Chapter 2
Fault Attacks, Injection Techniques and Tools for Simulation

Roberta Piscitelli, Shivam Bhasin and Francesco Regazzoni

2.1 Introduction

Embedded systems pervaded our live since few years. The applications where they are used are often safety critical, such as public transports or smart grids control, or handle private and sensitive data, such has medical records or biometrics information for access control. This trend is expected to even increase in the near future, when a large amount of embedded devices will be connected to the so called Internet of Things (IoT). If, on one side, the level of interoperability and connectivity which will be reached by the objects in the IoT will allow to offer a large variety of services, to increase the efficiency and to reduce the costs, on the other side, the envisioned applications require the device to include security functionality to guarantee the confidentiality of the processed data and the security of the overall infrastructure.

Designers anticipated these needs by augmenting several devices with state of the art cryptographic primitives: embedded processors included instructions to quickly encrypt and decrypt data and a number of low-cost accelerators were designed to boost the performance of secure protocols implemented in wireless sensor nodes. However, robust and mathematically secure cryptographic primitives are not sufficient to guarantee the security of embedded devices. In the past, cryptographics algorithm have been conceived to be robust only against mathematical attacks: their structure is realized to resist, among other, to linear and differential cryptanalysis, they were requested to resist brute force attacks, also considering the progress of the

R. Piscitelli (✉)
EGI.eu, Science Park 140, Amsterdam, The Netherlands
e-mail: roberta.piscitelli@egi.eu; roberta.piscitelli83@gmail.com

S. Bhasin
Temasek Labs@NTU, 21 Nanyang Link, Singapore 637371, Singapore
e-mail: sbhasin@ntu.edu.sg

F. Regazzoni
ALaRI-USI, via Buffi, 13, 6900 Lugano, Switzerland
e-mail: regazzoni@alari.ch

N. Sklavos et al. (eds.), *Hardware Security and Trust*,
DOI 10.1007/978-3-319-44318-8_2

technology, and the hardness of the computational problem involved should have been capable of guaranteeing long-term security.

The situation changed in the last to decades, with the advent, the rise and the develop of a novel form of attacks, called *physical attacks*. These attacks, instead of addressing the mathematical structure of the algorithm, try to extract information about the secret key exploiting the weaknesses of the implementation of the algorithm itself. To recover the secret data, the adversary can exploit either an additional information leaked by the device during the computation (for instance the power consumed by the device) or can actively induce an anomalous behavior capable of leaking secret information.

Physical attacks are particularly dangerous for embedded systems, as they are, potentially, "in the hand" of the adversary, which thus has the whole control over them. Physical attacks are usually divided into two classes: *passive attacks* and *active attacks*. Among the first ones, the most notable one is power analysis [1], in which the adversary measures the power consumed by certain number of encryptions computed using a known plaintext, makes an hypothesis on a small portion of the secret key, and used the previously collected power traces to verify the correctness of the hypothesis. Nevertheless, the time [2] needed to complete an encryption, the electromagnetic emission [3] of a device or even the photons emitted by transistors [4] were successfully used to recover secret data.

During active attacks, the adversary does not limit himself to the observation of information leakage but actively tampers with the target device. The most common form of active attack is fault injection. In this attack, the adversary forces the device to perform erroneous operations and he exploits the relation between the correct results and the incorrect ones to infer the secret key (or to significantly reduce the possible key space). Fault injection consists of two parts: the first is the injection of the fault into the device, in which the target device is induced into an anomalous behavior, the second is the attack itself, in which the erroneous output is used to extract secret information.

Fault injection attacks are extremely dangerous because they require a limited amount of time to be carried out and because they were proven to be effective even when performed with an extremely inexpensive equipment. Barenghi et al. [5], for instance, showed how, by underpowering an ARM9 embedded processor, it was possible to induce a number of errors sufficient to successfully attacks software implementations of the AES and the RSA algorithms. A similar approach was used also to attack an ASIC implementation of the AES algorithm.

Robustness against fault attacks is usually evaluated in laboratories, where a manufactured device is tested by mounting a number of known attacks. However, this is not the best solution for designers which needs to timely apply the proper countermeasure against these attacks. Even if the final prove of resistance can be obtained only with the direct evaluation of the manufactured device, it would be more effective to have an initial exploration of the resistance against fault attacks at design time. This, however, requires to have tools capable of simulating the behavior of a device under attack, at the needed resolution, and to have a methodology to compare different countermeasures.

This paper addresses the problem of fault attacks. First, we survey the most common methods used to inject the faults, highlighting the potentialities of the method and its cost. Then we summarize the type of fault attacks previously presented in literature, finally we introduce the design tools which can be used for simulating fault attacks and we discuss to which extend they are suitable for evaluating the sensitivity of a device against fault attacks.

2.2 Fault Injection Techniques

Fault attacks are active attacks, which need an adversary to induce errors into the target device, using some tampering means. This tampering can be accomplished in several ways, as extensively discussed in literature and illustrated in Fig. 2.1. In general tampering means or fault injection techniques are classified in two broad categories, i.e., global and local. Global fault injection are, in general, low-cost techniques which create disturbances on global parameters like voltage, clock, etc. The resultant faults are more or less random in nature and the adversary might need several injection, to find required faults. On the other hand, local techniques are more precise in terms of fault location and model. However, this precision needs expensive equipment.

