#SAT for Vulnerability Analysis of Security Components

Linus Feiten* Matthias Sauer* Tobias Schubert* Alexander Czutro* Victor Tomashevich‡
Eberhard Böhl† Ilia Polian‡ Bernd Becker*

* Albert-Ludwigs-University of Freiburg
Georges-Köhler-Allee 051
79110 Freiburg, Germany
{ feiten | sauerm | schubert | aczutro | becker }
@informatik.uni-freiburg.de

† Robert Bosch GmbH
72703 Reutlingen
Eberhard.Boehl@de.bosch.com

‡ University of Passau
Innstrasse 43
94032 Passau, Germany
{ ilia.polian | victor.tomashevich }
@uni-passau.de

***Abstract*—Vulnerability to malicious fault attacks is an emerging concern for hardware circuits that process sensitive data. We describe a new methodology to assess the vulnerability to such attacks, taking into account built-in protection mechanisms. Our method is based on accurate modeling of fault effects and their detection status expressed as Boolean satisfiability (SAT) formulae. Vulnerability is quantified based on the number of solutions of such formulae, which are computed by an efficient #SAT solver. We demonstrate the applicability of this method by analyzing a sequential pseudo random number generator and a combinatorial multiplier circuit both protected by robust error-detecting codes.**

Keywords: *Fault attacks, Satisfiability, Error detection, Vulnerability analysis, Random Number Generators*

I. Introduction

Integrated circuits increasingly process data that needs to be protected against unauthorized access; typically by cryptographic algorithms that are inherently hard to break. Attack scenarios that target the hardware implementation of an algorithm have been proposed and demonstrated in the past. These attacks establish and utilize *side channels* over which protected data can be read or written; e.g. analysis of power consumption, electromagnetic emissions or timing of operations. In order to manipulate the processed data, physical faults are injected into the hardware [1] and the effects of the modified calculation are observed. Successful fault-based attacks have been reported for a variety of systems, including RSA [2], AES [3], Trivium [4] and LED [5].

The protection of devices against fault-based attacks is based on error-detection. The information is processed in encoded form. When an error occurs, a non-codeword is calculated and identified by the error-detection circuitry, and an action like the system's shut-down may be triggered. Standard codes may not detect all faults which lead to the development of concepts specifically targeting malicious attacks, including robust codes [6].

The method described in this paper has first been introduced in [7]. It assesses the vulnerability of protected circuits to fault attacks by determining the probability that a modelled fault is not detected within a given number of clock cycles. We create a Boolean satisfiability (SAT) instance which formalizes this concept. Each satisfying assignment of this instance corresponds to a system state and a sequence of inputs for which the fault is not detected. We then apply a #SAT solver to count the number of such assignments. The result divided by the number of all states and input possibilities is the probability of a successful attack (i.e. undetected fault injection); assuming the states and input patterns are random and equiprobable. Such an assumption is valid for devices protected against power analysis attacks, because the attacker is not able to determine the exact system state before the fault injection.

The method is illustrated by two case studies: a small combinational multiplier used in an earlier work [8] and an industrial deterministic random number generator COSSMA [9], [10]. We perform design space exploration and identify large differences in vulnerability for designs with same cost, allowing the designer to make informed choices. In addition, we formally prove that the design with the largest possible protection detects all single and double faults in COSSMA.

The remainder of this work is structured as follows. Section II gives an overview over the used basic algorithms and is divided into an introduction to SAT and #SAT (Section II-A) and to the ECMS@ fault model (Section II-B). In Section III we then in detail present the method for combinational and sequential circuits. The case studies are presented in Section IV. Section V concludes the work.

II. Preliminaries

A. Introduction to SAT and #SAT

The vulnerability metrics defined further below are calculated by a method incorporating SAT-based automatic test pattern generation (ATPG). In ATPG, a circuit with a fault is given, and an input vector (test pattern) is returned for which the outputs of the fault-affected and fault-free circuit differ. SAT-based ATPG expresses this as a Boolean formula which is passed to a SAT solver that searches for a satisfying assignment of the formula's variables. The values of the returned test pattern are derived from the found assignment. If no such assignment exists, the fault is undetectable. While the SAT problem is NP-complete, recent powerful SAT algorithms including antom [11], glucose [12], lingeling [13] and miniSat [14] are capable of performing ATPG for multi-million-gate circuits [15].

