considering only the location parameters. This way, the location parameters within the initial population are well-distributed, and the evolution should have a better chance of observing more distant and distinct regions with *fail* response. This version of ES is named *ES grid*. We then combine both modifications into a third version of ES referred to as *ES grid gauss*. Lastly, we execute a GridES, similar to GridMA, where we run ES within each grid cell over the target area. The grid is split in the same manner as for GridMA. We ran the GridES with the *ES grid* version as it was the best considering the number of clusters, and it performed similarly to the best ES versions

Table 4. The average percentage of observed fault classes from all tested parameter combinations (6 000) using five different versions of ES algorithm on the same IC. The average is calculated over three runs.

	ES gauss	ES grid	ES grid gauss	GridES grid
fail comb.	0.61%	4.19%	0.82%	0.82%
mute	1.83%	6.31%	2.02%	1.68%
mute_pass	1.17%	2.85%	1.08%	0.94%
pass	96.39%	86.66%	96.08%	96.57%

considering the other two metrics. Note that the reported results from the new ES versions are mean values from three runs, while the previous experiments ran five times. From the results in Table 4, considering **the number of unique FI parameter combinations** with *fail* response, the initial ES version performs the best on average, followed by the *ES grid* version. The versions with Gaussian mutation perform more closely to the results observed with random search. However, applying grid initialization for the version with Gaussian mutation did help increase the number of observed vulnerabilities. Still, using uniform mutation proved better within these experiments. Similar to *ES gauss* and *ES grid gauss*, GridES obtained a similar number of faults as RS. Considering **the number of unique locations** with *fail* response, versions *ES grid* and *ES grid gauss* were better than the initial ES version, as seen in Table 5. On average, the number of unique locations is now closer to the best GridMA algorithm, and it remains in the scope of previously observed improvements over RS using any of the evolutionary approaches. Considering this metric, *ES gauss* and GridES perform similarly to RS. Gaussian mutation increases the location coverage to the same level as RS, as evident from the results with the *ES gauss* and *ES grid gauss*. Finally, we consider **the number of clusters** with *fail* response in Table 6, and the *ES grid* version found 9 clusters on average, while the GridMA, had 8.2 clusters which was the previous best result. We also note that all the ES versions observed more clusters than the initial version, and GridES had the same number on average. Thus, we improved the initial ES, with the crucial modification being the grid initialization. Gaussian mutation provided more randomness in the location parameters, which led to enhanced location coverage but less vulnerable parameter combinations and locations. However, interestingly, all ES versions provided more clusters than RS and MA, demonstrating the potential of diversity methods.

Table 5. The number of unique parameter combinations and *x-y* locations per fault class, and in total for all four algorithms. The average is calculated over three runs.

	ES gauss		ES grid		ES grid gauss		GridES grid	
Nb. comb. | Nb. loc.	6000	5869	6000	4235.5	6000	5857.3	6000	4322
Location coverage	0.9782		0.7059		0.9762		0.7203	
fail comb.	36.3	36	251	48	49	**48.7**	48.7	35.7
mute	110	107.3	378.5	124.5	121.3	121	101	78
mute_pass	70	69.7	171	86.5	64.7	64.7	56.3	47
pass	5783.7	5666	5199.5	4075	5765	5632	5794	4216.3

Table 6. The number of clusters based on the Mean Shift clustering algorithm over the unique *x-y* locations per fault class. The bandwidth size is 0.1. The number of clusters is averaged over three runs.

	ES gauss	ES grid	ES grid gauss	GridES grid
fail comb.	8.3	**9**	8.3	8
mute	11.3	10	9.6	10.3
mute_pass	9	12	10	9.3
pass	32.3	31.5	27.6	40

7 Conclusions and Future Work

Previous works show the benefits of algorithms such as memetic algorithm in finding more FI parameter combinations with vulnerabilities compared to commonly used random search. However, the observed results commonly come from a single sensitive region, and during security evaluation, we do not want to neglect possibly exploitable vulnerabilities. Thus, we propose diversity algorithms that promote diversity in the population of evolutionary algorithms and test the GridMA and Evolution Strategy and its variations. While we evaluate algorithms considering the number of unique FI parameter combinations as in related work, we additionally assess algorithm success based on the number of unique locations (*x-y*) and clusters with faults as two additional metrics that better align with the objective of finding multiple vulnerable regions. MA performs best only when the number of faults is concerned. However, GridMA and ES with grid initialization and Gaussian mutation (*ES grid gauss*) found more unique locations with faults. Nonetheless, all evolutionary algorithms, including MA, found around 30% more unique locations with *fail* responses than RS, performing similarly. Considering the number of clusters, MA performed the worst, while ES with grid initialization (*ES grid*) had the most clusters, followed by the GridMA algorithm and other ES versions. This work shows that the diversity approach helps find more distant locations with the desired outcome. However, the improvements are less