The kind of fault injected can be defined as fault model. The fault model has two important parameters, i.e., location and impact. Location means the spatial and

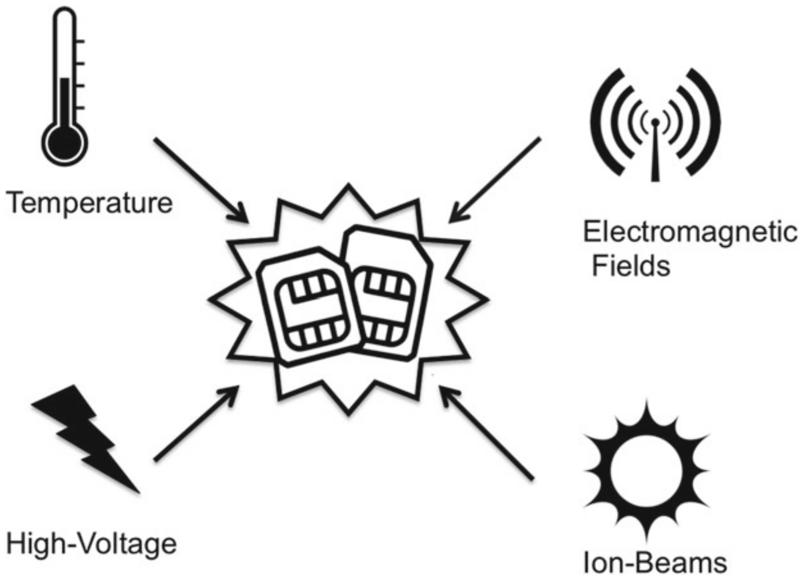

Fig. 2.1 An illustration of fault injection techniques

temporal location of fault injection during the execution of target algorithm. Depending on the type and precision of the technique, location can be at the level of *bit*, *variable* or *random*. Coming to the impact of fault, it is the affect on the target data. Commonly known fault injection impacts on target data can cause *stuck-at*, *bit-flip*, *random-byte*, or *uniformly distributed random value*. In the rest of the section, we summarize the most common techniques for injecting faults, highlighting for each of them, the main characteristics, the complexity, the cost, and the erroneous behavior introduced in the target device.

2.2.1 Fault Injection Through Power Supply

Embedded systems are often either battery operated or connected with an external power supply (latter, for instance, the case of smart cards). In this context, a natural and very inexpensive way to induce a malfunctioning is to alter the power supply coming from an external source. This alteration or disturbance can be performed in two different ways: *underfeeding* or *voltage glitch*. The typical effects caused by alteration of power supply are setup-time violation: flip-flops are triggered before the input signals reach a stable and correct value. Such fault techniques can be used to skip the execution of an instruction in the microprocessor code. The temporal precision of the fault injection depends on the accuracy of the voltage drop in duration and its synchronization with the target device. By underfeeding the target for a prolonged period, the adversary is able to insert transient faults with single-bit faults appearing first and increasing in multiplicity as the feeding voltage is further lowered. This requires only basic skill and can be easily achieved in practice without leaving evidence of tampering on the device. Alterations of power supply was exploited by Barenghi et al. [5] to attack software implementations of the AES and the RSA algorithms. The authors demonstrated that an embedded processor, in that case an ARM9, can be successfully attacked using very inexpensive equipment. Several other works demonstrated the feasibility of attacking the AES algorithm with fault induced by underfeeding. A possible example is the work of Selmane et al. [6]. The drawback of this technique is the time precision: the adversary can not control the exact time in which the fault happens. As a results, he must be capable of selecting the right faults and discard the ones which cannot be used during attacks.

2.2.2 Fault Injection Through Clock

Fault induced with clock are similar to fault induced with power supply. The typical target of these attacks are devices, such as smartcards, which use an external clock signal. The adversary can supply these devices with an altered clock signal, which contains, for instance, a clock pulse which is much shorter than what is expected in the normal clock. Pulses generated in this way are called clock glitches, are much shorter than the deviations normally tolerated by the smart cards [7, 8], and they can cause a setup-time violation or the skipping of instructions during the execution of a program [9]. The errors are transient and the device does not incur any damage; thus it is possible to induce faults at will without leaving any evidence of tampering. Overclocking the target for a prolonged period can also be used to inject transient faults. Much like underfeeding, overclocking leads to single-bit flips initially and higher multiplicity of faults at higher frequencies. This type of attack can be carried out with relatively low cost equipment.

2.2.3 Fault Injection Through Temperature

Electronic circuit are manufactured to work only on specific operating conditions identified by an upper and a lower temperature thresholds. Outside this range, there is no guarantee that the circuit continues to work as expected. Possible effects can be the random modification of the content of the memory cells or a limited functionality of the device. An adversary can induce faults into the device by exposing it to temperatures outside this range or by stressing it in order to increase its temperature. An example of this approach is the one proposed by Govindavajhala and Appel [10], where the chip was heated by executing a large number of load and store operations. The authors report a thermal fault induction attack against the DRAM memory chips of a common desktop PC. Using a 50 W light bulb and controlling its distance from the target, the authors reported around ten flipped bits per 32-bit word at 100 °C. Since the precision of the heating element is coarse grain, the controllability of faults is limited and impact is global. Moreover, excessive heating can cause permanent damage to the target.

2.2.4 Fault Injection Through Light

Optical attacks are a semi-invasive fault injection attack because they require the decapsulation of the target device, which is then hit with a light pulse or high-intensity laser. The light pulse or laser can be directed to the front or to the back side of the chip, depending on the type of the attack and the difficulty involved in each approach. In fact, with modern technologies, it can be difficult to reach the target cell from the front side of the chip, due to several metal layers of the chip itself. In order to obtain very precisely focused light pulses, the light emitted from a camera flash is concentrated with the aid of a precision optical microscope by applying it to the eyepiece after the device under attack has been carefully placed on the slide holder. In order to avoid over-irradiation of the device, which might lead to permanent damage to the circuit, care must be taken in selecting an appropriate magnification level for the microscope lens. The accuracy with light pulse is limited as the pulse gets scattered. Laser provides higher accuracy. It can also be used to attack through the back-substrate of the chip using near-infrared wavelengths. Optically induced fault require expensive equipment to be carried out. High-energy radiations like UV lamps can also create permanent faults. Nevertheless, they can be very precise both in the target, as it was demonstrated by Skorobogatov [11], and in time. This precision allow to change values at the granularity of a single RAM cell or even at of a register [12]. Light-based faults need medium to high expertise depending on the equipment. Laser is also capable of damaging the chip with over-radiation. The faults can be precise in time and space and also multiple faults can be injected using laser.

