SECURITY ANALYSIS OF AES USING FUNCTIONALITY FAULT MODEL

OLIVIER FAURAX

École des Mines de St Étienne - Site Georges Charpak, Laboratoire SESAM, Avenue des Anémones, 13120 Gardanne, FRANCE

faurax@emse.fr

TRAIAN MUNTEAN

Université de la Méditerranée, "Systèmes Informatiques Communicants", 13288 Marseille, FRANCE

traian.muntean@univmed.fr

Security of cryptographic circuits is a major concern. Smartcards are targeted by sophisticated attacks like fault attacks that combine physical disturbance and cryptanalysis. We propose a methodology and a tool (PAFI) to analyse the robustness of circuits under fault attacks using fault injection in simulation. The number of injection is reduced by taking into account the function of the latches in the whole circuit. We tested a circuit implementing the cryptosystem AES and showed that our approach reduces the number of fault injections to be performed (-80%). Moreover, most of the selected injection points are the ones that lead to known fault attacks (95%).

Keywords: security; fault attacks; fault injection.

1. Introduction

Security in small devices such as smartcards is critical due to the data they handle (ID, bank, pay TV, etc.) and the fact they are physically reachable by attackers.

A fault attack on an integrated circuit consists in a physical disturbance performed on one or more parts of it in order to analyze changes in the result. With results of related computations that differ only by one fault, one can be able to extract critical data by cryptanalysis. The first works about fault attacks are DFAs (*Differential Fault Analysis*) leading to attacks on RSA [1] and on DES [2]. Then, several attacks were performed on AES [345].

So, there is a need to handle this type of attacks before manufacturing the circuit and to find vulnerable part early in the design process. We suggest simulation-based fault injection that can be used before silicon IC, that generates reproductable faults in a less expensive way.

However, current circuit complexity (gates number, metal levels, etc.) does not allow simulation of all possible faults. Injection points number can be high, especially if all injection times are considered. This number increases too with fault multiplicity. For single faults, simulation time is linear in fault locations number and in fault times number.

This article is organized as follows : some related works on fault injection are commented in section 2. Then, we present our functional weighting methodology and associated tool (PAFI) in section 3. In the section 4, we discuss the results of this approach on the cryptosystem AES. To conclude, future works are described in section 5.

2. Related works

Fault injection tools and methods exist for about twenty years. We briefly present a non-exhaustive set of such tools.

MEFISTO [6] is one of the most important contribution as it was able to simulate multi-level faults on circuits described in VHDL, using saboteurs and mutants.

VERIFY [7] proposes a new fault injection technique by extending the VHDL signals syntax. This does not need recompilation as MEFISTO's mutants. Nevertheless, this technique needs a specific compiler that understands these extensions.

SINJECT [8] is a simulation-based injection tool that uses MEFISTO's mutants and saboteurs, adding support for non-synthetizable mixed-mode (VHDL/Verilog) circuit descriptions.

FITSEC [9] divides the circuit in order to emulate one part on a FPGA and simulates the remaining part. This technique drastically decreases the injection time by a factor of 100.

This tools enable one to conduct simulation-based fault injections without helping him to choose faults to reduce computation time.

Some other tools guide the user to a subset of possible injections. Two approaches have been proposed : to select injections that have a high probability of disturbing the system (to obtain a lower bound of the coverage rate) or to select representative injections (to obtain a coverage rate close to the real one).

DEPEND [10] reduces the number of faults by analysing the workload.

Güthoff and Sieh [11] analyse the execution of instructions without fault (golden run) on a processor to simulate faults only on registers used very often and only when theirs values make sense.

The technique of fault expansion [12] groups faults to simulate only one representative of each group. However, this method is only effective when each fault equivalence class is a significant portion of the fault population [13].

Our approach advances the state of the art because it is based on types of data stored in latches to deduce injection points of choice.

3. Methodology and tool

In this work, we investigate the criteria of injection points choice according to their functionality in the circuit. The target is to guide, *a priori*, the injection campaign to the faults that have a high probability to have an impact on the system behaviour that can be used to perform known DFAs. The methodology has been previously presented in [14] and will be summarized below. Then, a tool that implements it (PAFI) will be described.