significant than the difference between evolutionary algorithms and RS regarding the number of FI parameter combinations. Thus, while this work showcases the potential enhancement using the diversity approaches, future work could consider (μ, λ)-ES and more advanced diversity algorithms to provide more significant improvements.

References

1. Barenghi, A., Breveglieri, L., Koren, I., Naccache, D.: Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc. IEEE **100**(11), 3056–3076 (2012)
2. Beyer, H.G., Schwefel, H.P.: Evolution strategies-a comprehensive introduction. Nat. Comput. **1**, 3–52 (2002)
3. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems (1997)
4. Carpi, R.B., Picek, S., Batina, L., Menarini, F., Jakobovic, D., Golub, M.: Glitch it if you can: parameter search strategies for successful fault injection. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 236–252. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_16
5. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Mach. Intell. **24**(5), 603–619 (2002)
6. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018). https://doi.org/10.13154/tches.v2018.i3.547-572. https://tches.iacr.org/index.php/TCHES/article/view/7286
7. Fuhr, T., Jaulmes, E., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2013, USA, pp. 108–118. IEEE Computer Society (2013). https://doi.org/10.1109/FDTC.2013.18
8. Hooke, R., Jeeves, T.A.: "Direct search" solution of numerical and statistical problems. J. ACM **8**, 212–229 (1961)
9. Hutter, M., Schmidt, J.-M.: The temperature side channel and heating fault attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 219–235. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_15
10. Kim, C.H., Quisquater, J.-J.: Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 215–228. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72354-7_18
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
12. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
13. Krček, M., Ordas, T., Fronte, D., Picek, S.: The more you know: improving laser fault injection with prior knowledge. In: 2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), pp. 18–29. IEEE (2022)
14. Krček, M., Fronte, D., Picek, S.: On the importance of initial solutions selection in fault injection. In: 2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), pp. 1–12 (2021). https://doi.org/10.1109/FDTC53659.2021.00011

15. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, vol. 1, pp. 281–297 (1967)
16. Maldini, A., Samwel, N., Picek, S., Batina, L.: Genetic algorithm-based electromagnetic fault injection. In: 2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 35–42. IEEE (2018)
17. Moradi, M., Oakes, B.J., Saraoglu, M., Morozov, A., Janschek, K., Denil, J.: Exploring fault parameter space using reinforcement learning-based fault injection. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 102–109. IEEE (2020)
18. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller. In: 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 77–88. IEEE (2013)
19. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Caltech Concurrent Computation Program (2000)
20. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
21. Picek, S., Batina, L., Buzing, P., Jakobovic, D.: Fault injection with a new flavor: memetic algorithms make a difference. In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 159–173. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_11
22. Picek, S., Batina, L., Jakobović, D., Carpi, R.B.: Evolving genetic algorithms for fault injection attacks. In: 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1106–1111. IEEE (2014)
23. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: SoK: deep learning-based physical side-channel analysis. ACM Comput. Surv. **55**(11), 1–35 (2023)
24. Rais-Ali, I., Bouvet, A., Guilley, S.: Quantifying the speed-up offered by genetic algorithms during fault injection cartographies. In: 2022 Workshop on Fault Detection and Tolerance in Cryptography (FDTC), pp. 61–72. IEEE (2022)
25. Sedaghatbaf, A., Moradi, M., Almasizadeh, J., Sangchoolie, B., Van Acker, B., Denil, J.: DELFASE: a deep learning method for fault space exploration. In: 2022 18th European Dependable Computing Conference (EDCC), pp. 57–64. IEEE (2022)
26. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_2
27. Werner, V., Maingault, L., Potet, M.L.: Fast calibration of fault injection equipment with hyperparameter optimization techniques. In: Grosso, V., Pöppelmann, T. (eds.) CARDIS 2021. LNCS, vol. 13173, pp. 121–138. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97348-3_7
28. Wu, L., Ribera, G., Beringuier-Boher, N., Picek, S.: A fast characterization method for semi-invasive fault injection attacks. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 146–170. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40186-3_8