2.2.5 Fault Injection Through Electromagnetic Fields

Electromagnetic pulses can cause the change in the memory content or the malfunctioning of the device. This is due to the so called Eddy currents, which are created using an active coil [13]. Electromagnetic pulses can induce a fault which is extremely localized and precise (up to the level of a single bit), while the equipment needed to carry out this attack can be relatively cheap. Furthermore, the attack can be carried out without depackaging the chip. However, the adversary is required to know the details of the layout of the chip in order to identify the precise point of attack. The EM pulse can either be injected over the power trail of the chip, uniformly affecting the whole attacked device, or a smaller EM coil can be used to induct an additional current on a specific part of the circuit. The idea of this type of fault injection has been introduced by Quisquater and Samyde [13]. The authors demonstrated that it is possible to alter the computations of a cryptographic algorithm using an Electromagnetic probe and a camera flash (used to induce high voltage into the coil of the probe). This technique does not work efficiently with chips that employ grounded metal packaging (usually for heat sinking purposes), that also act as an EM shield, which needs the adversary to perform decapsulation (Table 2.1).

2.2.6 Fault Injection Through Focused Ion Beams

Focused Ion Beams (FIB) are a very expensive way to inject fault into the device, however they allows the attacker to arbitrarily modify the structure of a circuit. The adversary can cut existing wires, add connections, and operated through different layers. The capability of these tools are demonstrated in Torrance et al. [14], where the authors showed the reconstruction of a bus without damaging the contents of the memory. FIB equipment is expensive and needs high technical expertise, but the precision is extremely high. Current FIBs are able to operate with precision up to 2.5 nm, i.e., less than a tenth of the gate width of the smallest transistor that can currently be etched.

Table 2.1 Comparison of fault injection techniques

Technique	Type	Target			Precision		Equipment	Expertise
		Decapsulation	Invasive	Design details	Time	Space	Costs	
Clock glitch	Global	No	Non-invasive	Required	High	Low	Low	Moderate
Voltage glitch	Global	No	Non-invasive	Partial needed	Moderate	Low	Low	Moderate
Underfeeding	Global	No	Non-invasive	No	None	High	Low	Low
Overclocking	Global	No	Non-invasive	No	None	High	Low	Low
Temperature	Global	No	Semi-invasive	Required	None	Low	Low	Low
Global EM pulse	Global	No	Semi-invasive	Partial needed	Moderate	Low	Low	Moderate
Local EM pulse	Local	Sometimes	Semi-invasive	Required	Moderate	Moderate	Moderate	Moderate
Light pulse	Local	Yes	Semi-invasive	Required	Moderate	Moderate	Moderate	Moderate
Laser beam	Local	Yes	Semi-invasive	Required	High	High	High	High
Light radiation	Local	Yes	Invasive	No	Low	Low	Low	Moderate
Focused ion beam	Local	Yes	Invasive	Required	Complete	Complete	Very high	Very high

2.2.7 Comparison of Fault Injection Techniques

The previously introduced fault injection techniques have several parameters to define its application. The most important parameters are compared in Table 2.2.

2.3 Fault Attacks

Faults attacks have gained popularity as a serious threat to embedded systems over the last few years. Attacks can target a specific algorithm or generically modify the program flow to attacker's advantage. In the following, we refer the classification of attacks and the organization proposed by Karaklajić et al. [15]. In particular, three distinct classes of fault attacks are identified for embedded system.

2.3.1 Algorithm Specific Attacks

Fault attacks can be designed to exploit specific weaknesses of the target algorithm which are introduced by the injection of a fault. Several attacks targeting a large number of algorithms were presented in the past, the most common being the attacks against AES, RSA, and ECC.

Bloemer et al. in [16] proposed an attack on AES which exploit the change of a single bit after the first key addition. However, this attack can successfully recover a complete key only when the adversary has the possibility to inject a fault at a very precise timing and at a very specific position.

The security of asymmetric cryptosystems relies on problems which are mathematically hard to be solved. Fault attacks can be designed to weaken the problems and thus weaken the security of the algorithm based on that. A common target for such attacks are public-key cryptography algorithm, in particular RSA and ECC, as they are widely used for authentication, digital signature, and key exchange. RSA is based on exponentiation using a square and multiply (S&M) routine, while ECC is based on point-scalar multiplication using a double and add (D&A) routine. Both (S&M) and (D&A) have similar structure where the set of executed routine depends on the value of the processed bit of the secret.

Proposed attacks to these cryptosystems requires the attacker to change the base point of an ECC. As a result, the scalar point-multiplication will be moved to a weaker curve. The use of weak curve will make the problem of solving the discrete-logarithm problem of ECC manageable, and thus will lead to the recover of the secret [17]. The same attack can be carried out if the attacker manage to supply wrong parameters for the curve [17]. Other attacks proposed in the past showed that faults can be exploited to control few bits of the secret nonce in DSA and, which ultimately allows to recover the whole key [18]. Pairing algorithm are also vulnerable

Table 2.2 Comparison of fault injection mechanisms

Mechanism		Cost	Controllability	Trigger	Type	Repeatability	Injection time	Risk of damage	Runtime injection
Simulator	Static analisys								No
	Execution based	Med.	High	Yes	App. [lim.]	High	Med.	No	No
	Trace-based				Sys. [lim.]				Yes
	Transistor level								No
Software sim.	Compile time	Low	Low	Yes	App.	High	Low	No	No
	Runtime				OS				Yes
Low-level VM sim.		Med	High	Yes	OS Sys [lim].	Med.	High	No	Yes
Emulation		High	Med.	Med.	App. [lim.]	Med.	High	Yes	Yes
					Sys.				
Hardware		High	Med.	Med.	Sys.	Med.	High	Yes	Yes

to faults [19]: it was demonstrated that by modifying the loop parameter of a pairing algorithm, the secret point can be recovered.

2.3.2 Differential Fault Analysis

Differential fault analysis (DFA) is one of the most common exploits for cryptographic algorithms. The main idea behind DFA is the following: faults are injected in a device to alter the computation of the target algorithm. When a fault injection is successful, it is reflected on the output ciphertext. The attacker also computes the correct ciphertext for the same inputs. After collecting several correct/faulty ciphertext pairs, differential cryptanalysis techniques are applied to discard key candidates. With precise faults injection, the attacker can reduce the key candidates to a unique solution. DFA was introduced as a threat to symmetric block ciphers by Biham [20] and then extended to other symmetric primitives [21, 22]. The attack presented by Giraud [23] targets the state during the last round, and assumes that the fault alters only a single bit of the state, prior to the last SubBytes operation. The attack is thus applicable only to the last AES round that does not include the MixColumns operation, where the a single byte difference in the state will not spread to other bytes. Attacks proposed in the past also target a single bit or a single byte in the key-expansion routine of the target algorithm in order to recover the whole secret key [24].

