Fault Injection with a new flavor: Memetic Algorithms make a difference

Stjepan Picek[1], Lejla Batina[2], Pieter Buzing[3] and Domagoj Jakobovic[1]

[1] University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia
{stjepan.picek, domagoj.jakobovic}@fer.hr
[2] Radboud University Nijmegen, The Netherlands
lejla@cs.ru.nl
[3] Riscure BV, The Netherlands,
Buzing@riscure.com

Abstract. During recent years we observe an arms race between new creative methods for inserting effective faults and designing new countermeasures against such threats. Yet, even analyses of an unprotected smart card pose a problem for an analyst assuming constraints in time (or consequently, in a feasible number of measurements). In this paper we present a new kind of algorithm capable of finding faults in the black box test scenario - memetic algorithm. This algorithm combines the strengths of the following three algorithms: genetic algorithm, tabu search and local search. Furthermore, the same algorithm can be used if the goal is simply a rapid characterization of the search space. We compare our algorithm with random search and exhaustive search approaches. Experimental results show that our memetic algorithm is substantially more successful in both, locating faults and characterizing search space, than the other known methods. In reaching both goals, our memetic algorithm uses less than 300 measurements.

Keywords: Fault analysis, Glitches, Smart cards, Memetic Algorithms

1 Introduction

Smart cards and other small pervasive devices such as RFID tags are used daily by billions of users for applications such as public transportation, Internet banking, online shopping, etc. The exposure to numerous threats, mainly coming from the adversary aiming at physical security, have led to this becoming one of the most actively researched topics by both academia and industry in the past two decades.

Anderson and Kuhn [1] put some doubt on the claimed tamper-resistance of smart cards almost two decades ago. This paper was shortly followed by the set of techniques for tampering with smart cards by Kömmerling and Kuhn [2].

In general, the techniques for tampering can be classified as *passive* or *active* [3]. In passive techniques some side-channel information is monitored while

the card is supposed to work "normally". An example of these passive techniques is the analysis of power consumption, as introduced by Kocher et al. [4] or electromagnetic radiation [5]. In the case of active techniques, the device is not only monitored but also external interferences affect the normal behavior of the device. An example is Fault Injection (FI) attack. These interferences, the so-called *glitches*, can be of different nature: optical (laser pulses) and electrical glitches (voltage, clock), temperature changes, electromagnetic (EM) radiation, etc. They are used to cause malfunctioning, resulting in some cases in secret key recovery. Fault injection techniques by glitching are typically *non-invasive* techniques, in the sense that the smart card is not physically modified (in contrast to other *invasive* techniques that require hardware modifications).

A fault injection attack is considered to be successful if after exposing the device under attack to a specially crafted external interference, the device shows an unexpected behavior, which can be exploited by an attacker (e.g. leaking of sensitive information, bypassing security checks, etc.). However, this external insertion of signals has to be precisely tuned for the fault injection to succeed. As an example, a complete characterization of a clock signal glitch requires from the security analyst to define more than 10 parameters (related to clock signal voltage levels, time offset of the glitch, etc.).

Finding the correct parameters for a successful FI can be considered as a search problem where one aims to find, within minimum time, the parameter configurations which result in a successful fault injection [6]. The search space, considering all possible combinations of the values of interest for the fault injection is typically too large to perform an exhaustive search.

Heuristic search algorithms can reduce the search time considerably. In this paper we investigate the feasibility of genetic algorithms (GAs) in this application domain. More specifically, we compare a standard GA with an enhanced GA called a memetic algorithm (MA), which adds local search iterations to the process. The motivation behind MA is the fact that GAs are generally good at *exploration* of the search space, but can often be improved in terms of the *exploitation* aspect.

Apart from introducing a GA framework to the FI domain we also make a distinction between the aim of finding as many glitches as possible on one hand, and characterizing the parameter space on the other (with characterization we consider identifying promising parameter regions). By finding these promising parameter regions and focusing the search on those values we increase the probability of the glitch candidates producing a successful glitch.

