Hardware Designer's Guide to Fault Attacks

Duško Karaklajić, Jörn-Marc Schmidt* and Ingrid Verbauwhede, *Fellow*, IEEE

***Abstract*—Hardware designers invest a significant design effort when implementing computationally intensive cryptographic algorithms onto constrained embedded devices to match the computational demands of the algorithms with the stringent area, power and energy budgets of the platforms.**

When it comes to designs that are employed in potential hostile environments, another challenge arises: the design has to be resistant against attacks based on the physical properties of the implementation, the so-called implementation attacks. This creates an extra design concern for a hardware designer.

This paper gives an insight into the field of fault attacks and countermeasures to help the designer to protect the design against this type of implementation attacks. We analyze fault attacks from different aspects and expose the mechanisms they employ to reveal a secret parameter of a device. In addition, we classify the existing countermeasures and discuss their effectiveness and efficiency. The result of this work is a guide for selecting a set of countermeasures which provides a sufficient security level to meet the constraints of the embedded devices.

***Index Terms*—Embedded security, Fault attacks, Countermeasures, Classification.**

I. Introduction

FAULT attacks are introduced by Boneh et al. in 1997 [1] where a fault in a computation is used to attack an RSA implementation using the Chinese Remainder Theorem (CRT). That attack initiated the discovery of a considerable amount of different types of fault attacks. As new types are constantly being proposed and accumulated, designing a fault-attack secure cryptosystem becomes increasingly difficult. Even though there is a significant number of different countermeasures against fault attacks presented in the literature, it is not straightforward to select the proper set that can efficiently protect a device. The selected countermeasures must meet two main requirements: 1) the overhead they introduce must fit in the available area and energy budget of a device, and 2) they have to provide protection against all relevant attacks, i.e., the attacks that are a threat for the device under analysis. Furthermore, due to the huge variety of fault attacks, it is difficult to determine the complete set of relevant threats. Since there is no established strategy for securing a design against fault attacks, a designer can easily overlook protection against a relevant attack, thus lowering the security level of the device. On the other hand, even though there is no general strategy that ensures security against fault attacks, it is possible to determine certain directions in a design process which can lead to a protected implementation. In order to achieve such protection, a hardware designer should be able to answer the following questions:

- What kind of fault attacks exist?
- Which of them are a threat for the specific design?
- Which set of countermeasures thwarts the possible attacks?
- What is the cost and the effectiveness of the selected countermeasures?

This paper provides an overview of existing fault attacks and countermeasures, thus helping designers of cryptographic devices to answer the questions above. We start by classifying known fault attacks according to different criteria. Rather than analyzing each specific attack, we extract the common properties of all attacks and classify them into groups. By analyzing the fault injection mechanisms, and their precision and possible targets, the main principles the attacks employ to reveal secret information out of a design are exposed. By explaining the complete *fault's path* from the fault injection until the point when it is exploited to deduce the secret data, we make it easy to understand how the fault attacks are used to *break* cryptographic devices.

Based on the groups of attacks, the countermeasures against them are classified. We analyze the protection mechanism they use, their abstraction level and the part of a processor they protect, thereby dividing the existing countermeasures into a few categories. Also, we estimate their time and area overhead and analyze their effectiveness against different classes of attacks. The classification eases the selection of a set of countermeasures that meets the constraints of embedded devices, still providing an adequate security level. Finally, we analyze the effectiveness of the existing countermeasures against fault models enabled by advanced fault injection equipment and discuss future directions of protection mechanisms.

II. Fault Attacks

In this section the variety of fault attacks is introduced. We explain how an injected fault propagates from the physical transistor level where it is injected towards the higher abstraction levels. There it is exploited to derive secret information from the faulty computations. Rather than analyzing specific attacks, we focus on their general properties, such as precision, targets and techniques they use to deduce secret information from a faulty result. The classification of these properties is the starting point for their prevention.

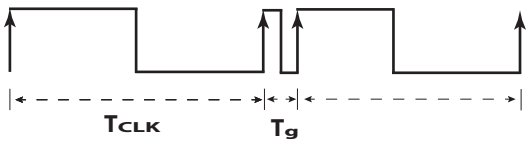
A. Physical Nature of Fault Injection

Since the first idea to exploit the occurrence of faults, various methods to inject a fault in a device have been presented.

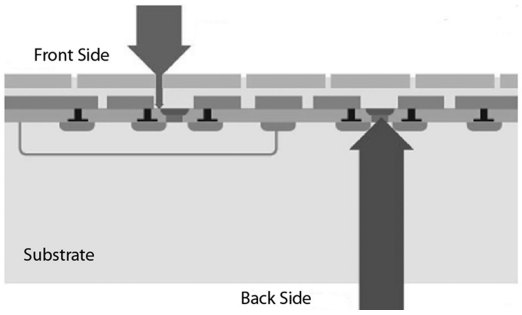
KU Leuven, ESAT/SCD-COSIC and IMINDS Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium Email: firstname.lastname@esat.kuleuven.be

* Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria Email: Joern-Marc.Schmidt@iaik.tugraz.at

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by the Flemish IMINDS projects, by the Flemish Government, FWO G.0550.12N, by the Hercules Foundation AKUL/11/19, by the European Commission through the ICT programmes under contract ICT-2007-216676 ECRYPT II, and under contract ICT-SEC-2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES)



(a) A glitch in the clock signal.



(b) The effect of a focused laser beam [2].

Fig. 1: Fault injection techniques.

We hereby give a brief overview of the most common fault injection techniques and comment on their properties.

Under-powering and Power spikes. Tampering with the power supply of a device is a low cost fault injection method. A possible way to provoke a faulty behavior is the under-powering of the device. Since there is no precise timing, the faults provoked by such a method tend to occur uniformly throughout the computations and the attacker must be able to successfully discard the erroneous results caused by undesired faults.

Another method to affect computations performed in the device is the induction of precise high variations in a power supply. Power spikes can cause a processor to skip or misinterpret an instruction, but also induce memory faults. For instance, if a processor reads a memory location at the time of a voltage spike, the wrong data may be gathered from the memory bus. Further, this fault injection technique is commonly exploited by the attackers who aim to tamper with a program counter or a loop bound [3], [4], [5]. Both listed fault injection techniques are easy to implement and require an attacker to be able to access the power supply line of the device.

Clock glitches. Devices that require an external clock generation can be attacked by supplying a deviated clock signal, i.e. a signal that contains one or several pulses much shorter than normal. As depicted in Fig. 1a, the period of the injected glitch, $\mathbf{T_g}$, is significantly shorter than $\mathbf{T_{CLK}}$. Such a glitch can cause a processor to execute the next instruction before the previous one was finished or to store invalid data in a memory location [3], [6]. In order to be able to induce such faults, an adversary should have a direct control over the clock line, which is typically the case when smart cards are attacked. The devices that use an internal clock generator can not be attacked by this method. Clock glitches are considered to be the simplest fault injection methods and can be induced by very cheap equipment, i.e., using low-end FPGA boards [7], [5].

Temperature attacks. Since electronic devices function correctly only in a certain temperature range, their exposure to too high or too low temperatures can induce faults [8], [1], [9]. This method is used to alter data stored in the memory, but is hard to focus on a particular portion of data.

Optical attacks. Optical faults are induced by exposing a decapsulated chip to a strong light source, e.g. a photo flash or a laser beam [10]. As semiconductors and conductors are inherently sensitive to laser ionization, it is possible to cause a switch of transistors when exposed to a light pulse. Using a focused laser beam, a single bit in a memory can be set or reset. When attacking a chip, an adversary can target either front side or back side of the chip (Fig. 1b). Even though the transistors are located at the front side, it may be difficult to reach them due to the metal layers placed on top. As an alternative, the transistors may be affected from the back side of the chip through the substrate, but a proper wavelength of the laser beam that ensures sufficient penetration depth must be used.

