MIPRO 2014, 26-30 May 2014, Opatija, Croatia

Evolving Genetic Algorithms for Fault Injection Attacks

Stjepan Picek and Lejla Batina
Radboud University Nijmegen, Institute for Computing and Information Sciences (ICIS)
Postbus 9010, 6500 GL Nijmegen, The Netherlands
Email: stjepan@computer.org, lejla@cs.ru.nl
Domagoj Jakobović
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, Zagreb, Croatia
Email: domagoj.jakobovic@fer.hr
Rafael Boix Carpi
Riscure BV, The Netherlands
Email: BoixCarpi@riscure.com

***Abstract*—Genetic algorithms are used today to solve numerous difficult problems. However, it is often needed to specialize and adapt them further in order to successfully tackle some specific problem. One such example is the fault injection attack where the goal is to find a specific set of parameters that can lead to a successful cryptographic attack in a minimum amount of time. In this paper we address the process of the specialization of genetic algorithm from its standard form to the final, highly-specialized one. In this process we needed to customize crossover operator, add a mapping between the values in cryptographic domain and genetic algorithm domain and finally to adapt genetic algorithm to work on-the-fly. For the last phase of development we plan to go to the memetic algorithm by adding a local search strategy. Furthermore, we give a comparison between our algorithm and random search which is the mostly employed method for this problem at the moment. Our experiments show that our algorithm significantly outperforms the random search.**

I. Introduction

Genetic algorithms (GAs) are widely used to solve many different optimization problems. Since they are very general solvers, often it is necessary to customize the algorithm for some specific problem. Although the "No free Lunch" theorem states that when averaged over all problems all algorithms behave the same, that is true only when we posses no knowledge about the problem [1]. In this paper we start with standard genetic algorithm, and on the basis of experimental results we specialize it further to better relate to the problem at hand.

We are surrounded by small smart devices in our lives such as smart cards (contact and contactless), RFID tags, mobile phones etc. and their security is of utmost importance for a user. Considering the security and privacy of those devices there exists a special class of attacks that a malicious adversary can try, aiming at weaknesses of implementations. Those so called implementation attacks are divided into passive and active. The former explore some passive physical channel that is available while the device is processing secret data (such as power consumption or timing) and the latter is actively disturbing the computation aiming at finely tuned faults [2], [3]. The fault has to be effectively inserted such that the device does not break while an error computation will occur. This fact allows for the adversary, after inserting several "useful faults", to learn the secret key used. The famous Bellcore attack shows that having one faulty RSA signature and a correct one allows the adversary to factor the RSA modulus and hence completely break the system [4].

In this paper we consider so called glitching attacks where a sudden change in the supply voltage of a smart card can have the desired effect of a fault. However, inserting "useful faults" is considered to be very hard as there are many parameters that have to be tuned such as timing, glitch length, glitch voltage etc. The search in this multi-dimensional space is typically performed as a random search. Our work aims at improving this and deriving more efficient methods in finding the right ranges of all relevant parameters.

A. Related Work

As mentioned above, the research on fault attacks started around two decades ago with the seminal paper of Boneh, DeMillo and Lipton [4]. They showed the potential of this type of attacks assuming that they are possible. However, it took several years for a publication on an actual practical fault attack by Aumuller et al. [5]. The authors published in 2002 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 clock and voltage lines of a cryptographic chip.

Van Woudenberg et al. describe a real attack scenario for an Optical Fault Injection attack where a laser is used to induce faults [6]. The authors also introduce the practical problem of setting the parameters for fault injection and they briefly discuss the lack of methodology.

Recently, a paper of Boix Carpi et al. [7] proposed a new algorithm that is a tailored search strategy but it is especially developed for finding the best choice of parameters for glitching. The authors also experiment with genetic algorithm and suggest investigating it further. We continue with this recommendation aiming at setting common grounds for this new line research that could help security evaluators to assure that smart devices preserve user's privacy.

B. Our Contribution

In this paper we experiment with genetic algorithms when looking for parameters that can lead to successful fault injection. For the algorithm to behave as expected we needed to create a custom version of genetic algorithm and also interface it with a hardware-based system that inserts and evaluates fault attempts.

The rest of this paper is organized as follows: in Section 2 we present experimental setup, whereas in Section 3 we give experimental results of several search strategies as well as discussion about the results. Finally, in Section 4 we give a conclusion.