1.1 Related Work

The concept of fault analysis-based attacks is known in the research community for around twenty years. Boneh, DeMillo and Lipton published an attack on RSA where they exploit hardware faults for cryptanalysis [7, 8]. Kömmerling and Kuhn present an extensive overview of techniques for fault injection and other tampering techniques and give ideas on how to mitigate some of them [2]. The paper highlights the case of power supply (VCC) fault injection (referred

to as *glitch attacks*) and emphasizes those as the ones most useful in practice. Aumüller et al. performed one of the first practical works on fault analysis, in which they describe a real-life scenario of the impact of injecting glitches in the VCC and clock lines of an IC [9]. They also suggest some countermeasures applicable in this specific case. Approximately at the same time, Skorobogatov and Anderson introduce optical (laser) fault injection, where they describe injecting faults with a laser on a decapsulated IC [10]. This technique is still very successful nowadays for defeating the security of many protected devices, but it is out of scope for this work. Van Woudenberg et al. describe a real attack scenario for an Optical Fault Injection attack [11]. The practical problem of setting the parameters for fault injection is introduced in their work and the authors briefly discuss the lack of methodology to solve it as the main direction they rely on is based on heuristics. In addition, the paper gives a nice overview of all the practical issues that arise during a real execution of the FI attacks on actual hardware. Balasch et al. explore the effects of glitches injected in the clock line of an IC [12]. This work is very interesting for identifying various effects that a glitch can cause on real hardware in terms of defining all possible outcomes of a successful fault injection. However, it has to be noted that current smart cards usually run on an internal clock which makes this FI technique infeasible. The work of Boix Carpi et al. deals with a similar problem to ours but the authors take a different approach [6]. They use a self-adapting search algorithm that shows some potential when considering only two parameters, glitch shape and length. As a future direction they mention one could try genetic algorithms. Additionally, the same authors present preliminary work with genetic algorithms in [13]. As an extension to this work, we consider three parameters and take the analysis to the next level by unleashing the full power of evolutionary computation in combination with other search techniques.

1.2 Our Contribution

There are two main contributions in this paper. As far as we know, we are the first to use a hybrid (memetic) algorithm [14] to look for successful glitches. Our memetic algorithm combines techniques from genetic algorithms, local search and tabu search. The second contribution is that we use the same algorithm not only to find faults, but also for the characterization of the search space with a minimal number of measurements.

The remainder of this paper is organized as follows: in Section 2 we give our problem statement and relevant properties of the search space as well as smart card details. In Section 3 we present description of algorithms we use, and in Section 4 we give experimental results and a discussion. Finally, in Section 5 we offer conclusions and future work directions.

2 Preliminaries

In this section we start with a short introduction to the smart card used. Afterwards, we give information about possible verdict classes and search space parameters.

2.1 Smart Card Details

In all our experiments we use smart cards that are based on ATMega163+24C256 IC, realized in CMOS technology. Those cards do not deploy any side-channel or fault injection countermeasure. All processing on the card is performed in software, and cards are running on an external 1 MHz clock frequency.

For the experimental purposes, we attack a vulnerable PIN authentication mechanism. The PIN authentication mechanism is implemented as follows:

```
for (i = 0; i < 4; i++)
{
    if (pin[i] == input[i])
        ok_digits++;
}
if (ok_digits == 4)  //LOCATION FOR ATTACK
    respond_code(0x00, SW_NO_ERROR_msb, SW_NO_ERROR_lsb); //PIN IS CORRECT
else
    respond_code(0x00, 0x69, 0x85); //PIN IS WRONG
```

In the above code we want to glitch the target (smart card) while it is executing the second if-statement. However, we want to emphasize that our approach makes no assumptions on the software that runs on the smart card. First of all, we regard the target as a black box and only hypothesize that there exists a weakness in the implementation and that we can roughly estimate its location in time. Secondly, the most difficult part of finding good FI parameters are the electrical properties like glitch voltage and length. The right values for these two dimensions will make the target physically behave in an unspecified way, and most importantly, they will do so on any smart card of the same make (or production batch), regardless of the implemented software.