The optical fault injection technique is constantly advancing. The light spot size on a chip die is shrinking and is now physically limited by the wavelength of the photons. A diode based laser can achieve a spot size of about $6 \times 1.4\mu m$ [2]. In addition, the triggering mechanism of laser stations keeps improving so that it provides very accurate timing of the light pulse and fast switching, which enables multi-glitching, i.e injecting multiple precise faults in a short time period.

Electromagnetic (EM) fault injection. An external electromagnetic field can cause the malfunctioning of an encapsulated chip or change memory content. It induces eddy currents on the chip surface, which can cause a single bit fault [11]. A very cheap but imprecise EM fault can be induced using a gas lighter [12].

All described fault injection methods share the same property: by manipulating the physical layer of a device, they cause the transistors to switch abnormally. However, the properties of the faults they induce are different. The first three methods do not require expensive equipment, but their effect can hardly be focused to a particular part of a device. On the other hand, optical and EM methods can affect a very restricted area, but require a more complex setup. The next section deals with the basic fault properties in more details.

B. Basic Fault Properties

A set of properties of an injected fault used to characterize an attack is called a *fault model*. It reflects a physical event–fault injection into a mathematical model. The following list describes the basic properties of fault injection. They can differ in the ability to control the location and the time precision, in the effect, in the number of affected bits, and in the duration of the effect [13], [14], [15].

Controllability over the fault location can be characterized as *precise control*, *loose control* or *no control*. An attacker is assumed to have *precise control* if he/she is able to affect a single specific bit of a specific variable. On the other hand, *loose control* assumes that it is possible to target a specific variable, but not its specific part. Finally, an attacker has *no control* over the fault location if he/she affects a variable at random.

Similarly, **controllability over the fault timing** can be classified as *no control*, *loose control* or *precise control*. An adversary which has *precise control* over the fault timing can affect a specific variable at a specific point in time. With *loose control* over the fault timing, an attacker is able to target a set of operations or clock cycles, but not a specific one. The controllability over the fault timing is of particular significance when embedded software implementations on microcontrollers are attacked since skipping of specified instructions is often required in that case.

By the **number of affected bits**, faults are classified as *single bit* faults, *word-size* faults and *variable-size* faults. The actual word size is dependent on the architecture of the target processor and typically is 8, 16, 32 or 64 bits.

The effect of a fault is determined by its manifestation in the chip and four types can be distinguished: *stuck-at fault*, *bit flip fault*, *set/reset fault* and *random fault*.

Considering the **duration of the injected fault** we differentiate between *transient*, *permanent* and *destructive* faults. The effect of a *transient* fault lasts for a limited period of time, after which the correct value of a variable is present again. An example of such a fault is affecting a variable while being transfered via a bus. It is assumed that a *transient fault* only affects one request of the target variable, while all the further requests result in the correct value again. On the other hand, *permanent faults* affect the target variable until it is explicitly overwritten, e.g. affecting a variable while being stored in memory. Finally, *destructive faults* damage the physical layer of a chip, causing a number of bits to be fixed at a specific value. An example of such faults are stuck-at faults in logic or memory, and this kind of fault cannot be reversed.

Another type of faults enabled by using SRAM-based FPGAs is called *remanent faults*. They are achieved by injecting an error in a configuration memory of the FPGA, thus changing the architecture of a design. By their duration, they are placed between the permanent and destructive faults and rely on the re-configurability of FPGAs.

C. Fault Targets

This section highlights the parts of a generic processor that are common targets of fault attacks. A generic processor consists of the following high level components: an I/O part which provides an interface with external components; a memory module for storing system parameters and intermediate results, as well as data and instruction processing modules (see Fig. 2). Attacks can be classified according to the following four main components of a processor: inputs, data part, storage and control part.

Attacks on input parameters. Depending on the device and its use, it may be possible to trigger very precise faults by manipulating the input parameters [16], [17], [18], [19], [20], [21]. This is possible if the adversary can choose the input for the computation to some extent, and if the implementation fails to check whether the input is valid. Since a cryptographic core is usually only a part of a bigger design, the input parameters are not necessarily supplied from the I/O part, but can also be read from a non-volatile memory.

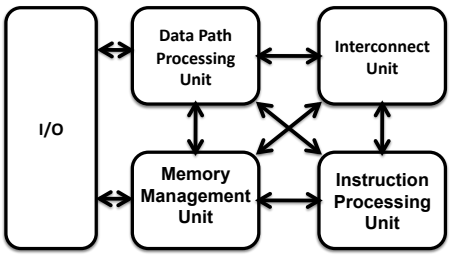


Fig. 2: Components of a generic processor

Attacks on data processing part. A device can be disturbed while performing a computation, which will cause an erroneous intermediate or final result [22], [23], [24], [25], [1], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36]. Note that the same effect can be caused if a fault is injected while the result is transferred via a bus, or even when it is stored in registers/memory.

Attacks on storage part. Since data storage parts usually occupy a significant part of a chip area and they have a regular structure, they are easily distinguishable from the other components. For this reason, they are a common target for fault injection. Errors in volatile storage can be exploited to alter the intermediate results of computations, while tampering with the non-volatile memory can affect the system parameters.

Attacks on instruction processing part. Attacking the program flow instead of computations is an effective attack method which targets the control part of a device instead of the datapath [37], [13], [38]. This includes making the processor skip and misinterpret certain instructions.

One desired effect of an injected fault can be obtained by targeting different processors parts. For instance, if an attacker aims to alter an intermediate result of a computation, he/she can achieve it either by affecting an actual computation or a result itself when stored in memory. Her/his choice is determined by the available fault injection equipment and by the required fault properties, but also by the way in which an injected fault is exploited to leak secret information from the device. The different ways to benefit from a faulty behavior is explained in the next section.

D. Exploitation of the Injected Faults

Faults are injected in the physical layer of a device, but they are manifested on a higher abstraction level that involves operations on sensitive data like encryption schemes and digital signatures. At this abstraction level, the results of faulty computations are exploited to leak information about the secret parameter. Without going into details about every specific attack, this section deals with the general principles used in exploiting the effects of an injected fault to break

a cryptographic system. In order to make the discussion easier to understand, the general methods are supported by representative examples of the existing attacks. We distinguish four classes:

Algorithm specific attacks. Security of public-key systems is commonly based on a mathematical problem which is assumed to be computationally hard to solve. For instance, Elliptic Curve Cryptography (ECC) is based on the elliptic curve discrete-logarithm problem (ECDLP) [39], while RSA makes use of the problem of large integer factorization [40]. In order to circumvent solving the hard problem directly, an adversary can try to inject a fault in the computation and transform the problem into an easily solvable one. The kind of fault that is injected and the way to exploit it depends on the algorithm. Hence, we denote such attacks as algorithm specific attacks.

A prominent example of such an attack is the manipulation of the input parameters of an ECC-based system: by choosing an invalid base point, the computation of a scalar multiplication is shifted from the original curve to a weak one, where ECDLP is easy to solve and the secret scalar can be recovered [41]. A similar effect can be achieved by supplying wrong parameters for the curve [41]. Similarly, a fault that sets a few bits of a secret nonce to a known value in the Digital Signature Algorithm (DSA) scheme can be exploited to leak the whole key using a small number of faulty signatures [24]. Furthermore, by tampering with the loop bound which is used as a parameter in some pairing algorithms [42], it is possible to leak a secret point.

Differential fault analysis (DFA). Differential fault analysis exploits differential information between correct and faulty ciphertexts to retrieve the secret key. An attacker first collects several fault-free ciphertexts and then several faulty ones that are generated from the same secret key and plaintext. Afterwards, a technique from cryptoanalysis, the differential cryptanalysis, is applied to construct and solve differential fault equations in order to deduce bits of the secret key. In contrast to a classical differential attack, where characteristics of the cipher have to be analyzed to find appropriate inputs and outputs, the differential fault analysis manipulates the computation to obtain required outputs. Differential Fault Attacks were first introduced for symmetric ciphers in [43], which initiated a discovery of a considerable amount of DFA techniques that can be applied to various symmetric key primitives [44], [45], but they are also a threat to PKC algorithms [31].

