# Revisiting Fault Adversary Models
## Hardware Faults in Theory and Practice

Jan Richter-Brockmann[1], Pascal Sasdrich[1], and Tim Güneysu[1,2]

[1] Ruhr-Universität Bochum, Horst-Görtz Institute for IT-Security, Germany
[2] DFKI, Germany
`firstname.lastname@rub.de`

**Abstract.** Physical attacks are serious threats to hardware implementations of any strong cryptographic primitive. Particularly, fault injection attack is considered as a powerful technique to successfully attack embedded cryptographic implementations since various fault injection mechanisms from simple clock glitches to more advanced techniques like laser fault injection can lead to devastating attacks, even with just a single successfully injected fault. Given these critical attack vectors, researchers in academia and industry came up with a long list of dedicated countermeasures to thwart such attacks.
However, the validation of proposed countermeasures is mostly performed on custom adversary models that are often not tightly coupled with the actual physical behavior of available fault injection mechanisms and techniques and, hence, fail to model the reality accurately. Furthermore, using custom models complicates comparison between different designs and evaluation results. As a consequence, we aim to close this gap by proposing a simple, generic, and consolidated fault injection adversary model in this work that can be perfectly tailored to existing fault injection mechanisms and their physical behavior in hardware. To demonstrate the advantages of our adversary model, we apply it to a cryptographic primitive (i.e., an ASCON S-box) and evaluate it based on different attack vectors. We further show that our proposed adversary model can be used and integrated into the state-of-the-art fault verification tool VerFI. Finally, we provide a discussion on the benefits and differences of our approach compared to already existing evaluation methods and briefly discuss limitations of current available verification tools.

**Keywords:** FIA · Fault Modeling · Adversary Model · LFI · EMFI · Clock Glitch · Voltage Glitch.

## 1 Introduction

Although designing and constructing secure cryptographic primitives, such as block ciphers, is a well-understood and matured problem [20], secure implementation of cryptographic primitives in the presence of physical adversaries is still an open challenge, even after two decades of academic and industrial research. In particular, rather than exploiting flaws in cryptographic primitives or schemes, physical adversaries commonly address and exploit vulnerabilities in physical instances of the cryptographic algorithms and functions.

Among all physical and implementation attacks, Side-Channel Analysis (SCA) and Fault Injection Analysis (FIA) have shown a large potential to be mounted successfully on various implementations of cryptographically strong primitives and functions. In particular FIA, classified as a set of active attacks, has gained increasing attention during recent years due to powerful advances in more cost-efficient equipment and more experienced adversaries. In the wake of this progress, a plethora of different attack vectors has been proposed, e.g., clock or voltage glitches [2,40], electromagnetic pulses [11,13,24,25], or focused photon injection using laser beams [10,30,34,38]. Naturally, different approaches to increase protection against FIA have been proposed at similar pace, mainly following the concepts of redundancy and (concurrent) error detection [1,32], error correction [29,36], or recently introduced, infective computation [17].

However, checking and verifying that an implementation is successfully protected against FIA is a manual, downstream, test-driven, and error-prone process. Further, quality of analysis and verification results comprehensively depends on the quality and accuracy of underlying adversary models. If the adversary models fail to reflect the practical realities and capabilities of an adversary, countermeasures and protection mechanisms might be inappropriate, can fail to provide the desired level of security, and ultimately the physical implementation is still vulnerable to FIA.

Given these observations and challenges, security should be considered during the entire development and life cycle of the implementation. More precisely, continuous analysis, evaluation, and verification of the design, even before deployment, can assist the designer to choose and implement countermeasures correctly. In addition, accurate description and modeling of the capabilities and limitations of the physical adversary and the physical environment will ensure appropriate protection of the physical implementation after deployment.

Currently, a wide range and variety of custom adversary models is used for evaluation and verification of protection mechanisms and often, with new countermeasures, new adversary models are proposed at the same time. Unfortunately, most of the adversary models are hardly compatible and do not allow fair and meaningful comparisons between different approaches and implementations. Ideally, a standardized model that is simple, generic, but allows customization would help to analyze, verify, and compare different implementations and countermeasures. Ultimately, designers would be able to choose countermeasures and protection mechanisms appropriately, and easily evaluate and verify the security level for the targeted practical environment and circumstances with minimal effort using the standardized adversary model tweaked for the given realities.

***Contribution:*** In this work, we review existing approaches and methods to inject faults into cryptographic implementations in order to consolidate existing adversary models and extract an unified adversary model for standardized fault analysis and verification. In particular, we introduce a generic and abstract adversary model that can be parameterized and instantiated to model different adversaries with varying capabilities and limitations. More precisely, we show how the generic adversary model can be customized to reflect and model common fault injection approaches, including (but not limited to) *clock* or *voltage* glitches, *electromagnetic pulses*, and focused *laser beams*, and apply each model to a practical example emphasizing similarities and differences of the different adversary model instances.

Eventually, our consolidated and unified adversary model can be used to establish a standardized evaluation and verification metric for FIA countermeasures that allows fair comparison in adversary capabilities and limitations as well as vulnerabilities and fault coverage of different protection mechanisms. In particular, our proposed adversary model facilitates application for design and verification through a simple, adaptable, and intuitive design. We demonstrate these features by integrating our fault model into the fault verification tool VerFI [4] and providing a case study on the lightweight block cipher LED [18].

***Outline:*** While Section 2 summarizes and reviews common (practical) fault injection mechanisms in detail, including *clock glitches*, *underpowering and voltage glitches*, *electromagnetic pulses*, and *optical fault injections*, Section 3 is dedicated to conception and discussion of our consolidated and unified adversary model. Besides, Section 3 introduces our considered circuit model, states initial assumptions, and introduces the generic but parametric adversary model. In Section 4, we provide practical instantiations of our approach with respect to the fault injection mechanisms presented in Section 2. Further, we demonstrate the practical integration of our adversary model to the fault verification tool VerFI in Section 5. Before concluding our work in Section 7, we compare our approach to existing fault models and briefly discuss limitations in Section 6.

## 2 Background

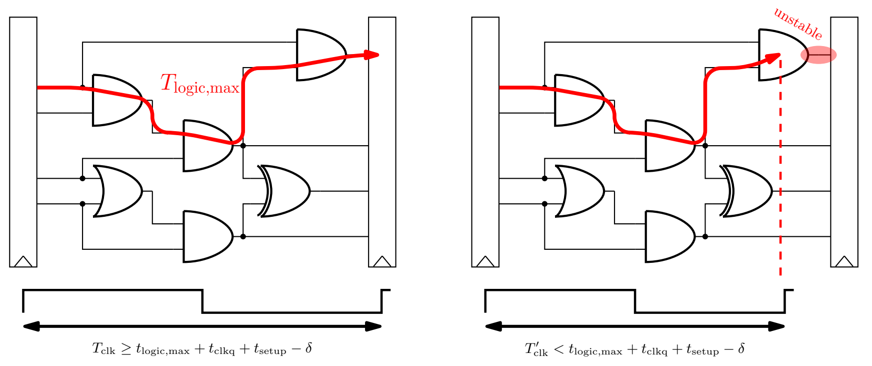
Before we start to summarize and review common fault injection mechanisms in detail, we introduce some notations which we will use throughout this work.

### 2.1 Notation

We denote functions by using sans-serif fonts. A multi-bit variable $\mathbf{x}$ is denoted in bold while single bit values and values from the vector $\mathbf{x}$ are indicated by $x_i$. Upper-case characters in calligraphic fonts are used to denote sets, e.g., $\mathcal{G}$.

### 2.2 Fault Injection Mechanisms

Over the last two decades, many different fault injection mechanisms were proposed and successfully established to attack hardware implementations of cryptographic algorithms. In this section, we summarize the most common techniques and explain the fundamental physical mechanisms of these attacks.


$T_{\text{logic,max}}$
unstable
$T_{\text{clk}} \geq t_{\text{logic,max}} + t_{\text{clkq}} + t_{\text{setup}} - \delta$
$T'_{\text{clk}} < t_{\text{logic,max}} + t_{\text{clkq}} + t_{\text{setup}} - \delta$

(a) Behavior under normal condition. (b) Behavior under clock glitch occurrence.

Fig. 1: Physical effects of clock glitches on operation of digital circuits.

**Clock Glitches.** Faulting digital circuits through generation and injection of *clock glitches* is considered as rather inexpensive technique for FIA. However, before we examine the physical fundamentals and mechanisms of intentional fault injection through clock glitches, we briefly review state-of-the-art literature with respect to FIA based on clock glitch generation.

*State of the Art.* In an early work on the general principles of fault injection via clock glitches, Agoyan et *al.* [2] demonstrated its effectiveness using the example of an Advanced Encryption Standard (AES) hardware implementation. Soon thereafter, Endo et *al.* [16] presented an on-chip clock glitch generator composed of Delay Locked Loops (DLLs) to test and validate the effectiveness of newly developed countermeasures addressing the threat of clock glitch insertion. In 2014, Korak et *al.* [21] increased the success rates of clock glitches in combination with heating of the device under attack. Although it was assumed that internal application of Phase-Locked Loops (PLLs) can easily defeat the threat of fault injection through clock glitches, Selmke et *al.* [35] recently presented successful fault injections using clock glitches on a microcontroller internally equipped with a PLL. Note, however, that this attack is still limited and only possible if an ongoing computation is not interrupted by the `LOCKED` signal of the PLL, as also noted by the authors.

*Physical Mechanism.* At a first glance, clock glitches may have limited relevance in real world scenarios when compared to other fault injection mechanisms covered later in this section. However, since clock glitch generators can be instantiated fairly easy in many common Field-Programmable Gate Array (FPGA) devices, allowing to create cost-efficient test setups and environments for countermeasure validation even for inexperienced designers, we opt to cover this mechanism and its physical mechanics in more detail in the following paragraph.

For this, Figure 1 schematically depicts the physical effects of clock glitches on the behavior and operation of digital circuits. Under normal operation conditions (Figure 1a), all signals can propagate through the combinational logic and settle to a stable state before the rising edge of the clock signal triggers the sampling process of the subsequent register. As a consequence, the (maximum) clock period $T_{\text{clk}}$ of a digital circuit is usually determined under the following conditions and assumptions:

$$T_{\text{clk}} + \delta \geq t_{\text{logic,max}} + t_{\text{clkq}} + t_{\text{setup}} \tag{1}$$

Here, $\delta$ denotes the clock skew, $t_{\text{logic,max}}$ the maximum propagation delay of the combinational logic, $t_{\text{clkq}}$ the delay of the register, and $t_{\text{setup}}$ the setup time for the input of the register.

Given that an adversary now can generate a clock glitch for an effective fault injection, the clock period $T'_{\text{clk}}$ is instantaneously decreased such that the inequality in Equation 1 is violated (but will hold again afterwards). Hence, for some primary input combinations the clock period might be to short to allow full propagation of the signals through the entire combinational logic and stabilization of the correct result at the input of the register. Figure 1b visualizes this behavior, eventually leading to the fact that the output of the considered gate is still independent of the current primary inputs and might lead to a faulty value sampled by the register at the arrival of the rising edge of the clock glitch.

**Underpowering and Voltage Glitches.** Similar to fault injection through clock glitches, *underpowering* and *voltage glitches* are also considered as rather inexpensive but effective methods for FIA. While underpowering considers the scenario of lowering the supply voltage of the target device throughout the entire computation process, voltage glitches only lower the supply voltage for a very limited period of time during the execution. Again, we briefly summarize state-of-the-art literature, before we discuss the physical fundamentals and mechanics of fault injection through underpowering and voltage glitching.