2.2 Verdict Classes and Boundaries

Fault injection testing equipment can output only verdict classes that correspond to successful measurements. There exist several possible classes for classifying a single measurement (i.e. attack attempt):

1. NORMAL: smart card behaves as expected and the glitch is ignored
2. RESET: smart card resets as a result of the glitch
3. MUTE: smart card stops all communication as a result of the glitch
4. INCONCLUSIVE: smart card responds in a way that cannot be classified in any other class

5. SUCCESS: smart card response is a specific, predetermined value that does not happen under normal operation

In the rest of this paper, we will consider RESET and MUTE classes as equivalent when interpreting the results. Additionally, when depicting graphs with measurement results, for each of classes we allocate a color. When the card responds NORMAL, we depict a green dot in the search space, for RESET/MUTE we depict a blue color, for INCONCLUSIVE yellow color and finally, for SUCCESS red color.

2.3 Search Space Parameters

There are multiple search space parameters that need to be set in the fault injection process. We informally divide those parameters into two groups. The first group consists of parameters that we influence with an external search space algorithm and are therefore of primary interest. The second group consists of parameters that we leave for fault injection framework to set randomly.

In the first group we include the following parameters: glitch length, glitch voltage and glitch offset. The glitch length refers to the time (ns) that the VCC line is perturbed. The glitch voltage is the number of miliVolts (mV) that is added to the VCC line. The glitch offset is the start time (ns) of the perturbation relative to the start of the clock cycle.

For example, suppose that the glitch length is 100 ns, the glitch voltage is -3 500 mV, the glitch offset is 250 ns, and the supplied VCC is 5 V. What will happen is that 250 ns after the start of the clock cycle the VCC line will be pulled down to 1.5 V for 100 ns, after which it will be restored to 5 V.

Those three parameters determine the *electrical* effect on the target: roughly speaking the product of glitch length and glitch voltage represents the amount of energy that is exerted on (or withheld from) the target. Since the current propagation within a clock cycle is not constant the offset timing is also important. We refer to the three parameters as the "shape" of the glitch. Note that this is a physical effect: if the glitch is too strong the target will "mute" or reset. On the other hand, if the glitch is too weak the target will not be disturbed, but if the glitch is of an appropriate shape the target will behave in an unspecified way. This is unrelated to the *logical* effect, which refers to the exact instruction that is being glitched.

The second parameter group covers the logical effect and it consists of the number of wait cycles and the number of glitch cycles. The wait cycles parameter refers to the number of clock cycles that are skipped before the glitch attack is performed, counting from the sending of the smart card command. The glitch cycles parameter specifies the number of successive clock cycles that are glitched.

For example, we could wait for 800 clock cycles and then apply the glitch in the next 5 cycles, assuming the relevant instruction is executed close to the foreseen time frame. The fact that the electrical effects can be observed independently of the logical effects allows the security analyst to work in two phases.

First he will find a glitch "shape" that triggers the target to behave in an unspecified way. In the second phase he can search for the right wait cycles and glitch cycles values while applying the right glitch. In this paper we disregard the logical parameters. We assume reasonable ranges for the wait/glitch cycles and select uniformly at random from these ranges.

A useful property of the glitch shape parameters is that they display locality. The glitch offset has only a small range of values that "work". On top of that, the glitch voltage and the glitch length shows a monotonic behavior: once a glitch voltage is strong enough to force a card reset, then bigger values will also force a reset. The same goes for the glitch length. In practice this means that there is a clear phase transition in the voltage/length dimensions between NORMAL results and RESET/MUTE results. Also, the class of SUCCESS results (when offset is also guessed correctly) is often located around this phase transition. Note however that the exact effect of a glitch is stochastic and a perfect separation of parameter regions is impossible.

3 Approach and Methods

Looking for spots that lead to the successful fault injection can be considered as an optimization problem where we want to find as much "good" spots as possible in the minimal amount of time. The complexity of the problem depends on the number of considered parameters. However, before trying to give an answer about appropriate methods, we offer an illustration of the difficulty of the problem.