A similar technique based on collecting several faulty plaintext-ciphertext pairs, and obtaining collisions in the ciphertexts, is exploited in [46] to derive information about a secret parameter. The difference between the correct and faulty outputs can also be used to characterize a device. In Fault Sensitivity Analysis (FSA), an attacker can deduce information about the secret parameter by correlating derived characteristics with precomputed ones that depend on estimated key bytes [47].

Tampering with the program flow. Instead of relying on faults in the processed data, attacks can also target the program flow [27], [28], [35], [36]. If a fault is injected in the program counter of a processor, it can cause an instruction to be skipped or misinterpreted, or executed multiple times. For example, it is possible to leak a secret exponent by skipping certain instructions of an exponentiation algorithm [28]. In addition, tampering with the loop counter or a branch mechanism may lead to the recovery of a pairing scheme's secret point [35] or reduce the number of rounds of a symmetric cipher [43]. Those attacks can also be employed in combination with other attacks to skip the final check of countermeasures. For example, if a device is protected by a validity check before the result is output, an attacker can inject a fault to skip this check [30].

Safe-error attacks. Another important class of fault analysis is safe-error attacks. They make use of a fault that, depending on they key, has or has no effect on the output result of the computation. In contrast to other attacks, a safe-error attack does not require a faulty output to derive information. It only uses information whether an error occurs in the output or not. Thus, it is possible to apply such an attack independent of the countermeasures that ensure a correct output of the computation. There are two types of safe-error attacks: computational, called C safe-error attacks, and memory, or M safe-error attacks. The C-type attack [13] induces any temporary random computational fault in an Arithmetic Logic Unit (ALU) of a device. Its objectives are dummy operations, which are usually introduced to achieve Simple Power Analysis (SPA) resistance. An adversary that induces transient faults in the ALU or the memory obtains the information whether the fault hits a dummy operation or not. Hence, if the position of dummy operations is key dependent, the adversary leaks information about the secret key with each injected fault. While the C safe-error attack exploits the weakness of an algorithm, the M safe-error attack employs a possible safe error in the implementation. The basic method of an M safe-error is key-dependent clearing of some memory blocks. Safe-error attacks are successfully mounted on both public-key [13], [30] and symmetric-key cryptosystems [48].

E. Feasibility of the Attacks

The considerations above show that an adversary can exploit an injected fault in several ways. However, each attack has different fault property requirements in order to be implemented in practice. Table I specifies the required fault properties for different classes of attacks. Some of them have strict demands in space/time controllability or fault effect. On the other hand, some can be implemented using various fault models, which is marked with multiple possibilities in the table. This leaves an attacker with the opportunity to choose a *weaker* fault model at the cost of increased post-processing time.

In addition to choosing an appropriate fault model, an attacker has another *degree of freedom*: to choose a part of a processor in which an error is injected. Table II lists different classes of fault attack and marks the processors parts that can be targeted for their practical implementation. Note that some attacks allow an adversary to choose a target for the fault injection among different processor parts. The presented tables show that, given a specific fault attack described as a mathematical model, an adversary has the flexibility to

TABLE I: Feasibility of Fault Attacks

Fault Attack	Requirements				
	Space	Time	#Bits	Effect	Duration
Algorithm Specific Attack	SC/LC	SC/LC	B/W/V	F/S	PF/TF
DFA	LC	LC	B/W	R	TF
Attacks on Program Flow	SC/LC	SC	B/W/V	F/S	PF/TF
Safe-Error Attacks	SC/LC	SC	B/W/V	R	TF

LC: **Loose Control**
SC: **Strong Control**
B: **Bit**
W: **Word**
V: **Variable**

PF: **Permanent Fault**
TF: **Transient Fault**
S: **Set**
R: **Randomize**
F: **Flip**

TABLE II: Targets of Fault Attacks

Attack	Target			
	Inputs	Datapath	Memory	Control
Alg. Specific	$\surd$	$\surd$	$\surd$	$\surd$
DFA	X	$\surd$	$\surd$	X
Control	X	X	$\surd$	$\surd$
Safe-Error	X	$\surd$	$\surd$	$\surd$

choose among different parts of a cryptographic processor, as well as among different fault models that can provide the implementation of the attack. It is usually determined by the available fault injection equipment and by the architecture of the device under attack. In any case, an attacker needs only one successful attack to break a device, while a designer must foresee all possible threats. The next section discusses ways to protect implementations against fault attacks, and comments on their effectiveness and efficiency.

III. Countermeasures against Fault Attacks

Deployment of novel and improved fault attacks drives the evolution of countermeasures. Since there exist no generic countermeasures which can prevent all attacks, the combination of different techniques is required to achieve a sufficient security level. As countermeasures come at a price, they are chosen to provide a good trade-off between hardware and performance costs and security level. In practice, countermeasures are intended to make an attack sufficiently expensive, not impossible, still keeping a protected design cost-effective and performing [2].

There are two major principles for protection of a cryptographic device against fault attacks, as shown in Fig. 3. First, a device can be protected by hardware countermeasures which are specifically constructed to prevent a certain type of fault injection. A prominent example of such countermeasures are passive and active shields. Passive shields are metal layers that cover a part of or the entire chip, thus preventing an optical fault injection or probing attacks [49]. An active shield consists of a wire mesh that runs signals over the chip surface and detects any interruption on a wire. Furthermore, a chip can be equipped with light sensors and anomalous frequency detectors which detect optical fault injection and glitches in the clock signal. The main drawback of the hardware countermeasures is their cost. In addition, since such protection aims

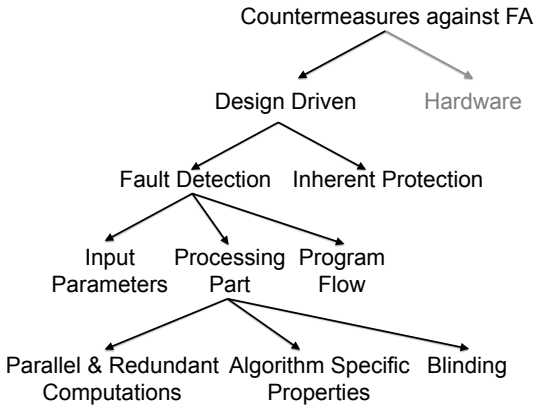


Fig. 3: A Classification Tree of Countermeasures against Fault Attacks.

at preventing a specific method of fault injection, it does not provide long term protection, as novel fault injection methods are frequently developed.

The focus of this paper is on the second approach, which aims at protecting a cryptographic algorithm and its implementation so that an injected fault is detected or so that a design is inherently protected against fault attacks. This type of countermeasure is defined as *design driven* in the text below.

There are two main principles used to construct design driven countermeasures: (1) employing redundancy to check whether the computation was tampered with, i.e. incorporate a fault check which detects and reports a fault, or (2) design the implementation to be inherently prone to attacks. Although the second method leads to a more efficient protection, it prevents only a limited set of attacks. Therefore, it is necessary to combine it with a fault detection mechanism in order to provide complete protection. Further, a fault can be detected in different parts of a processor: input part, processing part (memory + datapath), or in the program flow (Fig. 2). Finally, an actual fault detection mechanism can be implemented in different ways, as shown in the last stage of the classification tree in Fig. 3. Moreover, the listed countermeasures can be applied on different abstraction levels, which influences the effectiveness against certain attacks.

The rest of this section discusses the application of fault detection on each possible abstraction level and explains in

detail the countermeasure classes listed in the classification tree. Furthermore, we discuss their effectiveness against different classes of fault attacks and give an estimation of the performance overhead they introduce.

