






Quantitative Fault Injection Analysis

Jakob Feldtkeller¹ , Tim Güneysu^{1,2} , and Patrick Schaumont³ 

¹ Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany

{jakob.feldtkeller, tim.gueneysu}@rub.de

² DFKI, Bremen, Germany

³ Worcester Polytechnic Institute, Worcester, USA

pschaumont@wpi.edu

Abstract. Active fault injection is a credible threat to real-world digital systems computing on sensitive data. Arguing about security in the presence of faults is non-trivial, and state-of-the-art criteria are overly conservative and lack the ability of fine-grained comparison. However, comparing two alternative implementations for their security is required to find a satisfying compromise between security and performance. In addition, the comparison of alternative fault scenarios can help optimize the implementation of effective countermeasures.

In this work, we use quantitative information flow analysis to establish a vulnerability metric for hardware circuits under fault injection that measures the severity of an attack in terms of information leakage. Potential use cases range from comparing implementations with respect to their vulnerability to specific fault scenarios to optimizing countermeasures. We automate the computation of our metric by integrating it into a state-of-the-art evaluation tool for physical attacks and provide new insights into the security under an active fault attacker.

Keywords: Fault Injection Analysis · Fault Metric · Quantitative Information Flow

1 Introduction

Since their first publication in 1997 by Boneh et al. [12], Fault Injection Analysis (FIA) has become a fundamental part of the threat landscape for digital systems. In FIA, a malicious attacker disturbs the intended execution flow of a sensitive system to cause a denial of service, escalate privileges, or gain secret information. Such disturbance of execution is possible through fault injection, for example, via clock glitching [1], voltage glitching [50], electromagnetic pulses [6], or focused laser beams [52]. To thwart the exploitation of FIA, system designers use shields and sensors to prevent and notice a fault injection attempt or introduce redundancy in time, space, or information to detect the propagation of faults throughout the system after successful penetration.

However, the construction of FIA-secure systems requires clear criteria of vulnerability and security for a circuit under attack to direct the deployment of countermeasures. State-of-the-art security definitions for FIA focus on the

observability of faulty behavior at the outputs. In particular, the fault effect is propagated through the circuit and deemed insecure when the output gets affected and dedicated countermeasures are not triggered [4, 22, 43]. While helpful, this criterion lacks precision for FIA aimed at information leakage. Specifically, it is overly conservative in some cases and fails to detect practical attacks in others. On the one hand, some faults may propagate undetected to the outputs without affecting security. For example, injecting a fault directly into the output of some cryptographic cipher does not affect the security but is marked insecure by the criterion. Hence, the criterion is more a measure of the effectiveness of the countermeasures than of the circuit’s vulnerability. On the other hand, Statistical Ineffective Fault Analysis (SIFA) exploits the fact that a fault injection has no impact on the execution (ineffective fault) and is therefore not captured by the criterion at all while being a viable threat to implementation security. For SIFA, additional rules are introduced, such as checking the statistical dependency between secrets and fault detection behavior [31, 42]. In addition, the given criterion is binary in the sense that it only offers a categorization in the buckets *secure* and *insecure*. Hence, a qualitative comparison within one of those categories is not possible.

To overcome these limitations, we need a general metric to capture the security quantitatively in the context of information-leaking FIA. In particular, such a quantitative metric universally describes the secrecy loss caused by fault injection, i.e., not by providing a specific attack but by a quantification that is independent of the used analysis method. Such a quantitative metric can be used to identify the necessity of countermeasures and measure their effectiveness more tightly, leading to more optimal secure designs. It also allows the comparison of different designs for security, enabling a trade-off between performance (area, power, latency) and security. Such a trade-off could mean that a certain level of vulnerability is willingly accepted to gain some performance. However, it requires a realistic assessment of the cost in terms of security. Similarly, a designer (or attacker) can locate the most vulnerable fault positions to prioritize development efforts where they are most effective. Also, such a quantitative metric can be used by machine learning algorithms to learn the construction of secure designs in a fine-grained manner. Since machine learning provides no guarantees for the resulting designs, security criteria are required again to assess the result.

To summarize, the overall goal is to create a framework and automated tool for the computation of fault severity that requires minimal effort from the user, with domain- and design-specific information being derived automatically or provided with ease. The tool will be used in an automatic design or evaluation framework to determine realistic fault locations, compute a quantitative security metric for the threat, and use the security assessment to optimize circuit synthesis for both performance *and* security.

Contribution. In this work, we show how to use methods of Quantitative Information Flow (QIF) analysis [3, 53] to evaluate security in the context of information-leaking FIA (cf. Sect. 3). For that, we extend a commonly used fault model [44] with *probabilistic faults* and show how to model FIA based on stateless

information channels. Hence, we merge results from two scientific communities to define a quantitative vulnerability metric for FIA that separates the notion of secrecy-loss due to fault injection from specific attacks against specific implementations. We provide an algorithm for precise computation, exploiting the efficient representation of boolean functions via Binary Decision Diagrams (BDDs), and integrate this approach into the state-of-the-art evaluation tool for physical security VERICA [42] (cf. Sect. 4). In our implementation, we cover deterministic information channels only. However, we show that we can transform each probabilistic channel into a deterministic channel with external random inputs and, hence, this is no restriction in generality.

The proposed quantitative FIA metric can be used to evaluate the efficiency of fault attacks (cf. Sect. 5) or the quality of countermeasures (cf. Sect. 6). We use the analysis of attacks to showcase the accuracy of our metric achieved by the tight match between theory and practice. Afterward, the analysis of countermeasures focuses on the generation of new insights to deepen our understanding of secrecy loss due to active fault injection. In particular, we show that some of the recent findings (and foundations to some security proofs) on SIFA are flawed. Also, our quantitative metric allows identifying fault locations that reduce the amount of leakage caused by other faults. This enables the implementation of new defense mechanisms, where the structure of the circuit ensures that when there is a leaking fault, there is always a leakage-reducing fault active.

While there already exist occasional works in the literature that use QIF for the evaluation of FIA, these works are non-generic and limited to specific attacks, countermeasures, or ciphers (cf. Sect. 8.1). Hence, applying those methods to other scenarios requires significant effort and expert knowledge. In contrast, we provide a general evaluation method for automatic computation.

2 Preliminaries

The important notation used throughout this work is given in Table 1. In general, we write functions in sans serif font (e.g., F) and sets as upper-case characters using a calligraphic font (e.g., \mathcal{S}). We denote a distribution over a set \mathcal{S} by $\mathbb{D}\mathcal{S}$.

2.1 Fault Injection Analysis

Fault Injection Methods. In FIA, an adversary disrupts the normal execution of a system under attack to gain an advantage. Most fault injection methods require physical access to the attacked device and often manipulate the timing behavior of the circuit [44]. Prominent examples include clock glitching [1], which increases the clock frequency; voltage glitching [50], which increases the propagation delay of logic gates by lowering the supply voltage; electromagnetic pulses [6], which reset parts of the circuit and cause a race between clock and information signals; or focused laser beams [52], which temporarily affect the physical properties of transistors. Recently, however, more and more research has shown the ability to inject faults via software, allowing remote execution of

Table 1. Important notations used throughout this work.

Notation	Description
Circuit	C Digital logic circuit.
	C Information-theoretic channel.
	S, S, s Set of secrets, random variable of secrets, and secret value.
	\mathcal{X}, X, x Set of inputs (no secrets), random variable of inputs, and input value.
	\mathcal{R}, r Set of random values and specific random value.
	\mathcal{Y}, Y, y Set of outputs, random variable of outputs, and output value.
	Y', y' Random variable of faulty output, and faulty output value.
δ Detection flag.	
FIA	k Maximum number of simultaneous faults (security order).
	τ Fault type.
	\mathcal{F} Set of possible fault combinations.
	V Vulnerability metric (probability of correct guess in one attempt).
L Leakage metric (number of leaked bits).	

FIA. This can be done through the energy management system of modern Central Processing Units (CPUs) [55], through high-frequency accesses to memory locations [29], through valid but malicious bitstreams for Field-Programmable Gate Arrays (FPGAs) [28], or through randomly occurring faults in large systems [54]. In general, faults can be transient, i.e., having only a short-term effect, or permanent.

Analysis Methods. By exploiting faults, an adversary can bypass access-control mechanisms, cause denial of service, or obtain secret information. In this work, we focus on the leakage of sensitive information, where many analysis mechanisms are inspired by techniques from cryptanalysis. The first published fault attack by Boneh et al. [12] falls in the category of Algebraic Fault Analysis (AFA) [19]. It solves a system of equations depending on correct and faulty outputs and intermediate variables for the secret. Similarly, Differential Fault Analysis (DFA) [9] exploits known differentials between a correct and a faulty intermediate state to reduce the possible key space. In particular, only those key hypotheses remain where, for all pairs of correct and faulty outputs, the intermediate differential matches with the induced fault. The same idea can be applied to impossible differentials [8]. Collision-based Fault Analysis (CFA) [10] uses only correct and faulty output pairs where the output does not change despite the injected fault. The simplest example is a known fault injected into some key bit. If the output is the same for the correct and the faulty output, then the fault is a correct key guess. All the previous FIA techniques require pairs of correct and faulty outputs. The insight of biased FIA [37] is that only faulty outputs are needed if the fault injection causes some bias in an intermediate state, either because of

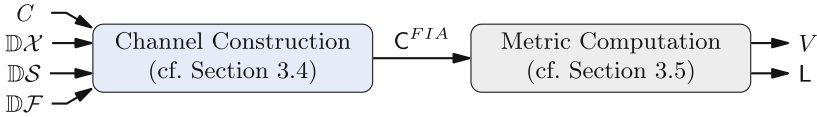


Fig. 1. Framework for quantitative fault-injection analysis.

dependencies between the secret and the fault occurrence or because of the fault itself is biased (e.g., set/reset faults). Then, only those key hypotheses remain that lead to the known bias in the intermediate state. SIFA [23] uses the same idea, but creates the bias by discarding all outputs where the fault has an effect.

2.2 Side-Channel Analysis and Masking

Another highly relevant attack is passive Side-Channel Analysis (SCA), where an adversary observes some physical characteristics such as timing [35], instantaneous power consumption [36], or electromagnetic emanations [27] to recover some processed secret, e.g., a cryptographic key. For arguing about SCA security theoretically, the d -probing model [33] was introduced, where an adversary gets access to d chosen intermediate values. As protection against such attacks, *boolean masking* [15] replaces each $x_i \in \mathbb{F}_2$ with a vector $\langle x_{i,0}, \dots, x_{i,n-1} \rangle \in \mathbb{F}_2^n$ with $n \in \mathbb{N}$, such that knowing any set of up to d shares $x_{i,j}$ does not reveal any information about x_i , and $x_i = \bigoplus_{j=0}^{n-1} x_{i,j}$. The optimal amount of shares is $n = d + 1$ (with less there is a trivial attack by probing all shares of some value). Similarly, the circuit is transformed into a *masked circuit* by transferring each operation to a set of operations that produces share vectors of the output from share vectors of the input.

3 A Vulnerability Metric for FIA

In the following, we describe the proposed vulnerability metric for FIA. For that, we start by defining our circuit, fault, and adversary models. Then we present our framework by showing how to construct an appropriate information channel and describe the actual computation of the vulnerability metric (cf. Fig. 1).

3.1 Circuit Model

Stateless Channel. We model a circuit C as a probabilistic information channel $C : \mathcal{X} \times \mathcal{S} \rightarrow \mathbb{D}\mathcal{Y}$ with a secret $s \in \mathcal{S}$ and some $x \in \mathcal{X}$ as inputs that produce a (probabilistic) output $y \in \mathcal{Y}$ [3]. Thereby, the adversary tries to learn the secret s and the output y should model all information that an adversary learns by observing the execution of the circuit. In general, we describe such a channel as a matrix, where each entry $C_{(x,s),y}$ gives the probability $\Pr[y \mid x, s]$. Hence, each row lists all the output probabilities given a specific input and sums up to 1. If $C(x, s)$ is uniquely defined, i.e., each row has exactly one entry equal to 1, we call the circuit deterministic.