*State of the Art.* The first successful fault injection using the mechanisms of *underpowering* was presented in 2008 by Selmane et *al.* [33]. Using a 130 nm Application-Specific Integrated Circuit (ASIC) embedding an AES engine on a smart card target device, the authors report a successful recovery of the secret key processed inside the AES encryption engine. Their evaluations further demonstrate the dependency between voltage level and success rate of fault injection through underpowering. However, since underpowering naturally effects the entire execution of a cryptographic algorithm, precisely injecting faults, e.g., in a specific iteration of the algorithm, is very difficult and hardly achievable. As a consequence, Zussa et *al.* [40] focused their investigations on the fault injection mechanism of temporary *voltage glitches* to disturb the execution of cryptographic algorithms. More precisely, the authors prove that the physical mechanisms of voltage glitches and underpowering can be traced back to timing violations, as explained in the following paragraph.

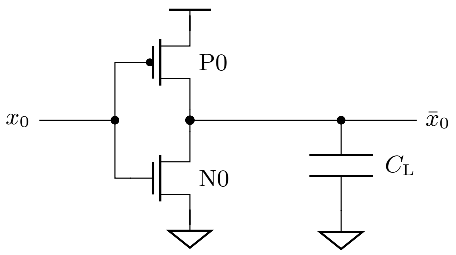


Fig. 2: Transistor-level schematic of a CMOS inverter.

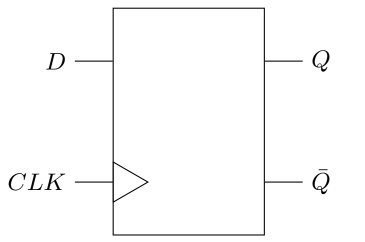
*Physical Mechanism.* Considering the example of simple Complementary Metal-Oxide-Semiconductor (CMOS) inverter at transistor level, as given in Figure 2, we briefly summarize the findings of [40] with respect to timing violations caused through voltage glitches (and underpowering). Assuming that each CMOS gate introduces some propagation delay upon signal switching, the propagation delay in case of a simple CMOS inverter can be explained through the switching process in the transistors. Exemplarily, we assume a switching activity from *low* to *high* at the output of the P-type Metal-Oxide Semiconductor (PMOS) transistor $P0$ in Figure 2. In this case, the propagation delay $t_{\mathrm{pLH}}$, as derived in [28], is given by the following equation:

$$t_{\mathrm{pLH}} = \frac{C_{\mathrm{L}}\left[\frac{2|V_{\mathrm{th,p}}|}{V_{\mathrm{DD}}-|V_{\mathrm{th,p}}|} + \ln\left(3 - 4\frac{|V_{\mathrm{th,p}}|}{V_{\mathrm{DD}}}\right)\right]}{\mu_{\mathrm{p}} C_{\mathrm{ox}} \frac{W_{\mathrm{p}}}{L_{\mathrm{p}}} \left(V_{\mathrm{DD}} - |V_{\mathrm{th,p}}|\right)}. \quad (2)$$

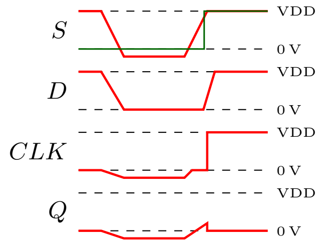
Here, $C_{\mathrm{L}}$ models the load of connected gates, $V_{\mathrm{th,p}}$ the threshold voltage of the transistor, $\mu_{\mathrm{p}}$ the mobility of the holes, $C_{\mathrm{OX}}$ the capacity of the gate oxide, and $W_{\mathrm{p}}/L_{\mathrm{p}}$ the ratio of the transistor dimensions.

Obviously, under a lower supply voltage $V_{\mathrm{DD}}$, the propagation delay of the inverter $t_{\mathrm{pLH}}$ increases. Further, similar equations can be derived for the N-type Metal-Oxide Semiconductor (NMOS) transistor and even for more complex gates than a simple inverter, resulting in the same effect and impact. Eventually, as the variation of the supply voltage affects all transistors and gates between two register stages, lowering the supply voltage through voltage glitches or underpowering will increase the maximum propagation delay of the combinational logic. As a consequence, the inequality in Equation 1 might be violated. Hence, as for clock glitches, the final result might not be stable at the input of the register resulting in the sampling of a faulty value.

**Electromagnetic Pulses.** Another approach for fault injection into embedded devices, having higher precision than clock or voltage glitches but still at reasonable equipment and expertise requirement [7], uses *electromagnetic pulses*. Again, we briefly review and summarize related state-of-the-art literature and discuss the physical mechanisms responsible for the manifestation of faults in digital circuits.



(a) D Flip-Flop.



(b) Simplified diagram of an EMFI.

Fig. 3: Physical effects due to faults caused by EMPs (adapted from [13, 14]).

*State of the Art.* Over the last years, the understanding of the underlying mechanism of faults caused by Electromagnetic Pulse (EMP) changed. While in 2012, Dehbaoui et *al.* [11] performed some experiments on microcontrollers and FPGAs leading to the conclusion that Electromagnetic Fault Injections (EMFIs) can be explained by timing violations, two years later, Ordas et *al.* [26] demonstrated that timing faults cannot capture and describe the complete behavior of EMFI. In the following years, they performed further experiments and eventually deduced a *sampling fault model* [24, 25]. Most recently, Dumont et *al.* [13, 14] were eventually able to explain the physical behavior for the sampling fault model and confirmed its correctness by conducting several theoretical simulations and additional practical experiments. For this, we will summarize the latest findings and explain the underlying physical mechanism responsible for fault injections caused by EMP in the following paragraph.

*Physical Mechanisms.* Any general EMFI setup usually consists of a ferrite core, a coil, and a voltage pulse generator to generate a magnetic field which can be used to induce a current in any wire loop based on the theory developed by M. Faraday. Particularly in Integrated Circuits (ICs), those wire loops can be found in the power and ground networks. Hence, the induced current leads to a voltage swing $S$ between the power and the ground grid (cf. Figure 3b for the effects of an undershoot).

However, in the following, we limit our explanations on D Flip-Flops (DFFs) (see Figure 3a), as they are the main elements in digital ICs susceptible to EMFI. The aforementioned voltage drop caused by an EMP consequently pulls the potential of the clock signal and the input signal $D$ down, as visualized in a simplified diagram in Figure 3b. More precisely, this behavior is caused by the falling edge of the swing $S$ and can therefore be associated with the first EMP generated by the probe. With the rising edge of $S$ (and the second EMP) the circuit starts to recover the original state.

Here, the authors of [13] describe the recovering phase as a race between the clock signal and the input signal $D$. A successful fault injection is performed only if the clock signal wins the race, meaning that the clock recovers faster than the input signal $D$, and therefore the DFF stores a faulted value (cf. Figure 3b).

Note, however, that not only a negative swing can be induced, but also a positive swing, then leading to an overshoot instead of undershoot. While the negative polarity often leads to bit-resets, the positive overshoot induces bit-set faults with higher probability. For more details, we refer the interested reader to the original work in [13, 14].

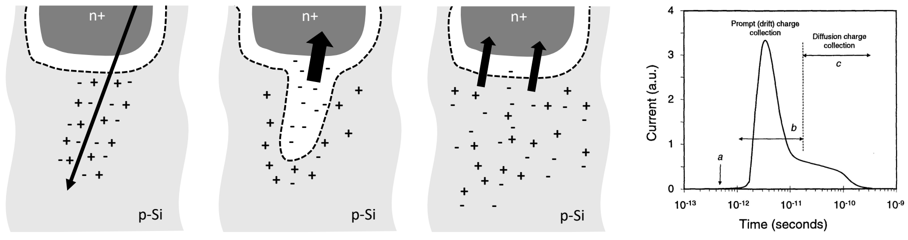
In summary, Dumont et *al.* showed that EMFI causes sampling faults which can also be modeled as set or reset faults in memory elements such as DFFs. Additionally, there most recent work in [13] also demonstrates a very fine locality of EMFI, surprisingly mostly independent of the electromagnetic probe geometry.

**Optical Fault Injections.** Optical fault injection based on focused laser beams was initially presented in 2002, in the seminal work of Skorobogatov and Anderson [38]. Since then, many follow-up works have been presented and improved the potential of laser-assisted fault injection. Before we summarize and explain the physical effects of laser-induced faults, we dedicate the next paragraph to the current progress and state-of-the-art research with respect to optical fault injection methods.

*State of the Art.* The first case study of laser fault injections, presented in the seminal work [38] of Skorobogatov and Anderson, was designed for a target platform built in a quite large 1 200 nm technology. However, in the following years, several other works studied the influence of laser beams to the operation of ICs and improved the application for lasers as a fault injection mechanism. For instance, in 2013, Roscian et *al.* [30] already targeted a 250 nm technology and performed investigations on the underlying fault model. Already in the following year, Courbon et *al.* [10] could demonstrate the tremendous accuracy of laser fault injection and used it to characterize registers instantiated in a 90 nm technology. Similarly, Selmke et *al.* [34] investigated the accuracy of laser-induced faults for a 45 nm technology, but concluded that precise fault injections into memory cells, by using laser beams, becomes more difficult for smaller technologies.

However, not only memory cells, but also any combinational gate of a digital IC is susceptible to laser-induced faults, as was shown in 2016 by Schellenberg et *al.* [31]. In this work, the authors used successful injections of faults to perform a *fault sensitivity analysis*, also possible for smaller target technologies. Most recently, Dutertre et *al.* [15] successfully performed fault injections on an AES implemented on a very small 28 nm technology. However, although the hardness of laser fault injection varies with the targeted technology node and geometry size, the basic fundamentals and physical effects can be traced back to the same phenomena.

*Physical Mechanisms.* Figure 4 exemplarily shows the fundamental physical effect when a focused laser beam hits and affects an NMOS transistor. More precisely, the laser beam starts an ionizing process in a PN-junction while along the



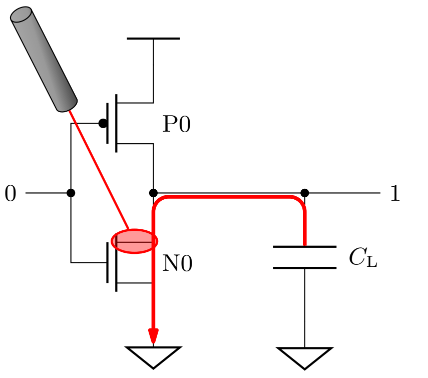
(a) Ionizing. (b) $I_{\mathrm{drift}}$. (c) $I_{\mathrm{diff}}$. (d) Transient current.

Fig. 4: Physical effect of a laser fault injection as introduced in [6].

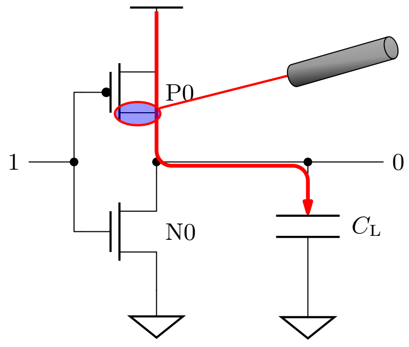
laser injection path, a dense distribution of electron-hole pairs is produced (cf. Figure 4a). Afterwards, the carriers are rapidly collected by the electric field and the charge is compensated, resulting in a permanently reduced voltage on that node while eventually, a temporary drift current arises as visualized in Figure 4b. However, shortly afterwards (usually at a magnitude of a few picoseconds) the funnel collapses and a small diffusion current dominates the collection process, which again is shown in Figure 4c. For clarification, the transient current flow over time is additionally plotted in Figure 4d (taken from [6, 39]).