A. Abstraction Level of Countermeasures

If a protection against fault attacks is addressed only when a design is finished, it usually leads to an inefficient and incomplete protection. Instead, the security should be incorporated in a design process and implemented with the rest of the system. Fig. 4 visualizes different abstractions levels that exist when a hardware module capable of running a cryptographic protocol is designed. The highest abstraction level deals with

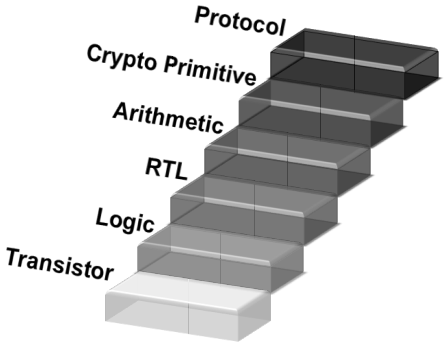


Fig. 4: Abstraction Levels in Cryptographic Hardware Design Process.

a cryptographic protocol, e.g. authentication protocol, and specifies all the operations required for its execution. A cryptographic protocol is based on cryptographic primitives, such as encryption, decryption, or digital signing [50]. Passing through the arithmetic, register-transfer level (RTL) and logic level, the design process reaches the physical, transistor layer. Each abstraction level provides an opportunity for the designer to apply specific performance optimization techniques, leading to fast, small, or low energy solutions [51], [52].

Similarly, countermeasures which detect faults can be incorporated in a design process and implemented on any layer. Since there is no countermeasure that protects against all fault attacks, parallel protection mechanisms are often required on different abstraction levels to achieve sound protection:

Protocol level. On the protocol layer, it is possible to design the usage of cryptographic primitives in a way that certain fault attacks are not possible any more. A prominent example is the so-called fresh re-keying, where a new key is generated for each run of the encryption, making differential attacks impossible [53].

Cryptographic primitive level. In order to protect a cryptographic primitive, the result of the algorithm can be verified before outputting it. For example, a DSA implementation can check whether the result is fault-free by verifying the signature with the public key before outputting it [54]. Further, the fact that encryption and decryption algorithms often use the same datapath, is exploited to check the correctness of the computations. It is possible to detect faults in the encryption process by decrypting the obtained ciphertext and comparing it to the original plaintext.

Algorithm level. In public key cryptography, the primitives are often based on modular exponentiation or scalar multiplication (ECC, RSA). Depending on the algorithm used, different fault detection methods can be applied, e.g. in case of the Montgomery powering ladder (MPL), a consistency check at the end is an efficient way to detect errors [55].

Arithmetic level. Each cryptographic algorithm is based on arithmetic and/or logical operations. In many secure processors, these operations already contain some type of protection. This can be achieved e.g. by redundant parallel computations or by means of error detection codes [56].

Circuit level. A design can also be protected at circuit level. Often, a special logic style is used to detect faults. For instance, an unused state in dual-rail logic can be employed to detect faults [57].

B. Fault Detection Mechanisms

A fault detection is the basis for most of the existing countermeasures against fault attacks. They incorporate a fault detection mechanism which detects and reports an error, thus preventing an erroneous result to be observed by an attacker. In what follows, we describe how fault detection mechanisms are used to protect basic parts of a generic processor: *input*, *processing*, and the *control part*[1].

1) Protection of Input Parameters: In some protocols, an adversary is allowed to supply parameters that are used for the computation later on. If an implementation misses to check whether these parameters are valid, the adversary can choose bogus inputs to attack the system [41].

If the inputs are not supplied externally, the system parameters are typically stored in non-volatile memory and are transferred to RAM for the computations. These parameters can also be targeted for attack, as errors that occur either in public or secret parameters can be exploited to reveal secret information. Since such errors can be injected even before traditional countermeasures that protect the integrity of an algorithm can take effect, the validity of input parameters must be explicitly checked before being used for computations. A common way of implementing such checks is by adding a Cyclic Redundancy Check (CRC) to each system parameter. After a parameter is read for a computation, its CRC is computed and compared to the one stored in non-volatile memory.

Error detection probability of the CRC is proportional to the amount of added redundancy. However, increased redundancy introduces additional hardware overhead, since more memory is required for its storage. Moreover, a module that computes CRCs has to be implemented. Therefore, the amount of redundancy is usually a result of a trade-off between the desired security level and the available hardware resources.

[1]The protection of memory is distributed over the protection of input and system parameters, which are stored in a non-volatile memory and the protection of a processing part, which detects errors in the intermediate results, typically stored in RAM or the register file.

2) Protection of the Processing Part: The processing part of a device is harder to protect than the parameters, since static codes are not sufficient. A fault injected in a device during the processing time may enable leakage of sensitive information. Furthermore, there is a big variety of attacks like modifying registers, disturbing the computation of the ALU, or modifying the program flow to miss instructions. In order to prevent such attacks, two methods have been proven effective: 1) parallel or redundant computations, and 2) checking algorithm-specific properties. In addition, techniques that aim at the prevention of side-channel attacks, such as blinding [58], [59], can make fault attacks more difficult.

Parallel and redundant computations. Concurrent error detection (CED) is a common mechanism used to prevent fault-based cryptoanalysis. It is based on introducing redundancy to check the correctness of a computation. A straightforward way to implement CED is to compute the same operation twice, either in parallel or consecutively. However, these methods introduce an overhead of at least 100% [60]. Methods that provide protection at a lower costs are parity-preserving logic [61], [62], [63] or arithmetic residue codes [64]. Parity preserving logic is an effective way to detect faults in symmetric ciphers [65], [66], [67].

Another efficient way to implement time redundancy- based CED involves encrypting data twice (encrypting and decrypting) and comparing the results. Such schemes can be optimized to reduce the time overhead [68] and they are a common protection for symmetric ciphers. For instance, the duplication can be applied on operation, round or full cipher level, as explained in [69]. For a detailed overview of the protection methods for symmetric ciphers, we refer the reader to [70].

Furthermore, algorithms themselves can be extended by redundancy to detect manipulations [71], [60], [72], [73], [64], [48], [74], [75]. Often, a homomorphic mapping of the data to a smaller set can be found. Hence, the algorithm can be performed on both representations, and it can be checked that the output corresponds to the smaller representation before outputting data. For example, in addition to the original elliptic curve, a curve of a smaller order is used to define a larger combined curve, which allows for checking the final result of the ECC computations efficiently [33]. Shamir [76] proposed a similar method for RSA implementations. The basic idea is to extend the field by a small modulus and compute the RSA in the enlarged group as well as in the group defined by the small modulus[2].

Checking algorithm specific properties. Some cryptographic algorithms already contain redundancy by construction, and that can be used for fault detection [31], [77], [78], [41], [79], [55], [80], [81], [82], [83]. For instance, checking if the result of a scalar multiplication still belongs to the original curve is a common fault detection method in the ECC implementations. Another widely exploited protection of the MPL [84] based exponentiation is checking a coherency between the intermediate variables in the algorithm. Checking if the difference between two intermediate points is exactly one base point is an efficient fault detection for ECC [85], [86]. Similarly, the coherence between the two intermediates can be used to detect faults in RSA implementations.

Blinding. An alternative to specific countermeasures that can help to complicate possible attacks is called *blinding*. It was introduced to prevent side-channel attacks but can also thwart fault attacks that require a precise fault injection since the variables are randomized. Blinding an exponent or a message can be applied to achieve side-channel and fault attack-resistant exponentiation algorithms, using a single countermeasure [58], [59].

Generally speaking, if an algorithm already provides specific properties that can be employed for checking, such a countermeasure introduces less hardware overhead compared to the parallel and redundant computations. However, the latter provide more generic solutions, since they do not depend on a specific algorithm.

3) Protection of Program Flow: In addition to the data, an adversary can also target the program flow (see Section II-D). The main idea is to exploit the faults that allow the selective execution of instructions in a program. A possible way to protect the integrity of a program flow is by calculating and checking its fingerprint using An+B codes [87], [88]. Namely, these codes provide a signature, which can be precomputed and verified to detect tampering with data or program flow.

C. Inherent Protection Mechanisms

In addition to fault detection mechanisms which report an injected fault, there exists a small set of countermeasures which are based on an intrinsic protection. Such countermeasures do not prevent the output of the erroneous result, but ensure that it does not leak any information about the secret parameter [89]. These methods do not need to be countermeasures in the classical sense; for example, choosing the parameters of an elliptic curve so that the curve has a strong twist prevents attacks that rely on moving the computation to the twist [34]. Hence, this set of attacks can be prevented without any overhead by choosing the curve parameters carefully. Similarly, storing a base point of an ECC cryptosystem in non-volatile memory instead of supplying it as an input parameter prevents an attacker in mounting fault sensitivity analysis (FSA) [47].