We assume a stateless channel, meaning that each input is processed independently of all previous inputs, i.e., there is no notion of time or order between different executions. In particular, the channel always accepts the same input multiple times and processes it in the same (probabilistic) way. This restriction has implications for the type of countermeasures captured by this model, as discussed below (cf. Sect. 3.4). A stateless channel does not restrict the circuit to be stateless, i.e., the circuit may contain memory elements such as registers. However, before feeding a new input to the circuit, all memory elements are reset to an initial value, so there is no dependency between different executions.

Directed Graph. To precisely model the ability of a faulting adversary, we need more internal information about the circuit than provided by a probabilistic channel. A common method in the literature is to model a circuit via a Direct Acyclic Graph (DAG) [44]. For this, we define a set of input and output gates $\mathcal{G}_{io} = \{\text{in}, \text{out}\}$ where *in* has no input and outputs a value from the finite field \mathbb{F}_2 , and *out* does the opposite. Further, without loss of generality, we define the set of combinatorial gates to be $\mathcal{G}_c = \{\text{inv}, \text{and}, \text{nand}, \text{xor}, \text{xnor}\}$ and the set of memory gates to be clocked registers $\mathcal{G}_m = \{\text{reg}\}$. To represent probabilistic circuits, we define a randomness gate $\mathcal{G}_{\text{rand}} = \{\text{rand}\}$ with no input that outputs an independent and uniformly chosen value at each clock cycle. Then, we model a circuit C as a *directed graph* $\mathcal{C} = \{\mathcal{G}, \mathcal{W}\}$, where vertices $g \in \mathcal{G} = \mathcal{G}_{io} \cup \mathcal{G}_c \cup \mathcal{G}_m \cup \mathcal{G}_{\text{rand}}$ represent logical gates and edges $w \in \mathcal{W}$ represent wires connecting two gates and carrying a value from the field \mathbb{F}_2 .

3.2 Fault Model

We assume a slightly modified version of the fault model from Richter-Brockmann et al. [44]. In this model, up to k faults are injected into gates, and affected gates are transformed to a different gate type specified by the fault type $\tau \in \mathcal{T}$ (cf. Fig. 2a and 2b). Typical fault types are *set*, *reset* (replacing the targeted gate with a constant one or zero, respectively), or *bit flips* (inversion of the gate). In contrast to Richter-Brockmann et al., we introduce a probabilistic notion of fault location. A fault can occur in a subset of gates $\mathcal{G}' \subseteq \mathcal{G}$. Then a fault is a tuple $f_i = (g, \tau)$ with $g \in \mathcal{G}'$ and $\tau \in \mathcal{T}$. We define a probability distribution $\mathbb{D}\mathcal{F}$ over the set \mathcal{F} of all fault combinations f with up to k faults. The combination with zero faults is always in $\mathbb{D}\mathcal{F}$, potentially with probability zero. Hence, each fault combination will occur with a certain probability defined by $\Pr[f] \in \mathbb{D}\mathcal{F}$, as depicted in Fig. 2c.

This model naturally expresses transient faults, i.e., faults that affect the circuit only for a short amount of time (at one invocation). Permanent faults can be modeled by altering the underlying circuit structure according to the fault (as a fault is inherently a gate transformation).

Motivation for Probabilistic Fault Model. In the context of FIA, we consider an adversary (cf. Sect. 3.3) who is deliberate in the choice of faults that they inject. Specifically, an adversary will always try to inject faults that maximize the gain,

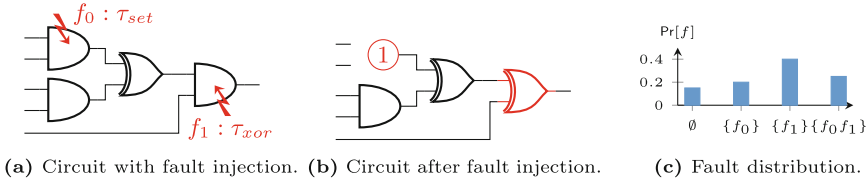


Fig. 2. In our fault model gates are transformed into other gate types depending on the fault type $\tau \in \mathcal{T}$. Each fault combination occurs with a probability $\Pr[f] \in \mathbb{D}\mathcal{F}$.

i.e., maximize the leakage of some secret value. However, in practice, an adversary is restricted in the means of fault injection, resulting in a certain imprecision in the fault location. While precise methods exist, e.g., via focused laser beams [52], those methods are expensive in equipment and challenging in execution. Other fault methods, e.g., clock or voltage glitching [1, 50], are much easier and cheaper but have a widely dispersed effect on the circuit. For example, the effect of clock glitching is determined by the timing behavior of each path given the current inputs together with the previously stored values of each register [44]. Similarly, a laser attack where the diameter of the laser beam is larger than the size of the transistor switches in the underlying technology may simultaneously affect multiple neighboring gates. Hence, mostly an attacker is not in total control of the effect the injected fault has on the circuit, which leads to a probabilistic fault behavior. We model this probabilistic behavior by $\mathbb{D}\mathcal{F}$. While we extend the fault model with probabilistic faults and use $\mathbb{D}\mathcal{F}$ in our subsequent analysis, we do not answer the question of how to come up with reasonable fault distributions. However, in practice, all fault attacks have implicit or explicit assumptions on $\mathbb{D}\mathcal{F}$ (e.g., uniformly distributed faults) which can be used for analysis. Ideally, deriving $\mathbb{D}\mathcal{F}$ would be part of a security-aware Electronic Design Automation (EDA) environment, which computes a reasonable approximation given some fault injection parameters. However, as this is a complex research question in its own right, we leave this for future work.

3.3 Adversary Model

The adversary \mathcal{A}_f gets access to a circuit C that can be invoked exactly twice with the same input, once without manipulation and once with manipulation via fault injection. For this, \mathcal{A}_f has access to the circuit structure as a directed graph and the corresponding channel matrix. The goal of \mathcal{A}_f is to learn the specific input $s \in \mathcal{S}$ of which \mathcal{A}_f has prior knowledge of the general distribution $\mathbb{D}\mathcal{S}$. In addition, \mathcal{A}_f knows the distribution over the other inputs $\mathbb{D}\mathcal{X}$ (which is independent of $\mathbb{D}\mathcal{S}$) and of the distribution over possible faults $\mathbb{D}\mathcal{F}$.

Usually, the distribution of secrets $\mathbb{D}\mathcal{S}$ is defined as a uniform distribution, meaning that the adversary has no prior knowledge about the secret. The distribution over the other inputs $\mathbb{D}\mathcal{X}$ can be adjusted to model the specific scenario of interest. For example, a uniform distribution to represent other secret values \mathcal{A}_f is not particularly interested in, a distribution with $\Pr[x] = 1$ for some $x \in \mathcal{X}$

to represent a known or adversary chosen input value, or something in between for scenarios where \mathcal{A}_f has some knowledge about the input, e.g., because of formatting or padding.

After the invocation of C , \mathcal{A}_f gets access to a fault-free output y and a faulty output y' . Providing \mathcal{A}_f with the pair (y, y') marks a powerful attacker who can observe the precise effect of the injected fault. Often this is justified by an adversary who can run a circuit multiple times with the same input (as the channel C is stateless). By using probabilistic channels, our model also accounts for circuits where different probabilistic choices are used to derive y and y' . Importantly \mathcal{A}_f is an information-theoretical adversary, and we do not restrict the computational power. Also, the adversary always tries to learn a secret and is not interested in behavior manipulation for other purposes, e.g., denial of service or bypassing access control.

3.4 Constructing a FIA Channel

Channel Composition. Channels can be combined into larger channels according to certain rules. In the following, we provide the composition rules for two types of two-channel compositions. Of course, these can be extended to the composition of any number of channels through iterative composition.

Parallel Composition. The simplest variant of channel composition is a parallel composition [3], where the adversary gets the output of two independent runs of two channels. The resulting channel matrix can be computed from the channel matrices being composed. In particular, given two channels $C^1 : \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2 : \mathcal{X} \rightarrow \mathcal{Y}^2$ the entries for the parallel channel matrix can be computed as $(C^1 \parallel C^2)_{x,(y_1,y_2)} = C^1_{x,y_1} \cdot C^2_{x,y_2}$.

Composition via Internal Probabilistic Choice. Another form of composition is internal probabilistic choice [3], where the adversary gets the output of only one of two channels, but does not know which one was chosen. Again, the resulting channel matrix can be computed from the two channels. Given two channels $C^1 : \mathcal{X} \rightarrow \mathcal{Y}^1$ and $C^2 : \mathcal{X} \rightarrow \mathcal{Y}^2$ the entries for the composed channel can be computed by

$$(C^1 \oplus_r C^2)_{x,y} = \begin{cases} \Pr[C^1]C^1_{x,y} + (1 - \Pr[C^1])C^2_{x,y} & y \in \mathcal{Y}^1 \cap \mathcal{Y}^2 \\ \Pr[C^1]C^1_{x,y} & y \in \mathcal{Y}^1 \setminus \mathcal{Y}^2 \\ (1 - \Pr[C^1])C^2_{x,y} & y \in \mathcal{Y}^2 \setminus \mathcal{Y}^1 \end{cases}$$

FIA Channel. We use the above composition rules to construct a channel that represents the adversary's view and knowledge. In particular, we construct individual channels C^f for all fault combinations $f \in \mathcal{F}$, such that C^f represents the circuit C under the fault combination f . All these channels are composed by internal probabilistic choice under the distribution $\mathbb{D}\mathcal{F}$ to model a randomly

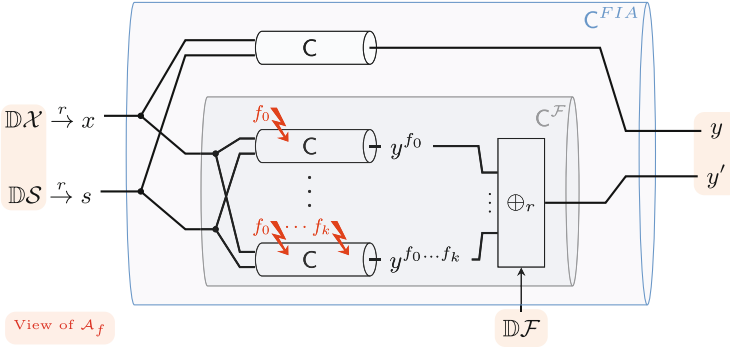


Fig. 3. Construction of a FIA information channel via composition.

selected fault scenario. The resulting channel C^F is then composed in parallel with the unaltered circuit channel C , to acknowledge the leakage of pairs of correct and faulty outputs. An overview of the construction is given in Fig. 3. As a result, the overall channel matrix can be computed as

$$C_{(x,s),(y,y')}^{FIA} = C_{(y,s),y} \sum_{f \in \mathcal{F}} \Pr[f] C_{(x,s),y'}^f$$

By providing the same input x and s to all channels, we can model the correct and incorrect channel executions as independent runs while still guaranteeing that the output pair (y, y') results from the same inputs.

Coverage of FIA Channel. The presented FIA channel covers a wide range of possible fault attacks and circuit structures. In particular, the channel represents all attacks that use only faulty outputs, use only correct outputs (where faulty outputs are suppressed), and use a combination of faulty and correct outputs. While the last attack scenario is trivially covered, the other two scenarios are included because the FIA channel is a composition of the channels that model the attacks where only one type of output is used. In particular, the composition strategies used do not reduce the amount of leaked information. The proposed channel is restricted in the sense that \mathcal{A}_f only has access to a single output pair (y, y') . However, we argue that the analysis of leakage for a single output pair is a good enough approximation of the severity of attacks using multiple output pairs since these attacks also rely on the existence of leakage for a single output pair. Similarly, an attacker using multiple faulty outputs y'_i for each non-faulty output y relays on the existence of leakage in the case of a single faulty y' .