As a consequence, the effect of producing a temporary drift current $I_{\mathrm{drift}}$ in a PN-junction of a transistor can be used to alter the state of a gate. For the sake of simplicity, we consider the CMOS inverter given in Figure 5 as a minimal example, where subsequent connected gates are simplified and modeled by a load capacity $C_{\mathrm{L}}$. As a first step, we assume the input of the inverter to be zero and output to be one, as visualized in Figure 5a. Once an adversary hits the drain region of the NMOS transistor with the help of a focused laser beam, the output state of the inverter may change. In particular, the high drift current through the transistor forces a discharge process of the output node, i.e., the electrical charge from $C_{\mathrm{L}}$ is moved such that the output changes from one to zero. However, this effect can only occur if the temporary drift current is larger than the current flowing through the PMOS transistor, which still conducts correctly. Hence, if the drift current $I_{\mathrm{drift}}$ collapses, the output node will eventually switch back to its former high level. This results in a temporary injected fault which is called Single Event Transient (SET) (or *transient* Single Event Upset (SEU) [27]). A similar effect occurs when the input of the inverter is one and the output is zero, but in this case the laser beam has to hit the drain region of the PMOS transistor (instead of the NMOS transistor) in order to switch the output node from zero to one, i.e., to load $C_{\mathrm{L}}$ [30].

In summary, we can state that this fault injection mechanism either causes bit-set or bit-reset faults considering the given inverter. Further, the bit-set or bit-reset faults can occur as temporary faults in both, combinational logic (i.e., logic gates) or in memory gates (e.g., registers). However, in case the attacker targets memory gates, the stored value will be altered, which is called a *static* SEU [27], as this transient fault cannot be recovered while transient faults in



(a) Sensitive drain region of an NMOS.



(b) Sensitive drain region of a PMOS.

Fig. 5: Effect of a laser fault injection exemplary shown with an inverter [30].

combinational gates may be recovered by sufficient long clock periods (in comparison to the duration of the fault).

**Miscellaneous Mechanisms** Besides the fault injection mechanisms introduced above, a few more techniques can be found in the literature, such as body biasing [22, 23], overpowering [8], temperature [19, 37], and X-Ray beams [3]. However, these techniques only have a minor or auxiliary role in practice in comparison to the mechanisms described above and therefore we decided to exclude them from following considerations in our work.

## 3 Concept

Given the broad range of physical fault injection mechanisms that we introduced in the previous section, our efforts in this section focus on a consolidated and unified model for fault injection adversaries, ideally covering all previously introduced concepts. For this, we start with formal definitions of fundamental concepts as well as initial assumptions and limitations that we consider for our adversary model. Then, based on those definitions and assumptions, we propose and describe our generalized fault injection adversary model in more detail.

### 3.1 Circuit Model

As this work focuses on fault injection techniques and adversaries for physical hardware and digital ICs, we first introduce our abstract model to describe the underlying circuit targeted by the adversary. For this, we assume that a digital circuit $\mathbf{C}$ is implemented to execute an arbitrary Boolean function $\mathsf{f} : \mathbb{F}_2^i \rightarrow \mathbb{F}_2^o$ with $i, o \geq 1$. Further, we can decompose the circuit $\mathbf{C}$ into atomic component blocks, called *gates*, which further can be distinguished with respect to purely *combinational gates*, as defined in Definition 1, and *sequential memory gates*, as given in Definition 2.

**Definition 1 (Combinational Gate).** *A* combinational gate $g_c$ *is a physical component in a digital logic circuit that evaluates its output as a pure (Boolean) function of the present inputs only (without any dependency on the history of inputs).*

In the context of this work, we further assume that the set of Boolean functions implemented by combinational gates is limited and given as $\mathcal{G}_c = \{\mathsf{not}, \mathsf{and}, \mathsf{nand}, \mathsf{or}, \mathsf{nor}, \mathsf{xor}, \mathsf{xnor}\}$. For the sake of simplicity, we further define more granular subsets to distinguish functions with fan-in of size 1 and 2, i.e., $\mathcal{G}_u = \{\mathsf{not}\}$ and $\mathcal{G}_b = \{\mathsf{and}, \mathsf{nand}, \mathsf{or}, \mathsf{nor}, \mathsf{xor}, \mathsf{xnor}\}$, such that $\mathcal{G}_c = \mathcal{G}_u \cup \mathcal{G}_b$.

**Definition 2 (Sequential Gate).** *A* sequential gate $g_r \in \mathcal{G}_s$ *is a physical, clock-synchronized, memory component in a digital logic circuit for which the output depends not only on the present inputs but also on the history of previous inputs.*

Hence, *sequential (memory) gates* are used to store intermediate results and to establish synchronization points in a digital circuit. In the context of this work, we model sequential gates as clock-dependent synchronization points that store a single Boolean variable $x \in \mathbb{F}_2$ with $\mathcal{G}_s = \{\mathsf{reg}\}$. Additionally, given the set of combinational gates $\mathcal{G}_c$ and the set of sequential gates $\mathcal{G}_s$, we define a set $\mathcal{G} = \mathcal{G}_c \cup \mathcal{G}_s$ to unite all valid gates of a digital circuit $\mathbf{C}$ in one set.

**Definition 3 (Circuit Representation).** *A digital circuit* $\mathbf{C}$ *is modeled by a Direct Acyclic Graph (DAG) formally described by* $\mathbf{D} = \{\mathcal{V}, \mathcal{E}\}$*, with* $\mathcal{V}$ *the set of vertices and* $\mathcal{E}$ *the set of edges. A single vertex* $v \in \mathcal{V}$ *represents a combinational or sequential gate and a single edge* $e \in \mathcal{E}$ *represents a wire carrying a digital signal, modeled as an element from the finite field* $\mathbb{F}_2$*.*

The formal definition of our considered circuit model is given in Definition 3. Note, however, that this work focuses on synchronous digital logic circuits only, hence, we always assume that inputs, outputs, and sequential gates are synchronized to a common clock signal.

## 3.2 Fault Model

For the description of fault events and fault propagation in digital logic circuits, we first introduce the two sets of *unary* and *binary* Boolean functions. More precisely, the set of *unary* functions is given as $\mathcal{U} = \{\, \mathsf{u} \mid \mathsf{u} : \mathbb{F}_2 \to \mathbb{F}_2 \,\}$ while the set of *binary* function is defined as $\mathcal{B} = \left\{\, \mathsf{b} \mid \mathsf{b} : \mathbb{F}_2^2 \to \mathbb{F}_2 \,\right\}$.

In general, for a given Boolean function $\mathsf{f} : \mathbb{F}_2^i \to \mathbb{F}_2^o$, we can construct $2^{o \times 2^i}$ distinct Boolean functions in $i$ variables and $o$ result values, i.e., we have $|\mathcal{U}| = 4$ distinct unary and $|\mathcal{B}| = 16$ distinct binary functions. Further, the specific assignments of all possible functions for $\mathcal{U}$ and $\mathcal{B}$ are presented in Table 1.

Table 1: Functions included in $\mathcal{U}$ and in $\mathcal{B}$.

| **Inputs** | | $\mathsf{u}_i(x_0) \in \mathcal{U}$ | | | | $\mathsf{b}_i(x_0, x_1) \in \mathcal{B}$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_0$ | $x_1$ | $\mathsf{u}_0$ | $\mathsf{u}_1$ | $\mathsf{u}_2$ | $\mathsf{u}_3$ | $\mathsf{b}_0$ | $\mathsf{b}_1$ | $\mathsf{b}_2$ | $\mathsf{b}_3$ | $\mathsf{b}_4$ | $\mathsf{b}_5$ | $\mathsf{b}_6$ | $\mathsf{b}_7$ | $\mathsf{b}_8$ | $\mathsf{b}_9$ | $\mathsf{b}_{10}$ | $\mathsf{b}_{11}$ | $\mathsf{b}_{12}$ | $\mathsf{b}_{13}$ | $\mathsf{b}_{14}$ | $\mathsf{b}_{15}$ |
| 0 | 0 | | – | | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | | – | | | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

As described before, we consider a limited set $\mathcal{G}_c$ of combinational gates for our circuit model. More specifically, we consider a single unary gate $\{\mathsf{not}\} \in \mathcal{G}_u$ that executes the unary Boolean function $u_0$ on its input. Further, the binary gates $\{\mathsf{and}, \mathsf{nand}, \mathsf{or}, \mathsf{nor}, \mathsf{xor}, \mathsf{xnor}\} \in \mathcal{G}_b$ execute the Boolean functions $\mathsf{b}_i$ with $0 \leq i \leq 5$ on their inputs respectively.

Then, given a DAG $\mathbf{D}$ modeling a digital circuit $\mathbf{C}$, we can associate each gate in the physical circuit, with a vertex $v \in \mathcal{V}$ of the graph, representing a combinational or sequential gate by a Boolean function from the sets $\mathcal{U}$ or $\mathcal{B}$. For this, we define the following golden mapping $\tau_{golden}$ from gate to vertex and associated Boolean function $\mathsf{f}_g$:

$$\tau_{golden}: \begin{array}{llll} \{\mathsf{not}\} \mapsto \{\mathsf{u}_0\} & \{\mathsf{and}\} \mapsto \{\mathsf{b}_0\} & \{\mathsf{or}\} \mapsto \{\mathsf{b}_2\} & \{\mathsf{xor}\} \mapsto \{\mathsf{b}_4\} \\ \{\mathsf{reg}\} \mapsto \{\mathsf{u}_1\} & \{\mathsf{nand}\} \mapsto \{\mathsf{b}_1\} & \{\mathsf{nor}\} \mapsto \{\mathsf{b}_3\} & \{\mathsf{xnor}\} \mapsto \{\mathsf{b}_5\} \end{array}$$

Given the gate-function mapping and the abstract representation of a digital circuit in terms of a DAG, we can formally describe the event, effects, and propagation of an injected fault.

**Definition 4 (Fault Event).** *A* fault event *can occur in a digital circuit* $\mathbf{C}$ *if and only if a gate* $g \in \mathcal{G}$ *within the circuit does not evaluate according to its associated Boolean function* $\mathsf{f}_g$*.*

**Definition 5 (Fault).** *A* fault *is defined by wrong values that manifest in primary sequential output gates* $g_r \in \mathcal{G}_s$*. Hence, a fault is always caused by a fault event.*

Considering our limited set of combinational and sequential gates, a fault event in a combinational gate occurs immediately if the considered $g \in \mathcal{G}_c$ evaluates to an incorrect result $z'$ with $z' \neq \mathsf{f}_g(\mathbf{x})$. For sequential and clock-synchronous gates $g_\mathrm{r} \in \mathcal{G}_s$, faults will only manifest in synchronization to the provided clock signal such that $z' \neq \mathsf{f}_r(x)$.

**Definition 6 (Fault Propagation).** *Given a digital circuit* $\mathbf{C}$ *modeled as DAG* $\mathbf{D}$ *and a manifestation of a fault event in a single gate, according to Definition 4. Then,* fault propagation *describes the effect of propagating incorrect intermediate results along the edges and vertices of the graph.*

However, if a fault event occurs in a gate of a digital circuit, i.e., the faulted gate evaluates incorrectly, this fault event may also have an impact on subsequent gates. More specifically, if a faulty signal $z'$ is input to further gates, these gates may evaluate correctly according to their associated function but still provide wrong results due to incorrect inputs. This effect in general is called *fault propagation* which is formally described by Definition 6. Two different scenarios of fault propagation are given in Example 1.