Further, a careful selection of the exponentiation algorithm can provide a protection against some attacks. For instance, in addition to high performance exponentiation, the MPL with no y coordinate provides a protection against sign change attack [33].

A very careful implementation is required to prevent safe-error attacks since they are based on a different principle. Compared to other fault attacks, they do not require a faulty output; the information regarding whether an error has occurred or not is enough. Hence, checking whether a result is correct is not enough to thwart such attacks. An efficient way to protect a design against both C and M safe-error attacks, is a proper selection of the exponentiation algorithm combined with an implementation flow that ensures security against these attacks [32], [90], [91].

[2]Note that if the computation is based on CRT, both groups can be enlarged by the same small prime and compared afterwards. This reduces the required overhead.

IV. Effectiveness and Cost of the Countermeasures

The previous considerations show that in the last decade many different proposals on how to protect implementations against fault attacks have been published. Each solution thwarts a specific set of attacks with a limited probability and comes with a certain overhead. In order to achieve sound protection of a design, a combination of several methods have to be employed. For selecting this set, a hardware designer should be aware of the possible attacks that are feasible in the intended use-case of the device. Based on the attacks that have to be prevented and the available margin in terms of costs, the designer can select an appropriate set of protection mechanisms.

Table III depicts the effectiveness of the mechanisms against different classes of fault attacks. It can be noticed that most

TABLE III: Effectiveness of the Countermeasures

Protection	Fault Attacks				
	Alg. Specific		DFA	SEA	Control
	Inputs	Datapath			
Parallel & Redundant	X	$\surd$	$\surd$	X	$\surd$
Alg. Specific Check	X	$\surd$	$\surd$	$?\surd$	$?\surd$
Protection of Inputs	$\surd$	X	X	X	X
Inherent Protection	$?\surd$	$\surd$	$\surd$	$\surd$	X

$\surd$: a countermeasure thwarts an attack.
$?\surd$: a countermeasure is effective in some special cases.
X: a countermeasure can not thwart an attack.

of the existing protection mechanisms aim to thwart the attacks on the processing part, algorithm specific attacks and differential fault attacks. On the other hand, the choice of countermeasures against the attacks on input parameters and safe-error attacks is very limited. If choosing an appropriate input parameter can not implicitly thwart an attack, the parameter must be explicitly checked before being used for computations. Similarly, there is no general method which provides the resistance against safe-error attacks. A device must be designed in such way that it provides an inherent resistance to these attacks.

In addition to effectiveness, a designer should have an estimation of the overhead that a certain class of countermeasures introduces before selecting it. Table IV gives an estimation of the area overhead and the throughput reduction introduced by different classes of countermeasures. Please note that the overhead of the specific countermeasures inside a class varies depending on the algorithm that should be protected and the required security level. For details on the specific countermeasures, we refer the reader to overviews given in [65], [92]. As expected, Table IV exposes the countermeasures based

TABLE IV: Cost Estimation of the Countermeasures

	Cost	
Countermeasure	**Area**	**Throughput**
Parallel Redundant Modules	$> 50\%$	$\approx 0\%$
Repetition	$\approx 0\%$	50%
Protection of Input Params	$< 10\%$	$< 10\%$
Algorithm Specific Check	$< 30\%$	$< 30\%$
Inherent Protection	-	-

on repetition and parallel computations as the most costly ones. If implemented naively, they introduce an overhead of 100% [85]. In practice, they exploit some algorithm-specific properties so that the overhad is reduced [61], [62], [63], [64]. Such kind of protection is typical for symmetric ciphers that share the datapath for encryption and decryption. Using pipelining, the repeated computation is done in parallel with the original one, thus reducing the time overhead [93].

Next, algorithm specific checks typically introduce low overhead, but it can vary depending on the protection capabilities and the specific deployment in the system. For instance, if the protection requires redundant memory locations, its area overhead can be up to 30% [78]. On the other hand, if the existing memory and datapath can be re-used, the area overhead can be as low as 1% [94]. The introduced time overhead mainly depends on the frequency of the countermeasure's activation. For example, if an error detection is activated only at the end of the algorithm, the time overhead could be less then 1% [95], [78], [94]. On the other hand, if a concurrent error detection is required, which implies the fault detection after each round of the algorithm, the time overhead is significantly increased [78].

Finally, the inherent protection mechanisms are the most efficient ones. As they are based on the proper selection of algorithms and system parameters, they introduce no area and time overhead. However, due to the limited number of the attacks they thwart, they must be accompanied by other countermeasures in order to provide a complete protection.

The comparative tables in this section show that only a proper combination of countermeasures against fault attacks can result in a secure design. While a design itself can be protected on different abstraction levels, the validity of the input parameters and the integrity of a program flow must be addressed separately. The precise overhead that a countermeasure introduces and the probability of detecting a specific attack are tightly coupled with the design, and are often subject of trade-offs. They have to be addressed separately for every concrete application.

A. Future Directions

Taking into account the inventiveness of attackers and ever increasing technical capabilities of the fault injection equipment, it woud be interesting to analyze the effectiveness of the countermeasures against such advanced adversarial model. To do so, we split the analysis into two parts: to tackle the problem of novel fault exploitations mechanisms, we first discuss the effectiveness of different classes of countermeasures against an emerging class of implementation attacks that combine the side-channel and fault analysis (combined attacks). Then, to address the problem of using advanced fault injection equipment, we examine the effectiveness of the countermeasures when an adversary is able to induce multiple precise faults. Finally, we discuss possible future research directions for securing designs against fault attacks.

Combined attacks combine the fault injection and side-channels, such as power consumption or electromagnetic radiation, to observe the effect of the injected fault. These attacks

are mounted on both public key [96], [97] and symmetric key cryptographic algorithms [98], [99]. Combined attacks exploit the fact that the injected error's effect exists from the moment the error is triggered and not only at the end of the computation. Therefore, they could be successfully applied despite fault detection at the end of the algorithm since all the necessary information are obtained through a side-channel during the execution. Obviously, the class of countermeasures that is based on fault detection at the end of algorithm (Section III) is not effective in this case. Instead, a protection that immediately detects a fault, such as concurrent error detection, must be employed [78]. Also, the countermeasures originally proposed to thwart side-channel attacks, such as blinding or infective computation [80], [100], could be used.

Another aspect of an advanced attacker model is the employment of *high end* fault injection equipment. Analyzing up-to-date fault injection techniques [2], [101], it can be concluded that they are advancing mainly in the following two aspects: 1) a triggering mechanism that enables the precise control over the fault's timing and 2) *fast switching* lasers, such as a diode laser, that can inject multiple faults in a very short time period[3]. Please note that we do not analyze the fault model in which an attacker is able to set/rest a single bit at will since we do not consider it to be practical. Even though the laser spot size is shrinking, so does the CMOS feature size, thus limiting the space precision of the fault injection.

In order to analyze how different classes of countermeasures deal with the presented fault injection advances, we construct Table V. It indicates that precise single faults that can be injected in a specified point of time, i.e. a part of a clock cycle, are a threat to the countermeasures that check the integrity of input parameters since they can alter a variable after its check is being performed. Similarly, parallel and redundant computations that tend to protect arithmetic modules, such as adders and multipliers, can be defeated in a similar way. On the other hand, if parallel and redundant computations are coupled with the algorithm-specific properties [33], [76] or if the algorithm-specific checks are employed, precise single faults are not effective. Since these countermeasures detect faults at the end of the algorithm, an adversary can not make use of a fault injected after the check.

TABLE V: Effectiveness of the Countermeasures against the Advanced Fault Model.