Since the only requirement for the channel is that it is stateless, a wide range of countermeasures can be evaluated. In particular, all countermeasures that do not depend on a stored state are covered by our model, e.g., detection, correction, and infection based on redundancy in time, space, or information. An example of a countermeasure that is out of scope is an implementation that stores all used inputs and never responds to an input a second time.

3.5 FIA Vulnerability and Leakage

Vulnerability. After constructing the appropriate channel for FIA, we can now compute the leakage of a given fault scenario and use this leakage as a metric for severity. In general, the *vulnerability* V is defined as the probability that an adversary can guess the secret in one attempt [3, 53]. In particular, an adversary would always guess the most likely value to maximize the probability of success and hence $V[S] = \max_{s \in \mathcal{S}} \Pr[S = s]$. However, we are interested in the vulnerability given some output observation, which is given by the *conditional vulnerability* [3, 53] with $V[S | Y] = \sum_{y \in \mathcal{Y}} \Pr[Y = y] \cdot \max_{s \in \mathcal{S}} \Pr[S = s | Y = y]$. Intuitively, the conditional vulnerability provides the expected probability that an adversary can guess the secret in one attempt, given a particular observation. Using the expected probability makes it a property of the channel rather than of a specific observation. Hence, the FIA vulnerability separates the secrecy loss caused by a fault injection from the specific attack and provides a general metric. Computing the conditional vulnerability is more practical when computed as $V[S | Y] = \sum_{y \in \mathcal{Y}} \max_{s \in \mathcal{S}} \Pr[Y = y | S = s] \Pr[S = s]$, which is equivalent using Bayes' Theorem [53]. Then, using the channel C^{FIA} defined above, we can derive the vulnerability for FIA as:

$$\begin{aligned} V[S | Y, Y'] &= \sum_{y, y'} \max_s (\Pr[s] \Pr[y, y' | s]) \\ &\stackrel{(*)}{=} \sum_{y, y'} \max_s \left(\sum_x \Pr[s] \Pr[x] \Pr[y, y' | x, s] \right) \\ &\stackrel{(**)}{=} \sum_{y, y'} \max_s \left(\sum_x \Pr[s] \Pr[x] \Pr[y | x, s] \sum_f \Pr[f] \Pr[y' | x, s] \right). \end{aligned}$$

* Law of total probability ** Channel decomposition

Information Leakage. Translating the vulnerability to a measure of bits leads to the *min-entropy* $H_\infty(S) = \log_2(1/V[S])$ or the *conditional min-entropy* $H_\infty(S | Y) = \log_2(1/V[S | Y])$ [3, 53]. Here, the min-entropy is an expression of the residual uncertainty of the secret for the adversary. Finally, we can compute the *information leakage* L as the difference between the uncertainty before and after the circuit execution, i.e., $L[S | Y] = H_\infty(S) - H_\infty(S | Y)$ [3, 53].

The information leakage is a measure of the information that an adversary can expect to learn about a secret after observing the execution of the system (in our case, the FIA channel), i.e., the leakage is weighted by the probability of occurrence. Thus, a non-integer value can be interpreted as: *There is a certain probability that some of the secret bits will be leaked to the adversary.*

3.6 Composition of Independent Fault-Channels

With the above-given definition of vulnerability, we can provide a meaningful definition of independence in the context of faults. In particular, two faults (or

sets of faults) are independent of each other if the vulnerability caused by both faults can be split into the product of the individual vulnerabilities for each fault (with a correction term for the general vulnerability of the secret). Hence, the definition mirrors the definition of independent probabilities.

Definition 1. For a given channel C two fault combinations $f_0, f_1 \in \mathcal{F}$ are independent iff

$$V[S | C^{f_0} \rightarrow (Y, Y')] \cdot V[S | C^{f_1} \rightarrow (Y, Y')] = V[S] \cdot V[S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')].$$

Intuitively, this means that the impact of f_0 does not interfere with the impact of f_1 and vice versa. This gets more obvious when considering the leakage of two independent faults, which gets additive. Hence, the two faults leak different, i.e., independent, bits about the secret.

Theorem 1. Two fault combinations $f_0, f_1 \in \mathcal{F}$ are independent iff $L[S | C^{f_0} \rightarrow (Y, Y')] + L[S | C^{f_1} \rightarrow (Y, Y')] = L[S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')]$.

Proof.

$$\begin{aligned} V[S | C^{f_0} \rightarrow (Y, Y')] \cdot V[S | C^{f_1} \rightarrow (Y, Y')] &= V[S] \cdot V[S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')] \\ \Leftrightarrow H_\infty(S | C^{f_0} \rightarrow (Y, Y')) + H_\infty(S | C^{f_1} \rightarrow (Y, Y')) \\ &= H_\infty(S) + H_\infty(S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')) \\ \Leftrightarrow H_\infty(S) - H_\infty(S | C^{f_0} \rightarrow (Y, Y')) + H_\infty(S) - H_\infty(S | C^{f_1} \rightarrow (Y, Y')) \\ &= H_\infty(S) - H_\infty(S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')) \\ \Leftrightarrow L[S | C^{f_0} \rightarrow (Y, Y')] + L[S | C^{f_1} \rightarrow (Y, Y')] &= L[S | C^{f_0 \wedge f_1} \rightarrow (Y, Y')] \end{aligned}$$

The property of independent faults can be used to analyze fault scenarios in isolation rather than in combination. Usually, the independence of faults follows from the underlying circuit structure, i.e., the propagation path of the faults does not cross. Then, the leakage can be computed independently for each circuit part and then be combined by Theorem 1 (cf. Sect. 5.1 and 5.2).

4 Methodology for Computation

Below we present how to efficiently compute the FIA vulnerability and leakage with Reduced Ordered Binary Decision Diagrams (ROBDDs) [2, 13], a canonical, graph-based representation of boolean functions based on the *Shannon Decomposition* (in accordance with the literature we refer to ROBDDs as BDDs throughout this work). The transformation from a circuit to a BDD is simple but limited to DAGs. As a consequence, no looping is allowed within the circuit, and thus no iterative circuits are supported. However, such circuits can be trivially supported by simple *loop unrolling*. In addition, we restrict ourselves to deterministic circuits/channels for simplicity. However, we show below how to transform any probabilistic channel into a deterministic channel, so this is not a real limitation of our approach.

4.1 Circuit Transformation

We begin by describing some basic circuit/channel transformations required as preprocessing for our main algorithm.

Deterministic and Probabilistic Channels. Our approach is tailored to deterministic channels. However, we show that any probabilistic channel can be transformed into an equivalent deterministic channel. As a result, the proposed analysis approach applies to *all* channels, after a preprocessing for probabilistic channels has been applied. The transformation is done by adding additional *random* input variables that externalize the random choices of the channel. The formal result is given in Theorem 2, where we use the notation $C \rightarrow y$ to indicate that the value y is an output of channel C .

Theorem 2. *Any probabilistic channel $C^P : \mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$ can be transformed to a deterministic channel $C^D : \mathcal{X} \times \mathcal{R} \rightarrow \mathcal{Y}$ with $\Pr[C^P \rightarrow y \mid x] = \Pr[C^D \rightarrow y \mid x]$.*

Proof. Let $C^P : \mathcal{X} \rightarrow \mathbb{D}\mathcal{Y}$ be a probabilistic channel with corresponding channel-matrix entries $0 \leq C_{x,y}^P \leq 1$. By definition of the channel matrix it holds that $C_{x,y}^P = \Pr[y \mid x]$. We start by defining a set of random elements \mathcal{R} such that there is a distinct $r_x \in \mathcal{R}$ for each $x \in \mathcal{X}$ with $\exists y \in \mathcal{Y} : C_{x,y}^P \notin \{0, 1\}$. Further, let all $r_x \in \mathcal{R}$ be of $\lceil \log_2(|\mathcal{Y}|) \rceil$ bits such that each value $r_x = i$ selects one output value $y_i \in \mathcal{Y}$. We define the distribution $\mathbb{D}\mathcal{R}$ such that $\forall r_x \in \mathcal{R}$ it holds that $\Pr[r_x = i] = C_{x,y_i}^P = \Pr[y \mid x]$. Now we define a deterministic channel $C^D : \mathcal{X} \times \mathcal{R} \rightarrow \mathcal{Y}$ with the following channel matrix entries:

$$C_{(x,r_x),y_i}^D = \begin{cases} 1, & \text{if } C_{x,y_i}^P \neq 0 \wedge r_x = i \\ 0, & \text{otherwise} \end{cases}$$

This is the required channel for the following reasons:

$$\begin{aligned} \Pr[C^D \rightarrow y_i \mid x] &\stackrel{(*)}{=} \sum_j \Pr[y_i \mid x \wedge r_x = j] \Pr[r_x = j] \\ &\stackrel{(**)}{=} \Pr[y_i \mid x \wedge r_x = i] \Pr[r_x = i] \\ &\stackrel{(***)}{=} \Pr[r_x = i] \\ &\stackrel{(***)}{=} \Pr[C^P \rightarrow y \mid x] \end{aligned}$$

* Law of total probability ** Definition $C_{(y,r_x),y_i}^D$ *** Definition $\Pr[r_x = i]$

Since the adversary \mathcal{A}_f has access to the channel matrix C^P , it is essential to provide \mathcal{A}_f with the distribution $\mathbb{D}\mathcal{R}$ after the transformation to C^D . Otherwise, the adversary would be less powerful than before the transformation,

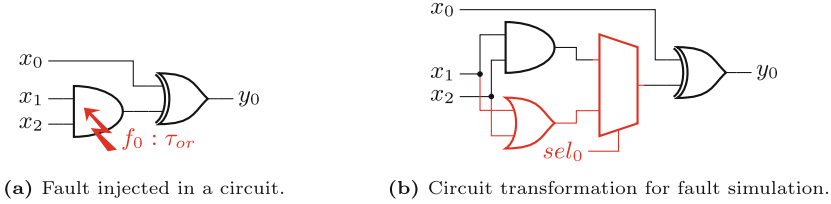


Fig. 4. A circuit under fault injection is adapted to include a MUX that selects between the correct and the faulty behavior of the circuit.

since the details of the probabilistic choices would be missing. In addition, the computation of the vulnerability in the channel C^{FIA} changes to:

$$V[S | Y, Y'] = \sum_{y, y'} \max_s \left(\sum_{x, r} \Pr[s] \Pr[x] \Pr[r] \Pr[y | x, s, r] \sum_f \Pr[f] \Pr[y' | x, s, r] \right) \tag{1}$$

Fault-Selection Variables. The adversary and fault model used allows \mathcal{A}_f to select a subset of gates and change them to different gate types according to the specified fault types. A trivial algorithm would instantiate a different circuit C^f for each combination of faults $f \in \mathcal{F}$ (as indicated in Fig. 3) and change the gates accordingly. These different circuit instances can be distributed in time or space. We introduce a more efficient method by proposing a way to encode all possible fault combinations into a single circuit representation. To do this, we introduce new *fault-selection signals* sel_g for each fault location $g \in \mathcal{G}'$. Then, for each fault location $g \in \mathcal{G}'$, we add a Multiplexer (MUX) to the circuit so that the select signal of the MUX is driven by the signal sel_g . Now, the MUX will pass the output of g if $sel_g = 0$, or otherwise select a fault type. For each fault type τ , a different gate is added to the circuit so that the gate type corresponds to τ , the inputs of the gate are the same as the inputs of g , and the output is connected to the MUX. An example instantiation is shown in Fig. 4. In this way, we can evaluate different combinations of faults $f \in \mathcal{F}$ by activating the corresponding fault selection signals sel_g . Of course, this increases the size of the circuit and thus the size of the associated BDD, but it eliminates the need to create each faulty circuit individually.