Vulnerability analysis performed in this article targets transient faults that are injected by a malicious attacker, and not manufacturing defects. Rather than finding one test pattern that identifies a fault, we are interested in the effects and detectability of the fault under a long stream of equiprobable inputs. The definition of "detection" in context of transient faults is less straight-forward than for permanent manufacturing defects and will be discussed later.

For this reason, we are utilizing an extension of SAT known as #SAT that counts the number of solutions satisfying a given formula. To the best of our knowledge, this is the first application of #SAT in the field of circuit design and analysis.

#SAT is much harder than SAT for a given formula. A conventional SAT solver essentially performs depth-first search based on the classical DPLL method, [16], [17] enriched by a number of performance enhancements, until *one* satisfying assignment is found. #SAT solvers, however, count the number of *all* solutions by decomposing the Boolean formula into disjoint components. The #SAT-solver used in this work, #antom, is built upon antom [11], a state-of-the-art SAT algorithm.

To represent different vulnerability-related conditions, a powerful fault modeling language is required. The next section provides details on the fault representation formalism used in this work.

B. Introduction to ECMS@

Our method employs the SAT-based ATPG tool TIGUAN [15] as the front-end to #antom (reviewed in the previous section). TIGUAN supports the ECMS@ (*Enhanced Conditional Multiple-Stuck-at*) model as a generic formalism to capture complex manufacturing defect mechanisms [18]. Here, we use it to create symbolic representations of fault attacks and their consequences. For simplicity, we assume attack-induced transient faults with duration of one clock cycle, although longer fault injections could be modeled as well.

An ECMS@ fault is given by r *condition lines* $a_1, \ldots, a_r$ ($r \geq 1$ or $r = 0$ i.e. no conditions), each one of them associated to a *condition* c_i; and by q *victim lines* $s_1, \ldots, s_q$, ($q \geq 1$), each one of them associated to a logical value b_j. The meaning of such a fault is as follows: if all conditions c_i are met at the same time, then each victim line s_j will change its value to b_j. Consequently, TIGUAN will attempt to generate a test pattern that satisfies all the conditions and makes sure that the changes on the victim lines will result in different output values in the fault-free and the fault-affected circuit. TIGUAN supports several types of conditions, and only the following subset of them is used for vulnerability analysis. Recall that a SAT-based ATPG keeps two logic values for each (relevant) signal line in the circuit: one for the fault-free circuit and one for the circuit affected by the fault. The ECMS@ model enables control of both these values.

- $c_i = \mathbf{0}$: Line a_i is set to 0 in the fault-free circuit;
- $c_i = \mathbf{1}$: Line a_i is set to 1 in the fault-free circuit:
- $c_i = \boldsymbol{F}$(*Fault-affected*): fault effect is visible on line c_i, that is, line a_i has different logic values in the fault-free and the fault-affected circuit;
- $c_i = \boldsymbol{NF}$(*Non-Fault-affected*): fault effect must not be propagated through a_i, the values in the two versions of the circuit are identical).
- $c_i = \boldsymbol{S0}$: Line a_i is set to 0 in the fault-free and in the fault-affected circuit;
- $c_i = \boldsymbol{S1}$: Line a_i is set to 1 in the fault-free and in the fault-affected circuit.

Note that conditions $\mathbf{0}$ and $\mathbf{1}$ do not impose requirements for the fault-affected version of the circuit. Their difference from $\boldsymbol{S0}$ and $\boldsymbol{S1}$ will play an important role in modeling of certain vulnerability scenarios.

To represent sequential behavior, values assumed by a line in different clock cycles are indicated by suffix @ followed by the number of the clock cycle. For example, the following ECMS@ fault describes the stuck-at-0 behavior on the signal line called *victim* in clock cycle 0 (i.e., this line will assume the logic-0 value in clock cycle 0), provided that the signal line *alert* is 0 in clock cycles 0 and 1, while the line *checked* is non-fault-affected in clock cycle 1:

$$alert@0\ \boldsymbol{S0},\ alert@1\ \boldsymbol{S0},\ checked@1\ \boldsymbol{NF} : victim@0\ sa0$$