	Faults	
Countermeasure	**Single**	**Multiple**
Protection of Input Params	$\surd^{\star}$	$\surd$
Parallel Redundant Modules	$\surd^{\star}$	X
Repetition	$\surd^{\star}$	X
Algorithm Specific Check	$\surd$	$\surd$
Inherent Protection	$\surd$	$\surd$

$\surd$: a countermeasure is still effective.
$\surd^{\star}$: a countermeasure is effective in some special cases.
X: a countermeasure is not effective.

[3]Approximately, a fault can be injected on every 40 nS [2]

Next, precise multiple faults seem to be the biggest threat to countermeasures based on parallel and redundant computations. If an attacker can inject two faults in a chosen time moment, he can cause the same error twice, thus making it undetectable. Further, if an attacker is able to induce two simultaneous faults, the probability of causing the same errors is even higher. The other classes of countermeasures are resistant to multiplie fault attacks, even though the error detection probability could be decreased in some particular cases.

The analysis above shows that the effectiveness of the countermeasures against fault attacks is time-limited and they have to be re-evaluated regularly. Even though there are some attempts to make a formal model for fault attacks, provable security against these attacks is not settled yet. Another research direction is a *hardware security assurance*: a process of identifying and fixing vulnerabilities in a design before it is released. So far, there exist two distinct approaches to this problem. The first one tries to give general guidelines for secure hardware development [51], [102], [52], [103], [89] while the other one aims at ensuring that a device is secure against particular attacks before it is released [90], [91], [104]. We believe that a combination of the general theoretical approach and the modeling based on the experience in protecting cryptographic devices against particular attacks could lead to a solution that provides a sufficient security level at the minimal cost.

V. Conclusion

In order to construct a fault-attack resistant embedded device, a hardware designer has to understand their threat in the first place. Even though there is a huge variety of fault attacks, we showed that they all share common properties. We classified the attacks and expose the mechanisms they employ to reveal sensitive information by extracting these properties. Moreover, we explained how an injected fault propagates form the physical layer to the protocol layer and highlighted the parts of a generic processor which are sensitive to faults. Hence, we provide an orientation to hardware designers for determining which attacks are a threat for their specific designs.

Furthermore, we analyzed possible ways to protect a device and explained methods to thwart fault attacks. It turns out that security against fault attacks must be incorporated in an early stage of the design process and has to be addressed on different abstraction levels. We showed possible ways to protect different parts of a generic processor and evaluated their effectiveness against various attacks. Finally, we estimated time and area overhead of each class of countermeasures, thus making it easier for a hardware designer to select the set of countermeasures that meets the performance constraints, still providing a sufficient level of security.

References

[1] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *16th Annual International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT 1997, Berlin, Heidelberg, 1997, pp. 37–51.

[2] J. Van Woudenberg, M. Witteman, and F. Menarini, "Practical Optical Fault Injection on Secure Microcontrollers," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011)*, Sept. 2011, pp. 91–99.

[3] O. Kömmerling and M. G. Kuhn, "Design Principles for Tamper-resistant Smartcard Processors," in *Proceedings of the USENIX Workshop on Smartcard Technology*. Berkeley, CA, USA: USENIX Association, 1999, pp. 2–2.

[4] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures," 2002.

[5] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC 2011. Washington, DC, USA: IEEE Computer Society, 2011, pp. 105–114.

[6] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb. 2006.

[7] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "An On-chip Glitchy Clock Generator for Testing Fault Injection Attacks," *Journal of Cryptographic Engineering*, vol. 1, pp. 265–270, 2011.

[8] I. Peterson, "Chinks in Digital Armor: Exploiting Faults to Break Smart-card Cryptosystems," *Science News*, vol. 151, pp. 78–79, 1997.

[9] S. Skorobogatov, "Low Temperature Data Remanence in Static RAM," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-536, Jun. 2002. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf

[10] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems -CHES 2002*, 2002, pp. 2–12.

[11] J.-J. Quisquater and D. Samyde, "Eddy current for Magnetic Analysis with Active Sensor," in *Esmart 2002, Nice, France*, Sept. 2002.

[12] J.-M. Schmidt and M. Hutter, "Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results," in *Austrochip 2007, 15th Austrian Workhop on Microelectronics, 11 October 2007, Graz, Austria, Proceedings*, J. W. Karl C. Posch, Ed. Verlag der Technischen Universität Graz, 2007, pp. 61–67.

[13] S.-M. Yen and M. Joye, "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis," *IEEE Transactions on Computers*, vol. 49, pp. 967–970, September 2000.

[14] S.-M. Yen, S. Kim, S. Lim, and S. Moon, "A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack," in *Proceedings of the 4th International Conference on Information Security and Cryptology*, ser. ICISC 2001. London, UK: Springer-Verlag, 2002, pp. 414–427.

[15] M. Otto, "Fault Attacks and Countermeasures," Ph.D. dissertation, University of Paderborn, 2005.

[16] M. Kara-Ivaniov, E. Iceland, and A. Kipnis, "Attacks on Authentication and Signature Schemes Involving Corruption of Public Key (Modulus)," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2008)*. IEEE Computer Society, 2008, pp. 108–115.

[17] C. Kim, P. Bulens, C. Petit, and J.-J. Quisquater, "Fault Attacks on Public Key Elements: Application to DLP-Based Schemes," in *Public Key Infrastructure*, ser. Lecture Notes in Computer Science, S. Mjlsnes, S. Mauw, and S. Katsikas, Eds. Springer Berlin / Heidelberg, 2008, vol. 5057, pp. 182–195.

[18] A. Berzati, C. Canovas-Dumas, and L. Goubin, "Public Key Perturbation of Randomized RSA Implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. Lecture Notes in Computer Science, S. Mangard and F.-X. Standaert, Eds. Springer Berlin / Heidelberg, 2010, vol. 6225, pp. 306–319.

[19] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin, "Fault Attacks on RSA Public Keys: Left-To-Right Implementations Are Also Vulnerable," in *Topics in Cryptology CT-RSA 2009*, ser. Lecture Notes in Computer Science, M. Fischlin, Ed. Springer Berlin / Heidelberg, 2009, vol. 5473, pp. 414–428.

[20] J.-P. Seifert, "On Authenticated Computing and RSA-based Authentication," in *Proceedings of the 12th ACM conference on Computer and communications security*, ser. CCS 2005. ACM, 2005, pp. 122–127.

[21] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier, "Why One Should Also Secure RSA Public Key Elements," in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2006*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds. Springer Berlin / Heidelberg, 2006, vol. 4249, pp. 324–338.

[22] J. Carrijo, R. Tonicelli, and A. C. A. Nascimento, "A Fault Analytic Method against HB+," *IEICE Transactions*, pp. 855–859, 2010.

[23] N. A. Howgrave-Graham and N. P. Smart, "Lattice Attacks on Digital Signature Schemes," *Des. Codes Cryptography*, vol. 23, pp. 283–290, July 2001.

[24] D. Naccache, P. Q. NguyŁn, and M. Tunstall, "Experimenting with Faults, Lattices and the DSA," in *Public Key Cryptography PKC 2005, volume 3386 of Lecture Notes in Computer Science*. Springer-Verlag, 2005, pp. 16–28.

[25] A. Pellegrini, V. Bertacco, and T. Austin, "Fault-based Attack of RSA Authentication," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 855–860.

[26] M. Joye, J.-J. Quisquater, F. Bao, and R. Deng, "RSA-type Signatures in the Presence of Transient Faults," in *Crytography and Coding*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1997, pp. 155–160.

[27] J. Schmidt and M. Medwed, "A Fault Attack on ECDSA," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*. IEEE, Sept. 2009, pp. 93 –99.

[28] J.-M. Schmidt and C. Herbst, "A Practical Fault Attack on Square and Multiply," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2008)*. IEEE, Aug. 2008, pp. 53–58.

[29] D. Wagner, "Cryptanalysis of a Provably Secure CRT-RSA Algorithm," in *Proceedings of the 11th ACM conference on Computer and Communications Security*, ser. CCS 2004. ACM, 2004, pp. 92–97.