*Example 1.* In this example we assume a gate $g \in \mathcal{G}_c$ producing a faulty output $z' \in \mathbb{F}_2$. The faulty output $z' = 1$ is the first input $x_0$ to a gate $g_2 = \{\mathsf{or}\}$ while the second input $x_1 = 1$. In this case, fault propagation will stop immediately, as the output of $g_2$ will be 1 regardless of the first input. However, given that $x_1 = 0$, then, upon correct inputs, $g_2$ would evaluate to 0, however, due to $x_0 = z' = 1$, the fault will propagate through $g_2$ and may affect further gates in the circuit.

**Definition 7 (Fault Scenario).** *We define a* fault scenario *as the occurrence of a fault event in a target gate $g \in \mathcal{G}$ under a given input $\mathbf{x} \in \mathbb{F}_2^i$ to the circuit* $\mathbf{C}$.

Hence, each specific fault event in a target gate $g \in \mathcal{G}$ creates for each valid input $\mathbf{x} \in \mathbb{F}_2^i$ an own fault scenario. Therefore, the input size $i$, the amount of considered gates, and the number of valid fault events for each gate (more details will be given in Section 3.3) determine the total amount of fault scenarios $N_{\text{scenario}}$ for a given circuit $\mathbf{C}$.

**Definition 8 (Fault Coverage).** *The* fault coverage *is defined by the term $f_{cov} = 1 - \frac{N_{not}}{N_{scenario}}$ where $N_{not}$ describes the amount of fault scenarios that are not covered by an applied protection mechanisms associated with a circuit* $\mathbf{C}$.

## 3.3 Consolidated Adversary Model

Introducing two abstraction levels for a digital logic circuit, we can explain and introduce our generic and consolidated fault injection adversary model. As a consequence, this allows us to create a dedicated fault injection adversary model that can be adjusted by a set of parameters introduced afterwards.

**Initial Assumptions.** For this, let us define and list some initial assumptions in order to provide a reasonably complex fault injection model for digital logic circuits. First, we assume that all primary inputs to a target circuit $\mathbf{C}$ are fault-free, since inputs that are already faulted can never be recognized by any fault model framework or by countermeasures that should be evaluated. Second, we do not consider any signal delays or timing behaviors of the circuit in our fault model, as these are generally technology-specific and undermine our attempt to create a generic and abstract model. Hence, we work on a netlist level which can be perfectly mapped to DAGs as described in Section 3.1. Third, for all of our analyses, we assume a worst-case scenario. For this, given that we do not consider any signal and gate delays per se, we have to assume that any path in our circuit can be the critical path. Even though this may lead to consideration of fault scenarios that cannot occur in a real world design, it ensures a full coverage of all (theoretical) faults, ultimately achieving a stronger assertion with respect to evaluation of resistance and protection mechanisms. Finally, we do not specifically consider *persistent* faults in our fault adversary model. If persistent faults should be modeled, it can still be accomplished within our assumption by triggering a specific fault event on each evaluation. This, however, is not part of a fault model but rather part of the utilized framework integrating the fault adversary model.

**Abstraction Levels.** The description of a circuit $\mathbf{C}$ as a DAG $\mathbf{D}$ allows us to separate the fault modeling into two abstraction levels – a *structural level* and a *functional level.* On the structural level we consider the edges and vertices of the DAG, i.e., the wires in $\mathbf{C}$ connecting the circuit gates. This gives us the possibility to model, describe and track the propagation of fault events through the entire circuit. Additionally, the structural level provides information about the placement of synchronization points, i.e., the sequential memory gates. This information is important since faults ultimately will manifest in register stages which we will demonstrate in Section 4. However, the actual fault events are injected directly in combinational gates $g_{\mathrm{c}} \in \mathcal{G}_c$ or in sequential memory gates $g_{\mathrm{r}} \in \mathcal{G}_s$ where both types of gates describe the functional level of $\mathbf{C}$ through the associated Boolean functions given in $\tau_{golden}$ (cf. Section 3.2).

**Modeling Faults.** On a very abstract (and simplified) level, we model a single fault event by altering the associate function of the target gate to an arbitrary function within the same domain, i.e., defined over the same number of inputs and outputs. In particular, faults injected into a gate $g_u \in \mathcal{G}_u$ or in a sequential memory gate $g_{\mathrm{r}} \in \mathcal{G}_s$ are modeled by exchanging the associated function with a function $\mathsf{u} \in \mathcal{U}$. Similarly, faults injected into a gate $g_b \in \mathcal{G}_b$ are modeled by exchanging the associated function of $g_b$ with a function $\mathsf{b} \in \mathcal{B}$.

In this sense, for each fault scenario, the DAG of the circuit is re-evaluated and updated, such that for each vertex $v \in \mathcal{V}$ of the graph, the associated functions are selected from $\tau_{golden}$ or a fault type is chosen from a *fault model* $\tau_{faulty}$, depending on whether the fault event occurred in the corresponding gate or not, such that:

$$v_g = \begin{cases} \tau_{golden}(g) & g \text{ is fault-free} \\ \tau_{faulty}(g) & g \text{ is faulted} \end{cases}, \forall g \in \mathbf{C}$$

Notably, this model is as generic as possible and provides the opportunity to map all well-known fault types to a circuit implemented in hardware. To this end, we further define a notation which allows us to denote mappings where several gates are mapped to the same Boolean function. For example, given a subset of gates $\mathcal{G}_{\mathrm{sub}} \subset \mathcal{G}_b$ and each of the gates $g \in \mathcal{G}_{\mathrm{sub}}$ should be mapped to the functions $\mathsf{b}_6$ and $\mathsf{b}_7$ in case a corresponding gate in $\mathbf{C}$ is faulty, we denote the underlying mapping $\tau_j$ as $\tau_j : \mathcal{G}_{\text{sub}} \mapsto \{\mathsf{b}_6, \mathsf{b}_7\}$ for a specific fault model $j$. However, to meet realistic scenarios for an actual attacker, we further introduce a set of parameters which allows us to constrain the generic model and customize it depending on given circumstances.

**Parametric Adversary Model.** To this end, we introduce the following three parameters to describe the limitations of an adversary: $n$, $t$, and $l$. While the first parameter $n$ defines the power of the attacker in terms of how many faults can be injected at the same time, i.e., the *number* of fault events, the second parameter $t$ defines the *type* of the fault events. Finally, $l$ limits the circuit *locations* where the fault events can occur, i.e., this parameter defines the type of circuit gates that can be targeted by the adversary. In the following we present more details about the three parameters and their design rationals.

*Number of Faults $n$:* The parameter $n$ sets the total number of fault events that can occur at the same time and therefore it constraints the power of an attacker in terms of simultaneously injected faults. Hence, when modeling adversarial fault injections in a digital circuit $\mathbf{C}$, $n$ can be selected from $\mathcal{N} = \{1, 2, ..., N\}$ with $N = |\mathcal{V}|$, i.e., $N$ is equal to the total number of combinational and sequential gates that are available in $\mathbf{C}$.

However, selecting $n \in \mathcal{N}$, we assume that an attacker is able to inject up to $n$ faults, meaning that we consider all possible fault scenarios with $n' \leq n$ fault events. This assumption is well established in literature when evaluating the fault coverage of a target countermeasure [29,32]. However, even if we select $n$ as an upper bound, we still might observe more than $n$ faults manifesting in a register stage or primary output due to fault propagation (cf. Definition 6). We further explain this phenomena in Example 2.

*Example 2.* For this example we assume that we model an attacker with $n = 1$ and that a fault is injected into the $\mathsf{nand}$ gate in Figure 6. Although $n$ is constrained by 1, the fault can propagate through the $\mathsf{and}$ and $\mathsf{xor}$ gate such that three faults would eventually manifest at the primary output register stage. Hence, $n$ only indicates the number of fault events an attacker is able to inject but it does not give any information about the total number of faults that will occur in the circuit. This behavior was also mentioned by Aghaie et *al.* in [1].

*Fault Type $t$:* The fault type $t$ can be selected from a set $\mathcal{T} = \{\tau_{sr}, \tau_s, \tau_r, \tau_{bf}, \tau_{fm}\}$ which contains different fault models $\tau_j$. Each of these fault models describes how a gate from a target circuit $\mathbf{C}$ is mapped to a function $\mathsf{u} \in \mathcal{U}$ or $\mathsf{b} \in \mathcal{B}$ in the resulting DAG $\mathbf{D}$. In this paragraph, we introduce common fault models used to describe different fault injection mechanisms.

For this, we define $\tau_{sr}$ as a fault model where each gate from $\mathcal{G}$ is mapped to the *set* or *reset* function. Particularly, a gate $g_u \in \mathcal{G}_u$ or a sequential memory gate $g_r \in \mathcal{G}_s$ is modeled by the function $\mathsf{u}_2(x) = 0$ or by $\mathsf{u}_3(x) = 1$ with $x \in \mathbb{F}_2$.

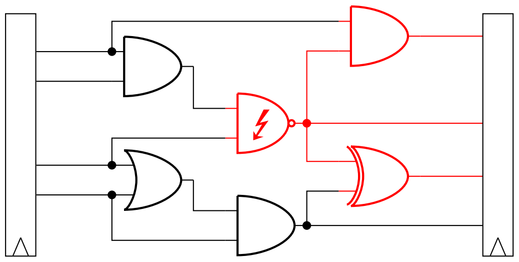


Fig. 6: Influence of a single fault on subsequent gates.

Hence, we apply a mapping that is described by $\{\mathsf{not}, \mathsf{reg}\} \rightarrow \{\mathsf{u}_2, \mathsf{u}_3\}$. Similarly, a faulty gate $g_b \in \mathcal{G}_b$ is modeled by one of the functions $\mathsf{b}_6(\mathbf{x}) = 0$ or $\mathsf{b}_7(\mathbf{x}) = 1$ with $\mathbf{x} \in \mathbb{F}_2^2$, describing a reset or set fault, respectively. In this case, the mapping is formally described by $\{\mathcal{G}_b\} \rightarrow \{b_6, b_7\}$. In essence, the mapping $\tau_{sr}$ serves as a base line for most of the fault injection mechanisms introduced in Section 2, however, more details about the connection between the physical behavior and the proposed parameter selections are given in Section 4.

In order to allow more fine grained evaluations, we additionally define the mappings $\tau_s$ and $\tau_r$ which describe only set or only reset fault events, respectively. Hence, the mapping $\tau_r$ defines $\{\mathsf{not}, \mathsf{reg}\} \rightarrow \{\mathsf{u}_2\}$ for unary gates and $\mathcal{G}_b \rightarrow \{\mathsf{b}_6\}$ for binary gates. Similarly, $\tau_s$ defines $\{\mathsf{not}, \mathsf{reg}\} \rightarrow \{\mathsf{u}_3\}$ for unary gates and $\mathcal{G}_b \rightarrow \{\mathsf{b}_7\}$ for binary gates. This distinction can be useful for specific primitives or technologies where a fault injection attack can either cause set or reset fault events only. Examples for such primitives are NOR flash memories where only bit-set faults occur as shown and explained in [9].

Another common fault model that can be found in the literature is based on bit-flips which we describe by the mapping $\tau_{bf}$. In this case we map each gate from $\mathcal{G}$ to its inverse gate resulting in the following fault model:

$$\tau_{bf}\colon \begin{array}{llll} \{\mathsf{not}\} \mapsto \{\mathsf{u}_1\} & \{\mathsf{and}\} \mapsto \{\mathsf{b}_1\} & \{\mathsf{or}\} \mapsto \{\mathsf{b}_3\} & \{\mathsf{xor}\} \mapsto \{\mathsf{b}_5\} \\ \{\mathsf{reg}\} \mapsto \{\mathsf{u}_0\} & \{\mathsf{nand}\} \mapsto \{\mathsf{b}_0\} & \{\mathsf{nor}\} \mapsto \{\mathsf{b}_2\} & \{\mathsf{xnor}\} \mapsto \{\mathsf{b}_4\} \end{array}$$

Thus, each gate is modeled by a function returning the inverse of the values that would be returned by the original gate.