DFA has also been reported on public-key algorithms like RSA [25]. Bellcore attack [26] attempts to factor the modulus n by injecting faults in exponentiation phase using the Chinese remainder theorem. DFA was further extended to fault sensitivity analysis (FSA [27]), which exploits the physical characteristics of faults like timing instead of faulty outputs.

2.3.3 Tampering with the Program Flow

Faults can also be injected to change the flow of an executed software code [28–31]. A fault in program counter is an obvious way to modify the program flow. For code implementing cryptographic algorithms, change in the program flow can be a security threat. Often an instruction skip lead to a wrong computation of critical portion of the algorithm, significantly weakening it. A notable example of this involves the exponentiation algorithm [28]. Similarly, fault on loop counter or branch selection were demonstrated to leak secret point of pairing schemes [30] or to reduce the number of encryption round in symmetric key algorithms [20], thus enabling classical cryptanalysis. Often fault attack countermeasures are based on redundancy with sanity check. Some efficient attacks simply try to skip the sanity check to bypass a deployed countermeasure [32]. This approach can be distinguished in two main categories: during compile time or during runtime. To inject faults at compile time, the program instruction must be modified before the program image is loaded and executed. Rather than injecting faults into the hardware of the target system, this method injects errors into the source code or assembly code of the target program to emulate the effect of hardware, software, and transient faults. During runtime, a mechanism is needed to trigger fault injection. Commonly used triggering mechanisms include, timeouts, exception/traps and code insertions. Although this approach is flexible, it has its shortcomings: first of all, it cannot inject faults into locations that are inaccessible to software. Second, the poor time-resolution of the approach may cause fidelity problems. For long latency faults, such as memory faults, the low time-resolution may not be a problem. For short latency faults, such as bus and CPU faults, the approach may fail to capture certain error behavior, like propagation.

2.4 Fault Injection Simulators and Their Applicability to Fault Attacks

In simulation-based fault injection, the target system as well as the possible hardware faults are modeled and simulated by a software program, usually called fault simulator. The fault simulation is performed by modifying either the hardware model or the software state of the target system. This means that the system could behave as if there was a hardware fault [13]. There are two categories of fault injection: runtime fault injection and compile-time fault injection. In the former, faults are injected during the simulation or the execution of the model. In the latter, faults are injected at compile time in the target hardware model or in the software executed by the target system. The advantage of the simulation-based fault injection techniques is that there is no risk to damage the system in use. In addition, they are cheaper in terms of time and efforts than the hardware techniques. They also have a higher controllability and observability of the system behavior in the presence of faults. Nevertheless, simulation-based fault injection techniques may lack in the accuracy of the fault model and the system model. In addition, they have a poor time-resolution, which may cause fidelity problems. Software fault injection is a special case of simulation-based fault injection where the target system is a large microprocessor-based machine that may include caches, memories, and devices, running a complex software. This technique is able to target applications and operating systems, which is not easy to do with the hardware fault injection.

Fault-injection simulators are attractive because they do not require expensive hardware. Moreover, they can be used to support all system abstraction levels, as applications and operative systems, which is difficult at hardware level. The controllability of fault-injection simulators is very high: given sufficient detail in the model, it is possible to modify any signal value in any desired way, with the results of the fault-injection easily observable regardless of the location of the modified signal within the model. The main goal of an early analysis of the resistance against fault attacks is to allow designers to easily identify the weakest point of their design, and to protect it with appropriate countermeasures. Although this approach is flexible, it has some shortcomings:

- Large development efforts are required, as they involve the construction of the simulation model of the system under analysis, including a detailed model of the processor in use. This increase the cost of using simulation-based fault-injection tools.
- Not all the fault attacks previously discussed can be simulated in the simulation model.
- The fidelity of the model strongly depends on the accuracy of the models used.
- High time consuming, due to the length of the experiment.

Some attacks, in particular setup-time violations, can be reliably simulated using state of the art EDA commodities. For some others, instead, it is impossible to have a complete simulation. It is however possible to model the type of error which will be induced into the device, and simulate the behavior of a device when a similar type of error occurs with cycle accurate or with behavioral simulators. The strategy usually adopted by these injection frameworks is to evaluate the effects, that the injected faults have on the final result of the computation. Designer then attempts to mount an attacks using the simulated data and can determine if the amount of information which will be available to the attacker will be sufficient to successfully extract secret information. In the rest of this section we revise known tools and approaches used in the past for injecting and simulating faults at different level of abstraction and we discuss their suitability for evaluating the resistance against fault attacks.

2.4.1 Weaknesses Identification with Static Analysis

Identification of portions of the circuit sensitive to fault attacks can be achieved using static timing analysis. Static timing analysis produces a very detailed timing characterization of the paths inside their design, highlighting the critical path and all the other paths which are very close to the critical one. Barenghi et al. [33] proposed to extract the worst-case delays associated with the input connections of the state and key registers. Static analysis was carried out with Synposys PrimePower, using as input the placed and routed netlist and the parasitics of the connections. The authors compared the ranking of sensitivity to attacks, obtained using static analysis, with the fault attacks mounted on a real device. Obtained results demonstrated that static timing analysis provides an effective way to estimate the worst case timings for the input lines of the state registers and pinpoint which ones are more likely to be vulnerable to setup-time violation attacks.

2.4.2 High-Level Simulation with Complex Fault Models

High-level simulators are system simulator which simulate the behavior of a device with the precision of a clock cycle. They can be execution based, when the benchmark is directly executed, or trace based, when the simulation is carried out using a trace of execution previously generated.

2.4.2.1 Fault Injection in Execution-Based Simulators

In these kind of simulators, a module for fault injection is integrated in the target design. The fault injection module can be integrated as dedicated module called *saboteur*. It is inactive during normal operation and can alter value or timing characteristics when active. Saboteurs can be inserted in series or parallel to the target design. Serial insertion, in its simplest form, consists of breaking up the signal path between a driver (output) and its corresponding receiver (input) and placing a saboteur in between. In its more complex form, it is possible to break up the signal paths between a set of drivers and its corresponding set of receivers and insert a saboteur. For parallel insertion, a saboteur is simply added as an additional driver for a resolved signal. The other approach of fault injection is using *mutants* which are inserted by modifying parts of the target circuit components. Those two approaches present the advantage of supporting all system abstraction levels: electrical, logical, functional, and architectural. Such approaches allow full reproduction of: *single-bit flips, selected bit alterations, data corruptions, circuit rewiring, clock alteration* and *instruction swaps* effects. However, theory require large development effort and cannot support fully randomisation and real-time features.