4.2 Computation of FIA Vulnerability

General Idea. A naive computation of the FIA vulnerability $V[S | Y, Y']$ starts by deriving the channel matrix C^{FIA} , as shown in Fig. 3, by iterating over all $x \in \mathcal{X}$, $s \in \mathcal{S}$, $r \in \mathcal{R}$, and $f \in \mathcal{F}$, before computing Eq. 1. Note, however, that after transforming the probabilistic channel C^{FIA} into a deterministic channel, all entries in C^{FIA} are either one or zero. Thus, the matrix is just an encoding of valid input/output pairs that can be efficiently encoded via a BDD. When computing Eq. 1, we ensure that we only iterate over valid pairs of (y, y') by fixing

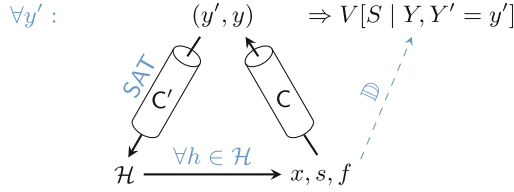


Fig. 5. High-level overview of the principle of Algorithm 1, with $\mathcal{H} = \mathcal{X} \times \mathcal{S} \times \mathcal{R} \times \mathcal{F}$.

the faulty output y' , determining all input and fault combinations that can lead to y' , and computing only the corresponding non-faulty outputs y . This general principle is illustrated in Fig. 5 and can be efficiently realized using BDDs. In addition, this approach allows for parallel and probabilistic computation of $V[S | Y, Y']$, as we discuss below and in Appendix A.

Deterministic Computation. Our BDD-based algorithm for computing the FIA vulnerability $V[S | Y, Y']$ is given in Algorithm 1. The use of BDDs allows the efficient inversion of a channel since the set of satisfying assignments of a function is given by all paths ending in the true-leave of the function's BDD (without storing the entire channel matrix).

The input to Algorithm 1 is a circuit C and a set of fault combinations \mathcal{F} . We first extend the circuit with fault selection variables (cf. Sect. 4.1) and construct the corresponding channel BDD C' . Then, according to Fig. 5, we iterate over all possible faulty output values y' and compute the corresponding part of the vulnerability. For the actual computation, we need all inputs $x \in \mathcal{X}$, $s \in \mathcal{S}$ in addition to all fault combinations $f \in \mathcal{F}$ that can lead to the faulty output y' . This gives us the set of satisfying assignments $\mathcal{H} = \mathcal{X} \times \mathcal{S} \times \mathcal{R} \times \mathcal{F}$ of the BDD that encodes the output y' . The set of satisfying assignments also contains the specific values for $r \in \mathcal{R}$ that lead to the faulty output. However, to get all possible pairs (y, y') we consider all possible assignments to r when computing y (cf. Line 12). In addition, since we have a deterministic channel, it holds that $\Pr[y' | x, s, r, f] = 1$ for $(x, s, r, f) \in \mathcal{H}$ while $\Pr[y' | x, s, r, f] = 0$ otherwise. Given the set \mathcal{H} , we can compute all pairs (y, y') with $\Pr[y, y' | x, s, r, f] = 1$ and then compute the vulnerability $V[S | Y, Y' = y']$.

Since Algorithm 1 isolates the computation of $V[S | Y, Y' = y']$ for each y' , it is easy to parallelize the computation and derive the overall vulnerability V by summing up all individual $V[S | Y, Y' = y']$ at the end.

Complexity. The runtime of Algorithm 1 depends on the circuit structure, the number of faults, and their locations. Without this information, it is hard to estimate the number of matching faults in Line 8 and the size of \mathcal{L} in each loop iteration. This makes the impact of the fault cardinality a function of the circuit and the fault locations. However, if we inspect the algorithm considering its use with deterministic circuits, we can see that throughout all iterations of the outer loop, the for loop in Line 6 is executed for all inputs x, s , since in a deterministic

Algorithm 1: Fault-vulnerability for deterministic circuits.

```

1 function fault_vulnerability( $C, \mathcal{F}$ ):
  // Get BDD of circuit
2  $C' \leftarrow \text{BDD}[\text{add\_fault\_mux}(C, \mathcal{F})]$ 
  // Compute vulnerability
3  $V \leftarrow 0$ 
4 for  $y' = 0$  to  $2^m$  do
  // Get satisfying assignment for given  $y'$ 
5  $\mathcal{H} \leftarrow \text{SAT}(\text{BDD}[C' = y'])$ 
6  $\mathcal{L} \leftarrow \emptyset$ 
7 for  $\forall (x, s) \in \mathcal{H}$  do
  // Compute  $\sum_f \text{Pr}[f] \text{Pr}[y' | x, s]$ 
8  $f_{pr} \leftarrow 0$ 
9 for  $f$  with matching  $(x, s)$  do
10  $f_{pr} \leftarrow f_{pr} + \text{Pr}[f]$ 
11 for  $\forall r \in \mathcal{R}$  do
12  $y \leftarrow C(x, s, r)$ 
  // Sum up according to matching  $y$ 
13 if  $(y, s) \in \mathcal{L}$  then
14  $\mathcal{L}_{y,s} \leftarrow \mathcal{L}_{y,s} + \text{Pr}[x] \cdot f_{pr} \cdot \text{Pr}[s]$ 
15 else
16  $\mathcal{L}_{y,s} \leftarrow \text{Pr}[x] \cdot f_{pr} \cdot \text{Pr}[s]$ 
  // Sum up everything
17 for  $\forall y \in \mathcal{L}$  do
18  $V \leftarrow V + \max(\mathcal{L}_y)$ 
19 return  $V$ 

```

circuit all inputs result in exactly one output value. Later, and within this for loop, there is a loop over all random values r . Thus, the algorithm is at least exponential in the number of input bits (secret, non-secret, and random).

4.3 Implementation

We integrated the computation of the FIA vulnerability and leakage, as described in Sect. 4.2, into the state-of-the-art verification tool VERICA [42]¹. VERICA is a BDD-based framework for verifying the independence of secrets and probe distributions for SCA, evaluating the impact of fault propagation on the circuit output for FIA, and the combination of both for Combined Analysis (CA). In contrast to the general fault model (cf. Sect. 3.2), we also allow faults at inputs to model cases where the analysis is performed for a part of a larger design, and thus faults can be placed outside of the analyzed section.

¹ <https://github.com/Chair-for-Security-Engineering/VERICA>.

Input/Output. VERICA receives as input a Verilog netlist of the design under test, an annotation file that defines the type of input/output signals (e.g., secrets, randomness, sharing, or replication), and a definition of the fault model, i.e., the transformation of gates under faults. In addition, to compute the fault vulnerability, a fault whitelist has to be provided that specifies the fault locations (gates) and the fault probabilities (for specific gates). The tool then outputs the vulnerability (and leakage) of the given scenario.

Restrictions to the Implementation. For simplicity and ease of use, we have made the following design choices for the practical implementation. First, we assume that all faults at different fault locations are independent of each other, i.e., the probability of two faults occurring together is the product of the individual probabilities. This simplifies the definition of the fault distribution $\mathbb{D}\mathcal{F}$ since only one fault probability per fault location needs to be defined. Second, we assume that all inputs $x \in \mathcal{X}$, $s \in \mathcal{S}$, and $r \in \mathcal{R}$ are drawn from an independent and uniform distribution. For s and r , this is the most natural choice for most real-world scenarios. For x , this restriction means that \mathcal{A}_f has no control or prior knowledge of the non-secret input, which limits the applicable scenarios. However, it is a reasonable assumption when x is the internal state of a cryptographic function. Both assumptions are not intrinsic to the way of computation, but a simplification to reduce the burden on the user. Of course, all of the above restrictions (Sect. 3 and Sect. 4) still apply.

5 Measuring the Efficiency of Fault Attacks

In the following section, we evaluate the described methodology for a quantitative FIA metric with respect to the match between theory and practice, by analyzing the efficiency of known fault attacks. To do so, we extend the fault model from Sect. 3.2 with additional faults on input values, i.e., each input bit can be manipulated with a *set*, *reset*, or *bit-flip* fault. While input faults have no counterpart in real-world attacks (an input fault is just another input), it is useful when analyzing only parts of circuits where input faults can occur through fault propagation. Indeed, in the following we only focus on scenarios where we analyze parts of the circuits and, hence, \mathcal{A}_f has no prior knowledge of the non-secret input x . All of our circuits are synthesized using the Synopsys Design Compiler with a subset of cells in the NanGate 45 nm Open Cell Library (OCL).

We report our results using the *leakage* $\mathbb{L}[S | Y, Y']$. In general, vulnerability and leakage are two representations of the same quantitative metric with different advantages and disadvantages. While the vulnerability provides a stand-alone metric for security, the leakage is more human understandable but requires knowledge of the bit width of the secret. We decided to focus on the leakage and provide the theoretical maximum leakage where necessary for interpretation.

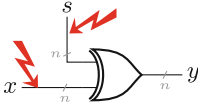


Fig. 6. Setup for key addition.

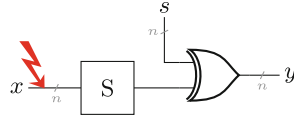


Fig. 7. Setup for DFA.

Table 2. Leakage after injecting faults into a key addition (cf. Fig. 6).

	Fault			Metric		
	location	Pr[f]	type	V^*	H_∞^*	L
input key	s_0	1.0	set	0.125	3.000	1.000
	s_0	1.0	reset	0.125	3.000	1.000
	s_0	1.0	flip	0.062	4.000	0.000
	s_0	0.5	set	0.094	3.415	0.585
input state	x_0	1.0	set	0.125	3.000	1.000
	x_0	1.0	reset	0.125	3.000	1.000
	x_0	1.0	flip	0.062	4.000	0.000
$k=2$	$s_0 s_1$	1.0 1.0	set	0.250	2.000	2.000
	$s_0 s_1$	1.0 0.5	set	0.188	2.415	1.585

* with $n = 4$.

Theoretical maximum: $L_{KeyAdd}^{max} = 4$.

5.1 Faulting Key Addition

We start our analysis with a simple key addition, i.e., an xor between some intermediate state x and some secret key s , as shown in Fig. 6. This is a well-understood construction with respect to FIA and serves as a trivial test of the soundness of our approach. The results for a 4-bit word are given in Table 2. As expected, *set/reset* faults on a key bit s_i lead to the leakage of one bit (the faulted key bit), while a bit flip does not reveal any secret information (because x is unknown). The reason is that for *set/reset* faults, the two outputs y and y' are equal if and only if the injected fault is ineffective. In contrast, a bit flip always results in an effective fault at the output. A similar effect can be achieved by corrupting a state bit x_i , as knowing two bits of an xor operation completely determines the third bit. An interesting property of the key addition is that each bit is processed individually, and thus faults injected into a key bit s_i only affect the leakage of that particular key bit. In other words, the injected faults are independent (cf. Definition 1) and it is sufficient to analyze a single construction and scale the leakage to the number of faulted key bits.

5.2 Differential Fault Analysis

One of the most common fault attacks on cryptographic ciphers is DFA [9], where an adversary \mathcal{A}_f gains access to a set of correct and faulty ciphertext

Table 3. DFA against PRESENT and DEFAULT with a single S-box (cf. Fig. 7). Each fault is a bit-flip with probability $\Pr[f_i] = 1$ at the indicated input x_i . In addition, we provide the number of key candidates left after observation of one pair (y, y') , where 256×16 means that in 256 cases for (y, y') there are 16 key candidates left.