Eventually, we intentionally leave space for custom definitions of fault models $\tau_{fm}$ in $\mathcal{T}$ to provide an adversary model that is as generic as possible while at the same time already covering common fault types and models. For this, we introduce one custom mapping $\tau_{nang15}$ in Section 4 and guide the reader through the process of precisely defining and modeling an attacker that uses a laser to inject fault events in a Nangate 15 nm technology.

*Fault Location $l$:* The fault location $l$ is the third parameter which is necessary to properly describe fault injections in our generic adversary model. We define the set $\mathcal{L} = \{\mathrm{c}, \mathrm{s}, \mathrm{cs}\}$ in order to distinguish between different areas on the structural level of a circuit $\mathbf{C}$. The first choice covers and models fault injections that

Table 2: Summary of available parameters to accurately model fault injections.

| **Param.** | **Options** | **Description** | |
|---|---|---|---|
| $n$ | $\mathcal{N} = \{1, 2, ..., N\}$ | Maximum number of fault events, $N = \lvert\mathcal{V}\rvert$ | |
| $t$ | $\mathcal{T} = \{\tau_{sr}, \tau_s, \tau_r, \tau_{bf}, \tau_{fm}\}$ | $\tau_{sr}$: | Fault model for set/reset fault |
| | | $\tau_s$: | Fault model for set faults |
| | | $\tau_r$: | Fault model for reset faults |
| | | $\tau_{bf}$: | Fault model for bit-flips faults |
| | | $\tau_{fm}$: | User-specified fault model |
| $l$ | $\mathcal{L} = \{\text{c}, \text{s}, \text{cs}\}$ | c: | Fault events occur in combinational gates only |
| | | s: | Fault events occur in sequential gates only |
| | | cs: | Fault events occur in all available gates |

solely affect combinational logic gates, i.e., gates from $\mathcal{G}_c$. Setting $l = \text{s}$, specifies fault injections where an attacker targets sequential memory gates $g_r \in \mathcal{G}_s$ only. Eventually, $l = \text{cs}$ models faults that can occur in both types of gates.

For reasons of clarity, Table 2 summarizes the available parameters with the corresponding options which are shortly described in the last column.

**Instantiating Adversary Models.** To bring together and connect the three introduced parameters $n$, $t$, and $l$, we define the function $\zeta(n, t, l)$. This allows us to instantiate different types of attackers and model the behavior of fault injections based on the committed parameter list. For example we can regulate the strength by changing the fault type $t$, or determine the accuracy of the fault injections setting $n$ as a powerful attacker may be able to precisely inject single bit faults. However, one of the main advantages of introducing $\zeta$ is that we create a basis to allow comparability between different designs which should be evaluated regarding their protection against fault injection attacks (under a given adversary model). In Example 3, we evaluate an exemplary circuit which is protected by duplication using our generic adversary model to transfer our definitions and notations to a practical instantiation.

*Example 3.* Figure 7 depicts an exemplary circuit $\mathbf{C}$ which describes a function $\mathsf{h} : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2^2$. The circuit is protected by duplication [5] against fault injection attacks. We now apply our generic adversary model to $\mathbf{C}$ in order to evaluate the protection level. In a first step we set $\zeta(1, \tau_{sr}, \text{c})$ meaning that an attacker injects up to one set or reset fault event in a combinational gate. For this, any comparator logic placed after the output register stage would detect all possible fault scenarios that can occur under the given adversary model. All together, there are four valid inputs $\mathsf{i} \in \mathbb{F}_2^2$ (due to our initial assumptions, the inputs are correct and the same for the original circuit and the duplication) whereas there are four different gates that could be faulted. This results in 16 possible fault scenarios for each fault type, i.e., 16 fault scenarios for set faults and 16 fault scenarios for reset faults which eventually results in a total amount of fault

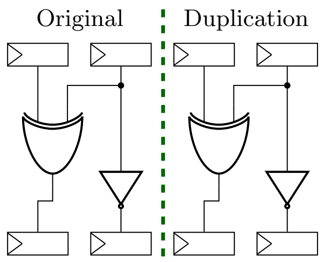


Fig. 7: Exemplary circuit to demonstrate the application of our generic fault model.

scenarios $N_{\text{scenario}} = 32$. We can conclude that $\mathbf{C}$ has a fault coverage of $100\,\%$ under $\zeta(1, \tau_{sr}, \mathrm{c})$.

Now we consider the adversary model $\zeta(2, \tau_{sr}, \mathrm{c})$ where an attacker is assumed to inject up to two set or reset fault events in combinational gates. Note that $\zeta(2, \tau_{sr}, \mathrm{c})$ also covers each fault event that can occur under $\zeta(1, \tau_{sr}, \mathrm{c})$ as explained above. Additionally, for each input $\mathbf{i} \in \mathbb{F}_2^2$ there are $\binom{4}{2} = 6$ fault locations for each fault type. In this case we have four different combinations of allocating the available fault types to the fault locations: two set faults, one set and one reset fault, one reset and one set fault, or two reset faults. All in all, this leaves us with $N_{\text{scenario}} = 4 \cdot 6 \cdot 4 + 32 = 128$ possible fault scenarios. From these fault scenarios, the circuit will not be able to detect eight faults. Here, reset faults in the not gates for the inputs $\mathbf{i} \in \{01, 11\}$ and set faults for the inputs $\mathbf{i} \in \{00, 10\}$ will not be detected. For faults in the xor gates, the inputs $\mathbf{i} \in \{01, 10\}$ will not be detected in case both gates will be faulted by reset faults and the inputs $\mathbf{i} \in \{00, 11\}$ will not be detected in case both gates will be faulted by set faults. To summarize, the adversary model $\zeta(2, \tau_{sr}, c)$ returns a fault coverage of $f_{\text{cov}} = 1 - 8/128 = 93.75\,\%$.

Note, however, that we did not have to consider fault propagation in this example since no output of a gate is an input to another gate.

## 4 Practical Instantiation

After we introduced our generic adversary model expressed through the corresponding function $\zeta$ to model adversaries with different capabilities, we show in this section how to map our theoretical considerations to real world fault injection mechanisms and how to model associated adversaries. Therefore, we establish a connection between available fault injection mechanisms introduced in Section 2.2 and our findings from Section 3.

Further, to demonstrate the practical application of $\zeta$, we consider the ASCON S-box [12] as an example for a cryptographic primitive and potential target of FIA. The corresponding S-box circuit is depicted in Figure 8, exhibiting some interesting properties that provide a good starting point for the application and discussion of our concept. First, this circuit already consists of both gate types,

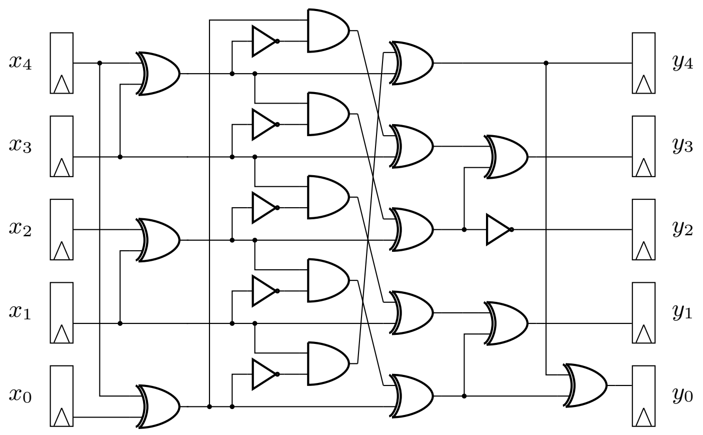


Fig. 8: ASCON S-box [12].

i.e., combinational gates from $\mathcal{G}_c$ and sequential memory gates from $\mathcal{G}_s$. Second, although the circuit is constructed on an almost regular pattern, fault propagation can be observed. More specifically, at the deepest logic level, the primary output $y_4$ is an input to the xor-gate which determines the output $y_0$. Hence, the structure of the ASCON S-box is perfectly suited to demonstrate and discuss different practical instantiations of our generic adversary model.

However, to facilitate comparison of different fault injection mechanisms and associated adversary models, we first define the total number of effective faults $N_{\text{effective}}$ as (single-bit) faults that eventually manifest in a primary output stage of a circuit $\mathbf{C}$. In our example, $N_{\text{effective}}$ can be at maximum five for each given fault scenario since the number of output registers is $o = 5$. Given that, the maximum number of possible faults $N_{\max}$ is limited by

$$N_{\max} = o \cdot N_{\text{scenario}} = o \cdot 2^i \cdot \sum_{g \in \mathbf{C}} |\tau_j(g)| \tag{3}$$

under a considered fault model $j$. Further, this allows us to introduce a circuit fault rate $r_{\text{fault}} = \frac{N_{\text{effective}}}{N_{\max}}$ to compare different instantiations of our adversary model function $\zeta$.

As indicated in Equation 3, we consider each valid input combination to determine the maximum number of possible faults $N_{\max}$ because each valid input creates an independent fault scenario. We explicitly decided to follow this approach for the considered example to allow a fairer comparison between the different models and instantiations of $\zeta$ at the end of this section. Note that for real-world applications (e.g., analyzing entire protected block-ciphers) with inputs $i \geq 64$ such an analysis is currently not possibly. However, this is not a limitation of our adversary model (since our model targets the description of fault occurrences in hardware gates) but a limitation of exhaustively simulating fault injections in all available gates for all valid inputs which is not part of the work and still an open research question.

### 4.1 Clock Glitches

As first step, in order to model clock glitches, we instantiate our adversary model as $\zeta(n, \tau_{sr}, \mathrm{c})$ with $n \in \mathcal{N}$. This setting perfectly models the underlying fault mechanism of clock glitches explained in Section 2.2. By setting $t = \tau_{sr}$ and $l = \mathrm{c}$, we consider *set* and *reset* fault events occurring in combinational gates $g \in \mathcal{G}_c$ only.

This choice can be justified when looking closer at the origin of faults injected by clock glitches. Particularly, if the attacker manages to decrease the clock period of a clock cycle, this makes registers sample their input signals early. Then, a fault occurs if the internal logic cannot propagate the correct signals timely and the current input to a register differs from the correct input. As a consequence, there are several possibilities how false inputs can occur, depending on the duration of the clock glitch. For shorter clock glitches, it is more likely that the propagation of the correct signal is interrupted in the first gates of the target circuit. Assuming that in total $n$ gates are affected by the clock glitch means that the output values of the considered gates are either one or zero (depending on the former state and previously processed date). However, this behavior can perfectly be modeled by set or reset fault events in the corresponding (combinational) gates. We further assume that the previous state already propagated through the circuit and is stable at the time of sampling. This assumption is valid since – as already mentioned above – the set or reset depends on previous data so that the (wrong) signal had enough time to travel through the logic.

For clarification of these decisions, we transfer this model to the ASCON S-box depicted in Figure 8. In our fault analysis, we always consider worst-case scenarios, as already mentioned in Section 3.3, which also includes that we model fault events independent of placing and routing of a circuit and that we neglect run times of gates and wires. Hence, when examining a target gate, we assume that – due to a worst-case place and route – the considered gate is located in the critical path. For example, in our modeling phase, we examine the second and gate from top in Figure 8 which means that we evaluate the primary outputs of the circuit when setting the output of the and gate to either zero or one. Hence, we assume that the propagation of the input signals is stopped in the and gate and that the output is constantly either zero or one. As this behavior also affects the subsequent xor-gates and the inverter, we further have to consider fault propagation during our analysis.