Existing available tools are: MEFISTO-C [34], VERIFY [35], HEARTLESS [36], GSTF [37], FTI [38], Xception [39], FERRARI [40], SAFE [41], DOCTOR [42]. A detailed overview can be found in [43].

Selected Simulators

Xception [39] is a software implemented fault injection tool for dependability analysis. It provides an automated test suite that helps in injecting realistic faults. It injects faults without any intrusion on the target system. No software traps are inserted and hence program can be executed in normal speed. It uses the advanced debugging and performance monitoring features that exist in processors to inject realistic faults by software, and to monitor the activation of the faults in order to observe in detail their impacts on the behavior of the system [39]. Xception is a flexible and low-costly tool that could be used in a wide range of processors and machines (parallel and real-time systems). In addition, it enables the definition of a general and precise processor fault model with a large range of fault triggers and oriented to the internal processor functional units.

2.4.2.2 Software Faults Emulation Tools

A few tools do exist to emulate the occurrence of faults in distributed applications. One of those tools is DOCTOR [42] (integrateD sOftware fault injeCTiOn enviRonment), that allows to inject faults in real-time systems. It supports faults in processor, memory and communication. The tool can inject permanent, transient, or intermittent faults. The fault scenarios that can be designed uses probabilistic model. While this suits small quantitative tests, repeatable fault injection capabilities are required for more complex fault scenarios.

SAFE [41] fault injection tool allows to automatically generate and execute fault injection tests. SAFE injects or detects software faults in C and C++ software, in order to force a software component failure, and to evaluate the robustness of the system as a whole. Injected faults are designed to realistically reproduce the real defects that hampers software systems, including issues affecting data initialization, control flow, and algorithms. Testing team can easily know how vulnerable the software is and fix it. The SAFE tool lets users customize which faults are injected.

2.4.2.3 Fault Injection in Trace-Based Simulators

An example of fault injection tools exploiting trace-based simulations is the one of Miele [44]. The tool analyzes the system-level dependability of embedded systems. The workflow is organized in three main phases: preliminary characterization of the system, setup of the experimental campaign, and execution of experimental campaign followed by results' post-processing. The designer specifies monitoring and classification actions at application and architecture levels. Debug-like mechanism allow to analyze the propagation of the errors in various functionalities of the executed application. The proposed approach is extremely suitable to reproduce the effects in simulation of *single-bit flips, selected bit alterations, data corruptions* and *instruction swaps*. Ferrari [40] (Fault-and-Error Automatic Real-Time Injection), developed at the University of Texas at Austin, uses software traps to inject CPU, memory, and bus faults. Ferrari consists of four components: the initializer and activator, the user information, the fault-and-error injector, and the data collector and analyzer. The fault-and-error injector uses software trap and trap handling routines. Software traps are triggered either by the program counter when it points to the desired program locations or by a timer. When the traps are triggered, the trap handling routines inject faults at the specific fault locations, typically by changing the content of selected registers or memory locations to emulate actual data corruptions. The faults injected can be those permanent or transient faults that result in an address line error, a data line error, and a condition bit error.

Jaca is a fault injection tool that is able to inject fault in object-oriented systems and can be adapted to any Java application without the need of its source code, but only few information about the application like the classes, methods, and attributes names [45]. Jaca has a graphical interface that permits the user to indicate the applications parameters under test in order to execute the fault injection [45]. Most of the fault injection tools are able to handle the injection of faults at low level of the software. Jaca differs from the other tools in the fact that it can perform both low-level fault injection, affecting Assembly language element (CPU registers, buses, etc.), and high-level fault injection affecting the attributes and methods of objects in a Java program.

The main advantage of using a trace-based simulator is the possibility of altering specific parts of the system without the need of altering the main structure of the system.

2.4.2.4 Software-Based Simulators

Software fault injection is a special case of simulation-based fault injection where the target system is a large microprocessor-based machine that may include caches, memories, and devices, running a complex software. This technique is able to target applications and operating systems, which is not easy to do with the hardware fault injection.

Selected Simulators
LFI is a tool to make fault injection-based testing more efficient and accessible to developers and testers [46]. LFI injects faults at the boundary between shared libraries and target programs, which permits to verify if the programs are handling the failures exposed by the libraries correctly or not. More in detail, LFI permits to automatically identify the errors exposed by shared libraries, find potentially buggy error recovery code in program binaries, and produce corresponding injection scenarios. Fault injection was rarely used in software development. LFI was developed in response to this. It permits to reduce the dependence on human labor and correct documentation, because it automatically profiles fault behaviors of libraries via static analysis of their binaries. The tool aims to provide testers an easy, fast, and comprehensive method to see how much the program is robust to face failures exposed between shared libraries and the tested programs [46].

Byteman [47] is a byte code injection tool developed to support Java code testing using fault injection technique. It is also very useful for troubleshooting and tracing Java program execution. Byteman provides a functions library which helps generating simple error conditions to complex error flows. Almost any Java code can be injected into the application in scope at the injection point. POJO (plain old java object) can be plugged in to replace built-in functions. Byteman works by modifying the bytecode of the application classes dynamically at runtime.

2.4.3 Low-Level Virtual Machine Simulation

Fault injection tools based on virtual machine can be a good solution. First, because they permit simulating the computer without having the real hardware system.

Moreover, they target hardware faults on the software level, and they allow observing complex computer-based systems, with operating system and user applications.

Virtualization is the technology permitting to create a virtual machine (VM) that behaves like a real physical computer with an Operating System (OS). It has an enormous effect in todays IT world since it ensures efficient and flexible performance, and permits cost saving from sharing the same physical hardware. The virtual machine where the software is running is called a guest machine, and the real machine in which the virtualization takes place is called the host machine. The words host and guest are used to make difference between the software that runs on the virtual machine and the software that runs on the physical machine.