[30] C. Kim, J. Shin, J.-J. Quisquater, and P. Lee, "Safe-Error Attack on SPA-FA Resistant Exponentiations Using a HW Modular Multiplier," in *Information Security and Cryptology - ICISC 2007*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4817, pp. 273–281.

[31] I. Biehl, B. Meyer, and V. Müller, "Differential Fault Attacks on Elliptic Curve Cryptosystems," in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO 2000. Springer-Verlag, 2000, pp. 131–146.

[32] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," in *International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES 2002. Springer-Verlag, 2003, pp. 291–302.

[33] J. Blömer, M. Otto, and J.-P. Seifert, "Sign Change Fault Attacks on Elliptic Curve Cryptosystems," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, vol. 4236, pp. 36–52.

[34] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette, "Fault Attack on Elliptic Curve Montgomery Ladder Implementation," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2008)*. IEEE Computer Society, 2008, pp. 92–98.

[35] D. Page and F. Vercauteren, "A Fault Attack on Pairing-Based Cryptography," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1075–1080, Sept. 2006.

[36] J.-M. Schmidt and M. Medwed, "Fault Attacks on the Montgomery Powering Ladder," in *International Conference on Information, Security and Cryptology - ICISC 2010*, ser. Lecture Notes in Computer Science, K.-H. Rhee and D. Nyang, Eds. Springer, 2010, to appear.

[37] S.-M. Yen, S. Moon, and J.-C. Ha, "Hardware Fault Attack on RSA with CRT Revisited," in *Information Security and Cryptology ICISC 2002*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, vol. 2587, pp. 374–388.

[38] J. Waddle and D. Wagner, "Fault Attacks on Dual-rail Encoded Systems," in *21st Annual Computer Security Applications Conference*, Dec. 2005, pp. 10–494.

[39] H. Cohen, G. Frey, and R. Avanzi, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman Hall/CRC, Boca Raton, 2006 .

[40] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, February 1978.

[41] M. Ciet and M. Joye, "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults," *Designs, Codes and Cryptography*, vol. 36, pp. 33–43, 2005.

[42] I. Duursma and H.-S. Lee, "Tate Pairing Implementation for Hyperelliptic Curves," in *Advances in Cryptology -ASIACRYPT 2003*, ser. Lecture Notes in Computer Science, C.-S. Laih, Ed. Springer Berlin / Heidelberg, 2003, vol. 2894, pp. 111–123.

[43] E. Biham and A. Shamir, "Differential Fault Analysis of Secret key Cryptosystems," in *Advances in Cryptology CRYPTO 1997*, ser. Lecture Notes in Computer Science, B. Kaliski, Ed. Springer Berlin / Heidelberg, 1997, vol. 1294, pp. 513–525.

[44] C. Giraud, "Differential fault analysis of the advanced encryption standard," in *Fault Analysis in Cryptography*, ser. Information Security

and Cryptography, M. Joye and M. Tunstall, Eds. Springer Berlin Heidelberg, 2012, pp. 55–72.

[45] M. Rivain, "Differential Fault Analysis of DES," in *Fault Analysis in Cryptography*, ser. Information Security and Cryptography, M. Joye and M. Tunstall, Eds. Springer Berlin Heidelberg, 2012, pp. 37–54.

[46] J. Blömer and V. Krummel, "Fault Based Collision Attacks on AES," in *Fault Diagnosis and Tolerance in Cryptography*, ser. Lecture Notes in Computer Science, L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, Eds. Springer Berlin / Heidelberg, 2006, vol. 4236, pp. 106–120.

[47] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault sensitivity analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. Lecture Notes in Computer Science, S. Mangard and F.-X. Standaert, Eds. Springer Berlin / Heidelberg, 2010, vol. 6225, pp. 320–334.

[48] J. Blömer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, R. Wright, Ed. Springer Berlin / Heidelberg, 2003, vol. 2742, pp. 162–181.

[49] H. Handschuh, P. Paillier, and J. Stern, "Probing Attacks On Tamper-Resistant Devices," in *Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, C. KoC and C. Paar, Eds. Springer Berlin / Heidelberg, 1999, vol. 1717, pp. 727–727.

[50] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[51] P. Schaumont and I. Verbauwhede, "Domain-specific codesign for embedded security," *IEEE Transactions on Computers*, vol. 36, no. 4, pp. 68 – 74, april 2003.

[52] I. Verbauwhede and P. Schaumont, "Design Methods for Security and Trust," in *Design, Automation and Test in Europe (DATE 2007)*. NICE,FR: IEEE, 2007, pp. 1–6.

[53] M. Medwed, F.-X. Standaert, J. Groschdl, and F. Regazzoni, "Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices," in *Progress in Cryptology AFRICACRYPT 2010*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6055, pp. 279–296.

[54] B. KaliskiJr and M. Robshaw, "Comments on Some New Attacks on Cryptographic Devices," *RSA Laboratories' Bulletin no.5*, 1997.

[55] C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1116–1120, sept. 2006.

[56] M. Medwed and J.-M. Schmidt, "Coding Schemes for Arithmetic and Logic Operations - How Robust Are They?" in *Information Security Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, pp. 51–65.

[57] S. Moore, R. Anderson, P. Cunningham, R. Mullins, and G. Taylor, "Improving Smart Card Security using Self-timed Circuits," in *Technology, Fourth AciD-WG Workshop, Grenoble, ISBN*, 2002, pp. 211–218.

[58] G. Fumaroli and D. Vigilant, "Blinded Fault Resistant Exponentiation," in *International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2006)*, ser. Lecture Notes in Computer Science, L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, Eds. Springer Berlin / Heidelberg, 2006, vol. 4236, pp. 62–70.

[59] A. Boscher, H. Handschuh, and E. Trichina, "Blinded Fault Resistant Exponentiation Revisited," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*. IEEE, Sept. 2009, pp. 3–9.

[60] A. Dominguez-Oviedo and M. Hasan, "Error Detection and Fault Tolerance in ECSM Using Input Randomization," *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 3, pp. 175–187, july-sept. 2009.

[61] G. G. Langdon and C. K. Tang, "Concurrent Error Detection for Group Look-ahead Binary Adders," *IBM Journal of Research and Development*, vol. 14, no. 5, pp. 563–573, Sep. 1970.

[62] S. Fenn, M. Gossel, M. Benaissa, and D. Taylor, "On-Line Error Detection for Bit-Serial Multipliers in GF(2m)," *Journal of Electronic Testing*, vol. 13, pp. 29–40, 1998.

[63] B. Ansari and I. Verbauwhede, "A Hybrid Scheme for Concurrent Error Detection of Multiplication over Finite Fields," in *International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2010)*, Oct. 2010, pp. 399–407.

[64] G. Gaubatz, B. Sunar, and M. G. Karpovsky, "Non-linear Residue Codes for Robust Public-key Arithmetic," in *Workshop on Fault Tolerance and Diagnosis in Cryptography (FTDC 2006)*. Springer, 2006, pp. 173–184.

[65] T. G. Malkin, F.-X. Standaert, and M. Yung, "A Comparative Cost/Security Analysis of Fault Attack Countermeasures," in *Proceedings of the Third international conference on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC 2006. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 159–172.

[66] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 492–505, Apr. 2003.

[67] G. Bertoni, L. Breveglieri, I. Koren, and P. Maistri, "An Efficient Hardware-based Fault Diagnosis Scheme for AES: Performances and Cost," in *International Symposium on Defect and Fault Tolerance in VLSI System*, oct. 2004, pp. 130–138.

[68] R. Karri, K. Wu, P. Mishra, and Y. Kim, "Concurrent Error Detection Schemes for Fault-based Side-channel Cryptanalysis of Symmetric Block Ciphers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1509–1517, Dec. 2002.

[69] ——, "Fault-based Side-channel Cryptanalysis Tolerant Rijndael Symmetric Block Cipher Architecture," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2001, pp. 427 –435.

[70] J.-M. Schmidt and M. Medwed, "Countermeasures for Symmetric Key Ciphers," in *Fault Analysis in Cryptography*, ser. Information Security and Cryptography, M. Joye and M. Tunstall, Eds. Springer Berlin Heidelberg, 2012, pp. 73–87.

