# **Short paper:** Diversity Methods for Laser Fault Injection to Improve Location Coverage

Marina Krček
Delft University of Technology
Delft, The Netherlands
m.krcek@tudelft.nl

Thomas Ordas
STMicroelectronics
Rousset, France
thomas.ordas@st.com

Stjepan Picek
Radboud University
Nijmegen, The Netherlands
stjepan.picek@ru.nl

***Abstract*—In the first step of fault injection attacks, it is necessary to perform fault injections for target characterization to improve the chances of finding vulnerabilities that can be exploited in the second step. The second step is the attack, where the vulnerabilities are used to break the security. This work considers the parameter search on the laser fault injection parameters. While this work can also be adjusted to find exploitable faults, the objective here is to find as many faults as possible on the target device. The goal comes from a security evaluation perspective to perform a successful target characterization. Previous works propose several methods, such as the memetic algorithm or hyperparameter tuning techniques. However, we notice a problem concerning the convergence of such methods to one specific target region, which is beneficial for an attack where one parameter combination could be enough. Indeed, these search algorithms lead to many observed vulnerabilities, but most seem to come from the same area on the target, which could mean some crucial vulnerabilities are missed. In this work, we propose considering the location coverage of the algorithms and offer two methods promoting diversity in tested parameter combinations to increase it while still finding many faults. We compare the grid memetic algorithm and evolution strategies to the performance of the memetic algorithm and random search. Our results show a benefit from introducing diversity to increase location coverage, but the overall number of vulnerabilities is decreased compared to the memetic algorithm. However, the number of unique locations with vulnerabilities is similar between the three evolutionary algorithms, with evolution strategies providing the most distant locations.**

***Keywords*-Laser Fault Injection, Parameter Search, Evolutionary Algorithms, Diversity, Location Coverage**

## I. Introduction

Small embedded devices often run cryptographic algorithms for security purposes. From Kerckhoff's principle, it is expected that without the secret key, the security is intact, even knowing all the other information about the cryptographic system. Thus, these algorithms are often mathematically secure, and brute force attack is not feasible. However, this is not the case. For example, implementation attacks, such as side-channel attacks (SCA) and fault injection (FI) attacks, can result in a successful security breach. Side-channel attacks are passive, with the attacker measuring the time [10], power consumption [9], or some other side-channel information 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, such that the attacker purposely interacts with the device to cause it to make errors during the execution of the underlining algorithm. Specifically, the attack can use external sources, such as electromagnetic radiation [15], lasers [20], temperature [7], and voltage glitching [8], to change data in the memory, skip instructions, or change the instructions.

We focus on laser fault injections (LFI), as introduced by Skorobogatov et al. [20]. 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 and the trigger can be placed at any point in the execution. As one can see, there are *plenty of 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 and 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) [2], statistical fault attack (SFA) [5], and statistical ineffective fault attacks (SIFA) [4]. 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. This work does not consider exploitable faults but focuses on a scenario where target characterization is needed, as in security evaluation, to improve the FI parameter search by finding more vulnerabilities and improving 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. If more options for laser settings are tested over the whole target area, this process becomes time-consuming, so a random search is applied as an alternative. However, both approaches could *miss interesting parameter sets* that lead to faults. In grid search, while the location is relatively thoroughly inspected, fixing the laser settings can contribute to overlooking some vulnerabilities. On the other hand, random search is unreliable since it can find many vulnerabilities or show a relatively low number of faults, which might not be an accurate representation. Therefore, we see an incentive to improve this process to search the FI parameter search space more efficiently and in an automated way.

Evolutionary algorithms (EAs) were explored for laser fault injection [12], voltage glitching [3], [18], [17], and electromagnetic fault injection [13], [19] 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 [21] and reinforcement learning [14] were also investigated. Additionally, the prediction ability of machine learning methods was explored for portability issues in the FI parameter search [11] and estimating the full target characterization [22].