Selected Simulators

LLVM (Low-Level Virtual Machine) is a compiler framework designed to support transparent, life-long program analysis, and transformation for arbitrary programs, by providing high-level information to compiler transformations at compile time, link-time, runtime, and in idle-time between runs [48, 49].

LLVM uses the LLVM Intermediate Representation (IR) as a form to represent code in the compiler. It symbolizes the most important aspect of LLVM, because it is designed to host mid-level analysis and transformations found in the optimizer section of the compiler. The LLVM IR is independent from the source language and the target machine. It is easy for a front end to generate, and expressive enough to permit important optimizations to be performed for real targets.

QEMU [50] is a versatile emulation platform with support for numerous target architectures like x86, ARM, MIPS and allowing to run a variety of unmodified guest operating systems. In [51], BitVaSim is proposed as a fault injection simulator on QEMU platform, for targets like PowerPC and ARM with built-in test software framework. BitVaSim can inject faults in any process, even pre-compiled software and allows a good degree of user configuration for fault injection. It can also be used for hardware targets in a virtual machine. In addition, unmodified operating systems and applications, especially the Built-In Test system can run on the prototype without intrusion. The described technique provides complete control over the target environment with fault injection process monitor and efficient feedback.

FAUMachine is a virtual machine that permits to install a full operating systems and run them as if they are independent computers. FAUMachine is similar in many aspect to standard virtual machines like QEMU [50] or VirtualBox [52]. The property that distinguishes FAUMachine from the other virtual machines is its ability to support fault injection capabilities for experimentation. FAUMachine supports the following fault types [53]:

- Memory Cells: such as transient bit flips, permanent struck-at faults, and permanent coupling faults.
- Disk, CD/DVD drive: such as transient or permanent block faults, and transient or permanent whole disk faults.
- Network: such as transient, intermittent, and permanent send or receive faults.

FAUMachine does not permit injecting faults in the CPU registers yet. Bit flips could be easy to implement. Stuck-at faults is also possible but it is much more complex since FAUMachine uses just-in-time compiling. In FAUMachine, the injection of fault could be done online via GUI, or defined (type, location, time, and duration of fault) via VHDL scripts. Compared to existing fault injection tools, FAUMachine is able to inject faults and observe the whole operating system or application software. Using the virtualization, this tool provides a high simulation speed for both complex hardware and software systems [53]. FAUMachine also supports automated tests, including the specification of faults to be injected.

2.4.4 Transistor Level Simulation

As previously discussed, setup-time violation can be induced by underfeeding the device. This attack can be completely simulated using SPICE level simulators, as proposed by Barenghi et al. [33]. The authors evaluated if transistor level simulator is capable of correctly predicting the fault patterns which were measured on a real device. The simulation was carried out using Synopsys Nanosim, a fast SPICE simulator, using the netlist and the parasitics generated by Cadence Encounter after place and route. The device was simulated for different voltages, ranging from 0.3 to 0.5 V. The simulation generated a number of faulty ciphertexts reasonably close to the one observed in the experiments, allowing to speculate that Nanosim is capable of predicting the setup-time violations measured in practice.

2.4.5 Emulation

Emulation-based fault injection has been introduced as a better solution for reducing the execution time compared to simulation-based fault injection. It is often based on the use of Field Programmable Gate Arrays (FPGAs) for speeding up fault simulation and exploits FPGAs for effective circuit emulation. This technique can allow the designer to study the actual behavior of the circuit in the application environment, taking into account real-time interactions. However, when an emulator is used, the initial VHDL description must be synthesizable.

Fault injection can be performed in hardware emulation models through compile time reconfiguration and runtime reconfiguration. Here reconfiguration refers to the process of adding hardware structures to the model which are necessary to perform the experiments. In compile-time reconfiguration, these hardware structures are added by the instrumentation of HDL models. The main disadvantage of compile-time reconfiguration is that the circuit must be resynthetised for each reconfiguration, which can impose a severe overhead on the time it takes to conduct a fault injection campaign.

2.5 Conclusions

Faults attacks are a powerful tool in the hand of adversaries, and they can have serious impacts on the security of embedded systems. Currently, most of the evaluation against fault attacks is done post-fabrication. However, it is important for designers to know the sensitive fault targets and possibly fix it at the design time. With this objective in mind, we summarize the most common fault attacks, the most frequently used fault injection techniques and the most used approach for fault simulations which could be used to evaluate the robustness of cryptographic circuits at design time.

Table 2.2 classifies the different fault injection methods. The approaches presented have different level of impact. Static timing analysis [33] provides an effective way for the designer to predict circuit paths which are likely to experience setup-time violations upon an attack, but it does not provide the possibility of simulating a fault.

Saboters and *mutant* fault injection approaches allow to properly simulate fault injection effects such as *single bit flips, selected bit alterations, data corruptions, circuit rewiring, clock alteration*, and *instruction swaps*. Furthermore, they provide full control of both fault models and injection mechanisms, together with maximum amount of observability and controllability. Essentially, given sufficient detail in the model, any signal value can be corrupted in any desired way, with the results of the corruption easily observable regardless of the location of the corrupted signal within the model. This flexibility allows any potential failure model to be accurately modeled. These methods are able to model both transient and permanent faults, and allow modeling of timing-related faults since the amount of simulation time required to inject the fault is minimal. The main drawback of those two approaches is given by the fact that only a predetermined set of faults can be injected, and new changes cannot be applied at runtime.

The main advantage of using a trace-based simulator as in [44] is given by the possibility of changing at runtime the execution traces, without a structural modification of specific components of the architecture or saboteurs. In this way, fault effects generation is easier and less time consuming. Moreover, it allows for time-specific fault attacks, since it is a cycle accurate based simulator. As main disadvantage, not all the current existing fault effects such as rewiring and clock delays can be effectively simulated, thus prohibiting an exhaustive analysis.

Virtual Machines-based tools can be a good solution for injecting faults. First, because they permit simulating the computer without having the real hardware system. Moreover, they target hardware faults on the software level, and they allow observing complex computer-based systems, with operating system and user applications. However, at the state of the art, they do not support yet fault models defined at software level, such as an instruction or a variable used in place of another.