Hence, applying the specific adversary model $\zeta(1, \tau_{sr}, \mathrm{c})$ to the ASCON S-box, we consider $2^5$ input combinations, 22 available gates to inject a set or reset fault event, and five output registers to observe a fault, resulting in a maximum number of $N_{\max} = 7\,040$ faults. Then, for each fault, we compare the resulting output $y_i'$ to the correct S-box output $y_i$ and for each output bit that is different to the correct one, we increase a fault counter which eventually results in $N_{\text{effective}} = 1\,040$ effective faults appearing at the output. To summarize, this analysis leaves the ASCON S-box circuit with a fault rate of $r_{\text{fault}} = 14.77\,\%$ under the given $\zeta(1, \tau_{sr}, \mathrm{c})$ adversary model.

## 4.2 Voltage Glitches

As mentioned in Section 2.2, the fundamental physical behavior of voltage glitches (or underpowering) is very similar to clock glitches and is caused through the violation of Equation 1. By lowering the voltage, the right side of this inequality is increased such that the sequential memory gates are triggered before all signals can propagate through the logic. As a result, this phenomena can be modeled by the same adversary model as clock glitches. Hence, we model fault injections caused by voltage glitches also by a $\zeta(n, \tau_{sr}, \mathrm{c})$ adversary with $n \in \mathcal{N}$.

Obviously, applying the voltage glitch adversary model to the ASCON S-box results in the same number of maximal possible faults $N_{\max}$ and the same number of effective faults $N_{\text{effective}}$ as described in Section 4.1. Hence, the fault rate for the ASCON S-box is also given by $r_{\text{fault}} = 14.77\,\%$ under the given $\zeta(1, \tau_{sr}, \mathrm{c})$ adversary model.

## 4.3 Electromagnetic Pulses

The modeling of fault injections caused by EMPs can be conducted by instantiating the adversary model function as $\zeta(n, \tau_{sr}, \mathrm{s})$ with $n \in \mathcal{N}$. In Section 2.2, we explained that EMPs induce a positive or negative voltage pulse in the target circuit. These pulses produce a set or reset fault event respectively. Hence, setting $t = \tau_{sr}$ perfectly covers the physical mechanism of EMFIs since this fault model maps the target gate function to the set or reset function. Additionally, the choice for selecting sequential memory gates as fault location $l$, matches the recently published results by Dumont et *al.* [13, 14] who refined and confirmed the model of sampling faults caused by EMFI.

Applying the fault model function to the considered example depicted in Figure 8, leaves us with $2^5$ input combinations, 5 gates for the set and reset faults (ignoring primary inputs as we assume inputs to be correct), and five output registers, which eventually results in $N_{\max} = 1\,600$ possible faults at the primary output. All together, there are $N_{\text{effective}} = 160$ effective faults resulting in a fault rate of $r_{\text{fault}} = 10\,\%$ for the ASCON S-box under the given $\zeta(1, \tau_{sr}, \mathrm{s})$ adversary model.

## 4.4 Optical Fault Injections

At last, considering the mechanism of optical fault injections, an appropriate modeling is possible by defining the adversary model function as $\zeta(n, \tau_{fm}, \mathrm{cs})$. This selection covers fault injections into combinational and sequential memory gates likewise. As described in Section 2.2, a focused laser beam on a digital circuit charges or discharges specific nodes on transistor level. Hence, targeting sequential gates, the value of a register can either be set or reset which needs to be covered in the custom fault mapping $\tau_{fm}$ defining $\{\mathsf{reg}\} \mapsto \{\mathsf{u}_2, \mathsf{u}_3\}$.

Additionally, the instantiation of the adversary function covers fault events that directly occur in the combinational logic. Here, we define specified mappings for $\tau_{fm}$ between the instantiated gates $g \in \mathcal{G}_c$ and the defined functions in $\mathcal{U}$ and

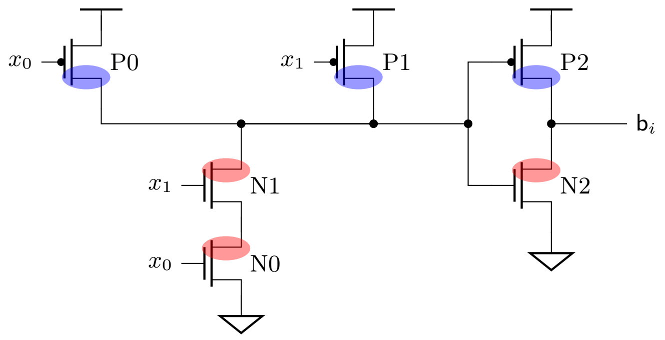


Fig. 9: AND gate from the 15 nm Open-Cell Library. Blue areas mark drain regions of PMOS transistors, red areas mark drain regions of NMOS transistors.

$\mathcal{B}$. The simplest example – faults occurring in a CMOS inverter – was already discussed in Section 2.2 where the mapping $\{\mathsf{not}\} \mapsto \{\mathsf{u}_2, \mathsf{u}_3\}$ is applied. For the remaining gates from $\mathcal{G}_b$, we now exemplary derive the mapping of an $\mathsf{and}$ gate designed in the 15 nm Open-Cell Library[3] which is depicted in Figure 9. Therefore, we will call the custom defined mapping $\tau_{fm}$ in the following $\tau_{nang15}$ as it is tailored to the given example.

The $\mathsf{and}$ gate consists of six transistors where three transistors are NMOS and three are PMOS transistors. In our model, we assume that an adversary can affect any number of transistors available in a target gate. Hence, our parameter $n$ only describes the number of fault events on gate level but does not distinguish the number of charged or discharged nodes. However, in case of the considered $\mathsf{and}$ gate, the attacker can easily change the function to a set or reset behavior by affecting the inverter stage. Additionally, it is possible to simultaneously inject a drift current $I_{\text{drift}}$ into the transistors N0 and N1 to force the gate to behave as an $\mathsf{or}$ gate. This is possible if $I_{\text{drift}}$ is larger than the current delivered by one of the PMOS transistors P0 and P1 such that the input node to the inverter can be discharged if either P0 or P1 conducts. In case both PMOS transistors conduct, the injected drift current would be too small to discharge the input node to the inverter so that the output of the gate would be zero. These observations lead us to the mapping $\{\mathsf{and}\} \mapsto \{\mathsf{b}_2, \mathsf{b}_6, \mathsf{b}_7\}$ which is added to $\tau_{nang15}$.

Fault mappings for all remaining gates from $\mathcal{G}$ can be derived in a very similar way. As we require a specific mapping for a $\mathsf{xor}$ gate in order to evaluate the example from Figure 8, the corresponding schematic is shown in Figure 10 in the appendix. Again, the attacker can easily generate set and reset fault events by charging or discharging the output node. However, there are other possible modifications which allow the attacker to force the gate to behave as a $\mathsf{nand}$ gate or as an $\mathsf{or}$ gate. The former change can be achieved by discharge the input node to the second stage, i.e., by shooting on N0 or N1. To force the gate to behave

[3] `https://si2.org/open-cell-library/`

Table 3: Intersection of fault injection models with available fault injection mechanisms for the ASCON S-box.

| | **Modeled Mechanism** | | |
|---|---|---|---|
| **Covered Mechanism** | *Laser* | *Clock/Voltage* | *EM* |
| *Laser* | 100 % | 63.4 % | 9.8 % |
| *Clock/Voltage* | 100 % | 100 % | 0 % |
| *EM* | 100 % | 0 % | 100 % |

as an or gate, the attacker has to hit one of the PMOS transistors P3 or P4. All together, we end up with the mapping $\{\mathsf{xor}\} \mapsto \{\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_6, \mathsf{b}_7\}$.

For the sake of completeness, we listed the corresponding mappings for all remaining gates from $\mathcal{G}_c$, describing all fault types in presence of Laser Fault Injections (LFIs) for the Nangate 15 nm technology, in the appendix in Table 5.

Evaluating the ASCON S-box, given the described adversary model instantiation $\zeta(1, \tau_{nang15}, \text{cs})$, can be accomplished by distinguishing between faults that are injected into sequential gates and combinational gates. The former case reveals the same fault rate as the evaluation under faults caused by EMPs. Hence, $N_{\text{max}} = 1\,600$ while there are 160 effective faults. The evaluation considering faults in combinational gates results in $N_{\text{max}} = 11\,360$ possible faults while there are $N_{\text{effective}} = 1\,480$ effective faults. Thus, adding the numbers of effective faults to $N_{\text{effective}} = 1\,640$ and possible faults to $N_{\text{max}} = 12\,960$ results in a total fault rate of $r_{\text{fault}} = 12.65\,\%$ for the ASCON S-box under the given $\zeta(1, \tau_{nang15}, \text{cs})$ adversary model.

### 4.5 Comparison of Fault Injection Mechanisms

Eventually, Table 3 compares the four considered and modeled fault injection mechanisms by assuming that the mechanisms are applied column-wise and an evaluation based on the introduced models is performed. Each row indicates – for the corresponding mechanism – the percentage intersection of the applied model. Particularly, when selecting the introduced function to model LFI, all other fault injection mechanisms are covered as well, meaning that a design, which is secure under the laser fault model, is also secure under all remaining models. Switching to the clock or voltage model, results in a 63.4 % coverage of all faults that can occur when applying the laser fault model. Note that this specific number only holds for the presented case of the ASCON S-box and depends on the utilized gates and the corresponding mapping functions. However, considering faults caused by EMPs, achieves the smallest intersection with the remaining fault injection mechanisms. Only 9.8 % of the faults caused by lasers are also covered when modeling electromagnetic based faults while clock or voltage glitches are not covered.

This comparison illustrates that the introduced adversary models can describe the different fault injection mechanisms in a finer grained fashion. A distinct differentiation between the available fault injection mechanisms based on the instantiated $\zeta$ is easily possible. It further demonstrates that a design, which is evaluated and secure in the laser fault model $\zeta(n, \tau_{nang15}, \mathrm{cs})$, is also protected against adversaries using one of the other fault injection mechanisms to attack a cryptographic hardware implementation. However, this does not hold vise versa so that a design, which for example is evaluated and secure under the EM fault model $\zeta(n, \tau_{sr}, \mathrm{s})$, is not necessarily secure against attackers using optical based fault injection mechanisms.

## 5 Case Study: Integration to VerFI

In this section we demonstrate the practical application of our new adversary model while integrating it to the state-of-the-art verification tool for fault injections VerFI [4].

**VerFI.** VerFI is an open-source tool to verify hardware countermeasures against fault injection attacks presented at HOST in 2019 [4]. The tool works on netlist-level and can be configured via a simulation file. This file contains information about the plaintext and key, which should be used for the analysis, different parameters that are cipher related (e.g., duration in clock cycles, port names, end condition), and parameters to define the fault injection. Given that, one can precisely specify the submodules which should be faulted, how many faults should be injected, and which fault injection type should be considered (toggle faults, stuck-at-0, stuck-at-1). Based on these information, VerFI analyzes the given circuit and reports the number of non-detected faults, detected faults, ineffective faults, and the total number of evaluated faults.