[71] R. Stern, N. Joshi, K. Wu, and R. Karri, "Register Transfer Level Concurrent Error Detection in Elliptic Curve Crypto Implementations," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2007)*. IEEE, sept. 2007, pp. 112–119.

[72] J. Francq, J.-B. Rigaud, P. Manet, A. Tria, and A. Tisserand, "Error Detection for Borrow-Save Adders Dedicated to ECC Unit," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2008)*. IEEE, Aug. 2008, pp. 77–86.

[73] E. Ozturk, G. Gaubatz, and B. Sunar, "Tate Pairing with Strong Fault Resiliency," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2007)*, vol. IEEE, sept. 2007, pp. 103–111.

[74] S. Liu, B. King, and W. Wang, "A CRT-RSA Algorithm Secure against Hardware Fault Attacks," in *International Symposium on Dependable, Autonomic and Secure Computing*. IEEE, 29 2006-oct. 1 2006, pp. 51–60.

[75] M. Joye and S.-M. Yen, "Secure Evaluation of Modular Functions," pp. 227–229, 2001.

[76] A. Shamir, "Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attack," November 1999.

[77] N. Ebeid and R. Lambert, "Securing the Elliptic Curve Montgomery Ladder against Fault Attacks," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*. IEEE, Sept. 2009, pp. 46–50.

[78] K. Ma and K. Wu, "LOEDAR: A Low Cost Error Detection and Recovery Scheme for ECC," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1–6.

[79] A. Boscher, R. Naciri, and E. Prouff, "CRT RSA Algorithm Protected Against Fault Attacks," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, ser. Lecture Notes in Computer Science, D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, Eds. Springer Berlin / Heidelberg, 2007, vol. 4462, pp. 229–243.

[80] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon, "RSA Speedup with Chinese Remainder Theorem Immune Against Hardware Fault Cryptanalysis," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 461–472, april 2003.

[81] C. H. Kim and J.-J. Quisquater, "How can we Overcome both Side Channel Analysis and Fault Attacks on RSA-CRT?" in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2007)*. IEEE, sept. 2007, pp. 21–29.

[82] M. Rivain, "Securing RSA against Fault Analysis by Double Addition Chain Exponentiation," in *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, ser. CT-RSA 2009. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 459–480.

[83] D. Vigilant, "RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2008*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 5154, pp. 130–145.

[84] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, 1987.

[85] A. Dominguez-Oviedo, "On Fault-based Attacks and Countermeasures for Elliptic Curve Cryptosystems," Ph.D. dissertation, University of Waterloo, Canada, 2008.
[86] D. Karaklajic, J. Fan, J.-M. Schmidt, and I. Verbauwhede, "Low-cost Fault Detection Method for ECC using Montgomery Powering Ladder," in *Design, Automation and Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.
[87] I. Proudler, "Idempotent AN codes," in *Colloquium on Signal Processing Applications of Finite Field Mathematics*. IEEE, Jun 1989, pp. 1–5.
[88] M. Medwed and J.-M. Schmidt, "Generic Fault Countermeasure Providing Data and Program Flow Integrity," in *Workshop on Fault Diagnosis and Tolerance in Cryptography, (FDTC 2008)*. IEEE, aug. 2008, pp. 68–73.
[89] S. Guilley, L. Sauvage, J.-L. Danger, and N. Selmane, "Fault Injection Resilience," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010)*. IEEE, Aug. 2010, pp. 51 –65.
[90] D. Karaklajić, J. Fan, and I. Verbauwhede, "Systematic Security Evaluation Method Against C Safe-error Attacks," in *International Symposium on Hardware-Oriented Security and Trust (HOST 2011)*. IEEE, June 2011, pp. 63–66.
[91] ——, "Systematic M Safe-error Detection in Hardware Implementations of Cryptographic Algorithms," in *International Symposium on Hardware-Oriented Security and Trust (HOST 2012)*. IEEE, June 2012, pp. 96–101.
[92] P. Maistri, "Countermeasures against Fault Attacks: the Good, the Bad, and the Ugly," in *IEEE International On-Line Testing Symposium (IOLTS)*, July 2011, pp. 134 –137.
[93] A. Satoh, T. Sugawara, N. Homma, and T. Aoki, "High-Performance Concurrent Error Detection Scheme for AES Hardware," in *International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES 2008. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 100–112.
[94] D. Karaklajić, J. Fan, J.-M. Schmidt, and I. Verbauwhede, "Low-cost Fault Detection Method for ECC using Montgomery Powering Ladder," in *Design, Automation & Test in Europe (DATE 2011)*. IEEE, March 2011, pp. 1–6.
[95] E. Wenger and M. Hutter, "A Hardware Processor supporting Elliptic Curve Cryptography for less than 9 kGEs," in *Proceedings of the International conference on Smart Card Research and Advanced Applications*, ser. CARDIS 2011. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 182–198.
[96] F. Amiel, K. Villegas, B. Feix, and L. Marcel, "Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis," *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010)*, pp. 92–102, 2007.
[97] J. Fan, B. Gierlichs, and F. Vercauteren, "To Infinity and Beyond: Combined Attack on ECC using Points of Low Order," in *International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES 2011. Springer-Verlag, 2011, pp. 143–159.
[98] T. Roche, V. Lomné, and K. Khalfallah, "Combined Fault and Side-channel Attack on Protected Implementations of AES," in *Proceedings of International conference on Smart Card Research and Advanced Applications*, ser. CARDIS 2011. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 65–83.
[99] C. Clavier, B. Feix, G. Gagnerot, and M. Roussellet, "Passive and Active Combined Attacks on AES Combining Fault Attacks and Side Channel Analysis," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010)*. IEEE, Aug. 2010, pp. 10–19.
[100] J. Blömer, M. Otto, and J.-P. Seifert, "A New CRT-RSA Algorithm Secure Against Bellcore Attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS 2003. ACM, 2003, pp. 311–320.
[101] E. Trichina and R. Korkikyan, "Multi Fault Laser Attacks on Protected CRT-RSA," *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, vol. 0, pp. 75–86, 2010.
[102] D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing Embedded Systems," *IEEE Security & Privacy*, vol. 4, no. 2, pp. 40–49, 2006.
[103] H. Khattri, N. Mangipudi, and S. Mandujano, "HSDL: A Security Development Lifecycle for Hardware Technologies," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2012)*, June 2012, pp. 116–121.
[104] T. Sugawara, T. Suzuki, and T. Katashita, "Circuit Simulation for Fault Sensitivity Analysis and its Application to Cryptographic LSI," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2012)*. IEEE, Sept. 2012.

Duško Karaklajić graduated in electrical engineering from University of Belgrade in 2008. He is currently pursuing the Ph.D. degree from the Electrical Engineering Department, KU Leuven, Belgium. He is a member of COSIC Research Group and his main research interest is physical security of embedded systems with the emphasis on fault attacks and countermeasures.

Jörn-Marc Schmidt studied Computer Science at the University of Mannheim, Germany, with concentration on Cryptography and received his diploma (corresponds to master) in 2006. In 2007, he became member of the VLSI and Security group at the IAIK, where he was working on theory and practice of fault attacks. Jörn-Marc received his PhD degree in October 2009 from Graz University of Technology, Austria. Since 2010, he leads the Secure Entities for Smart Environments (SEnSE)-Group at IAIK.

Ingrid Verbauwhede received the Ph.D. degree from KU Leuven, Belgium, in 1991. She is currently a Professor at the KU Leuven and an adjunct professor at UCLA, CA. She joined the COSIC research group in 2003 where she leads the embedded security research group. She was a post-doctoral visiting researcher and lecturer at University of California at Berkeley and worked for TCSI and Atmel in Berkeley, CA. She is a member of the Royal Flemish Academy of Belgium for Science and the Arts. Her main interest is in the design and design methods for secure embedded circuits and systems. For a complete list of her publications and patents, please visit *www.cosic.be*.