In a realistic setting that we consider, the glitch voltage parameter ranges from -5 000 mV to -50 mV, with a minimal step of 50 mV. The second parameter, glitch length, ranges from 2 ns to 150 ns with a step of 2 ns. If we conduct experiments where we are interested in only those two parameters and do exhaustive search, we need 7 400 measurements. If we assume that each measurement lasts one second, this gives us total time of two hours. However, if we add just one more parameter, e.g. glitch offset that goes from 100 ns to 400 ns with the 1 ns step, then we have in total 2.2 million measurements. This equals to more than 600 hours of measurement time. Naturally, one also needs to take into account possible consequences for a smart card if it is tested for more than 600 hours and the fact there are several more parameters of interest not even mentioned in this calculation.

The first objective targets at finding as many successful parameter combinations as possible, without regarding their values and their relation to each other. The second objective, on the other hand, aims to map the parameter space into regions with the same behavior outcome of the smart card. It is expected that the parameter combinations that result in the same behavior form regions in the search space which are adjacent to regions with different target behavior. Our experiments show that the region boundaries cannot be described with linear functions, and it is exactly along the boundaries that the successful attacks could be performed. Therefore, the objective of the search space characterization

is to provide boundaries between different regions with as few measurements as possible.

When looking for faults we can expect that more attempts should be made with parameter values that resemble those that led to a fault, but from the other perspective, the analysis will use more measurements and will result in other regions less analyzed. On the other hand, when looking for a region of interest, we can expect that the algorithm will also find faults, but that behavior should not be specially rewarded.

3.1 Genetic Algorithm

Genetic Algorithms (GAs) belong to a subclass of evolutionary algorithms where the elements of the search space S are arrays of elementary types [15].

In our approach we need to change several parts of a standard GA in order to work with this specific problem setting. A 'standard' GA assigns fitness values to different points in the search space (individuals or potential solutions) and maintains a population of those, usually initialized randomly. A potential solution in this context represents the values of the three parameters in our search space. In each iteration (generation) it selects the better ones and eliminates the worse, combines different individuals to produce new ones (using the crossover operator) which replace the eliminated ones and randomly changes parts of new individuals (using mutation).

First, we need to map verdict classes to fitness values. Since the objective is to *maximize* the value of fitness function, we give higher values to verdict classes that are of a bigger importance. Observing that we are looking for parameters that behave differently from NORMAL behavior, for NORMAL class we give the smallest value of 1. RESET and MUTE classes we consider the same and we give them a value 2 since we expect to find faults in areas between NORMAL and RESET. Finally, for SUCCESS class we give a value 3. Since we are not able to define the INCONCLUSIVE class, we also assign it the same value as for the RESET/MUTE class.

Next, instead of a standard crossover operator we use the custom version - local crossover (LC). In this operator, the first crossover point (potential solution) is chosen randomly. The second point is chosen so the LC operator crosses two points that belong to *different* classes, and it generates a new offspring point between the parents. The position of the offspring point is chosen on the basis of the number of solutions in complete population that belong to the parents classes: a child is proportionally closer to the parent with the class that is *less represented*, i.e. the class with the smaller number of individuals in the population. Only in the case when the first parent belongs to SUCCESS class, this operator tries to find a second parent in the same class. A mutation is conducted by adding some random value to the parameters. We present a GA with aforementioned modifications as Algorithm 1.

Algorithm 1 Genetic Algorithm.

Require: crx_count = 0, mut_count = 0
  repeat
    select first parent
    if first parent of SUCCESS class **then**
      try to find matching second parent
    else
      try to find second parent of different class
    end if
    perform crossover (depending on parent classes)
    copy child to new generation
    crx_count = crx_count + 1
  until $p_c * N \neq$ crx_count
  repeat
    select random individuals for tournament
    copy best of tournament to new generation
    mut_count = mut_count + 1
  until $(1 - p_c) * N \neq$ mut_count
  perform mutation on new generation with probability p_m
  evaluate population