**Adjustments to VerFI.** In order to demonstrate the application of our proposed adversary model, we adapted VerFI such that it was able to work with user defined fault mappings. Therefore, we modified the `library`-file and extended the parameter list for each gate by fault mappings which are specified in form of Boolean expressions. Consequently, we adapted the parsing function which reads in the gates from the `library`-file and stores the corresponding parameters. The required expressions describing the fault mappings are evaluated and stored in Look-Up Tables (LUTs) which are used in the fault simulation step to generate the outputs of the faulty gates. Within this fault simulation step each valid combination of fault mappings for a set of target gates (gates in which faults are currently injected) is analyzed before the next set of target gates is determined.

**Analyzing a protected LED implementation.** To demonstrate the evaluation of a protected block cipher, we selected an implementation of the lightweight block cipher LED64 [18] taken from Impeccable Circuits [1]. The authors published a list of hardware implementations of common block ciphers where each

Table 4: Fault analysis of LED64 using VerFI. The top three rows are results produced by using the proposed fault models in VerFI. The lower three rows report results obtained from an adapted version of VerFI reflecting our generic adversary model.

| **Fault Model** | **Detected** | **Non-detected** | **Ineffective** | **Scenarios (sum)** |
|---:|:-:|:-:|:-:|:-:|
| $\zeta(4, \tau_{bf}, \text{cs})$ | 97 428 | 3 598 | 1 064 | 102 090 |
| $\zeta(4, \tau_{s}, \text{cs})$ | 96 660 | 497 | 4 933 | 102 090 |
| $\zeta(4, \tau_{r}, \text{cs})$ | 87 372 | 49 | 14 669 | 102 090 |
| $\zeta(4, \tau_{sr}, \text{c})$ | 548 672 | 3 584 | 64 832 | 617 088 |
| $\zeta(4, \tau_{sr}, \text{s})$ | 1 520 | 14 | 162 | 1 696 |
| $\zeta(4, \tau_{nang15}, \text{cs})$ | 14 383 842 | 72 462 | 1 245 232 | 15 701 536 |

cipher is implemented with different levels of protection[4]. We decided to use the LED implementation where each state nibble is protected by four bits of redundancy. We constrained the allowed area for fault injections to the xor-gates adding the multiplication results in MixColumns for one resulting nibble and to the following 4-bit state register in the data path as well as in the redundancy. All together, the total number of target gates consists of 32 xor-gates and eight registers. The target circuit was analyzed for $n = 4$ while injecting faults in clock cycle 31 only. The plaintext and key were fixed to the values

```
plaintext = 0x0123456789ABCDEF      key = 0xDEADBEEFDEADBEEF
```

for all following analyses. Hence, compared to the previous examples, we performed a non-exhaustive evaluation with respect to the inputs.

Table 4 summarizes the evaluation results provided by the adjusted version of VerFI[5]. The upper three rows report the results for the fault models which where originally provided with VerFI (toggle, stuck-at-1, stuck-at-0) instantiated with our adversary model. The number of fault scenarios is the same for all three cases and is given by $\sum_{k=1}^{4} \binom{32+8}{k}$ since setting $n = 4$ also includes fault injections with $n < 4$.

The lower three rows summarize the VerFI report for adversary models instantiated for clock and voltage glitches, for electromagnetic pulses, and for laser fault injections, respectively. The number of fault scenarios for the clock glitch model results in $\sum_{k=1}^{4} \binom{32}{k} \cdot 2^k$. For each combination of $k$ target gates there are $2^k$ possibilities to combine set and reset faults. There are 14 fewer non-detected faults compared to the bit-flip model since the registers are not allowed to be faulted in the clock glitch model. Switching to $\zeta(4, \tau_{sr}, \text{s})$ (i.e., modeling faults caused by electromagnetic pulses), results in $\sum_{k=1}^{4} \binom{8}{k} \cdot 2^k$ fault scenarios while only 14 scenarios are not-detected. This number depends on the underlying linear code which in this case has 14 valid codewords with a Hamming weight of

[4] All these implementations can be found in the authors' git repository: `https://github.com/emsec/ImpeccableCircuits`

[5] Detailed results (also for $n < 4$) can be found in Table 6 in the Appendix.

four. Evaluating the countermeasure based on the adversary model describing laser fault injections, leads to the largest number of fault scenarios. The number is given by

$$\sum_{k=1}^{4}\sum_{j=0}^{k}\binom{32}{j}\cdot\binom{8}{k-j}\cdot 2^{(k+j)}.$$

The first binomial coefficient describes the number of faults occurring in the combinational gates while the second binomial coefficient determines the number of sequential gates that are faulted. The last term determines the combinations of fault mappings that exists for one combination of $k$ target gates. However, evaluating the target countermeasure, results in the largest number of non-detected faults which mainly is caused by the increased number of fault scenarios.

Based on these results, it can clearly be seen that our model provides a more fine-grained classification of fault models. A target design can be analyzed under different adversary models which are tailored to the most common fault injection mechanisms. Additionally, using these results to compare the security to other protection schemes, is much more consistent and straightforward to accomplish.

## 6 Discussion

After all, we would like to briefly summarize and discuss the benefits of a unified adversary model to describe fault injection attacks. First, using a unified fault adversary model allows a *distinct evaluation* of developed countermeasures and protection mechanisms under the same assumptions. Second, while our adversary model perfectly describes actual fault injection mechanisms, it is also designed to work as *generic* and *simple* as possible. This makes an application easy to use and allows a straightforward instantiation in practice. Third, the application of a consolidated fault adversary model enables the possibility to *compare* proposed countermeasures based on the same assumptions and limitations with respect to the attacker. In this sense, our model strives to fulfill these criteria since only a limited number of parameters is necessary to describe and model the adversary in a very *compact* way. In essence, the user directly conceives the properties of the instantiated adversary model and can compare it to evaluations of protection schemes under a similar adversary model what makes it highly *expressiveness*. Fourth, due to the generic form, the adversary model can naturally be adapted to other technologies and hardware primitives (e.g., different memory technologies). With this property the model achieves a high *transferability* being able to customize it to the given circumstances and environments.

**Expansion to more advanced logic gates.** Even given that we limited our fault model in Section 3 to unary and binary logic gates, it could be easily and without any restrictions expanded to more advanced logic gates. This includes standard logic gates (e.g., and, or) with more than two inputs and optimized gates like and-or-invert. To expand our adversary model, the user defines additional sets containing all functions with $i$-bit inputs with $i > 2$. The corresponding set would then contain $2^{(2^i)}$ different functions that would transform the $i$-bit input to a 1-bit output. Given the additional sets of functions, the used mappings in $\tau$ need to be adapted in order to accurately describe the occurring faults.

**Limitations of VerFI.** Despite the fact that we were able to integrate our adversary model to the fault verification tool VerFI, we see some limitations with respect to the evaluation results. In VerFI, the user can just evaluate the given design by fixing the plaintext and key to a constant value. This covers not all fault scenarios and could lead to false positives when evaluating a countermeasure against fault injection attacks. Additionally, beyond the practical evaluation using VerFI of this work we see further potential for performance improvements in order to evaluate larger parts of the target design within the same run.

**Comparison to existing models.** The most common and already existing fault models are restricted to toggle, stuck-at-1, and stuck-at-0 faults (in our model we call them bit-flip, set, and reset faults, respectively). Compared to these approaches our model can analyze fault injections in digital circuit more precisely and in more detail incorporating the physical behavior of the different fault injection mechanisms. However, an argument against our model could be that an user of a fault verification tool would still cover all worst-case fault scenarios by applying a bit-flip model resulting in much less combinations of fault mappings that need to be tested. The bit-flip model comes with the disadvantage of insufficient precision regarding the description of fault injections. First, it is not possible to distinguish between different fault injection mechanisms. Second, faults in some hardware primitives cannot be accurately modeled like the in Section 3.3 mentioned NOR flash memory. Third, besides these arguments, our proposed fault adversary model enables the user to precisely reconstruct the cause of failures in a developed countermeasure.

**Limitations of our proposed model.** Despite these clear advantages, our adversary model is not reflecting parameters of the technology node and the physical layout of integrated circuits. Since the model considers hardware designs on netlist level, a detailed integration of technology related parameters is not (yet) possible. Additionally, place-and-route information cannot be used in the evaluation phase resulting in worst-case assumptions for e.g., critical paths. Even if these limitations lead to an increased number of fault scenarios and therefore to an increased number of fault combinations needed to be tested, the revisited fault adversary model describes fault injections more precisely while ensuring a thorough coverage of occurring fault types.

## 7 Conclusion

By reviewing and summarizing existing fault injection mechanisms developed over the last two decades, we created a basic understanding of the physical behavior appearing on the actual hardware when attacking cryptographic implementations. Subsequently, we introduced a generic and abstract (but simple) fault adversary model which can freely be parametrized by selecting three parameters describing the number of fault events, the fault types, and the fault locations. Given that, we connected the practical fault injection mechanisms with the theoretical introduced fault adversary model and accurately described how it has to be instantiated to provide a perfect mapping between theory and practice. This connection gave us the opportunity to demonstrate the application of the adversary models – instantiated to model different attack mechanisms – to a practical case study of a protected design of the lightweight cipher LED. This case study was accomplished by extending the fault verification tool VerFI by our proposed adversary model. Eventually, we discussed the advantages and benefits of using our consolidated fault adversary model and limitations in existing state-of-the-art verification tools.

## Acknowledgment

The work described in this paper has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

## References

1. Aghaie, A., Moradi, A., Rasoolzadeh, S., Shahmirzadi, A.R., Schellenberg, F., Schneider, T.: Impeccable Circuits. IEEE Trans. Computers **69**(3), 361–376 (2019)
2. Agoyan, M., Dutertre, J.M., Naccache, D., Robisson, B., Tria, A.: When Clocks Fail: On Critical Paths and Clock Faults. In: International Conference on Smart Card Research and Advanced Applications. pp. 182–193. Springer (2010)
3. Anceau, S., Bleuet, P., Clédière, J., Maingault, L., Rainard, J., Tucoulou, R.: Nanofocused X-Ray Beam to Reprogram Secure Circuits. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 175–188. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_9, `https://doi.org/10.1007/978-3-319-66787-4_9`
4. Arribas, V., Wegener, F., Moradi, A., Nikova, S.: Cryptographic Fault Diagnosis using VerFi. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020. pp. 229–240. IEEE (2020). https://doi.org/10.1109/HOST45689.2020.9300264, `https://doi.org/10.1109/HOST45689.2020.9300264`
5. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE **94**(2), 370–382 (2006)
6. Baumann, R.C.: Soft Errors in Commercial Integrated Circuits. International Journal of High Speed Electronics and Systems **14**(02), 299–309 (2004)
7. Beckers, A., Kinugawa, M., Hayashi, Y., Fujimoto, D., Balasch, J., Gierlichs, B., Verbauwhede, I.: Design Considerations for EM Pulse Fault Injection. In: International Conference on Smart Card Research and Advanced Applications. pp. 176–192. Springer (2019)