This example illustrates that the conditions in an ECMS@ fault do not necessarily have to describe the physical cause of the fault. Here, the fault effect occurs in clock cycle 0 whereas two of three conditions involve a later clock cycle (namely 1). Such modeling is understood by TIGUAN, and it is also meaningful to represent vulnerability conditions. In this example, *alert* might be a system-wide signal that is 1 if a fault is detected, and *checked* might be a line attached to a checker. Therefore, this fault describes a situation where a fault on *victim* in cycle 0 is not detected (no alert) within clock cycles 0 and 1, if *checked* is unchanged in cycle 1 but not necessarily in cycle 0. If the original version of TIGUAN finds a test pattern (here, a sequence of two input vectors and the initial state vector) that detects this fault, then this test sequence exposes a vulnerability: a fault attack that is undetected for two clock cycles. TIGUAN integrated with the #SAT solver yields the total number of such sequences, which can be used to quantify the probability of such vulnerability as formalized in the next section.

III. #SAT-BASED VULNERABILITY ANALYSIS

In this work we consider two metrics that quantify the susceptibility of a circuit to malicious fault attacks: the *vulnerability vul* and the *attack success rate ASR*. Both metrics assume that the circuit is protected by error-detecting circuitry and quantify the chances of an undetected fault injection. Vulnerability relates the frequency of such successful attacks among all possible scenarios, including fault injections which were masked and had no effect at all, while ASR [8] considers only attacks resulting in visible effects on system level. In the following, we formalize these metrics for combinational and sequential circuits.

A. Combinational case

Let C be an n-input combinational circuit realizing the Boolean function Z. We assume that C is equipped with error-detecting circuitry connected to an output ED. In this work, we consider attacks modeled as faults f that flip single or multiple signal lines of the circuit (also referred to as victim lines) to logic-0 or logic-1 for the duration of one clock cycle. In combinational circuits, such faults correspond to single or multiple stuck-at faults. As introduced above, let Z^f be the Boolean function of C when fault f is injected. When test pattern $t \in \mathbb{B}^n$ is applied to the circuit's inputs and fault f is present, the value $ED(t, f)$ at output ED equals 1 if the fault is detected or 0 if the fault is not detected.

For a fault f, a test pattern t may belong to one of three classes:

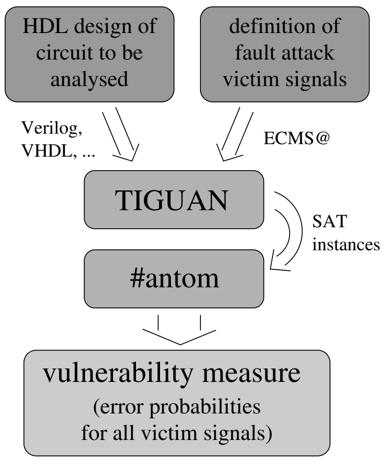
HDL design of circuit to be analysed
definition of fault attack victim signals
Verilog, VHDL, ...
ECMS@
TIGUAN
SAT instances
#antom
vulnerability measure
(error probabilities for all victim signals)

Figure 1. Method overview

1) No effect: $Z(t) = Z^f(t)$ (this implies $ED(t, f) = 0$)
2) Detected attack: $Z(t) \neq Z^f(t)$, $ED(t, f) = 1$
3) Successful undetected attack: $Z(t) \neq Z^f(t)$, $ED(t, f) = 0$

Let the number of patterns for which fault f is detected be $\delta(f)$, and let the number of patterns for which fault f results in a successful undetected attack be $\sigma(f)$. Now we can formally define the vulnerability and the attack success rate for a fault f in a combinational circuit:

$$vul(f) = \frac{\sigma(f)}{2^n} \qquad ASR(f) = \frac{\sigma(f)}{\delta(f) + \sigma(f)}$$

$vul(f)$ quantifies the probability of a successful (undetected) fault injection assuming that the attacker cannot freely control the inputs of the block C and can thus encounter any of the 2^n possible input combinations with equal probability at the time of his attack. This assumption is valid as the blocks being analyzed are often parts of a larger circuit such that no direct physical access is possible. While it is possible to use side-channel analysis, e.g., observing the power consumption, to at least derive the applied input combination, circuits including the COSSMA generator, used for evaluation later in this article, are designed to have no or little information leakage through side channels, thwarting such approaches. In contrast, $ASR(f)$ is restricted only to attacks that do have visible effects ($Z^f(t) \neq Z(t)$) and quantifies the share of such attacks that are not detected.