3.2 Tabu Search

Since there exist only a few different verdict classes (and consequently, only a limited number of different fitness values) it is expected that a number of same solutions will emerge that may be tested repeatedly. Since each such solution leads to a unnecessary measurement, we adopt a technique from Tabu Search (TS) optimization method. Tabu Search works by declaring certain solution candidates that have already been visited as *tabu* and therefore not to be visited again [15]. The advantage of using the TS method is twofold in our case: first, we lower the total number of measurements performed and second, when not revisiting already visited locations, the algorithm is less likely to get stuck in a local optimum. We implement Tabu Search by using a list which stores all the solutions that have been already measured and allocated fitness values. If a new solution is created that is on the list, it is not measured but discarded immediately. Note that we do not implement all TS functionalities, but only those related with keeping the tabu list.

3.3 Local Search

Local search (LS) is a metaheuristic method for solving computationally hard optimization problems [14]. Local search algorithms work on a single solution (instead of multiple solutions) and generally transcend only to neighbors of the current solution. It moves in the space of candidate solutions by applying local changes until it finds an optimal solution or the time bound is elapsed [14]. In our experiments we use one version of the divide-and-conquer algorithm where

each new solution is located in the middle of parent solutions (binary search). In order to behave in such a manner, we need to define what is the space of candidate solution, i.e. in what neighborhood it can operate. To this end, we need to define an appropriate distance metric.

Two solutions are neighbors if they are at the distance smaller than d. In this paper we experiment with Euclidean [16] and Manhattan [17] distance metrics. Since the search space parameters are of different magnitudes, we use a normalized search range of $[0, 1]$.

Euclidean distance between two points $\boldsymbol{a}$ and $\boldsymbol{b}$ in an n-dimensional space is equals $d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$ [16].

Manhattan distance between two points is the sum of absolute differences of their Cartesian coordinates and it equals $d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n} |a_i - b_i|$. [17].

In our experiments we use a local search algorithm after each GA generation. The local search works on all pairs of individuals that are closer than the distance d and it runs while the distance between solutions is larger than the resolution r. With the resolution parameter we control how precise the characterization of the search space should be. We note that it was necessary to add two parameters to control the local search algorithm. The distance parameter d ensures that only individuals that are closer than d can participate in local search. The resolution parameter r controls when the LS should stop operating on each pair of individuals. Pseudocode for the LS is given in Algorithm 2.

Algorithm 2 Local Search algorithm.

```
create pool with all individuals ind
for all ind in the pool do
  select ind_tmp from the pool
  d = distance (ind, ind_tmp)
  if d > resolution and d < distance and class (ind) != class (ind_tmp) then
    make pair
    remove individuals from pool
  end if
end for
for all pairs do
  d = distance (ind_1, ind_2)
  if d > resolution then
    create point in between points
    call evaluator
    replace parent from the same class as offspring
  else
    remove pair
  end if
end for
```

3.4 Memetic Algorithm

Memetic Algorithms (MAs) represent a synergy between evolutionary algorithms (or any other population-based algorithm) and local improvement algorithms [15]. Most MAs can be interpreted as search strategies in which a population of solutions cooperate and compete [14].

In our experiments, the memetic algorithm is a combination of three aforementioned algorithms: genetic algorithm, tabu search and local search. Each of those algorithms should lend its strength to obtain a new, synergistic one that is more powerful than any of them individually. Genetic algorithms give their strength when finding promising regions in search space. Local search improves the convergence speed when looking for SUCCESS points (or regions between two verdict classes) and tabu search reduces the number of measurements by avoiding duplicate measurements.

4 Experiments and Results

In this section we present details about our experimental setup and the parameters considered. Afterwards, we present our results and give a short discussion. **Common parameters** for all experiments are given in Table 1.

Table 1. Common parameters.