Fault		PRESENT		DEFAULT		
		L	Key Candidates	L	Key Candidates	
input	-	0.000	256×16	0.000	256×16	
	$x_0 x_1 x_2 x_3$	2.000	256×4	0.000	256×16	
	x_0	2.000	256×4	1.000	256×8	
	$x_0 x_3$	2.585	$128 \times 2, 128 \times 4$	0.000	256×16	
	$x_1 x_2$	2.585	$128 \times 2, 128 \times 4$	0.000	256×16	
	$x_0 x_1 x_2$	2.585	$128 \times 2, 128 \times 4$	1.000	256×8	
	x_3	2.585	$128 \times 2, 128 \times 4$	1.000	256×8	
	x_1	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	x_2	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_0 x_1$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_0 x_2$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_1 x_3$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_2 x_3$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_0 x_1 x_3$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_0 x_2 x_3$	2.807	$192 \times 2, 64 \times 4$	1.000	256×8	
	$x_1 x_2 x_3$	3.000	256×2	1.000	256×8	
	Theoretical maximum: $L_{PRESENT}^{max} = 4, L_{DEFAULT}^{max} = 4.$					

pairs (y, y') and uses statistical analysis to reduce the search space for the secret key s . For DFA to work, \mathcal{A}_f must inject bit-flip faults into an intermediate state with a subsequent nonlinear layer. For block ciphers, this is commonly done by attacking the last-round key and targeting the faults at the input of the last-round S-boxes. Hence, we have $y = S(x) \oplus s$ and $y' = S(x \oplus \Delta) \oplus s$ for an intermediate state x and a fault difference Δ (a potential last-round linear layer can be removed by choosing the corresponding output bits accordingly). More advanced attacks use the same principle but inject faults in earlier rounds to exploit fault propagation in the given cipher structure to affect multiple S-boxes at once. Therefore, to analyze the susceptibility of ciphers to DFA, it is sufficient to analyze the S-box and key addition of the last round, as shown in Fig. 7, with bit-flip faults on the input bits x_i . This simplification removes the propagation of faults to the input of the last-round S-boxes. While it removes a potential dependency between faults at the S-box input this is a common way to analyze DFA [48]. Similar to the key addition analyzed in the last section, the last-round S-boxes are also in parallel to each other, and faults injected into different S-boxes are independent of each other. Hence, we can restrict our analysis to a single instance and scale the leakage to the full set of affected S-boxes.

4-Bit S-boxes. The results for two 4-bit S-boxes are given in Table 3, along with the number of key candidates remaining after observation of one pair (y, y') . In particular, we analyze the S-boxes of PRESENT [11], a cipher with a focus on implementation efficiency in hardware, and DEFAULT [5], which is specifically designed to resist DFA. First, we observe that the leakage metric is directly related to the number of remaining key candidates. For example, for PRESENT we have a leakage of $L = 2$ when flipping the bit x_0 . For the same scenario, it holds that for all possible pairs of (y, y') , 4 out of 16 key candidates remain after observing (y, y') , which means that \mathcal{A}_f learns exactly 2 bits of the key. For the leakage metric, a higher value is better for the adversary, corresponding to a lower number of remaining key candidates. In addition, our metric also correctly measures the improved resistance of the DEFAULT S-box against DFA.

8-Bit S-boxes. We also analyzed AES regarding its susceptibility to DFA by using the AES S-box in the construction of Fig. 7. Interestingly, for all possible input differentials (with $\Pr[f] = 1$), we computed a leakage of $L = 6.989$ bits (where 8 is the theoretical maximum), i.e., all faults are equally bad and leak almost all possible key bits. This is confirmed by the number of key candidates left which is $1024 \times 4 \wedge 64512 \times 2$ for all fault scenarios. This behavior changes when each bit flip occurs with a probability $\Pr[f] < 1$, e.g., a bit-flip probability of $\Pr[f] = 0.5$ leads to a leakage of $L = 6$ when only bit x_7 can flip and $L = 0$ when all bits can flip. However, further analysis is needed to obtain realistic fault scenarios (cf. Sect. 8.2) and their leakage behavior.

5.3 Statistical Ineffective Fault Analysis

Another popular fault attack is SIFA [23], which exploits the knowledge that a fault is injected into some intermediate state but has no effect on the output. This is useful in situations where faulty outputs are suppressed, so \mathcal{A}_f only knows correct outputs and whether a fault is ineffective (has no effect). A *sufficient* condition for a SIFA vulnerability is a statistical dependency between the detection behavior and the secret. That is if an attacker can learn something about the secret just by observing the detection behavior (in the form of a detection flag or suppression of incorrect outputs), SIFA is possible. However, contrary to claims in the literature [31], we later show that this is not a *necessary* condition for SIFA, i.e., security does not follow from the absence of dependency.

In the following, we show that our tool accurately detects leaks from SIFA when there is a statistical dependency between the detection flag and secrets. To do this, we implement two instances of a design and feed the outputs to a detection module, as shown in Fig. 8. Then, faults are injected into only one instance to maintain the correct functionality of the detection module. By considering only the detection flags as outputs, we can ensure that the only leakage detected by our metric results from a SIFA vulnerability.

For the evaluation, we selected three circuits discussed by Daemen et al. [20] based on masking (cf. Sect. 2.2). On its own, masking is not sufficient to protect

Table 4. SIFA against *ISW* multiplication, and χ_3 and χ_5 with masked Toffoli gates (cf. Fig. 8). All faults occur with probability $\Pr[f] = 1$ and no randomness source is faulted.

	Fault		Metric L		
	location	type	<i>ISW</i>	χ_3	χ_5
internal $k = 1$	any	set	0.000	0.000	0.000
	any	reset	0.000	0.000	0.000
	any	flip	0.000	0.000	0.000
input max	$s_{0,0}$	set	0.585	0.000	0.000
	$s_{0,0}$	reset	0.585	0.000	0.000
	$s_{0,0}$	flip	1.000	0.000	0.000
$k = 2$	$\text{inv}(s_{0,1})$ $\text{inv}(s_{0,0})$	set	–	0.585	0.322

Theoretical maximum: $L_{ISW}^{max} = 2$, $L_{\chi_3}^{max} = 3$, $L_{\chi_5}^{max} = 5$.

against SIFA, but it provides valuable properties by ensuring that intermediate values are independent of secrets [23]. Below, we consider the circuit’s unshared inputs as secrets, use a security order $d = 1$, and implement a detection module for each share index individually to preserve the independence properties (cf. Fig. 8).

ISW Multiplication. The first circuit we analyze is the ISW multiplication [33] with additional registers to prevent SCA leakage from glitches. This circuit implements a masked **and**, where each input and output is masked with two shares. As can be seen in Table 4, there is no leakage for any internal fault with cardinality $k = 1$ (excluding randomness generation). However, when some input $s_{0,0}$ (first share of first input) is faulted, we see leakage due to a dependency between the secret values and the detection signal [20], i.e., $\Pr[s_0 \cdot s_1 = 1] \neq \Pr[s_0 \cdot s_1 = 1 \mid f \text{ ineffective}]$. Thus, the composition of the ISW multiplication is not SIFA-secure. From the structure of the circuit follows that faults in any input behave the same.

The inferred leakage is specific to the given implementation since implementation changes change the possible fault locations. For example, the Synopsys Design Compiler replaces $\text{and}(s_{0,0}, s_{1,1})$ with $\text{nor}(\text{inv}(s_{0,0}), \text{inv}(s_{1,1}))$ for timing and area optimization unless instructed otherwise. However, this potentially allows \mathcal{A}_f to fault the gate $\text{inv}(s_{0,0})$, which effectively introduces a fault at input $s_{0,0}$ (with the corresponding leakage behavior).

S-boxes from XOODOO and KECCAK. To protect against SIFA, Daemen *et al.* [20] propose circuits constructed by the composition of masked Toffoli gates. A Toffoli gate computes the term $x_0 \oplus (x_1 \cdot x_2)$ and has the property that all injected effective faults (set, reset, bit flip) are effective faults at the output. This guarantees that there is no dependency between the detection behavior

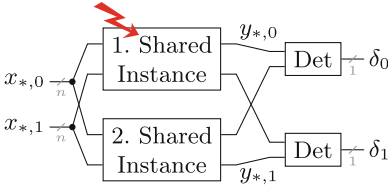


Fig. 8. Setup for SIFA.

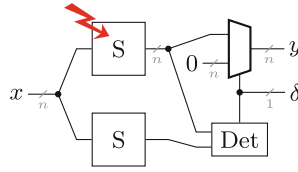


Fig. 9. Setup for analysis of detection.

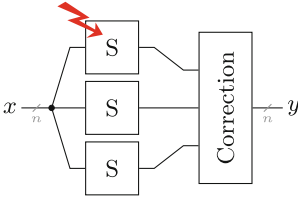


Fig. 10. Setup for analysis of correction.

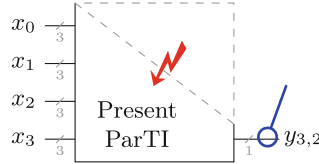


Fig. 11. Setup for CA.

and the secrets since all effective faults are always detected. Two S-boxes that can be easily implemented with Toffoli gates are those from XOODOO [21] and KECCAK [7]². The analyzed instances are claimed to be first-order SIFA secure, i.e., there should be no leakage for a single fault. Our metric accurately shows the independence of the detection behavior and the secrets for $k = 1$, while leakage can be seen for two injected faults (cf. Table 4).

6 Evaluating the Quality of Countermeasures

The quantitative FIA metric cannot only be used to measure the efficiency of fault attacks but also to evaluate the quality of countermeasures or the general resistance of designs against FIA. The last section used known attack vectors to highlight the close match between theory and practice. In contrast, we now show that meaningful insights can be drawn even under the constraints of scalability.

6.1 Detection/Correction

Various countermeasures against FIA have been proposed in the literature based on redundancy in time, space, or information. Common to all countermeasures is that redundancy is used either to react to the detection of faults or to directly correct faults that occur. To evaluate the effectiveness of countermeasures, we use the setting shown in Fig. 7, but with countermeasures applied. As S-box, we choose the PRESENT S-box because of its interesting leakage behavior for DFA (cf. Sect. 5.2), which we implemented according to Cassiers et al. [14].

² We implemented both S-boxes with Toffoli gates in parallel (instead of sequential, as in [20], to get the correct output) and without any registers.

Table 5. Leakage for faults injected in a PRESENT S-box & KeyAdd with countermeasures (cf. Fig. 9 and 10). All faults occurs with probability $\Pr[f] = 1$.

		Fault		Metric L			
		location	type	Plain	Detection (both - y , δ)	Detection (no flag - y)	Detection (flag only - δ)
input max	x_2	set	3.000	1.000	0.954	0.000	0.000
	x_2	reset	3.000	1.000	0.954	0.000	0.000
	x_2	flip	2.807	0.000	0.000	0.000	0.000
internal max	$\text{inv}(x_1)$	set	3.000	1.000	0.954	0.000	0.000
	$\text{inv}(x_1)$	reset	3.000	1.000	0.954	0.000	0.000
	$\text{and}(x_2, \bar{x}_1)$	flip	3.000	1.000	0.954	0.000	0.000

Theoretical maximum: $L_{PRESENT}^{max} = 4$.

Detection/Correction. The most basic countermeasures are based on repetition (in space or time) of the computation, which allows comparing the result of different instances to either detect faults (inequality) or correct them directly (majority voting). To detect an arbitrary set of k faults $k + 1$ instances are required, while for correction, $2k + 1$ instances are required. In Table 5, we show the leakage behavior for countermeasures with $k = 1$ and spatial repetition, where we have chosen one of the locations with the highest leakage for a given scenario. With respect to the countermeasures, we distinguish between *plain* (no countermeasure, no repetition - cf. Fig. 7), *detection* (2 repetitions with output set to 0 on fault detection - cf. Fig. 9), and *correction* (3 repetitions with majority voting - cf. Fig. 10). To get a fine-grained analysis of the detection, we have three different versions of the detection circuit: (i) detection flag δ and output y are given to \mathcal{A}_f , (ii) only the output y is given to \mathcal{A}_f while the detection flag is suppressed, and (iii) only the detection flag δ is given to \mathcal{A}_f while the output is suppressed (cf. Sect. 5.3). We further distinguish between faults at inputs and faults in the internal structure (including the detection/correction logic).