This work indicates a new perspective somewhat disregarded in related works and proposes a way to address the issue. The issue with aforementioned search algorithms, such as genetic and memetic algorithms, is convergence to one area explored more since they lead to *one* optimal solution. Despite this being a suitable method for the attacker's goal, we consider target characterization in security evaluation. We want to identify all possible faults, but considering an exhaustive search is not feasible, we investigate guided techniques to identify most existing vulnerabilities. Previous works show a significant increase in the observed faults clustered in one sensitive region [3], [13]. Thus, we propose including the location ($x$-$y$) coverage as an additional factor to measure the success of the algorithms for parameter search. We investigate the performance of several algorithms: random search, memetic algorithm, and two novel algorithms not explored before in the FI setting. The new algorithms are Grid Memetic Algorithm (GridMA) and Evolution Strategies (ES). The new algorithms are considered as they promote the diversity of the parameter combinations. By promoting diversity during execution, we aim to achieve better location coverage and find more distant vulnerabilities. Improved location coverage provides more confidence that no vulnerable area is overlooked. Experiments are performed with laser fault injections but should apply to other fault injection types.

Our main contributions are:

- We raise an issue with the current methods proposed for fault injection parameter search and investigate location coverage as an additional metric of the algorithm performance for the FI parameter search.
- We provide two methods that promote diversity in the total parameter combination tested. Promoted diversity helps with confidence that fewer vulnerabilities are missed during the target characterization.
- We show that location coverage is increased using diversity methods. From 38% coverage with the memetic algorithm, we get 70% coverage using evolution strategies. Evolution strategies also find more distant fault locations on average. All three evolutionary algorithms find $\approx 30\%$ more unique locations with *fail* outcome than random search.

## II. Preliminaries

### A. Random Search (RS)

The random search (RS) approach randomly selects the parameters for testing, where each parameter value has the same probability of being selected. We also ensure that unique parameter combinations are considered, avoiding repetition.

### B. Memetic Algorithm (MA)

Memetic algorithm (MA) [16] is a genetic algorithm with local search. The invocation of the local search can be done in different places. A genetic algorithm (GA) is a population-based algorithm inspired by biological evolution. The population is a set of individuals that represent solutions for specific optimization problems addressed with the algorithm. With a defined solution representation, the algorithm goes through its process, which starts with generating the initial population using an initialization procedure. Commonly, the initialization procedure is random sampling, which was also used in our case. The learning process uses GA operators - selection, crossover, and mutation. The selector operator selects specific solutions for the next operator. Usually, we choose the best solutions we want to reproduce as they could lead to even better solutions. For the selection, we need to have each solution's fitness value. The fitness value comes from evaluating the solution with a fitness function specific to the optimization problem. Thus, fitness is a relative measure to compare one solution to the rest of the current population or a different solution. Selected solutions are called parent solutions, which are then recombined into one or more solutions using the crossover operator. These created solutions are called offspring. The third GA operator is the mutation operator. As in biology, mutation introduces some random variations in the offspring solutions. Mutation can be done by modifying some parts of the solutions or changing the offspring with an entirely new randomly generated solution. The mutation probability is kept low to not converge to the random sampling method. One can implement each of the explained operators differently, but they all lead to creating a new population that continues to another generation/iteration of the algorithm. To make sure that we do not lose the best-observed solutions, a common mechanism to use is elitism. Elitism explicitly keeps one or more best solutions from the current population for the next population. The iteration ends with selecting some solutions for improvements using local search. In this work, we use the memetic algorithm from [12]. There, the local search is the Hooke-Jeeves algorithm, an optimization algorithm that does not require derivatives of the objective function [6].

### C. Grid Memetic Algorithm (GridMA)

We propose an algorithm that makes a grid over the target area we want to explore and then runs the previously explained MA in each grid region. We call this approach Grid Memetic Algorithm (GridMA). The idea is to directly force the algorithm to dedicate time and evaluations to all target regions in the same manner to decrease the chances that vulnerabilities in specific *x-y* locations are overlooked. For example, if the target area is divided into a $3 \times 3$ grid, resulting in nine independent regions, we run MA nine times for a single run of GridMA. Hyperparameters of the MA are then adjusted as the search space size becomes reduced, as we can use fewer iterations, smaller populations, etc. The algorithm is a minor modification to the existing MA, but it was a first step to test the performance where we directly force the algorithm to address all regions of the target.