Finally, for certain fault injection techniques, a complete simulation at SPICE level is possible. The drawback of this approach is the time required for the simulation, which can be prohibitive.

References

1. Kocher P, Jaffe J, Jun B. Differential power analysis. In: Advances in CryptologyCRYPTO99. Springer; 1999. p. 388–97.
2. Kocher PC. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in CryptologyCRYPTO96. Springer; 1996. p. 104–13.
3. Rohatgi P. Electromagnetic attacks and countermeasures. In: Cryptographic engineering. Springer; 2009. p. 407–30.
4. Schlösser A, Nedospasov D, Krämer J, Orlic S, Seifert J-P. Simple photonic emission analysis of AES. In: Cryptographic hardware and embedded systems—CHES 2012. Springer; 2012. p. 41–57.
5. Barenghi A. Bertoni GM, Breveglieri L, Pelosi G. A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA. J Syst Softw. 2013;86(7):1864–78.
6. Selmane N, Guilley S, Danger J-L. Practical setup time violation attacks on AES. In: Seventh European dependable computing conference, EDCC-7 2008, Kaunas, Lithuania, 7–9 May 2008, IEEE Computer Society; 2008. p. 91–6.
7. Otto M. Fault attacks and countermeasures. PhD thesis, Universit at Paderborn; 2005.
8. International organization for standardization. ISO/IEC 7816-3: electronic signals and transmission protocols. 2002. http://www.iso.ch.
9. Balasch J, Gierlichs B, Verbauwhede I. An in-depth and black-box characterization of the effects of clock glitches on 8-bit MCUs. In: Breveglieri L, Guilley S, Koren I, Naccache D, Takahashi J, editors. 2011 workshop on fault diagnosis and tolerance in cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011. IEEE; 2011. p. 105–14
10. Govindavajhala S, Appel AW. Using memory errors to attack a virtual machine. In: Proceedings of the 2003 IEEE symposium on security and privacy, SP '03. Washington, DC, USA: IEEE Computer Society; 2003. p. 154.
11. Skorobogatov S. Optical fault masking attacks. In: Breveglieri L, Joye M, Koren I, Naccache D, Verbauwhede I, editors. 2010 workshop on fault diagnosis and tolerance in cryptography, FDTC 2010, Santa Barbara, California, USA, 21 August 2010. IEEE Computer Society; 2010. p. 23–9
12. Barenghi A, Breveglieri L, Koren I, Naccache D. Fault injection attacks on cryptographic devices: theory, practice, and countermeasures. Proc IEEE. 2012;100(11):3056–76.
13. J-J Quisquater, D Samyde. Eddy current for magnetic analysis with active sensor. In: Esmart 2002, Nice, France, 9 2002.
14. Torrance R, James D. The state-of-the-art in IC reverse engineering. In: Cryptographic hardware and embedded systems-CHES 2009. Springer; 2009. p. 363–81.
15. Karaklajic D, Schmidt J-M, Verbauwhede I. Hardware designer's guide to fault attacks. IEEE Trans VLSI Syst. 2013;21(12):2295–306.
16. Blömer J, Seifert J-P. Fault based cryptanalysis of the advanced encryption standard (AES). In: Wright RN, editor. Financial cryptography, 7th international conference, FC 2003, Guadeloupe, French West Indies, January 27–30, 2003, revised papers. Lecture notes in computer science, vol. 2742. Springer; 2003. p. 162–81.
17. Ciet M, Joye M. Elliptic curve cryptosystems in the presence of permanent and transient faults. Des Codes Crypt. 2005;36(1):33–43.
18. Naccache D, Nguyen PQ, Tunstall M, Whelan C. Experimenting with faults, lattices and the DSA. In: Vaudenay S, editor. Public key cryptography—PKC 2005, proceedings of the 8th international workshop on theory and practice in public key cryptography, Les Diablerets, Switzerland, January 23–26, 2005. Lecture notes in computer science, vol. 3386, Springer; 2005. p. 16–28.
19. Duursma IM, Lee H-S. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In: Laih C-S, editor. ASIACRYPT 2003, proceedings of the 9th international conference on the theory and application of cryptology and information security: advances in cryptology,

Taipei, Taiwan, November 30–December 4, 2003. Lecture notes in computer science, vol. 2894. Springer; 2003. p. 111–23.