3.1. *Functional weight of latches*

Our model is transient faults that ends in a change of the system's state. The global state of a circuit is stored in the set of its latches. So, the faults of our model are the ones that perturb these memory elements. To distinguish critical latches, our approach adds a weight to every latch in order to represent its impact on the circuit behaviour. This weight will help the user to choose important latches to perturb during the simulation.

The weight takes into account structural and functional factors of circuits. Structural factors can be deduced from the circuit netlist : logical cone associated to a latch, type of latch, etc. There are not treated in this paper. Functional factors deal with the use of the latch regarding the whole circuit : data type, critical value for circuit behaviour, etc. These informations must be supplied by the user as they cannot be deduced from the circuit description and are needed to compute the functional part of the weight.

We consider three critical data types : secret data, control data and outputs data. Secret data has to be isolated from the outside in order to avoid an information leakage. If there exists observable data whose value depends on secret data, it has to be protected. Indeed, a wrong result generated by a perturbation could allow to deduce the secret data by cryptanalysis. During the circuit execution, control data is the result of tests and drives the circuit behaviour. A perturbation on corresponding latches impacts security, for example by validating wrong authentication or by giving access to protected resources. Outputs data are easily readable. A perturbation that modify only a part of the result facilitates cryptanalysis. Dusart, Letourneux and Vivolo [4] show this on AES by modifying only a quarter of the result (4 bytes out of 16).

The first step of our weighting method (that is described in section 3.3) is about adding a weight to latches containing these three data types.

3.2. *Related perturbations*

Latches are connected by logical cones. If a perturbation on a latch changes the circuit behaviour, it is likely that perturbations on latches of its propagation path can induce a modification of the circuit behaviour.

Faults on these latches are called *related perturbations* and are divided in two groups : *anterior perturbations* and *posterior perturbations.*

3.2.1. *Anterior perturbations*

A fault injected on the input latches of the logical cone of B can have a consequence on the value of B. This latches are noted A_i and these perturbations are called *anterior perturbations* because B is on the propagation path of the A_i.

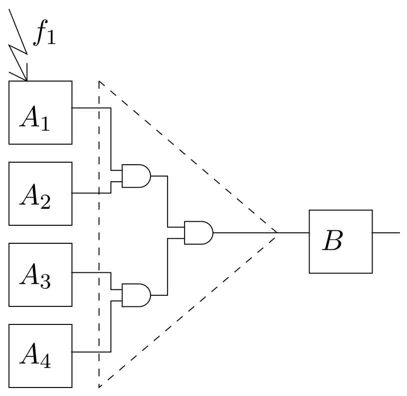

Fig. 1. Example of anterior perturbation

In the case of figure 1, a value change of one of the A_i latches is equivalent to a fault on the B latch. If B contains some secret data, an addressing fault is an anterior perturbation, as the behaviour is modified even if the latch containing the secret data was not directly affected by the fault. if B contains some control data, an anterior perturbation will modify the circuit behaviour if the test result is changed. It is also possible to have a circuit deadlock which is acceptable if it does not supply a wrong result.

An injected fault near the outputs will provide a partially wrong result that can lead to a differential analysis. That is why outputs data are considered as critical : anterior perturbations can leak informations.

Thus, the weight of a latch B can induce a cascade phenomenon on the weight of the previous latches on the propagation path (A_i).

3.2.2. *Posterior perturbations*

In the same way, perturbating output latches of the logical cones whose B is an input will probably change the behaviour of a circuit part. These latches are noted C_i and these perturbations are called *posterior perturbations* because the C_i are in the propagation path of B.

Figure 2 shows that a value change of one of the C_i latches is equivalent to a fault on the B latch for a part of the circuit (the one depending on C_1). For example, perturbing a latch containing a temporary result depending on secret data can reveal informations. Works of Yen and Joye [15] show that injection of a

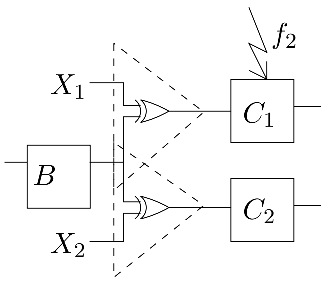

Fig. 2. Example of posterior perturbation

safe error (i.e. error that does not lead to a wrong result) on a temporary result during exponentiation can provide secret informations. A posterior perturbation on control data can completely change the behaviour of a circuit part. This can lead to an unexpected state and/or an illegal state according to the security specifications of the system. With posterior perturbations, a weight on a latch B can induce a weight on the latches of the logical cones depending on it (i.e. the C_i).