### D. Evolution Strategies (ES)

Evolution Strategies (ES), as GAs, belong to evolutionary algorithms since the idea for the method also comes from evolution [1]. The first version of ES consisted of a parent solution from which we obtained one offspring using a procedure like mutation. Then, the better solution is kept, and the better solution continues to go through the same process until some condition is met. From this first concept, ES developed through time, and now, in a more general setting, we have $(\mu \overset{+}{,} \lambda)$-ES. With this notation, the first version is (1+1)-ES since only one parent and one offspring exist. Thus, $\mu$ is the number of parents, and $\lambda$ is the number of offspring. Additionally, we can have $\mu/\rho$ notation for parents, where we have $\mu$ parents, but if we use crossover, then $\rho$ individuals from that set of parents are taken for crossover. In the notation, we use $+$ or , to mark if we select solutions for the following generation from parents and offspring $(\mu + \lambda)$ or discard the parents $(\mu)$ and keep the offspring regardless of fitness. With the described notation, we use the $(\mu + \lambda)$-ES. In the case of a single parent and a single offspring, the convergence is again for one best solution. Thus, we diversify by setting a $\mu > 1$. This can be considered as the population size used in MA. The initial set is at different locations, and in each iteration, we generate new offspring from each of those parents. The described process can lead to distinct clusters with the local optima. Therefore, using ES, diverse solutions are kept in the population that evolves through iterations. With this, we also expect to decrease the chances of overlooking vulnerabilities from the target.

## III. Experimental Setup

### A. 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. Revealing this information may provide malicious attackers with what kind of bench they could use to attack the products. The target for our experiments is an integrated circuit (IC) constructed with 40nm technology. Since we use lasers for fault injection, mechanical thinning was part of the preparation for the experiments, which is a standard procedure. During security evaluation, test programs are 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). The target has no security countermeasures as the purpose is target characterization rather than attack breaking the device's security. The implementation is displayed in Pseudocode 1. The `trigger_event` function is a monitored event that is used to inject faults at the desired time - on loading a data word into a register (marked with a comment in Pseudocode 1). The implementation is in the C programming language. Three main functions are visible, where the first one is the trigger event for triggering the laser shot, and then the loading to the register is done. The injection is aimed during the execution of that method. 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, we categorize this as a fault class *mute*.

Pseudocode 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 [11], [13]. We use a subset of the available values for each of the five parameters, defined according to the known layout. 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 takes around 643 days if we consider that one laser shot takes $\approx 0.15$ seconds. Additionally, since we perform the laser shot several times with the same parameters, this would increase the necessary time to complete the exhaustive search. Thus, there is certainly a need for optimized search methods to make the security evaluation feasible for many products.

### B. Algorithm Details

For all algorithms, we set that the maximum number of FI parameter evaluations is 30 000, where we inject five times with the same parameter combination. Thus, we evaluate 6 000 unique parameter combinations. The number of evaluations is set as a limit for algorithm execution time, as in previous works, it was shown that approximately that many evaluations were used for successful convergence. We can obtain different fault class responses since we perform five measurements with the same parameter combination. We differ slightly in the fault classification from the related work. In our results, we display classes so that if there was even just one response with *fail*, we consider it critical and count under *fail comb.* notation meaning *fail combination*. We, therefore, disregard which exact combination occurred when there was a *fail* and count it all in one class. If we were considering a specific attack, these classes might be different, and we could change the setup. Since we are in a security evaluation scenario, any occurrence of a *fail* response is critical. Other classes we display are with 5/5 times *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.

Fitness function for all algorithms is calculated as $\frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F}$, where $f_P, f_M$, and $f_F$ represent the fitness values for fault classes *pass*, *mute*, and *fail*, respectively. $N_P, N_M$, and $N_F$ represent the number of times the *pass*, *mute*, and *fail* class occurred out of the number of measurements for a specific parameter set. The sum of $N_P, N_M$, and $N_F$ is the number of measurements per parameter combination. This method is used in previous works, e.g., [12], [11]. The values for $f_P, f_M$, and $f_F$ are 1, 2, 10, respectively. The values are slightly different as we make a larger difference between the fitness of each class, forcing any *fail* combination to have a larger fitness.