Parameter	Parameter Value
Tournament size	3
Population size	30
Stopping criterion	10 generations
Mutation rate	0.1
Glitch length	[2, 150] ns
Glitch voltage	[-5 000, -50] mV
Glitch offset	[100, 400] ns
Glitch cycles	random from [1, 10]
Wait cycles	random from [750, 850]

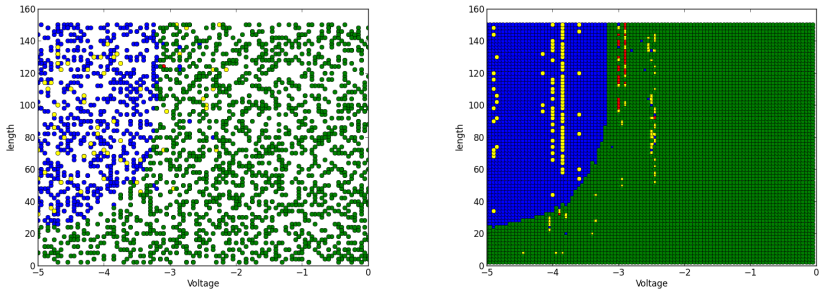
As it can be observed in Table 1 we use a small number of generations and a small population size since we are interested in a rapid characterization or finding faults. Indeed, if one has sufficient time at his disposal no method can outperform exhaustive search.

4.1 Experimental Results

When conducting experiments, we compare our results with random search and exhaustive search methods. Here we give the results for the two methods.

Random Search. In this method search space parameters are chosen uniformly at random. Figure 1(a) displays random search with 2 500 measurements.

Exhaustive Search. In order to check the full characterization of search space we also run an exhaustive search algorithm. Here, parameters of interest are glitch voltage, length and offset. Since there are too many possible solutions for any realistic exhaustive search we conduct exhaustive search for glitch length and voltage while other parameters are chosen uniformly at random. Figure 1(b) shows the results of 7 500 measurements.



(a) Random search, 2 500 measurements (b) Exhaustive search, 7 500 measurements

Fig. 1. Measurements for random and exhaustive search methods.

Next, we present results of our new algorithms separately for the case where the goal is to find as many faults as possible and for the case where the goal is the characterization of search space. After a short tuning phase we set distance d parameter to the value of 0.3 and resolution r parameter to the value of 0.1 since with those values we observe the best behavior. However, our experiments also show that these parameters are quite robust and small changes in values do not significantly change algorithm performance.

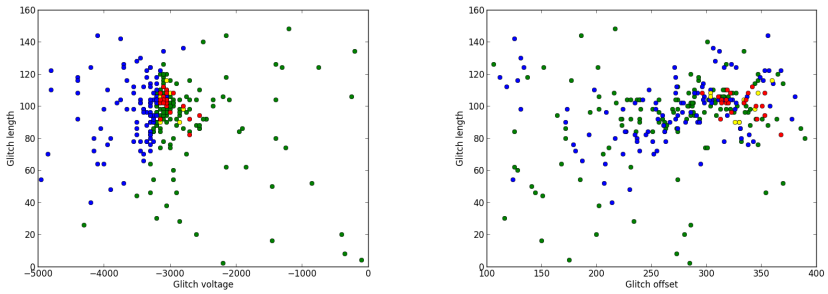
4.2 Finding Faults

When the goal is finding faults, we conduct several runs of different algorithm versions and then we present averaged values. Columns Normal, Reset and Success show average number of NORMAL, RESET/MUTE and SUCCESS measurements. In Table 2 we give the results for three different versions of our algorithm where we can see that GA+TS+LS algorithm with Euclidean distance metric finds the most SUCCESS points on average.

In Figures 2(a) and 2(b) we give an example of one run of GA+TS+LS algorithm with Euclidean distance and 250 measurements. In this experiment, with 250 measurements in total, we found 21 glitches which represents 8.5% of total measured points.

Table 2. Average results of experiments.