3.3. *Weight determination method*

Weights will decrease while going up or down the propagation paths. They will be combined in order to reveal potentially interesting injection points.

In the first step, the user provides the initial couple list {latch, weight} for the latches dealing with secret, control or outputs data. This is the list of all critical latches and their associated weights.

From these informations and the circuit netlist, the second step is to compute the weight for related latches (A_i and C_i). This couples {latch, weight} are then added to the list. This second step is iterated enough to obtain a stable list (i.e. that is not modified between two iterations).

Finally, it provides as a result a list of couples {latch, weight} that guides the user's choice of fault injection to simulate for the circuit validation. The computation is more formally described in [14].

This method is generic as there is no hypothesis on the type of the considered circuit. Moreover, contrary to the first step that needs user intervention, the second step can be entirely automatic. PAFI, a tool that implements this methodology, is presented in the next section.

3.4. *Prototype of Another Fault Injector (PAFI)*

The purpose of PAFI (Prototype of Another Fault Injector, figure 3) is to use unmodified modelization of the circuit to take into account very accurate details of the circuit (gates, delays). Our approach is to help the user focusing on most relevant faults and reducing the injection time by handling repetitive tasks.

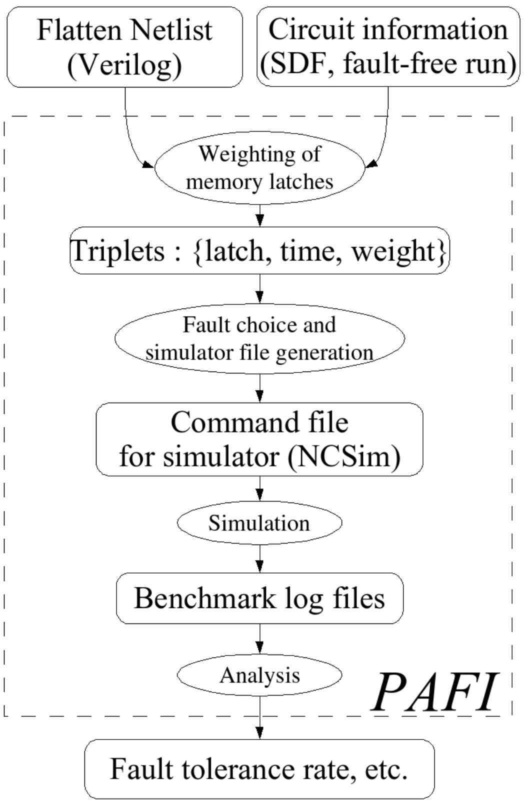
Flatten Netlist (Verilog)
Circuit information (SDF, fault-free run)
Weighting of memory latches
Triplets : {latch, time, weight}
Fault choice and simulator file generation
Command file for simulator (NCSim)
Simulation
Benchmark log files
Analysis
PAFI
Fault tolerance rate, etc.

Fig. 3. PAFI

The circuit is defined in a Verilog netlist. The purpose of the first step is to get user-defined functionnal weight, parse the netlist and find the latches and their logical cone. Then, the weight for each latch is computed with the method of the previous section.

At the same time, the netlist can be simulated without injecting fault in order to provide information about the fault-free run. This information can be used to take into account dynamic activation of regions of the circuit, discarding faults occuring in inactive regions. The faults become dependent of the time: that is why couples {latch, weight} become triplets {latch, time, weight} on figure 3. Faults depending on time are not used in this paper and will be adressed in future works.

Then, the possible faults and their weights are presented to the user that can choose the amount of injection to perform, depending on the time he can spend and the accuracy he needs.

From the chosen list of faults, our tool generates a command file for the simulator used (Cadence NCSim). This command file describes the simulations and the fault injections to be done. As we do not modify the circuit, the only versatile fault injection possibility is to use the built-in command of the simulator. This part of the tool is specific to the simulator.

However, our tool is designed to be modular : each step is performed by an independant program whose results are stored in an XML files. This allows to easily replace one module to adapt it to another simulator, for example.

During each simulation, the circuit under test is manipulated by a benchmark that provide input signals, check output signals and log results on files. The benchmark is written in VHDL and use the textio package to write files.

A custom analyser reads these log files and produces the expected computation of safety rate and user-defined analysis. The outcome can be graphically displayed, for example using Gnuplot (Cf. figure 4).