II. Experimental Setup

In this section we describe the fault injection scenario we follow, parameters of the problem and mappings between the solutions. On a more abstract level, the goal is to interface the system for injecting faults and evaluate them with the genetic algorithm framework. The role of genetic algorithm is to evolve new fault parameters and send those to the fault injection framework for an analysis. Motivation lies in the fact that finding such values is a difficult problem and there is no algorithm, except exhaustive search, that can do it. However, exhaustive search is often not a viable choice since there are constraints regarding the time that attacker has at his disposal. Furthermore, it is somewhat surprising that random search is regarded as the baseline strategy for this problem and as such is widely used.

Since random search is considered as a viable choice, that is also a pointer that research can go to the evolutionary computation area. We opted for genetic algorithms since they are directed random search method. In terms of an optimization problem, we want to minimize the time needed to find a set of parameters that can lead to the successful fault injection attack when we work in black-box scenario (no specific information about the device we attack).

A. Search Space Parameters

There are multiple parameters that need to be adjusted for a successful fault injection. Generally, they can be divided into those related with glitch shape, time and environment. In this work, we do not include environment or timing parameters in our experiments. The reason is the possibility to separate the search into two phases where in first phase we look for the adequate glitch shape and in the second phase for the correct timing periods to apply those glitches. Environmental parameters are not of interest since they can change even within the search and it is not clear how each of those changes affects the search.

Glitch shape parameters can be divided to glitch voltage [V] and glitch length [ns] parameters. With these assumptions, the goal is finding a suitable values for voltage and length that would provoke a successful fault injection. A pair of values for glitch voltage and length is considered a single point in search space (a possible solution).

B. Verdict Classes

Fault injection testing equipment and tool for the evaluation of introduced faults can output only verdict classes for each attempt. Therefore it is necessary to map each of those verdict classes to fitness values a GA can work with. There exist several possible classes for classifying a single measurement (attack attempt) [7]:

1) NORMAL: the target behaves as expected (the glitch is ignored)
2) RESET: the target resets as a result of the glitch
3) MUTE: the target stops all communication as a result of the glitch
4) CHANGING: multiple measurements in the same point yield responses that classify into different classes.
5) SUCCESS: the target response is a specific, predetermined value that does not happen under normal operation

If we define $F()$ as a function that scores the goodness of a point, and the higher $F()$ value is desired, then the formula for a single point classification is

$$\begin{aligned} F(SUCCESS) > F(CHANGING) > \\ (F(NORMAL) == F(RESET) == F(MUTE)). \end{aligned} \tag{1}$$

C. Interfacing Frameworks

For the fault injection (FI) framework we use Inspector program from Riscure BV. For genetic algorithms test suite we use the Evolutionary Computation Framework (ECF) [8]. ECF is a C++ framework intended for the application of any type of the evolutionary computation, developed at the University of Zagreb.

Since fault injections need to be done on the basis of the results from genetic algorithm, that makes it necessary for the fault injection system and genetic algorithm framework to communicate in real time. To be able to do that, we made an interface between those two systems.

The genetic algorithm is set to output one file for whole generation in each iteration of the evolution process. The fault injection software reads it and conducts the attacks with the values (search points) specified in the file. After each of the attacks, the FI software assigns a verdict class to each point and outputs appropriate parameter values and fitness values to a file. From that file the GA framework reads the fitness values and proceeds with the evolution.

The file system is chosen as a means of communication on the basis of the duration of candidate points evaluation; since each injection attempt can take up to several seconds, there is no need for faster process communication.

D. Target of Evaluation

We conduct all experiments with a smartcard based on an ATMega163+24C256 IC in CMOS technology. This card does not have any side-channel countermeasures or fault-injection countermeasures. All processing on the card is performed in software, and the card is running on an external 4MHz clock frequency. The code we are attacking is the PIN authentication mechanism.

III. Search strategies and Results

Finding the correct parameter setting is a highly non-deterministic process, and countermeasures just add up to the complexity of this problem. As a result of this, security analysts usually set a value range for each parameter. Then, the fault injection setup is left testing thousands of different parameter configurations within given ranges for a-posteriori analysis. That search is typically done with strategies that involve a high search space complexity. Examples are Monte Carlo search for a successful setting within a set of value ranges, or test all possible value combinations within value ranges given a value step. Additionally, the parameter tuning (if performed) is usually done manually. As a result, the testing process can take days.

The analyst possesses some knowledge about the problem that is generally applicable, for instance, it can be assumed that two classes will always have the same relative positioning (NORMAL with low absolute values for glitch voltage/length, and RESET/MUTE class for high absolute values). It is not so interesting to search in the areas that are far from the decision boundary since it can be assumed that there are no outliers. However, there is no prior knowledge about the positions of those regions.