We observe that the detection countermeasure cannot prevent all existing leaks. This is due to the occurrence of ineffective faults and the resulting biased intermediate state, as exploited by SIFA [23]. Implementing the attack in the given scenario reduces the key space to 8 out of 16, which perfectly matches the inferred leakage of $L = 1$. Also, there is no leakage for bit flip faults at the inputs, since these do not introduce any bias [23] and all faults are effective faults at the output [20]. However, we observe no leakage when we look only at the detection flag, which means that there is no statistical dependency between the detection behavior and the secrets. From this, we conclude that there are designs vulnerable to SIFA whose detection behavior is independent of secrets, directly contradicting *Proposition 1* of Hadzic et al. [31] which claim the opposite. Significantly, this has implications for verification methods of SIFA security that reduce to this proposition (e.g., [31, 42]). In cases where there is some leakage for detection, it is always $L = 1$ (we observed the same behavior when using the AES S-box). Whether this is a coincidence or a structural property requires further investigation. For the same cases, suppressing the detection

Table 6. Leakage of faults injected in masked PRESENT S-box with key addition (Fig. 7 with masking) for \mathcal{A}_f obtaining shared outputs ($\mathbb{L}[\{s_{i,j}\} \mid y, y']$) and unshared outputs ($\mathbb{L}[s \mid y, y']$), respectively. All faults occur with probability $\Pr[f] = 1$.

		Fault		$\mathbb{L}[\{s_{i,j}\} \mid y, y']$		$\mathbb{L}[s \mid y, y']$	
		location	type	DOM	HPC ₂	DOM	HPC ₂
input state max ($k = 1$)	$x_{2,0}$	set	2.000	2.000	2.000	2.000	
	$x_{2,0}$	reset	2.000	2.000	2.000	2.000	
	$x_{2,0}$	flip	2.807	2.807	2.807	2.807	
	$x_{2,0} \ x_{2,1}$	set	3.170	3.170	3.000	3.000	
	$x_{2,0} \ x_{2,1}$	reset	3.170	3.170	3.000	3.000	
	$x_{2,0} \ x_{2,1}$	flip	1.585	1.585	0.000	0.000	
secret max	$s_{0,0}$	set	0.000	0.000	0.000	0.000	
	$s_{0,0}$	reset	0.000	0.000	0.000	0.000	
	$s_{0,0}$	flip	0.000	0.000	0.000	0.000	
	$s_{0,0} \ s_{0,1}$	set	1.000	1.000	1.000	1.000	
	$s_{0,0} \ s_{0,1}$	reset	1.000	1.000	1.000	1.000	
	$s_{0,0} \ s_{0,1}$	flip	0.000	0.000	0.000	0.000	
rand. max	r_0	set	1.322	1.322	0.000	0.000	
	r_0	reset	1.322	1.322	0.000	0.000	
	r_0	flip	2.000	2.000	0.000	0.000	
internal max	$\text{inv}(x_{3,0})$	set	1.907	1.907	1.585	1.585	
	$\text{inv}(x_{3,0})$	reset	1.907	1.907	1.585	1.585	
	$\text{inv}(x_{3,0})$	flip	2.700	2.700	2.322	2.322	
	$\text{and}(x_{3,1}, \overline{x_{1,0} \oplus x_{2,0}})$	set	1.700	–	1.700	–	
	$\text{and}(x_{1,0} \oplus x_{2,0}, r_0)$	set	–	1.700	–	1.700	

Theoretical maximum: $\mathbb{L}_{PRESENT}^{max} = 4$.

signal only marginally reduces the leakage, since \mathcal{A}_f can no longer trivially distinguish between 0 as a valid or suppressed faulty output. Finally, as expected, there is no leakage when the correction is used.

Masking. As mentioned above, masking is not a countermeasure against FIA, but against passive SCA, although it has some advantageous properties for the prevention of SIFA. However, real-world implementations must withstand a wide variety of attacks, so it is interesting to analyze masking from a FIA perspective. To do so, we consider the circuit shown in Fig. 7 with masking using $d = 1$ and instantiating the S-box with composable-secure *DOM* [24] and *HPC₂* [14] gadgets (i.e., small but secure subcircuits that can be securely combined). In Table 6, we show the corresponding leakage, again selecting a location with the maximum leakage for a given fault scenario (except for the two simultaneous faults in the input state x , where we selected both shares of x_2).

In general, masking does not seem to harm FIA security. In particular, faults in the input state x have the same effect as without sharing (cf. Table 5). Flipping one share results in a bit-flip in the unshared value, while flipping both shares results in no fault. If the secret itself is faulted, it is necessary to set/reset both shares of the same secret s_i , which results in resetting the unshared secret bit (cf. Sect. 5.1). Faulting random inputs r_i (required to refresh the masking after non-linear operations) does not result in a leak after observing the unshared output. This is expected since the value of the randomness has no functional impact. Also, the leakage decreases as one moves from looking at the shared outputs to the unshared outputs since there is less information in the unshared values. Finally, for this particular design, the type of gadgets has no impact on the leakage behavior. However, this may be different when considering fault combinations from real-world attack scenarios (e.g., clock glitching - cf. Sect. 8.2).

6.2 Detailed Analysis of Combined Vulnerabilities

Real-world circuits are not only exposed to FIA and SCA individually but potentially also to the combination of both attacks. While the VERICA tool [42] can verify security in such a CA setting, it does not provide a quantitative security assessment for vulnerable designs. Similar to the general FIA setting, we can also apply our quantitative metric to CA by computing $V[S | \mathcal{P}, \mathcal{P}']$, where the adversary receives a set of probes \mathcal{P} instead of the outputs Y . Such an analysis may be more efficient than for general FIA (for a given probe set) since it is worthwhile to consider an internal subset of the design structures (which is probed) instead of the entire circuit. In general, security against CA seems to be quite expensive [25], and we hope that a dedicated quantitative analysis can instruct more efficient protection mechanisms with an acceptable leakage level.

For our case study, we chose a ParTI [49] implementation of a PRESENT S-box. ParTI is a protection scheme that combines masking (in particular Threshold Implementation (TI) [18]) with error detection codes to protect against both FIA and SCA individually. However, the scheme does not claim security against combined attacks. We use an implementation with $k = 1$ and $d = 1$, for which we ran VERICA to find a suitable probe position (output $y_{3,2}$) and removed all parts not related to that probe (cf. Fig. 7). We consider all unshared inputs as secrets.

ParTI Implementation of PRESENT Sbox. In Table 7, we show the leakage for all unshared-input bits together (s) and for each unshared-input bit individually (s_i). Concerning the fault locations, we selected locations with the maximum leakage when one input is faulted with a given fault type, the minimum leakage when two inputs are faulted with a given fault type such that the single-bit fault with the most leakage is in the fault pair and the maximum leakage for an internal fault. First, we observe that some fault locations and types leak more across all inputs than the sum of all individual leakages. For example, while $L[s | y_{3,2}] = 0.129$ for reset faults in $s_{3,0}$ and $s_{3,1}$, the same scenario leads to $\sum_i L[s_i | y_{3,2}] = 0,087$. Thus, \mathcal{A}_f learns something about the combination of secrets even if the individual secret is securely hidden, e.g., if it is more likely that $s = \langle 0, 0, 0, 0 \rangle$, even if the distribution over each s_i is uniform. For some of

Table 7. Leakage of Combined Analysis on ParTI implementation of the PRESENT S-box when the output $y_{3,2}$ is probed (cf. Fig. 11). All faults occur with probability $\Pr[f] = 1$.

		Fault		Metric $L[t \mid y_{3,2}, y'_{3,2}]$				
		location	type	$t = s$	$t = s_0$	$t = s_1$	$t = s_2$	$t = s_3$
input max		$s_{3,0}$	set	0.392	0.322	0.044	0.044	0.000
		$s_{3,0}$	reset	0.392	0.322	0.044	0.044	0.000
		$s_{1,0}$	flip	0.585	0.585	0.000	0.000	0.000
		$s_{1,0} s_{3,0}$	set	0.492	0.459	0.000	0.044	0.000
		$s_{1,0} s_{3,0}$	reset	0.492	0.459	0.000	0.044	0.000
		$s_{0,0} s_{1,0}$	flip	0.585	0.585	0.000	0.000	0.000
input min with max $k = 1$		$s_{3,0} s_{0,1}$	set	0.170	0.170	0.000	0.000	0.000
		$s_{3,0} s_{3,1}$	reset	0.129	0.087	0.000	0.000	0.000
		$s_{1,0} s_{0,1}$	flip	0.000	0.000	0.000	0.000	0.000
internal max		$\text{xor}(t_0, \overline{s_{2,0} \oplus s_{3,2}})$	set	0.833	0.807	0.000	0.000	0.000
		$\text{nand}(s_{2,0}, s_{1,2} \oplus s_{3,0})$	reset	0.858	0.807	0.000	0.000	0.022
		$\text{nor}(\overline{s_{1,0}}, t_1)$	flip	1.000	1.000	0.000	0.000	0.000

$$t_0 = s_{2,0} \cdot (s_{1,2} \oplus s_{3,0}); t_1 = s_{0,1} \oplus s_{1,2} \oplus s_{3,0} \oplus x_{2,0}.$$

Theoretical maximum: $L_{PRESENT}^{max} = 4$.

the given fault scenarios, the sum of the individual leakages is larger than the combined leakage. However, since the difference is only marginal, we explain this by rounding errors throughout the computation.

Second, we see that adding additional faults to a fault scenario can have different effects. In Fig. 12, we provide additional insight into three of the given scenarios. Specifically, we show the leakage over the change in fault probability for two faults, where the x-axis is always the probability for a fault in $s_{1,0}$. While adding a bit-flip fault in $s_{0,0}$ has no effect (cf. Fig. 12a), a set fault in $s_{3,0}$ increases the leakage (cf. Fig. 12b). However, the reverse is also possible, i.e., a fault that reduces or even eliminates the leakage (e.g., bit-flip $s_{0,1}$ - cf. Fig. 12c). This gives us a new protection scheme against FIA by constructing a design structure that ensures that when a fault occurs in $s_{1,0}$, there is also a fault in $s_{0,1}$. Since most faults are related to the timing behavior of a circuit [44], this can be achieved by delaying some of the signals in the circuit. We can also see that, in general, for a single fault, a lower fault probability means a lower leakage, which follows directly from the equation for the vulnerability (cf. Eq. 1).

7 Performance of Prototype Implementation

In the following, we provide some insight into the performance of our implementation. We run all experiments on a 64-bit Linux Operating System (OS) environment on an Intel Xeon E5-1660v4 CPU with 16 cores, a clock frequency of 3.20 GHz, and 128 GB of RAM.

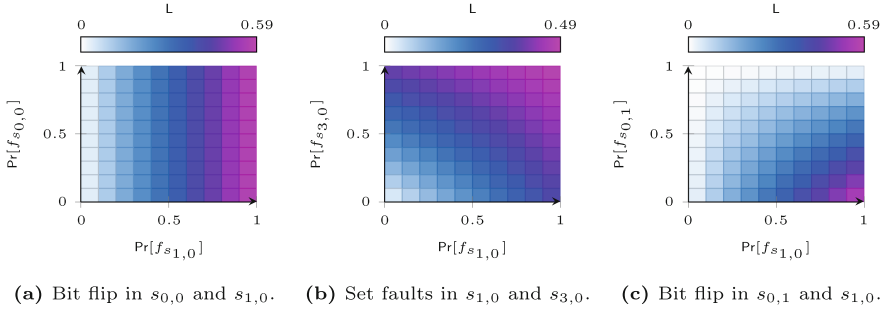


Fig. 12. Leakage behavior of fault combinations under changing fault probability.

Table 8. Execution time for vulnerability computation on a 64-bit Linux OS executing on an Intel Xeon E5-1660v4 CPU with 16 cores, a clock frequency of 3.20 GHz, and 128 GB of RAM.