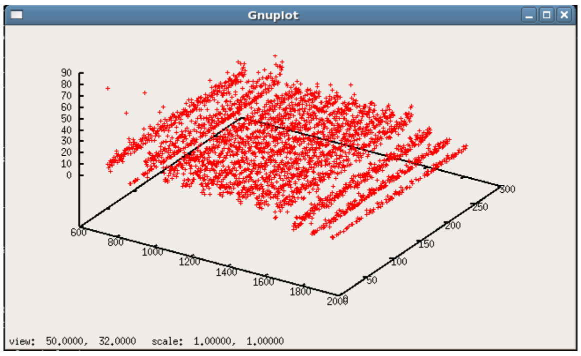

Fig. 4. Output of PAFI using Gnuplot

4. Case study : AES

4.1. *Implementation of AES*

AES [16] is a well-known cryptographic block cipher. It is a substitution-permutation network that takes as input a 128-bit plain text of and a 128-, 192- or 256-bit cipher key.

In this paper, we consider the variant with a 128-bit key length. This AES is made of 10 rounds where the input is scrambled using a round key that is computed from the precedent round key. The first input is the plain text and the first round key is the cipher key.

The part of the round that deals with the data can be divided in four steps :

AddRoundKey, SubBytes, ShiftRows, MixColumns. The 128 bits of data are viewed as a 4x4 matrix of 16 bytes. AddRoundKey computes the XOR between the input and the round key. SubBytes performs a substitution over $GF(2^8)$ of each byte. ShiftRows operates on each 4-byte row : it cyclically shifts the 4 bytes of a row by 0, 1, 2, 3 respectively. MixColumns applies a linear transformation over $GF(2^8)$ on each 4-byte column.

The MixColumns operates on columns. ShiftRows makes a column to become a diagonal. These two steps provide a diffusion from a byte over all the bytes of the data is 2 rounds (2 MixColumns + 1 ShiftRows).

Our AES circuit takes four 32-bit words for the data and four 32-bit words for the cipher key. Then it calculates the result and output it as four 32-bit words.

Our benchmark is to input known data and cipher key, to log the output (if any) and to compare it with the expected result.

Using a clock at 10 MHz, data and key are loaded in 750 ns. The computation occurs between 750 ns and 1950 ns, with a clock cycle of 100 ns. The results is expected at 1950 ns (first 4 bytes) until 2350 ns.

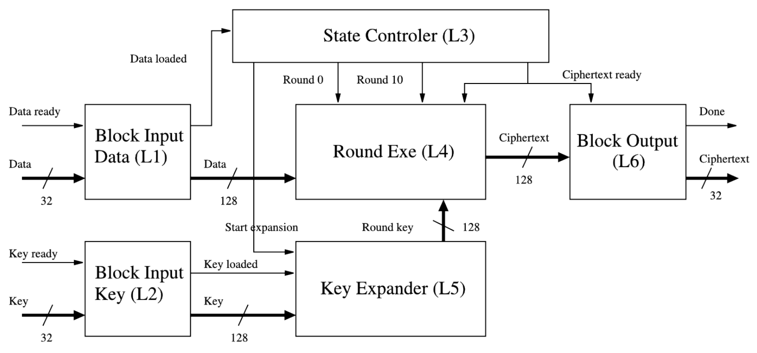

Fig. 5. Architecture of AES

The AES is composed of six parts labeled L1 to L6, shown on figure 5.

- L1 and L2 are input blocks for the plain text and the cipher key. Data and key are 128-bit long but the input is 32-bit wide, so L1 and L2 have to buffer the data and key during the loading process.
- L3 is a state controler that drive the synchronisation between the other parts.
- L4 contains the four AES sub-operations on data.
- L5 is a block that computes round key from the key provided.
- L6 outputs the 128-bit long result into four 32-bit words in a similar way

as L1 and L2 for the input.

4.2. *Sensitive latches of AES*

The key and all the round keys are secret data because with one round key, it is possible to compute the initial cypher key. During the computation, the current round key is stored in 128 latches of L5. The AES must be safe against attack with known clear texts where the attacker knows the text that is ciphered. That us why the data is not considered to be secret.