All the points are measured three times to enable the detection of changing behavior (the CHANGING class). Since a CHANGING class simply corresponds to different behavior in subsequent probes, a minimum of two is required for enabling detection of that class. The motivation for this lies in the fact that for some unstable protected targets more measurements per point are required to mitigate some side-effects during data acquisition.

A. Monte Carlo Search

This search strategy consists of performing measurements with randomly selected parameter combinations within the given initial search space. The random distribution for selecting values is considered to be uniform for each parameter present in the search space. This strategy is considered as the baseline search strategy.

A typical result for Monte Carlo search is given in Figure 1. All graphs are produced automatically by Inspector program and they depict only NORMAL, RESET/MUTE and SUCCESS verdict classes (CHANGING class is not shown even if found by the algorithm). Green dots represent solutions from NORMAL verdict class, BLUE represent solutions from RESET/MUTE and red dots represent solutions that belong to SUCCESS verdict class.

Two test runs are performed with the Monte Carlo strategy; a short one with 3072 measurements and a longer one with 76800 measurements. The short test completed with no SUCCESS measurement, and only a longer 76800 measurements

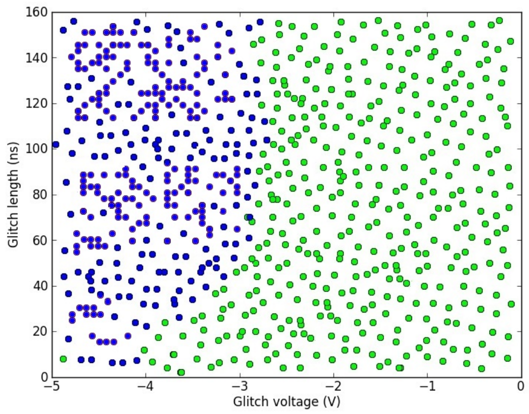

Fig. 1. Plot of measurement classification for Monte Carlo search.

TABLE I
Genetic algorithm parameters

Parameter	Parameter Value
Tournament size	3
Glitch length	[2, 150]
Glitch length resolution	2
Glitch voltage	[-5, -0.05]
Glitch voltage resolution	0.05
Population size	30
Stopping criterion	Stagnation in 10 generations
Mutation rate	0.1

test run produced 11 SUCCESS measurements. Because of the random nature of parameter values selection, there is also a significant number of repeating solutions.

B. Genetic Algorithm

For a genetic algorithm to work, we need to map verdict classes to distinct fitness values. Since we set the problem to be maximization, better classes will have higher fitness values. Mapping using in the initial version of the algorithm is NORMAL, RESET or MUTE verdict classes have value 1, CHANGING verdict class has value 8 and SUCCESS verdict class has value 10. Formally, the GA aims to find a (glitch length, glitch voltage) pair such that the fitness value F is maximal. Parameters of the search are given in Table I. Some of those values are set after the parameter tuning phase where we experimented with different population sizes and stopping criteria.

Since there are only a few different verdict classes, it means that a big portion of the search space will be impossible to differentiate. Therefore, we developed a version of tournament generational algorithm that is more appropriate for this specific problem. A single generation of the algorithm is described with the pseudocode as given in Algorithm 1.

In all the experiments, the population in the first generation is created randomly by the GA framework, where the initial fitness is zero for all individuals and then is sent to file to get the actual fitness values after the evaluation.

Algorithm 1 Genetic Algorithm - initial version

```
Require: count = 0, crx_count = 0
  repeat
    randomly select 3 individuals
    select best of the 3 individuals
    copy best to new population
    count = count + 1
  until $N \neq$ count
  replace old with the new generation
  repeat
    randomly select parents $p_1, p_2$
    $child_1$ = crossover $(p_1, p_2)$
    $child_2$ = crossover $(p_2, p_1)$
    replace $p_1$ with $child_1$
    replace $p_2$ with $child_2$
    crx_count = crx_count + 1
  until N * $p_c \neq crx\_count$
  perform mutation on all individuals with $p_m$
  evaluate population
```

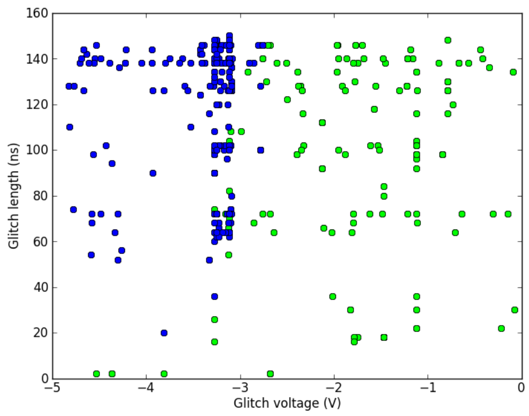

Fig. 2. Measurements for GA, initial algorithm.