To calculate the values σ, δ, vul and ASR for faults from a given fault list F, the flow from Figure 1 is employed. The circuit is represented by a hardware description language such as VHDL. Then, the following two ECMS@ faults are obtained for every single or multiple stuck-at fault $f \in F$. ECMS@ fault f_σ has the victim lines from f and one condition $ED = \boldsymbol{S0}$. A test pattern t detects f_σ if it detects f and at the same time results in $ED(t, f) = 0$, i.e., error-detection circuitry misses this fault although it is visible at an output. ECMS@ fault f_δ again reuses the victim lines from f and has condition $ED = \boldsymbol{F}$. This fault is detected by all patterns that detect the original fault f and result in different values on output ED in absence and presence of f; since ED is always 0 in absence of faults, this implies that the fault must be detected.

Then, TIGUAN is applied to map the ATPG problems for the two faults f_σ and f_δ, to SAT instances, and both resulting SAT instances are given to the #SAT solver #antom. The number of satisfying assignments for the SAT formula originating from the ATPG problem for fault f_σ corresponds to the number of patterns for which a successful attack is not detected, i.e. $\sigma(f)$. Analogously, $\delta(f)$ is the number of satisfying assignments for ATPG applied to f_δ. Knowing $\sigma(f)$, $\delta(f)$ and the number of inputs n, calculating $vul(f)$ and $ASR(f)$ is straightforward, using the formulae given above.

One objective of our analysis is to calculate the vulnerability of individual circuit locations in order to identify the likely attack targets ("weakest links"). This knowledge can be used to guide re-synthesis or apply low-level protection to the identified "weakest links" until vulnerability and/or ASR becomes satisfactory for all elements in the circuit. Several faults may be related to the same circuit location, such as the single-stuck-at 1 and 0 faults on the same signal line v. Assuming that the attacker cannot control the polarity of the injected fault, it is useful to obtain the vulnerability of v by aggregating the vulnerabilities for both faults (indicated by $v/1$ and $v/0$ for brevity):

$$vul(v) = \frac{vul(v/1) + vul(v/0)}{2}.$$

If multiple circuit locations $v_1, \ldots, v_q$ are targeted, the vulnerabilities of all 2^q stuck-at faults of multiplicity q are aggregated:

$$vul(v_1, \ldots, v_q) = \frac{1}{2^q} \sum_{(b_1,\ldots,b_q) \in \mathbb{B}^q} vul(v_1/b_1, \ldots, v_q/b_q).$$

For double-attack analysis performed in this work the vulnerability of a pair v_1 and v_2 of circuit lines is $vul(v_1, v_2) = 1/4 \cdot (vul(v_1/1, v_2/1) + vul(v_1/1, v_2/0) + vul(v_1/0, v_2/1) + vul(v_1/0, v_2/0))$. It is straightforward to extend this formalism from vul to ASR.

B. Sequential case

A sequential circuit has n inputs, m outputs and y flip-flops. A fault that affects a sequential circuit may manifest itself in the clock cycle in which it occurred or in a subsequent clock cycle. Consequently, it can be detected immediately or in a later clock cycle. In this work, we require that a fault is detected during a limited and finite period of time (number of clock cycles); faults that are not detected during that detection period are considered undetected. The definition of an error-detection period is important as the attacker requires finite time to perform his malicious calculation; if the fault has been identified long after it has occurred, it may not be possible any more to prevent damage.

To model fault effects confined to k clock cycles upon fault injection, we extend the definitions of vul and ASR to sequential circuits that run for k clock cycles. Then, the circuit processes $y + n \cdot k$ logical values (the initial state and the input vector during each of the k clock cycles) and produces $m \cdot k$ output values in total. It is