There are several control latches in the circuit. Most of the parts of the circuit has a state machine. L1 and L2 needs to count the number of 32 bits word that has been memorized. L3 is the state machine that drives the whole circuit. L5 contains a counter of the round. L6 has a state machine to convert the 128-bit result to four 32-bit words.

The output of the circuit are 128 of the latches of the output buffer (L6). They will be available to the user at the end of the computation.

The initial weight of these latches has been set to 1. When propagated through logical cones, the propagated weight has been reduce by a factor of 0.8. This means that A_i and C_i will get the weight of B reduced by a factor of 0.8. When two or more latches induce several propagated weight on a latch, they are added. If B_1 and B_2 are connected to A, the weight of A will be the sum of the propagated weight of them, that is, $0.8 * weight(B_1) + 0.8 * weight(B_2)$.

At the end of the computation, several latches have a high weight:

- the latches of L4 are in propagation path of the secret round key of L5 and of the state machine of L3
- the state machine of L5 that is in the propagation path of L2 and L3
- the state machine of L3 that is in the propagation path of L1 and that propagates on L6

In order to evaluate the speedup and soundness of our approach, two injection campaigns has been performed: one exhaustive injection campaign and one only on the sensitive latches of the AES.

4.3. *Exhaustive injection on AES*

We injected on the 665 latches of the design during the computation that take 13 clock cycles. This leads to 13*665 simulations, injecting one bit-flip for each simulation.

The results are shown on figure 6. 60.2% of injected faults do not lead to faulty results and 27.6% of faults induce 16-byte faulty outputs.

The faults injected at the beginning of the computation (750 and 850 ns) produce errors that are dependent of the design of the AES and are not relevant here due to the fact that the first round of the AES starts at 950 ns.

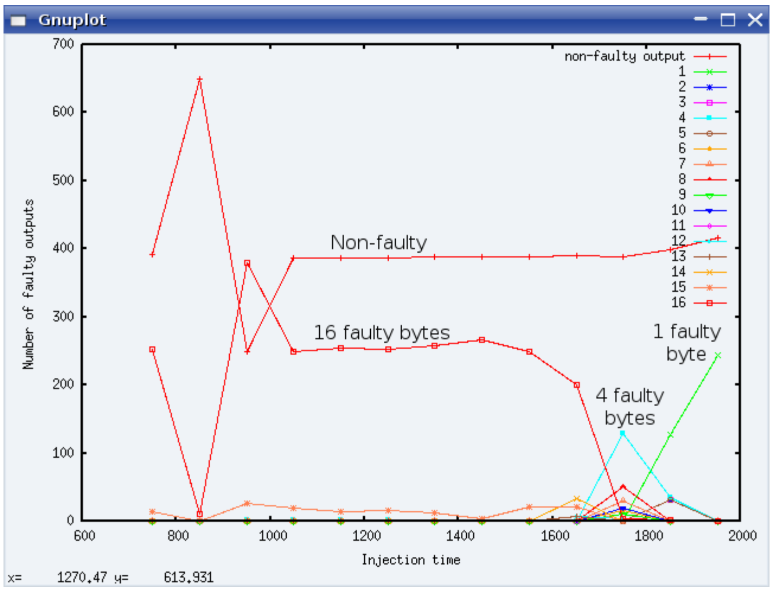

Fig. 6. Faulty output bytes on AES

Between 950 ns and 1650 ns, results are fault-free or 16-byte faulty outputs. Some injections near the end of the computation generate 4-byte errors: this verify the diffusion properties of AES (4 bytes by round, all bytes after 2 rounds).

The outputs with only one faulty byte at the end are due to an injection just before the output of the result.

4.4. *Injection in sensitive part using PAFI*

The purpose of the injection in sensitive part is to see if the faulty results can be used to recover part of the key.

Injections in the 128 latches of L4 produces faults on 1 bit (injection during last round, 1850ns), 4 bytes (last but one round, 1750ns) or 16 bytes (other cases). The faults on 4 bytes, that can also be seen on figure 6, can be exploited with fault attacks of [5].

The injection on the state machine of L5 do not lead immediatly to known fault attacks. However, some of these injections perturbate the round counter and can lead to a fault on a line of the round key. When one of these faults is performed during the computation of one of the two last rounds key, the computation shows differences that can be exploited by a variant of [3].