Result from initial version of GA where the stopping criterion is 10 generations without improvement is depicted in Figure 2.

It can be observed that the algorithm outputs a valid point. By valid it is understood that the value is in the boundary between the two unimodal components. If speaking in terms of classifiers, the output point would be on top of the decision boundary between the two classes. The algorithm finished when no better fitness values could be found.

In Figure 3 we see a result of the same experiment but with 50 generations without improvement as a stopping criterion.

After running the first set of experiments we noticed some undesired effects. Once the algorithm finds a point with a high fitness value, it seems to fix one variable and explores the full range of the other. For this problem, spreading the search so much only in two directions is not so desirable in the sense that points too distant from each other that belong

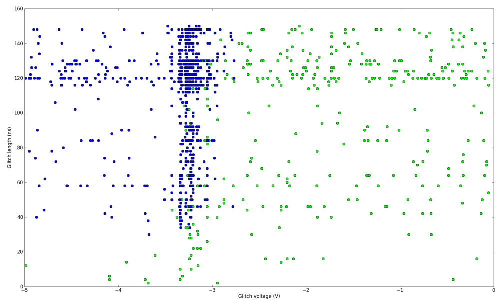

Fig. 3. Measurements for GA, initial algorithm, long run.

to the CHANGING class will likely classify to one of the low fitness valued classes (NORMAL or RESET/MUTE). To put it in other words, the decision boundary between classes will always have the same shape (convex, monotonically increasing curve and ideally it would mimic an exponential curve), so the direction that contains points belonging to the CHANGING class is not explored in depth by a vertical or horizontal line.

GA would be more successful in terms of finding more points that classify to CHANGING class by exploring the surroundings of a point that classifies into CHANGING inside a neighborhood of distance d (increasing d if needed) from it. Another effect that we observed is that once the GA finds a point belonging to the CHANGING class, the rest of the search space is ignored. This elitism is good in a sense that we want the search to be as short ass possible, but often it results in not exploring interesting parts of the search space sufficiently.

C. *Improving Crossover and Parameter Tuning*

Evolutionary algorithms in general (and GA in specific) do optimization by default, which means they always try to find points with a higher fitness value. The behavior of current algorithm is expected, especially with such small variances in fitness values. However, as we want to find as many points that belong to a certain class, we change the crossover operator to be more adequate.

New crossover operator crosses two points that should be of *different* classes, and it would choose a point between them that is closer to the better "parent" point. In this way we can point the algorithm to the desired region. Results for the new algorithm with improved crossover operator are shown in Figure 4. Although it may seem that the results depicted in Figure 4 are not better than those with the original algorithm, we remind that the graph does not show CHANGING class.

D. *Optimization + Classification Algorithm*

In a classical black-box optimization scenario, the best possible fitness value is not known in advance. The optimization algorithm will therefore always strive to find points with better values, regardless of how good the current solution is. If the best value is known, or if the goal is to find a solution of an acceptable predefined fitness value, then the algorithm is

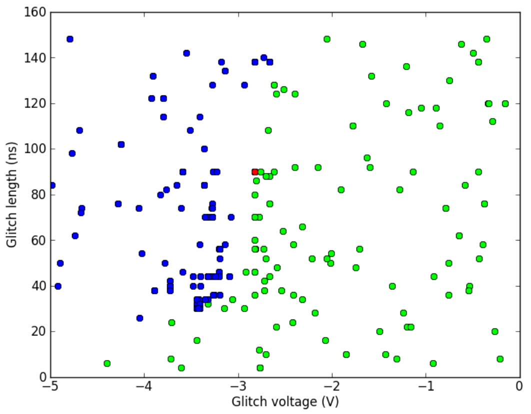

Fig. 4. Measurements for GA with modified crossover.

usually configured to stop as soon as it finds a single solution of the desired quality.

In this application we diverge from the described principle in the following ways:

1) we define a *target fitness* level where all the solutions having fitness equal or better than the target value are regarded as equally good and called *target points*;
2) instead of satisfying with a single target solution, the algorithm should find as many distinct target points as possible;
3) points worse than target value are divided into a number of *subclasses* corresponding to their fitness (we presume there is a small number of discrete fitness values).

The fitness mapping in this version is also not the same as in previous version of the algorithm, to allow distinction of different subclasses. Here we opted for a small change in values when mapping from verdict classes, so that NORMAL class retains the value of 1, RESET or MUTE classes have value 2, CHANGING class has value 8.5 and the target fitness level is set at 8. This allows the algorithm to perceive NORMAL and RESET/MUTE points as different subclasses, and all points that classify to CHANGING as target class points.