	Design					Fault			Metric		
	Description	in_x, r	in_s	out	comb.	reg.	k	$Pr[f]$	type	time	L
PRESENT *	Plain	4	4	4	26	0	1	0.7	flip	0.82 s	2.561
	Plain	4	4	4	26	0	4	0.7	flip	0.80 s	2.407
	DOM	8	4	4	91	16	1	0.7	flip	1.15 s	0.766
	DOM	8	4	4	91	16	4	0.7	flip	3.56 s	0.766
	HPC ₂	8	4	4	115	44	1	0.7	flip	1.16 s	1.036
	HPC ₂	8	4	4	115	44	4	0.7	flip	2.69 s	1.514
	2×	8	8	8	52	0	1	0.7	flip	0.81 s	2.561
	2×	8	8	8	52	0	4	0.7	flip	0.90 s	3.124
	3×	12	12	12	78	0	1	0.7	flip	8.57 s	2.561
	3×	12	12	12	78	0	4	0.7	flip	38.70 s	3.124
AES *	4×	16	16	16	104	0	1	0.7	flip	44.30 min	2.561
	4×	16	16	16	104	0	4	0.7	flip	3.50 h	3.124
	Plain	8	8	8	143	0	1	0.7	flip	0.84 s	7.043
	Plain	8	8	8	143	0	4	0.7	flip	1.03 s	5.712
	Detection	8	8	8	319	0	1	0.7	flip	0.96 s	0.000
	Detection	8	8	8	319	0	4	0.7	flip	1.62 s	0.000
	Correction	8	8	8	464	0	1	0.7	flip	0.97 s	0.000
	Correction	8	8	8	464	0	4	0.7	flip	5.88 s	0.000
KECCAK **	2×	16	16	16	288	0	1	0.7	flip	1.22 h	7.043
	2×	16	16	16	288	0	4	0.7	flip	6.89 h	13.359
	1 B. Input	0	10	10	1175	0	1	0.7	flip	1.99 s	9.743
	1 B. Input	0	10	10	1175	0	4	0.7	flip	3.36 s	9.970
	2 B. Input [†]	10	10	10	2392	0	1	0.7	flip	5.59 min	7.995
	2 B. Input [‡]	10	10	10	2392	0	4	0.7	flip	59.11 min	6.643

*S-box & KeyAdd; ** $b = 25$, $r = 10$; [†]single core.

In Table 8, we show the execution time together with important characteristics of the analyzed circuits (for randomly selected faults). The results clearly confirm the complexity considerations in Sect. 4.2, i.e., the execution time depends strongly (exponentially) on the number of input bits. We are able to analyze a design with 32 inputs, 16 outputs, and 104 gates ($4\times$ PRESENT S-box & KeyAdd) in 45 min with a single fault and in about 4 hours with four faults. A design with the same input and output sizes but more than twice the number of combinatorial gates ($2\times$ AES S-box & KeyAdd) takes about twice as long. At the same time, the size of the circuit does not have such a significant impact on the execution time, and a design with over 2000 gates can be evaluated in less than 6 min with one fault and in less than 1 h with four faults, even on a single core (for 16 cores, we ran out of memory because the BDD library used requires a copy of the BDD for each core). The parallel implementations of S-box & KeyAdd could also be analyzed separately due to their independence with respect to the leakage (cf. Sect. 5.1), but we show them here to give a sense of scalability.

8 Related and Future Work

In the following section, we will discuss related and future work of the presented methodology.

8.1 Related Work

Quantitative Information Flow. Early methods for QIF relied on *Shannon entropy* along with *mutual information* to measure the flow of information [16, 17]. Intuitively, the Shannon entropy is a measure of uncertainty because it provides the minimum number of bits required to encode a given piece of information. However, it has been shown that the security guarantees derived from Shannon entropy do not generally provide meaningful results [53]. To better capture the information leakage, Smith proposed the *min-entropy* as a measure of uncertainty [53], which was later refined and extended (see Alvim et al. [3] for an overview). We, therefore, rely on the min-entropy as presented in Sect. 4.

QIF for Hardware. The first attempt at quantitative analysis in the context of hardware was done by Mao et al. [39], who evaluated the leakage caused by the timing behavior (in the number of cycles) of an algorithmic hardware implementation in terms of mutual information. An extension to general information flow with automatic integration into a Hardware Description Language (HDL) was proposed by Guo et al. with QIF-Verilog [30]. They define operation-specific rules for leakage propagation from inputs to outputs of logical operations. These rules are inspired by Smith’s vulnerability metric V , but not an exact computation. Later, Reimann et al. extended this approach with QFlow [41] by computing the vulnerability V and the leakage L of subparts (groups of operations) and combining them via a Markov chain. However, this requires the assumption of

independence for all inputs to a component of the Markov chain, which is usually not given. In contrast, we analyze the entire system at once, leading to an exact computation with weaknesses in scalability. It remains an open question how the approaches for HDLs can be tailored to the computation of V in the context of FIA.

QIF for Fault Injection Analysis. Information-theoretic metrics and QIF have also been used in the context of FIA. Sakiyama et al. [48] used Shannon entropy to determine the information leakage in DFA given a design as shown in Fig. 7. They used a handcrafted analysis to analyze and improve existing attacks against AES. Later, Liu et al. [38] extended this approach to general Substitution-Permutation Networks (SPNs) with potential FIA countermeasures. However, they assume the independence of internal signals for computational efficiency. In another line of research, Patranabis et al. [40] analyze the security of a specific infection scheme in software using the mutual information between an output differential $y \oplus y'$ and the key s . An infection scheme tries to make a faulty output y' useless for \mathcal{A}_f by randomizing the effect of the fault. Hence, security is achieved when the mutual information is equal to zero. A more general methodology has been proposed by Feng et al. [26] for build-out infections, i.e., infections applied after an unmodified cipher implementation. In particular, they consider an attacker who obtains an output differential $y \oplus y'$ after a single fault injection. Using the structure of build-out infection schemes, the authors decompose the security analysis into the contribution of the unprotected cipher and the contribution of the infection scheme. In addition, common infection schemes are a composition of simple randomness-based operations that can be analyzed individually. While the methodology covers a wide range of infection schemes, the preparation must be tailored to the individual cipher and infection scheme and requires considerable expertise. In contrast to these works, we use QIF based on the min-entropy to develop a general metric for FIA not tailored to specific attacks and circuit structures. In addition, we introduce the notion of probabilistic faults into the fault model, allowing the analysis of more realistic fault scenarios.

Fault Analysis Tools. In recent years, the research community has focused on the development of automatic tools for fault susceptibility. The first set of tools focuses on the construction of potential key distinguishers for DFA, i.e., differentials between correct and incorrect intermediate states, and the evaluation of the associated attack complexity. For this purpose, XFC [34] uses classical IFA in a high-level cipher description. For the same purpose, ExpFault [47] uses system simulation in combination with data mining techniques. Of course, these methods are limited to DFA. The second set of tools takes a more general approach by analyzing the impact of faults on the output, i.e., distinguishing between detected, ineffective, and effective faults. To do this, VerFi [4] uses traditional simulation techniques to evaluate the system behavior, while FIVER [43] uses symbolic analysis based on BDDs. As an extension to FIVER, VERICA [42] com-

compares output distributions to verify security against SIFA. Except for the SIFA extension, these tools assume that all faulty outputs are equally dangerous, and are therefore unable to make a qualitative comparison between two different fault scenarios. However, since these tools are much more efficient than the proposed FIA vulnerability calculation, they can be used as a preliminary analysis step to extract interesting fault scenarios. As a third set of tools, recent works compare ciphertext distributions with a t-test for different fault locations with fixed secrets or different secrets with fixed fault locations [45, 46]. The ciphertexts for the analysis are generated by non-exhaustive simulation and a design is classified as vulnerable to FIA when the two ciphertext distributions can be distinguished according to the t-test. Again, the result does not allow a quantitative analysis, as the t-test provides a score for the confidence of the classification and not for the difference of the distributions.

8.2 Future Work

The first and most obvious shortcoming of our proposal is scalability. While we have shown that valuable insights can be gained from essential cryptographic components of small or medium size, more efficient methods of evaluation are needed for larger structures. Thus, we need some notion of composability to reduce the complexity of a single analysis and still be able to conclude complex structures. One approach in this direction is the construction of checkpoints (as proposed by Shahmirzadi et al. [51]) that isolate two parts of a design with respect to faults. Of course, this approach imposes additional overhead on the design, and more efficient methods are desirable.

In this work, we consider an adversary who has access to exactly one pair of correct and faulty outputs (y, y') . While this seems to be a good approximation for the general vulnerability of a design to FIA, since more advanced attacks rely on the existence of leakage in this simple case, a criterion for multiple output pairs may provide more fine-grained insights. A trivial way to achieve this is to combine different FIA channels into one large channel that produces a set of output pairs (y_i, y'_i) . However, since the complexity is already high for a single output pair, it is prohibitive for additional pairs and more efficient approaches are needed. In addition, we only consider state-less channels, which limits the type of circuits and countermeasures that can be analyzed. Thus, extending the concept to state-full channels is a valuable generalization.

Our fault model is based on a distribution over a set of faults $\mathbb{D}\mathcal{F}$, without providing any instructions on how to come up with this set and distribution of faults. Indeed, this is a complex and challenging problem in itself. We envision a tool that analyzes a given circuit structure with respect to a given fault scenario and returns a set of likely fault combinations and an estimate of their distribution. Combined with our approach, this would provide a powerful tool for real-world evaluation of security in the context of FIA.

9 Conclusion

In this work, we have shown how methods from QIF can be used to establish a quantitative metric for the security of a circuit against FIA. This metric allows for fine-grained analysis of existing and new defense mechanisms, thus enabling a trade-off between performance and security. Although computationally expensive, the proposed method can provide new insights and enhance the understanding of FIA and related countermeasures. For example, we were able to find incorrect assumptions in the context of SIFA security and enable the identification of security-enhancing faults.

Acknowledgments. We would like to thank Jan Richter-Brockmann and Pascal Sasdrich for fruitful discussions on fault security and support with the tool VERICA. The work described was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972, by the German Federal Ministry of Education and Research BMBF through the projects VE-HEP (16KIS1345) and 6GEM (16KISK038) and by the European Commission under the project CONVOLVE (101070374). This research was also supported in part by NSF Award 2219810.

A Probabilistic Computation

A.1 Methodology

The isolated computation of $V[S | Y, Y' = y']$ for each $y' \in \mathcal{Y}$ allows not only the parallel computation of the vulnerability but also a probabilistic computation. Here, instead of computing the exact value of $V[S | Y, Y']$, we can approximate it using a subset of \mathcal{Y} . Specifically, we use the *Monte-Carlo method* [32], where the mean of a set of samples is used to estimate the mean of a probability distribution. This is a good approximation if the sample set is large enough and each sample is chosen independently. The quality of the approximation is given by the Confidence Interval (CI), which provides a range in which the true distribution mean lies with a certain probability (given by the confidence level).