Algorithm	Normal (%)	Reset (%)	Success (%)
GA+TS	58.08	39.97	1.94
GA+TS+LS, Euclidean	55.29	41.87	2.84
GA+TS+LS, Manhattan	62.76	36.45	0.78



(a) Glitch voltage vs. glitch length (b) Glitch offset vs. glitch length

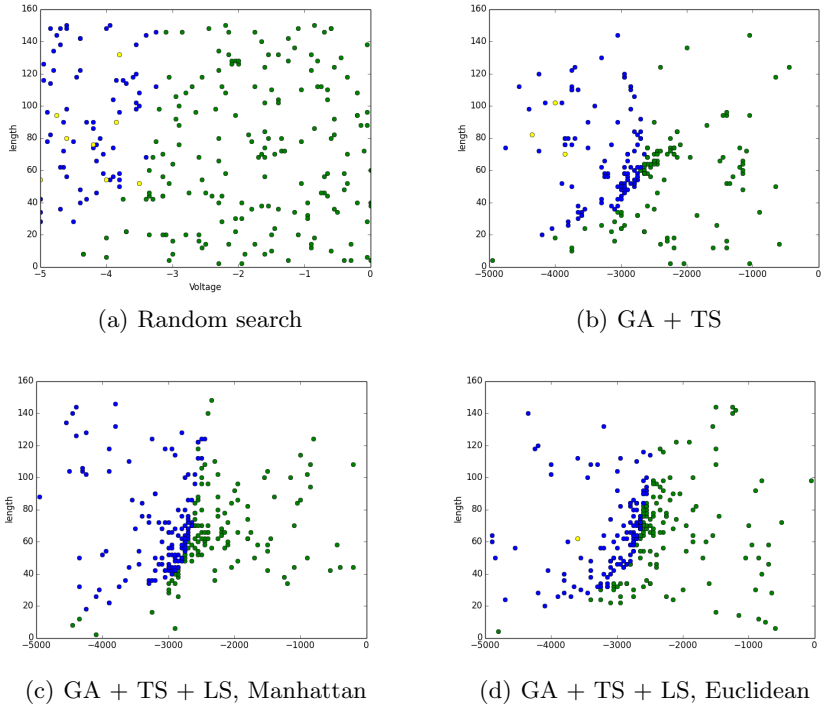
Fig. 2. GA+TS+LS, 250 measurements.

4.3 Search Space Characterization

When the goal is to characterize the search space, or more precisely the region between NORMAL and RESET/MUTE classes, we are not interested in SUCCESS points. Therefore, we can treat them as NORMAL or RESET/MUTE points (and give them fitness values 1 or 2, respectively). Again in this case, the number of measurements is set to 250. In Figures 3(a) to 3(d) we present results for search space characterization with four different algorithms.

We see that random search is not capable to characterize interesting regions with a small number of measurements. A combination of GA, TS and LS algorithms with Manhattan distance performs best since it accurately describes the longest part of the interesting region.

When observing differences in regards to distance metrics, we see that Euclidean distance gives better results when finding faults while Manhattan distance is better in space characterization scenario. However, this observation should be considered *cum grano salis* since we use the same distance value in both cases. It can be concluded that the smaller distances are better when looking for specific points (smaller distances are to be expected in Euclidean metric due to the squaring operation of normalized values) while bigger distance values can cover more space and characterize it better as it can be seen from the case where we use Manhattan distance. As evident from the results, the memetic algorithm behaves much better than the genetic algorithm considered (although, the GA we use is specialized and already behaves much better than the standard GA).



(a) Random search (b) GA + TS

(c) GA + TS + LS, Manhattan (d) GA + TS + LS, Euclidean

Fig. 3. Algorithms for the space characterization, 250 measurements.

It is very difficult to give a meaningful comparison between the efficiency of our algorithm and for instance algorithms presented in [6, 13] due to several reasons. First, we add one more dimension (glitch offset) to the search space and thus we render some of the operators from previous works non applicable. Moreover, our problem is much more difficult due to the extra dimension. Although exhaustive search in two dimensions (with some parameter steps) would take several hours we still consider it to be a realistic approach while with three parameter dimensions this problem becomes completely non practical in a realistic environment. With the increase of the number of parameters, the methods presented here need no additional adjustment and should prove even more efficient with regard to random search, which remains to be addressed.