*1) MA Hyperparameters:* We use a population of size $100$, with an *elite_size* of 10. The initialization method is random sampling without allowing duplicates. The selection operator is the roulette wheel. We use uniform crossover and mutation with a mutation probability of $0.05$. Lastly, the local search is the Hooke-Jeeves algorithm. Before evaluating the whole population, we perform sorting using a greedy approach based on the Manhattan distance between different locations of the FI parameter combinations in the population. The termination condition is the number of evaluations. Defined hyperparameters stay the same in our experiments and are taken based on information from previous work [12].

*2) GridMA Hyperparameters:* The GridMA algorithm's hyperparameters were explored more than MA, as it is a new method. We performed quick tuning guided by experience until we reached a satisfying convergence. Here, we describe hyperparameters for the final version of the GridMA, whose results are shown in Section IV. The MA running in each grid has the same hyperparameters as described in III-B1, except for some hyperparameters that we were able to decrease since the areas explored are smaller. Thus, the population size is set to 30, with an $elite\_size$ of 5. We divide the area in $4 \times 4$ grid, running 16 MA algorithms in total for one run of GridMA. Since we keep the number of evaluations the same - 6 000 parameter combinations, each region can evaluate only 375 FI parameter combinations.

*3) ES Hyperparameters:* Evolution Strategies are a new approach, so we first explore different hyperparameters until we reach competitive results. However, no extensive tuning was performed, as we quickly got the reported performance. More tuning could lead to better results. The reported results come from ES with $40$ parent solutions and $5$ offspring. Initialization is random sampling, while the number of evaluations is the same and is the termination condition. As in MA and GridMA, when we evaluate the entire set of solutions, we first sort them using the same greedy approach based on the different locations' Manhattan distance. We do not have crossover in this algorithm, but there is a uniform mutation, where the mutation probability is $0.4$, much higher than with MA. Since there is no crossover, mutation is the only way to introduce changes. Therefore, we use more significant probability as the probability is for each specific dimension of the parameter combinations, where with $40\%$, we will perform the mutation from uniformly distributed possible values of the given parameter. While forcing small local changes around the parent is commonly used, there are more non-vulnerable areas than vulnerable ones. So, in the FI parameter search, we consider allowing more 'jumps' will benefit this domain. However, in future work, we plan to test a more innovative mutation operator, which starts with local changes but also allows for more significant 'jumps'.

## IV. Experimental Results

We run all the algorithms five times for more relevant observations and report the average numbers. First, we look at the percentage of observed *fail* responses as in previous work. The average percentages are shown in Table I. Compared to previous work [12], we see a similar increase in observed *fail* responses between random search (RS) and memetic algorithm (MA). We find more than $50\times$ more FI parameter combinations leading to *fail* response with MA. On the other hand, in the two new methods that provide more diversity in the population of the FI parameters, we observe less percentage of *fail* responses. Considering that we use the same number of evaluations, 6 000, we get fewer *fail* responses compared to the previous MA algorithm. Compared to random search, we still find $\approx 12\times$ more *fails* with GridMA, and $\approx 7.5\times$ more with the ES search. The decrease in the percentage comes from the fact that we force the GridMA to look even where there might be no *fail* outcomes, and for ES, since we use only mutation, the algorithm has more randomness than MA. However, as mentioned, we try to address an issue not previously discussed when evaluating the search algorithms for FI parameter search.

TABLE I: The average percentage of observed fault classes from all tested parameter combinations (6 000) using four different algorithms on the same IC. The mean 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% |