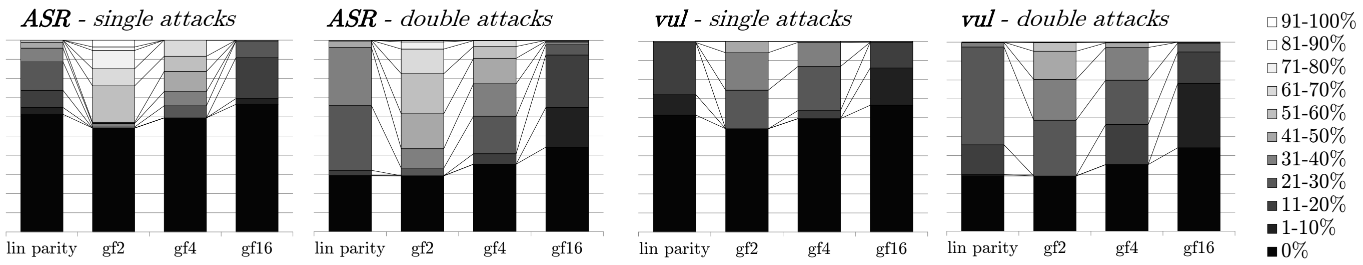

Figure 2. Distributions of $vul(f)$ and $ASR(f)$ for single and double stuck-at fault injections in the Braun multiplier

possible to map a sequential circuit to its k-step clock cycle expansion: a combinational circuit that behaves identical to the original sequential circuit during k clock cycles [19]. The flow described in the previous section can then be applied to this circuit.

One specific feature of the sequential COSSMA checker design evaluated later in this article is its ability to flexibly select outputs that are checked in individual clock cycles (further details are provided below). Instead of explicitly modeling the checker and the ED signal line, we considered value modifications at the checked outputs as fault detections. This necessitated several slight changes to the ECMS@ modeling, which are discussed in Section IV-B.

IV. Applications and Experimental Results

A. Braun multiplier

We used a small combinational Braun 4×4 multiplier circuit [20] protected by the information redundancy scheme such as in [8] to evaluate our flow for combinational circuits. The combinational core is equipped with a check bit predictor and a consistency checker according to an error-detecting code. As in [8], four codes have been considered: one linear (parity) code and three non-linear robust codes over GF(2), GF(4) and GF(16), respectively. ASR has been estimated in [8] by an FPGA-assisted fault-injection experiment while the results reported here are exact. Note that the circuit in [8] has been synthesized using only inverters, NAND2 and NOR2 gates to facilitate electrical analysis performed in that work. For our analyses in this work we synthesized it with the full set of logic gates, so that the gate-level net-lists are not identical to [8] and the results not directly comparable.

Table I contains results for the four versions of the Braun multiplier considering single and double attacks. As explained in Section III-A, a single attack targets a single signal line in the circuit, and the analysis considers both single-stuck-at faults on that line. A double attack targets two lines simultaneously, and the analysis is based on the four possible double-stuck-at faults. Columns 2 and 3 present the total number of attack scenarios under these assumptions, and also the number of scenarios for which the number of successful undetected attacks σ was 0. This is the case for a large number of attacks based on a single fault injection. If the attacker has capabilities to inject two faults simultaneously, most combinations of signal lines result in effects that are missed by the error-detecting mechanism with non-zero probability. The run times reported in Columns 4 and 5 are small and scale favorably with the number of #SAT instances.

Table II contains specific ASR and vul results for the Braun multipliers. Columns 2 and 3 contain averages over all attack targets (signal lines for single attacks and signal line pairs for double attacks). As expected, ASR (which excludes faults with no effect) has larger absolute values than vul. Columns 4 and 5 show the worst (largest) ASR and vul encountered during analysis. All circuits contain lines with a significant vulnerability according to both metrics.

Figure 2 shows the detailed break-down of attack targets in the different multiplier versions, according to classes defined with respect to their vul or ASR values. Lighter colors indicate the share of signal lines with greater vulnerability, darker colors indicate those with smaller vulnerability. Consider for example the *ASR - single attacks* bar of GF(16): we see more than half of the bar colored in the darkest color black. This means that more than half of the signal lines have an ASR value of 0%. The remainder of the bar consists of colors representing ASR values of 1 – 10%, 11 – 20%, and 21 – 30% respectively. This shows, that this 4-checkbit nonlinear code over GF(16) offers a much better protection than e.g. its GF(2) counterpart, where a lot more signals

Table I
Braun multipliers: Non-vulnerable locations and run-times

Braun multiplier	Victims with $\sigma = 0$ of all modeled targets		#antom CPU time to calculate all σ, δ	
Checker	single attacks	double attacks	single attacks	double attacks
lin parity	103/168	4100/14028	6 s	2058 s
GF(2)	91/168	4095/14028	5 s	1436 s
GF(4)	113/190	6328/17955	6 s	2527 s
GF(16)	153/230	11628/26335	5 s	2734 s