20. Biham E, Shamir A. Differential fault analysis of secret key cryptosystems. In: Kaliski BS Jr., editor. CRYPTO '97, proceedings of the 17th annual international cryptology conference on advances in cryptology, Santa Barbara, California, USA, August 17–21, 1997. Lecture notes in computer science, vol. 1294. Springer; 1997. p. 513–25.
21. Tunstall M, Mukhopadhyay D, Ali S. Differential fault analysis of the advanced encryption standard using a single fault. In: Ardagna CA, Zhou J, editors. Proceedings of the 5th IFIP WG 11.2 international workshop on information security theory and practice. Security and privacy of mobile devices in wireless communication, WISTP 2011, Heraklion, Crete, Greece, June 1–3, 2011. Lecture notes in computer science, vol. 6633. Springer; 2011. p. 224–33.
22. Rivain M. Differential fault analysis of DES. In: Joye M, Tunstall M, editors. Fault analysis in cryptography. Information security and cryptography. Springer; 2012. p. 37–54
23. Giraud C. Dfa on aes. In: Advanced encryption standard—AES, 4th International conference, AES 2004. Springer; 2003. p. 27–41.
24. Biehl I, Meyer B, Müller V. Differential fault attacks on elliptic curve cryptosystems. In: Bellare M, editor. CRYPTO 2000, proceedings of the 20th annual international cryptology conference on advances in cryptology, Santa Barbara, California, USA, August 20–24, 2000. Lecture notes in computer science, vol. 1880. Springer; 2000. p. 131–146.
25. Berzati A, Canovas C, Goubin L. Perturbating RSA public keys: an improved attack. In: Oswald E, Rohatgi P, editors. CHES 2008, proceedings of the 10th international workshop on cryptographic hardware and embedded systems, Washington, D.C., USA, August 10–13, 2008. Lecture notes in computer science, vol. 5154. Springer; 2008. p. 380–395.
26. Boneh D, DeMillo RA, Lipton RJ. On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy W, editor. Advances in cryptology—proceedings of the EUROCRYPT '97, international conference on the theory and application of cryptographic techniques, Konstanz, Germany, May 11–15, 1997. Lecture notes in computer science, vol. 1233. Springer; 1997. p. 37–51.
27. Li Y, Sakiyama K, Gomisawa S, Fukunaga T, Takahashi J, Ohta K. Fault sensitivity analysis. In: Mangard S, Standaert F-X, editors. CHES 2010, proceedings of the 12th international workshop on cryptographic hardware and embedded systems, Santa Barbara, CA, USA, August 17–20, 2010. Lecture notes in computer science, vol. 6225. Springer; 2010. p. 320–34.
28. Schmidt J-M, Herbst C. A practical fault attack on square and multiply. In: Breveglieri et al. [56], p. 53–8.
29. Schmidt J-M, Medwed M. A fault attack on ECDSA. In: Breveglieri L, Koren I, Naccache D, Oswald E, Seifert J-P, editors. Sixth international workshop on fault diagnosis and tolerance in cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009. IEEE Computer Society; 2009. p. 93–9.
30. Page D, Vercauteren F. A fault attack on pairing-based cryptography. IEEE Trans Comput. 2006;55(9):1075–80.
31. Schmidt J-M, Medwed M. Fault attacks on the montgomery powering ladder. In: Rhee KH, Nyang DH, editors. Information security and cryptology—ICISC 2010: 13th international conference, Seoul, Korea, December 1–3, 2010, revised selected papers. Lecture notes in computer science, vol. 6829. Springer; 2010. p. 396–406.
32. Kim CH, Shin JH, Quisquater J-J, Lee PJ. Safe-error attack on SPA-FA resistant exponentiations using a HW modular multiplier. In: Nam K-H, Rhee G, editors. ICISC 2007, proceedings of the 10th international conference on information security and cryptology, Seoul, Korea, November 29–30, 2007. Lecture notes in computer science, vol. 4817. Springer; 2007. p. 273–81.
33. Barenghi A, Hocquet C, Bol D, Standaert F-X, Regazzoni F, Koren I. A combined design-time/test-time study of the vulnerability of sub-threshold devices to low voltage fault attacks. IEEE Trans Emerg Top Comput. 2014;2(2):107–18.
34. Folkesson P, Svensson S, Karlsson J. A comparison of simulation based and scan chain implemented fault injection. In: Digest of papers: FTCS-28, the twenty-eigth annual international symposium on fault-tolerant computing, Munich, Germany, June 23–25, 1998. IEEE Computer Society; 1998. p. 284–93.

35. Sieh V, Tschäche O, Balbach F. Verify: evaluation of reliability using vhdl-models with embedded fault descriptions. In: FTCS. IEEE Computer Society; 1997. p. 32–6.
36. Rousselle C, Pflanz M, Behling A, Mohaupt T, Vierhaus HT. A register-transfer-level fault simulator for permanent and transient faults in embedded processors. In: DATE; 2001. p. 811.
37. Baraza JC, Gracia J, Gil D, Gil PJ. A prototype of a VHDL-based fault injection tool: description and application. J Syst Arch. 2002;47(10):847–67.
38. López C, Entrena L, Olías E. Automatic generation of fault tolerant VHDL designs in RTL. In: FDL (Forum on Design Languages), Lyon, France, September 2001.
39. Carreira J, Madeira H, Silva JG. Xception: software fault injection and monitoring in processor functional units; 1995.
40. Kanawati GA, Kanawati NA, Abraham JA. Ferrari: a flexible software-based fault and error injection system. IEEE Trans Comput. 1995;44(2):248–60.
41. Cotroneo D, Natella R. Fault injection for software certification. In: IEEE security and privacy, special issue on safety-critical systems: the next generation, vol. 11(4). IEEE Computer Society. p. 38–45.
42. Han S, Rosenberg HA, Shin KG. Doctor: an integrated software fault injection environment; 1995.
43. Ziade H, Ayoubi RA, Velazco R. A survey on fault injection techniques. Int Arab J Inf Technol. 2004;1(2):171–86.
44. Miele A. A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems. Microprocess Microsyst Embed Hardw Des. 2014;38(6):567–80.
45. de Moraes RLO, Martins E. JACA—a software fault injection tool. In: DSN. IEEE Computer Society; 2003. p. 667.
46. Marinescu PD, Candea G. LFI: a practical and general library-level fault injector. In: DSN. IEEE; 2009. p. 379–88.
47. Dinn AE. Flexible, dynamic injection of structured advice using byteman. In: Proceedings of the tenth international conference on aspect-oriented software development companion, AOSD' 11. New York, NY, USA: ACM; 2011. p. 41–50.
48. Lattner C, Adve V. LLVM: a compilation framework for lifelong program analysis and transformation. In: Proceedings of the international symposium on code generation and optimization: feedback-directed and runtime optimization, CGO '04. Washington, DC, USA: IEEE Computer Society; 2004. p. 75.
49. Kooli M, Benoit P, Di Natale G, Torres L, Sieh V. Fault injection tools based on virtual machines. In: 2014 9th international symposium on reconfigurable and communication-centric systems-on-chip (ReCoSoC), May 2014. p. 1–6.
50. Bellard F. QEMU, a fast and portable dynamic translator. In: Proceedings of the annual conference on USENIX annual technical conference, ATEC '05. Berkeley, CA, USA: USENIX Association; 2005. p. 41.
51. Wan H, Li Y, Xu P. A fault injection system based on QEMU simulator and designed for bit software testing. Appl Mech Mater. 2013;347–350:580–7.
52. Watson J. Virtualbox: bits and bytes masquerading as machines. Linux J. 2008(166).
53. Potyra S, Sieh V, Cin MD. Evaluating fault-tolerant system designs using faumachine. In: Guelfi N, Muccini H, Pelliccione P, Romanovsky A, editors. EFTS. ACM; 2007. p. 9.
54. Breveglieri L, Gueron S, Koren I, Naccache D, Seifert J-P (eds). Fifth international workshop on fault diagnosis and tolerance in cryptography, 2008, FDTC 2008, Washington, DC, USA, 10 August 2008. IEEE Computer Society; 2008.