In security evaluation, there should be a certain confidence that not many vulnerabilities are missed during the assessment of the IC. That was the reason to explore algorithms that promote diversity in the population to produce vulnerabilities distant in the utilized 5D space. Most importantly, we want distant solutions when looking at the observed vulnerabilities' location (*x*-*y*). Thus, in Table II, 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. Again, these numbers are average on five runs with each algorithm. 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 by dividing the number of unique locations (2D) by the number of total tested unique 5D parameter combinations. The numbers of the location coverage are in the row with the name *Nb. loc./Nb. comb.*. This number shows how much area we cover within the tested parameter combinations. The numbers are quite small if we look at all possible locations directly. To put it into perspective, from all possible combinations ($\approx$ 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 still has the best result. In the table, we compare where the difference is more prominent, as we want to see how much diversity is there in an algorithm without forcing specifically location diversity instead of whole parameter combination diversity. The results show that RS has the best coverage with 97.95% as the algorithm has no guidance. We force unique 5D parameter combinations but not unique locations. The worst location coverage is with MA (38.24%), which was noticed and raised as an issue and motivation for this work. GridMA and ES improve coverage with 46.36% for GridMA and 70.29% for ES. Looking at the number of locations with *fail* response compared to the number of tested locations, we have 0.62%, 2.11%, 1.77%, and 1.11% *fail* locations for RS, MA, GridMA, and ES, respectively. Thus, using any of the evolutionary methods, we observe more locations with *fail* but no increase with diversity methods compared to MA. However, we get that only 2.38% of observed *fail* responses come from unique locations when using MA. On the contrary, RS has 98.38%. GridMA and ES improve from MA with 10.88% and 16.45% for GridMA and ES, respectively. We confirm the assumption that exploiting the same and nearby locations by testing different laser settings brings MA to find many vulnerable parameter combinations. Comparing the unique locations with *fail* response between MA, GridMA, and ES, we see that the algorithms find a similar number of unique spots with *fail* - around 48, which is around 30% more than with RS (36.4).

Finding distant vulnerable locations is considered more valuable, so we compare algorithms based on that information.

TABLE II: The number of unique parameter combinations and *x*-*y* locations per fault class, and in total for all four algorithms.

| | RS | | MA | | GridMA | | ES | |
|---|---|---|---|---|---|---|---|---|
| Nb. comb. \| Nb. loc. | 6000 | 5877.2 | 6000 | 2294.6 | 6000 | 2781.4 | 6000 | 4217.4 |
| Nb. loc./Nb. comb. | 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 |

We calculate the number of clusters based on the Manhattan distance between two subsequent points. First, we sort the 2D location points in the same manner for all algorithms, and the Manhattan distance is calculated between them. Since we sort the points, the points belong to the same cluster if the Manhattan distance is lower than a certain threshold. On the other hand, if the distance is larger than a threshold, a new cluster is formed. Table III shows the number of clusters averaged over five runs. Thus, if the number of clusters is more significant, the vulnerabilities are observed in more distant locations on the target, which is the desired objective. We count as a cluster when the Manhattan distance between two points is at least ten steps. We also tested with 100 steps distance, and while the number of clusters was smaller, the relative relation stayed the same. These distances were reasonable for our setup, but the threshold distance should be adjusted depending on the size of the area one is testing. The results show that ES

TABLE III: The number of clusters based on Manhattan distance between the unique *x*-*y* locations per fault class. The threshold distance to count as a cluster is ten steps. The number of collections is averaged over five runs.

| Locations | RS | MA | GridMA | ES |
|---|---|---|---|---|
| *fail comb.* | 28.4 | 24.6 | 28.2 | 34 |
| *mute* | 55.8 | 36.4 | 43.4 | 66.4 |
| *mute_pass* | 36.4 | 30.4 | 37.8 | 49.2 |
| *pass* | 3770.8 | 1545.8 | 1716 | 2543.8 |

finds the most clusters with *fail* outcome, implying that the observed locations are more distant than with other algorithms. Comparing the number of unique locations with the number of clusters, we have 78.02%, 50.83%, 57.32%, and 72.34% locations are at least ten steps distant clusters for RS, MA, GridMA, and ES, respectively.

To conclude, we compare the algorithms in seven different aspects. As is done before, we compare the number of parameter combinations with the *fail* response to the number of all tested parameter combinations. We also compare the number of locations with *fail* to the number of tested locations in percentage and average numbers. From these three cases, we can rank the algorithms so that MA performs best (1.3), followed by GridMA (1.6), ES (3), and RS (4). We compare the location coverage by calculating the percentage of unique locations divided by unique 5D parameter combinations. Then we look at the coverage of tested locations compared to all possible locations. Lastly, we compare the number of clusters and the percentage of points forming the clusters. In this case, the average rank over those four cases is 1.25 for RS, 1.75 for ES, GridMA with a rank of 3, and MA with a rank of 4.