Table II
Braun multipliers: Average and worst-case ASR and vul

Braun multiplier	single attacks	double attacks	single attacks	double attacks
Checker	Average ASR over all victims		ASR of "weakest link" victim	
lin parity	9.8%	21.5%	52%	70%
GF(2)	29.7%	36.8%	97%	97%
GF(4)	19.1%	23.4%	70%	80%
GF(16)	6.0%	8.1%	50%	59%
	Average vul over all victims		vul of "weakest link" victim	
lin parity	4.6%	16.5%	50%	51%
GF(2)	15.1%	24.9%	50%	73%
GF(4)	10.8%	16.8%	50%	61%
GF(16)	4.0%	6.2%	50%	42%

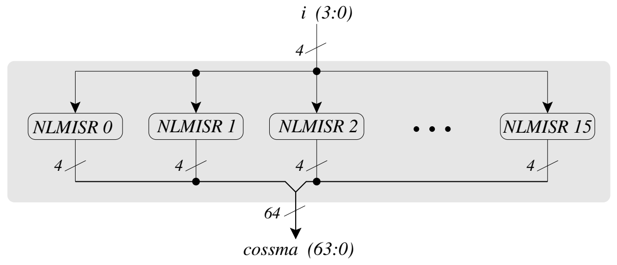

Figure 3. 64-bit COSSMA consisting of 16 4-bit NLMISRs

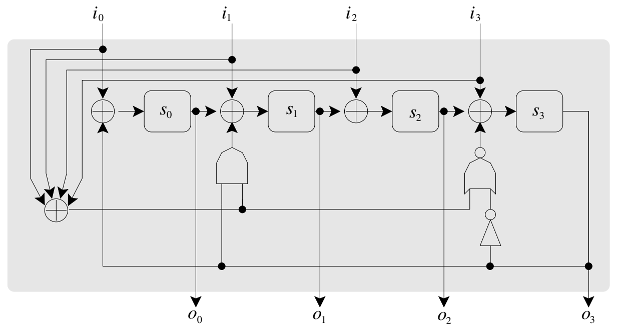

Figure 4. 4-bit NLMISR

have greater ASR values indicated by large portions of the bar colored in light colors. The superiority of GF(16) is shown by both ASR and vul values, likewise for single and double faults.

B. COSSMA

COSSMA (COmplete Set of State MAchines) [9] is a deterministic random bit generator designed for use in security-critical circuits [10]. It is protected against power analysis by special state encoding using identical Hamming weights for its states, and against fault attacks by a self-testing constant weight code checker [21].

The overall design of COSSMA is shown in Figure 3. It is composed of 16 identical non-linear multiple-input signature registers (NLMISRs), all sharing the same four input bits. One NLMISR is shown in Figure 4. It has four flip-flops and supports two primitive polynomials, $1 + x^3 + x^4$ and $1 + x + x^4$. Switching between polynomials is controlled by the XOR of all four input bits.

The 16 NLMISRs are initialized with pairwise different four-bit states. It can be shown that such 16 NLMISRs running in parallel will have 16 pairwise different states in every clock cycle. Since the internal states of all 16 NLMISRs are directly connected to COSSMA's primary outputs, the 64-bit COSSMA output is always a permutation of 16 four-bit sequences. As a consequence, the Hamming weight of the output is always 32. Moreover, when considering only the sub-set of 16 COSSMA outputs that are connected to a specific flip-flop within each NLMISR (e.g., the rightmost flip-flop in all 16 NLMISRs), the Hamming weight of these 16 outputs must be 8. This property is utilized to detect fault-based attacks.