8. Canivet, G., Maistri, P., Leveugle, R., Clédière, J., Valette, F., Renaudin, M.: Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA. J. Cryptol. **24**(2), 247–268 (2011)
9. Colombier, B., Menu, A., Dutertre, J.M., Moëllic, P.A., Rigaud, J.B., Danger, J.L.: Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 1–10. IEEE (2019)
10. Courbon, F., Loubet-Moundi, P., Fournier, J.J., Tria, A.: Adjusting Laser Injections for Fully Controlled Faults. In: International workshop on constructive side-channel analysis and secure design. pp. 229–242. Springer (2014)
11. Dehbaoui, A., Dutertre, J.M., Robisson, B., Tria, A.: Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AEs. In: Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 7–15. IEEE (2012)
12. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1. 2 (2016), `https://competitions.cr.yp.to/round3/asconv12.pdf`
13. Dumont, M., Lisart, M., Maurine, P.: Modeling and Simulating Electromagnetic Fault Injection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2020)
14. Dumont, M., Lisart, M., Maurine, P.: Electromagnetic Fault Injection: How Faults Occur. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 9–16. IEEE (2019)
15. Dutertre, J.M., Beroulle, V., Candelier, P., De Castro, S., Faber, L.B., Flottes, M.L., Gendrier, P., Hely, D., Leveugle, R., Maistri, P., et al.: Laser Fault Injection at the CMOS 28 nm Technology Node: an Analysis of the Fault Model. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 1–6. IEEE (2018)
16. Endo, S., Sugawara, T., Homma, N., Aoki, T., Satoh, A.: An on-chip glitchy-clock generator for testing fault injection attacks. Journal of Cryptographic Engineering **1**(4), 265 (2011)
17. Gierlichs, B., Schmidt, J., Tunstall, M.: Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In: Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings. pp. 305–321 (2012)
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 326–341. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_22, `https://doi.org/10.1007/978-3-642-23951-9_22`
19. Hutter, M., Schmidt, J.M.: The Temperature Side Channel and Heating Fault Attacks. In: International Conference on Smart Card Research and Advanced Applications. pp. 219–235. Springer (2013)
20. Knudsen, L.R., Robshaw, M.: The Block Cipher Companion. Information Security and Cryptography, Springer (2011)
21. Korak, T., Hutter, M., Ege, B., Batina, L.: Clock Glitch Attacks in the Presence of Heating. In: Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 104–114. IEEE (2014)
22. Maurine, P., Tobich, K., Ordas, T., Liardet, P.Y.: Yet another fault injection technique: by forward body biasing injection. In: YACC'2012: Yet Another Conference on Cryptography (2012)

23. O'Flynn, C.: Low-Cost Body Biasing Injection (BBI) Attacks on WLCSP Devices. In: Liardet, P., Mentens, N. (eds.) Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12609, pp. 166–180. Springer (2020). https://doi.org/10.1007/978-3-030-68487-7_11, `https://doi.org/10.1007/978-3-030-68487-7_11`
24. Ordas, S., Guillaume-Sage, L., Maurine, P.: EM Injection: Fault Model and Locality. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 3–13. IEEE (2015)
25. Ordas, S., Guillaume-Sage, L., Maurine, P.: Electromagnetic Fault Injection: the Curse of Flip-flops. Journal of Cryptographic Engineering **7**(3), 183–197 (2017)
26. Ordas, S., Guillaume-Sage, L., Tobich, K., Dutertre, J.M., Maurine, P.: Evidence of a Larger EM-induced Fault Model. In: International Conference on Smart Card Research and Advanced Applications. pp. 245–259. Springer (2014)
27. Petersen, E.: Single Event Effects in Aerospace. John Wiley & Sons (2011)
28. Razavi, B.: Fundamentals of Microelectronics. Wiley (2008)
29. Richter-Brockmann, J., Sasdrich, P., Bache, F., Güneysu, T.: Concurrent Error Detection Revisited: Hardware Protection against Fault and Side-Channel Attacks. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–11 (2020)
30. Roscian, C., Sarafianos, A., Dutertre, J.M., Tria, A.: Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 89–98. IEEE (2013)
31. Schellenberg, F., Finkeldey, M., Gerhardt, N., Hofmann, M., Moradi, A., Paar, C.: Large Laser Spots and Fault Sensitivity Analysis. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 203–208. IEEE (2016)
32. Schneider, T., Moradi, A., Güneysu, T.: ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In: Annual Cryptology Conference. pp. 302–332. Springer (2016)
33. Selmane, N., Guilley, S., Danger, J.L.: Practical Setup Time Violation Attacks on AEs. In: 7th European Dependable Computing Conference. pp. 91–96. IEEE (2008)
34. Selmke, B., Brummer, S., Heyszl, J., Sigl, G.: Precise Laser Fault Injections into 90 nm and 45 nm SRAM-Cells. In: International Conference on Smart Card Research and Advanced Applications. pp. 193–205. Springer (2015)
35. Selmke, B., Hauschild, F., Obermaier, J.: Peak Clock: Fault Injection into PLL-Based Systems via Clock Manipulation. In: Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop. pp. 85–94 (2019)
36. Shahmirzadi, A.R., Rasoolzadeh, S., Moradi, A.: Impeccable Circuits II. 57th Annual Design Automation Conference 2020, DAC 2020 **2020** (2020)
37. Skorobogatov, S.: Local Heating Attacks on Flash Memory Devices. In: International Workshop on Hardware-Oriented Security and Trust. pp. 1–6. IEEE (2009)
38. Skorobogatov, S.P., Anderson, R.J.: Optical Fault Induction Attacks. In: International workshop on cryptographic hardware and embedded systems. pp. 2–12. Springer (2002)
39. Wang, F., Agrawal, V.D.: Single Event Upset: An Embedded Tutorial. In: 21st International Conference on VLSI Design (VLSID 2008). pp. 429–434. IEEE (2008)
40. Zussa, L., Dutertre, J.M., Clediere, J., Tria, A.: Power Supply Glitch Induced Faults on FPGA: An In-Depth Analysis of the Injection Mechanism. In: On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International (2013)

## A Additional Schematic

Figure 10 shows the schematic of a xor gate designed in the Nangate 15 open cell technology. The gate consists of four NMOS transistors and four PMOS transistors where the drain regions are marked by red and blue ellipses, respectively.

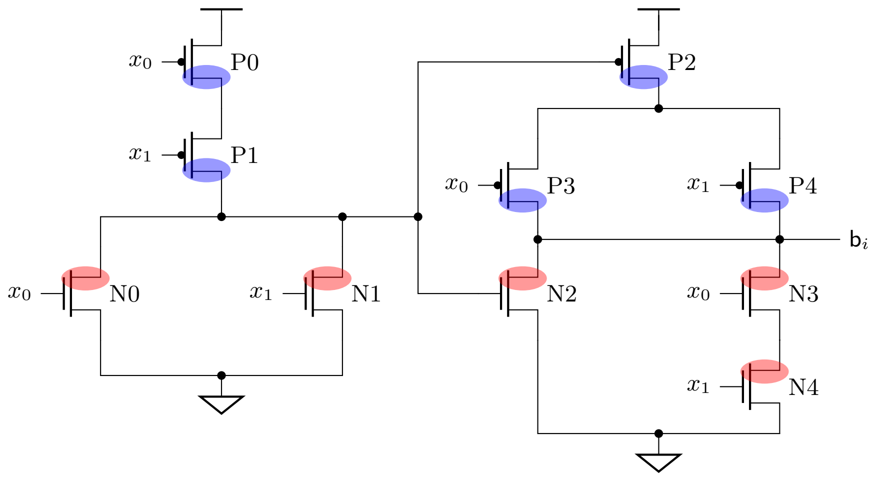


Fig. 10: XOR gate from the Open Nangate 15 technology.

## B Mappings for Nangate 15 Technology

Table 5 states the faults types for the Nangate 15 open cell technology for an attacker that uses LFI. The first column shows the available gates of the technology. The second column indicates the mappings to the functions from $\mathcal{U}$ and $\mathcal{B}$ to describe the possible fault events while the last column states the corresponding Boolean functions behind the functions from $\mathcal{U}$ and $\mathcal{B}$.

Table 5: Fault types for LFI on a Nangate 15 technology.

| | **Mapped Functions** | |
|---|---|---|
| **Gate** | *Functions from $\mathcal{U}$ and $\mathcal{B}$* | *Description* |
| not | $\{\mathsf{u}_2, \mathsf{u}_3\}$ | {set, reset} |
| and | $\{\mathsf{b}_2, \mathsf{b}_6, \mathsf{b}_7\}$ | {or, set, reset} |
| nand | $\{\mathsf{b}_3, \mathsf{b}_6, \mathsf{b}_7\}$ | {nor, set, reset} |
| or | $\{\mathsf{b}_0, \mathsf{b}_6, \mathsf{b}_7\}$ | {and, set, reset} |
| nor | $\{\mathsf{b}_1, \mathsf{b}_6, \mathsf{b}_7\}$ | {nand, set, reset} |
| xor | $\{\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_6, \mathsf{b}_7\}$ | {nand, or, set, reset} |
| xnor | $\{\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_6, \mathsf{b}_7\}$ | {nand, or, set, reset} |

## C Detailed Reports from VerFI Case Study

Table 6: Detailed results of the VerFI case study discussed in Section 5.

| **Fault Model** | **Detected** | **Non-detected** | **Ineffective** | **Scenarios (sum)** |
|---|---|---|---|---|
| $\zeta(1, \tau_{bf}, \mathrm{cs})$ | 40 | 0 | 0 | 40 |
| $\zeta(2, \tau_{bf}, \mathrm{cs})$ | 772 | 0 | 48 | 820 |
| $\zeta(3, \tau_{bf}, \mathrm{cs})$ | 10 652 | 0 | 48 | 10 700 |
| $\zeta(4, \tau_{bf}, \mathrm{cs})$ | 97 428 | 3 598 | 1 064 | 102 090 |
| $\zeta(1, \tau_{s}, \mathrm{cs})$ | 24 | 0 | 16 | 40 |
| $\zeta(2, \tau_{s}, \mathrm{cs})$ | 666 | 0 | 154 | 820 |
| $\zeta(3, \tau_{s}, \mathrm{cs})$ | 9 710 | 0 | 990 | 10 700 |
| $\zeta(4, \tau_{s}, \mathrm{cs})$ | 96 660 | 497 | 4 933 | 102 090 |
| $\zeta(1, \tau_{r}, \mathrm{cs})$ | 16 | 0 | 24 | 40 |
| $\zeta(2, \tau_{r}, \mathrm{cs})$ | 514 | 0 | 306 | 820 |
| $\zeta(3, \tau_{r}, \mathrm{cs})$ | 8 230 | 0 | 2 470 | 10 700 |
| $\zeta(4, \tau_{r}, \mathrm{cs})$ | 87 372 | 49 | 14 669 | 102 090 |
| $\zeta(1, \tau_{sr}, \mathrm{c})$ | 32 | 0 | 32 | 64 |
| $\zeta(2, \tau_{sr}, \mathrm{c})$ | 1 472 | 0 | 576 | 2 048 |
| $\zeta(3, \tau_{sr}, \mathrm{c})$ | 34 752 | 0 | 6 976 | 41 728 |
| $\zeta(4, \tau_{sr}, \mathrm{c})$ | 548 672 | 3 584 | 64 832 | 617 088 |
| $\zeta(1, \tau_{sr}, \mathrm{s})$ | 8 | 0 | 8 | 16 |
| $\zeta(2, \tau_{sr}, \mathrm{s})$ | 92 | 0 | 36 | 128 |
| $\zeta(3, \tau_{sr}, \mathrm{s})$ | 484 | 0 | 92 | 576 |
| $\zeta(4, \tau_{sr}, \mathrm{s})$ | 1 520 | 14 | 162 | 1 696 |
| $\zeta(1, \tau_{nang15}, \mathrm{cs})$ | 76 | 0 | 68 | 144 |
| $\zeta(2, \tau_{nang15}, \mathrm{cs})$ | 7 720 | 0 | 2 520 | 10 240 |
| $\zeta(3, \tau_{nang15}, \mathrm{cs})$ | 405 616 | 0 | 63 824 | 469 440 |
| $\zeta(4, \tau_{nang15}, \mathrm{cs})$ | 14 383 842 | 72 462 | 1 245 232 | 15 701 536 |