## V. Conclusions and Future Work

Previous works show the benefits of algorithms such as memetic algorithm in finding vulnerabilities with FI parameter combinations compared to commonly used random search. These works consider the number of unique parameter combinations with *fail* responses for evaluation and comparison. This work shows that coverage of the search is also crucial information. Indeed, in security evaluation, we do not want to neglect possibly exploitable vulnerabilities. Thus, we propose diversity algorithms that promote variety in the population of evolutionary algorithms. One such algorithm is evolution strategies. Additionally, we reused the MA for the Grid Memetic Algorithm approach to force the location coverage improvement. We evaluate algorithms with seven scenarios, where three are directly concerned with the number of observed *fail* responses, and the other four measure coverage with different metrics. As in previous works, MA shows the best performance when only the number of faults is concerned, even when locations are compared. However, the difference between the other two evolutionary approaches is smaller when the location is considered. On the other hand, cases measuring coverage show that RS has the highest coverage, followed by ES, while MA shows the worst results. The best coverage comes from no guidance in the RS that would confine the search to some regions. However, our results show that despite having lower location coverage, all evolutionary algorithms still find around 30% more unique locations with *fail* responses than RS. Additionally, the ES algorithm had the most significant number of clusters on average, meaning that the locations observed with *fail* response are more distant than those found with other algorithms.

We raise a critical concern for FI parameter search improvements. Different algorithms can be performed depending on the objective of the parameter search, and this work provides insights into different algorithms for a different intent than before. In future work, we plan to define a metric that considers the number of faults and coverage for evaluating FI parameter search algorithms more systematically, mitigating the need to compare and rank based on several distinct aspects. Additionally, we will consider more advanced diversity algorithms that could provide better results on coverage and the number of vulnerabilities without limiting the approach to force diversity in locations rather than the entire parameter set.

## References

[1] Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural computing **1**, 3–52 (2002)

[2] Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems (1997)

[3] 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: International Conference on Smart Card Research and Advanced Applications. pp. 236–252. Springer (2013)

[4] 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, Issue 3**, 547–572 (2018). https://doi.org/10.13154/tches.v2018.i3.547-572, https://tches.iacr.org/index.php/TCHES/article/view/7286

[5] 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. p. 108–118. FDTC '13, IEEE Computer Society, USA (2013). https://doi.org/10.1109/FDTC.2013.18, https://doi.org/10.1109/FDTC.2013.18

[6] Hooke, R., Jeeves, T.A.: " direct search" solution of numerical and statistical problems. J. ACM **8**, 212–229 (1961)

[7] Hutter, M., Schmidt, J.M.: The temperature side channel and heating fault attacks. In: International Conference on Smart Card Research and Advanced Applications. pp. 219–235. Springer (2013)

[8] Kim, C.H., Quisquater, J.J.: Fault attacks for crt based rsa: New attacks, new results, and new countermeasures. In: IFIP International Workshop on Information Security Theory and Practices. pp. 215–228. Springer (2007)

[9] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999)

[10] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology. p. 104–113. CRYPTO '96, Springer-Verlag, Berlin, Heidelberg (1996)

[11] 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)

[12] 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

[13] 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)

[14] 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)

[15] 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)

[16] Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Caltech Concurrent Computation Program (10 2000)

[17] Picek, S., Batina, L., Buzing, P., Jakobovic, D.: Fault injection with a new flavor: Memetic algorithms make a difference. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 159–173. Springer (2015)

[18] 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)

[19] 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)

[20] Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: International workshop on cryptographic hardware and embedded systems. pp. 2–12. Springer (2002)

[21] Werner, V., Maingault, L., Potet, M.L.: Fast calibration of fault injection equipment with hyperparameter optimization techniques. In: International Conference on Smart Card Research and Advanced Applications. pp. 121–138. Springer (2021)

[22] Wu, L., Ribera, G., Beringuier-Boher, N., Picek, S.: A fast characterization method for semi-invasive fault injection attacks. In: Cryptographers' Track at the RSA Conference. pp. 146–170. Springer (2020)