To compute the FIA vulnerability probabilistically, we randomly select N faulty output values $y' \in \mathcal{Y}$ and compute $V[S | Y, Y' = y']$ (as done in Algorithm 1). We then estimate the overall vulnerability by scaling the mean of the samples by the number of existing faulty outputs y' :

$$V[S | Y, Y'] \approx |\mathcal{Y}| \frac{\sum_{i=0}^N V[S | Y, Y' = y'_i]}{N} \quad (2)$$

Then the CI can be calculated using the Central Limit Theorem as $(\mu - z \frac{\sigma}{\sqrt{N}}, \mu + z \frac{\sigma}{\sqrt{N}})$, where μ is the sample mean, σ is the sample standard deviation, and z is the z-score of the confidence level. The z-score of common confidence levels is 1.64 for a 90% confidence level, 1.96 for a 95% confidence level, and 2.57 for a 99% confidence level. For efficient computation, an iterative formula for mean

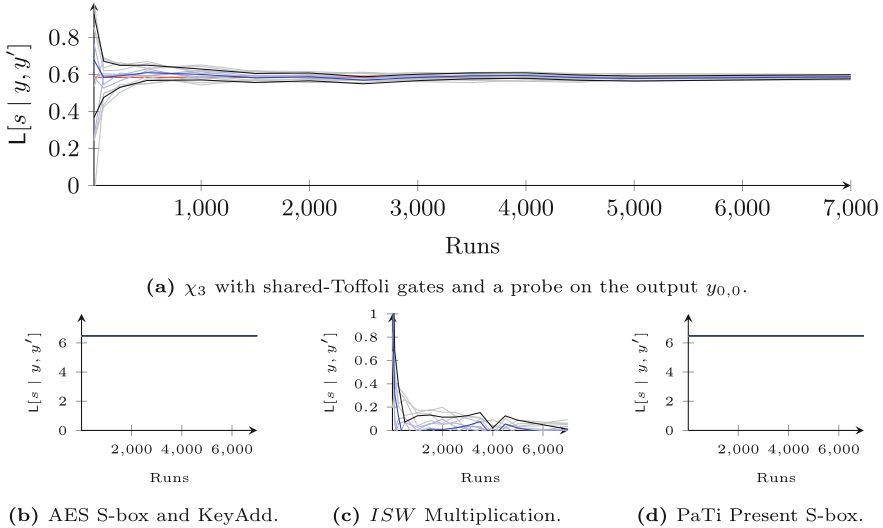


Fig. 13. Leakage over the number of runs for probabilistic computation with a set fault injected to the first input (i.e., x_0 or $x_{0,0}$) Leakage is given in blue while upper and lower bounds of the confidence range (95%) are given in black. Lighter colors represent different executions. The precise leakage is marked in red. (Color figure online)

and variance can be used. The CI is defined for the mean vulnerability and therefore must be scaled up for the overall vulnerability, similar to Eq. 2. This results in a CI that grows with the number of possible output values.

A.2 Evaluation

In Fig. 13, we show the convergence of the estimated leakage to the real leakage over the number of executions for four different circuits. While the estimation improves with an increasing number of executions, several thousand executions are required to obtain a high-confidence result. Thus, this only becomes interesting for circuits with a high number of output bits, and for most of the designs we analyzed, the exact computation is faster than running the probabilistic algorithm so often. Interestingly, however, there are some cases where the probabilistic algorithm yields the exact leakage after only one iteration (cf. Fig. 13b and 13d). This is the case when the vulnerability is the same for all possible output values, i.e., the expression $\max_s (\sum_x \Pr[s] \Pr[x] \Pr[y | x, s] \sum_f \Pr[f] \Pr[y' | x, s])$ is the same for all (y, y') . Further investigation is required to determine the set of circuits for which this holds. If this can be easily determined, the computation can be accelerated significantly, e.g., running 10 iterations for the $4 \times$ PRESENT S-box & KeyAdd takes only 3.87 s instead of about 3.5 h to get the exact leakage for four faults.

References

1. Agoyan, M., Dutertre, J.-M., Naccache, D., Robisson, B., Tria, A.: When clocks fail: on critical paths and clock faults. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 182–193. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12510-2_13
2. Akers, S.B.: Binary decision diagrams. *IEEE Trans. Computers* **27**(6), 509–516 (1978)
3. Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: *The Science of Quantitative Information Flow. Information Security and Cryptography*. Springer, Cham (2020). <https://doi.org/10.1007/978-3-319-96131-6>
4. Arribas, V., Wegener, F., Moradi, A., Nikova, S.: Cryptographic fault diagnosis using VerFI. In: HOST 2020, pp. 229–240. IEEE (2020)
5. Baksi, A., et al.: DEFAULT: cipher level resistance against differential fault attack. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 124–156. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_5
6. Beckers, A., et al.: Design considerations for EM pulse fault injection. In: Belaïd, S., Güneysu, T. (eds.) CARDIS 2019. LNCS, vol. 11833, pp. 176–192. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-42068-0_11
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 313–314. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_19
8. Biham, E., Granboulan, L., Nguyen, P.Q.: Impossible fault analysis of RC4 and differential fault analysis of RC4. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 359–367. Springer, Heidelberg (2005). https://doi.org/10.1007/11502760_24
9. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997). <https://doi.org/10.1007/BFb0052259>
10. Blömer, J., Krummel, V.: Fault based collision attacks on AES. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 106–120. Springer, Heidelberg (2006). https://doi.org/10.1007/11889700_11
11. Bogdanov, A., et al.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhe, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_31
12. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_4
13. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers* **35**(8), 677–691 (1986)
14. Cassiers, G., Grégoire, B., Levi, I., Standaert, F.: Hardware private circuits: from trivial composition to full verification. *IEEE Trans. Computers* **70**(10), 1677–1690 (2021)
15. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26

16. Clark, D., Hunt, S., Malacaria, P.: Quantitative analysis of the leakage of confidential data. In: Workshop on Quantitative Aspects of Programming Languages, QAPL 2001, Satellite Event of PLI 2001, Firenze, Italy, 7 September 2001, pp. 238–251 (2001)
17. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. *J. Comput. Secur.* **15**(3), 321–371 (2007)
18. Cnudde, T.D., Nikova, S.: More efficient private circuits II through threshold implementations. In: FDTC 2016, pp. 114–124. IEEE Computer Society (2016)
19. Courtois, N., Jackson, K., Ware, D.: Fault-algebraic attacks on inner rounds of DES. In: E-Smart 2010 Proceedings: The Future of Digital Security Technologies. Strategies Telecom and Multimedia (2010)
20. Daemen, J., Dobraunig, C., Eichlseder, M., Groß, H., Mendel, F., Primas, R.: Protecting against statistical ineffective fault attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**(3), 508–543 (2020)
21. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xooff. *IACR Trans. Symmetric Cryptol.* **2018**(4), 1–38 (2018)
22. Dhooghe, S., Nikova, S.: My gadget just cares for me - how NINA can prove security against combined attacks. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 35–55. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40186-3_3
23. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(3), 547–572 (2018)
24. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(3), 89–120 (2018)
25. Feldtkeller, J., Richter-Brockmann, J., Sasdrich, P., Güneysu, T.: CINI MINIS: domain isolation for fault and combined security, pp. 1023–1036. ACM CCS (2022)
26. Feng, J., Chen, H., Li, Y., Jiao, Z., Xi, W.: A framework for evaluation and analysis on infection countermeasures against fault attacks. *IEEE Trans. Inf. Forensics Secur.* **15**, 391–406 (2020)
27. Gandolfi, K., Moutrel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
28. Gnad, D.R.E., Oboril, F., Tahoori, M.B.: Voltage drop-based fault attacks on FPGAs using valid bitstreams. In: 27th International Conference on Field Programmable Logic and Applications, FPL 2017, Ghent, Belgium, 4–8 September 2017, pp. 1–7 (2017)
29. Gruss, D., Maurice, C., Mangard, S.: Rowhammer.js: a remote software-induced fault attack in JavaScript. In: Caballero, J., Zurutuza, U., Rodríguez, R.J. (eds.) DIMVA 2016. LNCS, vol. 9721, pp. 300–321. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40667-1_15
30. Guo, X., Dutta, R.G., He, J., Tehranipoor, M.M., Jin, Y.: QIF-Verilog: quantitative information-flow based hardware description languages for pre-silicon security assessment. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, 5–10 May 2019, pp. 91–100 (2019)
31. Hadžić, V., Primas, R., Bloem, R.: Proving SIFA protection of masked redundant circuits. In: Hou, Z., Ganesh, V. (eds.) ATVA 2021. LNCS, vol. 12971, pp. 249–265. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88885-5_17
32. Hutchinson, M.: A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Commun. Stat. Simul. Comput.* **19**(2), 433–450 (1990)

33. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
34. Khanna, P., Rebeiro, C., Hazra, A.: XFC: a framework for eXploitable fault characterization in block ciphers. In: Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, 18–22 June 2017, pp. 8:1–8:6 (2017)
35. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
36. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
37. Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta, K.: Fault sensitivity analysis. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 320–334. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_22
38. Liu, Q., Ning, B., Deng, P.: Information theory-based quantitative evaluation method for countermeasures against fault injection attacks. *IEEE Access* **7**, 141920–141928 (2019)
39. Mao, B., Hu, W., Althoff, A., Matai, J., Oberg, J., Mu, D., Sherwood, T., Kastner, R.: Quantifying timing-based information flow in cryptographic hardware. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015, Austin, TX, USA, 2–6 November 2015, pp. 552–559 (2015)
40. Patranabis, S., Chakraborty, A., Mukhopadhyay, D.: Fault Tolerant Infective Countermeasure for AES. *J. Hardw. Syst. Secur.* **1**(1), 3–17 (2017)
41. Reimann, L.M., Hanel, L., Sisejkovic, D., Merchant, F., Leupers, R.: QFlow: quantitative information flow for security-aware hardware design in verilog. In: 39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, 24–27 October 2021, pp. 603–607 (2021)
42. Richter-Brockmann, J., Feldtkeller, J., Sasdrich, P., Güneysu, T.: VERICA - verification of combined attacks: automated formal verification of security against simultaneous information leakage and tampering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**(4), 255–284 (2022)
43. Richter-Brockmann, J., Rezaei Shahmirzadi, A., Sasdrich, P., Moradi, A., Güneysu, T.: FIVER - robust verification of countermeasures against fault injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**(4), 447–473 (2021)
44. Richter-Brockmann, J., Sasdrich, P., Güneysu, T.: Revisiting fault adversary models - hardware faults in theory and practice. *IEEE Trans. Computers* **72**, 1–14 (2022)
45. Saha, S., Alam, M., Bag, A., Mukhopadhyay, D., Dasgupta, P.: Learn from your faults: leakage assessment in fault attacks using deep learning. *J. Cryptol.* **36**(3), 19 (2023)
46. Saha, S., Kumar, S.N., Patranabis, S., Mukhopadhyay, D., Dasgupta, P.: ALAFA: automatic leakage assessment for fault attack countermeasures. In: Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, 02–06 June 2019, p. 136 (2019)
47. Saha, S., Mukhopadhyay, D., Dasgupta, P.: ExpFault: an automated framework for exploitable fault characterization in block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**(2), 242–276 (2018)

48. Sakiyama, K., Li, Y., Iwamoto, M., Ohta, K.: Information-theoretic approach to optimal differential fault analysis. *IEEE Trans. Inf. Forensics Secur.* **7**(1), 109–120 (2012)
49. Schneider, T., Moradi, A., Güneysu, T.: ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Robshaw, M., Katz, J. (eds.) *CRYPTO 2016*. LNCS, vol. 9815, pp. 302–332. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_11
50. Selmane, N., Guilley, S., Danger, J.: Practical setup time violation attacks on AES. In: *EDCC-7 2008*, pp. 91–96. IEEE Computer Society (2008)
51. Shahmirzadi, A.R., Rasoolzadeh, S., Moradi, A.: Impeccable circuits II. In: *DAC 2020*, pp. 1–6. IEEE (2020)
52. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) *CHES 2002*. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_2
53. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) *FoSSaCS 2009*. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00596-1_21
54. Sullivan, G.A., Sippe, J., Heninger, N., Wustrow, E.: Open to a fault: on the passive compromise of TLS keys via transient errors. In: *31st USENIX Security Symposium, USENIX Security 2022*, Boston, MA, USA, 10–12 August 2022, pp. 233–250 (2022)
55. Tang, A., Sethumadhavan, S., Stolfo, S.J.: CLKSCREW: exposing the perils of security-oblivious energy management. In: *26th USENIX Security Symposium, USENIX Security 2017*, Vancouver, BC, Canada, 16–18 August 2017, pp. 1057–1074 (2017)