Next, in our approach we set strict constraints on the available number of measurements which was not the case in previous works (there the goal was a minimal number of measurements without explicitly stating the minimal number). Furthermore, as GAs use information from known solutions in future generations, after finding several faults we can expect to find asymptotically more faults in future generations and that probability increases with the number of generations. In related work there is also a distinction that they conduct three measurements per point to check for CHANGING class [6]. In our approach we

do not consider CHANGING class and consequently we do not conduct multiple measurements of the same points.

Lastly, based on the results in [6] it seems that some of the SUCCESS points that are found and taken into account in statistics are actually repeated measurements of the same points. Since in our approach Tabu Search renders that impossible, it would not be possible to compare those results without removing TS constraint from our algorithm. As evident from our results, TS on average reduces the total number of measurements by more than 20% which results in more unique points our algorithm can generate.

5 Conclusions and Future Work

In this work we revisit the problem of fiddling with multiple parameters for successful fault injection. Our experiments with the memetic algorithm show that one can successfully find faults with a limited number of measurements. Additionally, our algorithm can be used to characterize interesting search space regions. Both scenarios are explored with a "small" i.e. feasible number of measurements. By adding more measurements (and therefore GA generations) we obtain even better results since the GA works by using existing solutions to find new, better solutions. We do not claim that the GA (or the memetic algorithm) is the best possible method, but we demonstrate there are nature-inspired algorithms that can significantly improve the FI process.

Glitch testing in this work has only been performed on a target with no countermeasures. Since it is expected that the search space is affected by such countermeasures, e.g. glitch sensors, the applicability of this approach in a real world attack scenario remains to be assessed. A possible step in our research is therefore to experiment with smart cards on which countermeasures against FI are implemented.

Acknowledgments

This work was supported in part by the Technology Foundation STW (project 12624 - SIDES), The Netherlands Organization for Scientific Research NWO (project ProFIL 628.001.007) and the ICT COST action IC1204 TRUDEVICE.

References

1. R. Anderson and M. Kuhn, "Tamper resistance — a cautionary note," in *In Proceedings of the Second Usenix Workshop on Electronic Commerce*, 1996, pp. 1–11.
2. O. Kömmerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, ser. WOST'99. Berkeley, CA, USA: USENIX Association, 1999, pp. 2–2.

3. S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
4. P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '99. London, UK, UK: Springer-Verlag, 1999, pp. 388–397.
5. J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," in *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, ser. E-SMART '01. London, UK, UK: Springer-Verlag, 2001, pp. 200–210.
6. R. B. Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub, "Glitch it if you can: Parameter search strategies for successful fault injection," in *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, 2013, pp. 236–252.
7. D. Boneh, R. DeMillo, and R. Lipton, "New threat model breaks crypto codes," *Bellcore 85 Press Release*, 1996.
8. D. Boneh, R. A. Demillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults." Springer-Verlag, 1997, pp. 37–51.
9. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault attacks on RSA with CRT: Concrete results and practical countermeasures," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '02. London, UK, UK: Springer-Verlag, 2003, pp. 260–275.
10. S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '02. London, UK, UK: Springer-Verlag, 2003, pp. 2–12.
11. J. van Woudenberg, M. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, 2011, pp. 91–99.
12. J. Balasch, B. Gierlichs, and I. Verbauwhede, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs," in *Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 105–114.
13. S. Picek, L. Batina, D. Jakobovic, and R. B. Carpi, "Evolving genetic algorithms for fault injection attacks," in *2014 Proceedings of the 35th International Convention, MIPRO 2014, Opatija, Croatia, May 26-30, 2014*. IEEE, 2014.
14. F. W. Glover and G. A. Kochenberger, Eds., *Handbook of Metaheuristics*, 1st ed., ser. International Series in Operations Research & Management Science. Springer, Jan. 2003, vol. 114.
15. T. Weise, *Global Optimization Algorithms - Theory and Application*, 2nd ed., 2009, online available at http://www.it-weise.de/.
16. R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2d euclidean distance transform algorithms: A comparative survey," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 2:1–2:44, Feb. 2008.
17. E. F. Krause, *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*, ser. Dover Books on Mathematics. Dover Publications, 1988.