It is possible to attach a 64-bit constant weight code checker [21] to the outputs of COSSMA to check that the number of logic-1 bits is 32; if it is not, a fault due to a

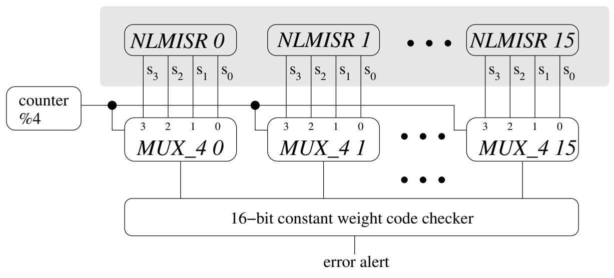

Figure 5. 16-bit checker connected to COSSMA via multiplexers

malicious attack or a different reason is detected. A lower-cost solution that checks only a sub-set of 16 output bits in each clock cycle has been proposed in [9]; the hardware implementation is shown in Figure 5. In clock cycle 0, the sub-set consisting of 16 output bits connected to bits s_0 of the 16 NLIMSRs is checked (they must contain eight logic-1 and logic-0 values), in clock cycle 1, bits s_1 of the NLMISRs are checked, and so forth. We call this architecture *4-bit single-checker*.

In this work, we considered the 4-bit single-checker, the complete checker (which evaluates all 64 outputs of COSSMA in all clock cycles), which we call *quadruple-checker*, and three "intermediate"-checker designs that evaluate two or three different NLMISR output bits per cycle.

To calculate vul and ASR for COSSMA, we leveraged the fact that all 16 NLMISRs are structurally identical. For this reason, it is sufficient to model the behavior of a single NLMISR without sacrificing accuracy. Moreover, instead of explicitly modeling the constant weight code checker and its output ED, we observe that any output deviation on a checked output is detected, with two important exceptions described next.

First, as COSSMA and all NLMISRs are sequential, the error may be detected in subsequent clock cycles. As suggested in [9], we assumed four clock cycles to be the error-detection period mentioned in Section III-B. Second, faults that impact two checked outputs with different polarities are not detected. For instance, a fault that flips s_0 from 0 to 1 and simultaneously flips s_1 from 1 to 0 is *not* detected (in clock cycle 0), if both s_0 and s_1 are checked. We addressed this problem by modifying the SAT instances to exclude such fault detections.

The results are reported in Tables III and IV, with layouts identical to Tables I and II, respectively. It can be clearly seen that the triple-checker is better than both double-checkers which in turn clearly outperform the single-checker.

Table III
4-bit-NLMISR: Non-vulnerable locations and run-times

COSSMA 4-bit NLMISR	Victims with $\sigma = 0$ of all modeled targets		#antom CPU time to calculate all σ, δ	
Checker	single attacks	double attacks	single attacks	double attacks
Single	0/20	0/190	52 s	2037 s
Double-Interlaced	13/20	48/190	18 s	649 s
Double-Neighbor	3/20	3/190	41 s	1374 s
Triple	13/20	51/190	34 s	1154 s

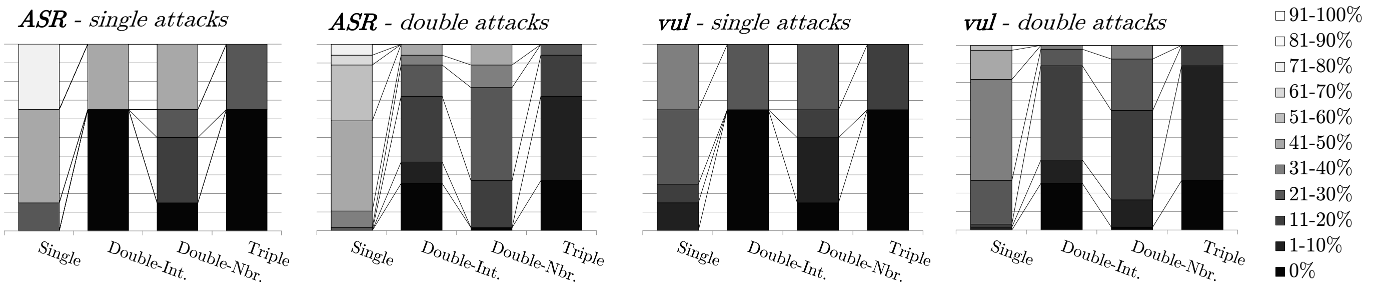

Figure 6. Distributions of $vul(f)$ and $ASR(f)$ for single and double stuck-at fault injections in the 4-bit NLMISR

Note that the NLMISR with the single-checker has not a single line with a perfect protection, i.e., $\sigma = 0$. The quadruple-checker, not shown in the table, detects all single and double attacks. Even a double attack cannot modify an output such that its Hamming weight remains constant over four clock cycles. The double-interlaced-checker is significantly better than the double-neighbor-checker despite identical hardware cost. This is further substantiated by the detailed break-down in Figure 6.