The next generation is produced in part by crossing the points from the current population of size N, and in part by selecting the better individuals by tournament selection operator. The proportion of crossover is controlled by the crossover probability (p_c), and the remaining individuals are selected with the tournament operator.

The selection of parents in the crossover operator is designed to promote diversity and explore boundaries between classes (points of different fitness values). The first parent is selected randomly from current generation; then, if the first parent is worse than the target level, the second parent is randomly selected from another class (point with a different fitness value). If, on the other hand, the first parent is of the target class, the algorithm tries to select another target class point as the second parent. If no such individual exists, a random parent is taken otherwise.

TABLE II
COMPARISON OF THE MONTE CARLO AND GA IN FINAL VERSION

Strategy	Measurements	#SUCCESS
Monte Carlo	3072	0
GA + classification	1560	8

The crossover operation also acts depending on the parent classes. If crossing points belong to different classes, the child is generated as a point between the parents in search space, but proportionally closer to the parent whose class is represented with a smaller number of individuals in current population. In this way, the search is guided towards the class that covers a smaller area of the search space. If crossing points are from the same class, the child is generated randomly between the parents in the search space.

After the crossover and tournament selection, a mutation operator is invoked on the new set of individuals, where the total number of mutations is determined with the mutation probability parameter (p_m). Algorithm 2 is a pseudocode for one iteration of the modified genetic algorithm.

Algorithm 2 Genetic Algorithm + classification

```
Require: crx_count = 0, mut_count = 0
  repeat
    select first parent
    if first parent of target 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
```

A typical result of the modified GA version is shown in Figure5. In this case the algorithm succeeded in finding the most target class points in a limited number of measurements.

In Table II we give the comparison of the successfulness of Monte Carlo and genetic algorithm that uses modified crossover operator and classification.

IV. Conclusions

In this paper we explore a new option to successfully find the correct parameters for the fault injection. Currently employed methods can produce results, but are slow and often require more time than the security analyst is willing to spare.

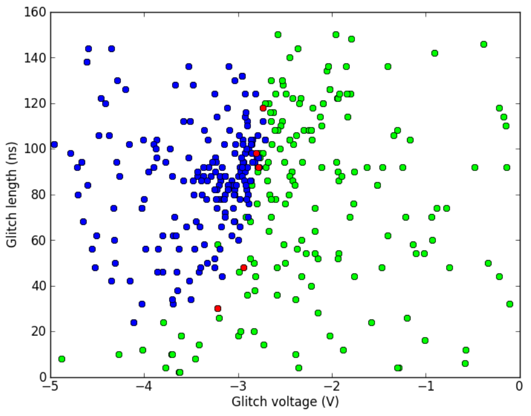

Fig. 5. Measurements for GA + classification.

We experimented with genetic algorithm where we started from its standard form and finished with a specialized algorithm. The results are promising and indicate that additional research is required in this area.

As a possible avenue for future research we plan to experiment with a memetic algorithm. That algorithm would comprise of the genetic algorithm in the last version plus an addition of a local search. When the algorithm finds a point that belongs to class INTERESTING (or better) from that point will inspect n points within distance d. If the new point belongs to a different class, the algorithm will keep that point and halve the distance value d for the next iteration of the algorithm. Additionally, since the genetic algorithm approach is working better than the baseline search strategy, it would be interesting to see how it behaves with fault injections that are based on electromagnetic radiation. The process would be similar to the scenario we describe here, only it would search for the parameters related with electromagnetic radiation.

Acknowledgements

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] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, April 1997.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology: Proceedings of CRYPTO'99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed., no. 1666. Springer-Verlag, 1999, pp. 388–397.

[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] R. D. D. Boneh and R. Lipton, "New threat model breaks crypto codes," *Bellcore 85 Press Release*, 1996.

[5] 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.

[6] 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.

[7] R. Boix Carpi, S. Picek, L. Batina, F. Menarini, D. Jakobovic, and M. Golub, "Glitch it if you can: Novel parameter search strategies for successful fault injection," in *to appear in the proceedings of 12th Smart Card Research and Advanced Application Conference (CARDIS), Lecture Notes in Computer Science*, vol. xxxx, Berlin, Germany, Nov. 2013, pp. yyy–zzz.

[8] D. Jakobovic and et al., "Evolutionary computation framework," Jan. 2014. [Online]. Available: http://gp.zemris.fer.hr/ecf/