Injections of fault in the state machine of L3 can reveal lots of useful information for an attacker or produce output that cannot be used to perform DFAs. When injecting fault on the counter of L3, one can change the number of rounds. One of these fault can reduce the number of round to 0, and the circuit output the plain text XORed with the key. On the opposite, if the fault is injected on a latch set to 1, the number of rounds will increase and it will be harder to use the output.

The results are presented in table 1. The number of injections has been reduced from 665 to 136 (-80%). In the selected injections, a significant amount of them are exploitable. The methodology helps in finding the injections that lead to fault attacks and discards fault-free injections.

	Exhaustive	Functionnal weight
Total	665	136
Fault-free	388 (58%)	0
Exploitable	130 (20%)	129 (95%)

Table 1. Number of injection in latches

5. Conclusion

In this work, we investigate the potential speedup and soundness of a fault injection methodology. This methodology reduces the fault injection space by taking into account the functionnality of the possible fault injection points. The associated tool, PAFI, has been used to test the approach on the cryptosystem AES.

The number of injections, compared to an exhaustive injection, is divided by 5, while keeping most of the fault injections that lead to known DFAs.

Further studies will be focused on combining physical and functional fault models. Then, we will use dynamic data to make our fault model more relevant and define a multiple fault model.

References

1. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. *Lecture Notes in Computer Science*, 1233:37–51, 1997.
2. Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
3. Christophe Giraud. Dfa on aes. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.
4. P. Dusart, G. Letourneux, and O. Vivolo. Differential fault analysis on a.e.s. Cryptology ePrint Archive, Report 2003/010, 2003. http://eprint.iacr.org/.
5. Gilles Piret and Jean-Jacques Quisquater. A differential fault attack technique against spn structures, with application to the aes and khazad. In *Cryptographic Hardware*

and Embedded Systems – CHES 2003, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

6. Eric Jenn, Jean Arlat, Marcus Rimen, Joakim Ohlsson, and Johan Karlsson. Fault injection into VHDL models: The MEFISTO tool. In *Proceedings of the 24th International Symposium on Fault Tolerant Computing, (FTCS-24), IEEE, Austin, Texas, USA*, pages 66–75, 1994.
7. Volkmar Sieh, Oliver Tschäche, and Frank Balbach. Verify: Evaluation of reliability using vhdl-models with embedded fault descriptions. In *FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 32–36, Washington, DC, USA, 1997. IEEE Computer Society.
8. Hamid R. Zarandi, Seyed Ghassem Miremadi, and Alireza Ejlali. Dependability analysis using a fault injection tool based on synthesizability of hdl models. In *DFT '03: Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 485–492. IEEE Computer Society, 2003.
9. A. R. Ejlali, G. Miremadi, H. R. Zarandi, G. Asadi, and S. B. Sarmadi. A hybrid fault injection approach based on simulation and emulation co-operation. In *DSN-2003: Proceedings of the 2003 International Conference on Dependable Systems and Networks*, pages 479–488, Washington, DC, USA, 2003. IEEE Computer Society.
10. Kumar K. Goswami, Ravishankar K. Iyer, and Luke Young. Depend: A simulation-based environment for system level dependability analysis. *IEEE Trans. Comput.*, 46(1):60–74, 1997.
11. Jens Güthoff and Volkmar Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *FTCS '95: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 196–206, Washington, DC, USA, 1995. IEEE Computer Society.
12. D.T. Smith, B.W. Johnson, J.A. III Profeta, and D.G. Bozzolo. A method to determine equivalent fault classes for permanent and transient faults. In *Proceedings of the IEEE Reliability and Maintainability Symposium, 1995*, pages 418–424, Washington, DC, USA, 1995. IEEE Computer Society.
13. Wei Wang, Kishor S. Trivedi, Babubhai V. Shah, and Joseph A. Profeta III. The impact of fault expansion on the interval estimate for fault detection coverage. In *FTCS '94: Proceedings of the 24th International Symposium on Fault-Tolerant Computing (FTCS '94)*, pages 330–337, Austin, TX, USA, 1994. IEEE Computer Society.
14. Olivier Faurax, Laurent Freund, Assia Tria, Traian Muntean, and Frédéric Bancel. A generic method for fault injection in circuits. In *IWSOC 2006: Proceedings of the 6th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 211–214. IEEE Circuits and Systems Society, 2006.
15. Sung-Ming Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Comput.*, 49(9):967–970, 2000.
16. J. Daemen and V. Rijmen. Aes proposal: Rijndael, 1998.