V. Conclusion

We demonstrated a new method to calculate the exact probabilities of physical faults yielding undesired behavior. The exactness is achieved by #SAT procedures that leverage recent advances in Boolean satisfiability solving. The method can be used to evaluate a circuit's vulnerability to fault attacks. We demonstrated the practical applicability of the method by analyzing several combinational and sequential designs protected by error-detecting circuitry. The computation time for all case studies was manageable and future work has to explore possible scaling boundaries. The analysis automatically identified properties of the circuit which were not known before, like the probability of an uninformed attack circumventing different checker implementations.

Acknowledgement

Parts of this work were supported by the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) under grants BE 1176-15/2, PO 1220-2/2, GRK 1103 and SFB/TR 14 AVACS.

Table IV
4-bit-NLMISR: Average and worst-case ASR and vul

COSSMA 4-bit NLMISR	single attacks	double attacks	single attacks	double attacks
Checker	Average ASR over all victims		ASR of "weakest link" victim	
Single	52.8%	50.5%	75%	75%
Double-Interlaced	17.5%	16.5%	50%	50%
Double-Neighbor	27.9%	26.2%	50%	50%
Triple	8.8%	8.1%	25%	25%
	Average vul over all victims		vul of "weakest link" victim	
Single	25.2%	34.4%	38%	56%
Double-Interlaced	8.8%	11.5%	25%	38%
Double-Neighbor	14.0%	17.9%	25%	38%
Triple	4.6%	5.6%	13%	19%

References

[1] H. Bar-El and D. Naccache, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[2] C. H. Kim and J. J. Quisquater, "Fault attacks for CRT based RSA: New attacks, new results, and new countermeasures," *Lecture Notes in Computer Science*, vol. 4462, pp. 215–228, 2007.

[3] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," *Lecture Notes in Computer Science*, vol. 6633, pp. 224–233, 2011.

[4] M. S. E. Mohamed, S. Bulygin, and J. Buchmann, "Improved differential fault analysis of Trivium," in *COSADE*, 2011.

[5] P. Jovanovic, M. Kreuzer, and I. Polian, "A fault attack on the LED block cipher," in *COSADE*, 2012.

[6] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *DSN*, pp. 93–101, 2004.

[7] L. Feiten, M. Sauer, T. Schubert, A. Czutro, E. Böhl, I. Polian, and B. Becker, "#SAT-Based Vulnerability Analysis of Security Components — A Case Study," in *Int'l Symp. on Defect and Fault Tolerance*, October 2012.

[8] V. Tomashevich, S. Srinivasan, F. Foerg, and I. Polian, "Cross-level protection of circuits against faults and malicious attacks," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, pp. 150 –155, june 2012.

[9] E. Böhl and P. Duplys, "Fault attack resistant deterministic random bit generator usable for key randomization," in *IEEE Int'l Online Testing Symp.*, pp. 194–195, 2011.

[10] E. Böhl and M. Ihle, "A fault attack robust TRNG," in *IEEE Int'l Online Testing Symp.*, 2012.

[11] T. Schubert, M. Lewis, and B. Becker, "antom — Solver Description," in *SAT Race*, 2010.

[12] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI* (C. Boutilier, ed.), pp. 399–404, 2009.

[13] A. Biere, "Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010," tech. rep., Institute for Formal Models and Verification, Johannes Kepler University, 2010.

[14] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT* (E. Giunchiglia and A. Tacchella, eds.), vol. 2919 of *Lecture Notes in Computer Science*, pp. 502–518, Springer, 2003.

[15] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker, "Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability Analysis," *International Journal of Parallel Programming*, vol. 38, pp. 185–202, June 2010.

[16] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[17] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, vol. 5, pp. 394–397, 1962.

[18] A. Czutro, M. Sauer, T. Schubert, I. Polian, and B. Becker, "SAT-ATPG Using Preferences for Improved Detection of Complex Defect Mechanisms," in *VLSI Test Symp.*, April 2012.

[19] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[20] J. Charles Roth and L. Kinney, *Fundamentals of Logic Design.* Cengage Learning, 2009.

[21] S. Tarnick, "Design of embedded constant weight code checkers based on averaging operations," in *IEEE Int'l Online Testing Symp.*, pp. 255–260, 2010.