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. Fault injection attacks are considered as powerful techniques 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. Given these critical attack vectors, researchers came up with a long list of dedicated countermeasures to thwart such attacks.
However, the security 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, 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 that can be perfectly tailored to existing fault injection mechanisms and their physical behavior in hardware. To demonstrate the advantages, we apply it to a cryptographic primitive and evaluate it based on different attack vectors. We further show that our proposed adversary model can be 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.

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 [KR11], secure implementation of cryptographic primitives in the presence of physical adversaries is still an open challenge, even after two decades of 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 [ADN+10, ZDCT13], electromagnetic pulses [DDRT12, DLM20, OGSM15, OGSM17], or focused photon injection using laser beams [SA02, RSDT13, CLMFT14, SBHS15]. Likewise, many different analysis techniques ranging from Differential Fault Analysis (DFA) [BS97], over Ineffective Fault Attack (IFA) [Cla07] and Statistical Fault Attack (SFA) [FJLT13], to Statistical Ineffective Fault Analysis (SIFA) [DEK+18, DEG+18] has been presented to exploit injected faults. Naturally, different approaches to increase protection against FIA have been proposed at similar pace, mainly following the concepts of redundancy and (concurrent) error detection [SMG16, AMR+19], error correction [SRM20, RBSBG20], or infective computation [GST12].

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 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 environment will ensure appropriate protection of the implementation after deployment.

Currently, a wide range 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 comparison 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 metric for FIA countermeasures that allows fair comparison in adversary capabilities and limitations as well as vulnerabilities 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 [AWMN20] and providing a case study on the lightweight block cipher LED.

Outline: While Section 2 reviews common fault injection mechanisms in detail, Section 3 is dedicated to conception and discussion of our consolidated adversary model. Besides, Section 3 introduces our considered circuit model, states initial assumptions, and introduces the generic but parametric model. In Section 4, we provide practical instantiations of our approach with respect to the mechanisms presented in Section 2. Further, we demonstrate the integration to the 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

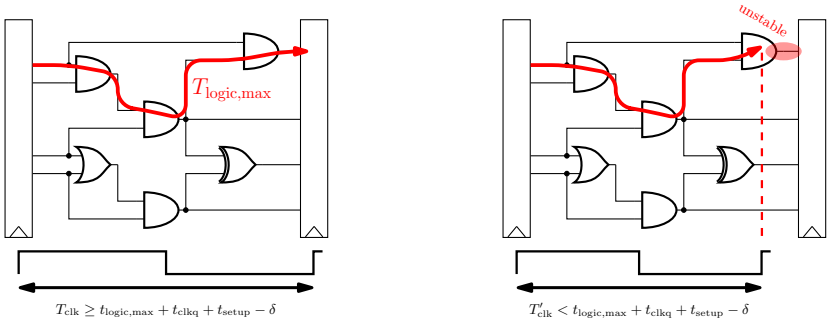
Before we start to summarize and review common fault injection mechanisms in detail, we introduce some notations.

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 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. We survey the most common techniques and explain the fundamental physical mechanisms.



(a) Normal condition. (b) Clock glitch occurrence.

Fig. 1: Physical effects of clock glitches on 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.* [ADN+10] demonstrated its effectiveness using the example of an Advanced Encryption Standard (AES) hardware implementation. Soon thereafter, Endo et *al.* [ESH+11] 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.* [KHEB14] 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.* [SHO19] 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 (since they can be prevented by using the `LOCKED` signal of PLLs as described above) 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 Arrays (FPGAs), allowing to create cost-efficient test setups for countermeasure validation, we opt to cover this mechanism 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_{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, δ denotes the clock skew, $t_{\text{logic,max}}$ the maximum propagation delay of the combinational logic, t_{clkq} the delay of the register, and t_{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'_{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 too short to allow full propagation of the signals through the entire combinational logic and a stabilization of the correct result at the input of the register is not guaranteed. 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.* [SGD08]. 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.* [ZDCT13] 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.

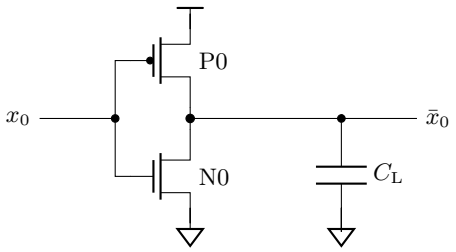


Fig. 2: Transistor-level schematic of a CMOS inverter.

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 [ZDCT13] 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. Exemplary, 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_pLH, as derived in [Raz08], is given by the following equation:

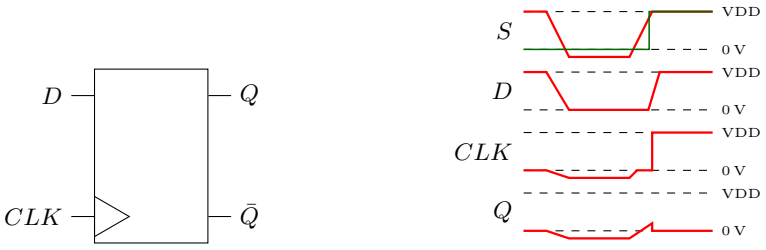
$$t_\mathrm{pLH} = \frac{C_\mathrm{L}\left[\frac{2|V_\mathrm{th,p}|}{V_\mathrm{DD}-|V_\mathrm{th,p}|} + \ln\left(3 - 4\frac{|V_\mathrm{th,p}|}{V_\mathrm{DD}}\right)\right]}{K_p\left(V_\mathrm{DD} - |V_\mathrm{th,p}|\right)}. \tag{2}$$

Here, C_L models the load of connected gates, $V_\mathrm{th,p}$ the threshold voltage of the transistor, and $K_p = \mu_\mathrm{p} C_\mathrm{ox}\frac{W_\mathrm{p}}{L_\mathrm{p}}$ the gain of the PMOS transistor.

Obviously, under a lower supply voltage V_DD, the propagation delay of the inverter t_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 [BKH+19], uses *electromagnetic pulses*. Again, we briefly summarize related state-of-the-art literature and discuss the physical mechanisms responsible for the manifestation of faults.

State of the Art. Over the last years, the understanding of the underlying mechanism of faults caused by Electromagnetic Pulses (EMPs) changed. While in 2012,



(a) D Flip-Flop. (b) Simplified diagram.

Fig. 3: Physical effects due to faults caused by EMPs [DLM19, DLM20].

Dehbaoui et *al.* [DDRT12] performed some experiments on microcontrollers and FPGAs leading to the conclusion that Electromagnetic Fault Injection (EMFI) can be explained by timing violations of the critical path (as for clock glitches), two years later, Ordas et *al.* [OGST+14] 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* [OGSM15,OGSM17]. Most recently, Dumont et *al.* [DLM19,DLM20] were eventually able to explain the physical behavior for the sampling fault model and confirmed its correctness by conducting several 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 Mechanism. Any EMFI setup usually consists of a ferrite core, a coil, and a voltage pulse generator to establish a magnetic field 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 are the power and ground networks, where the induced current leads to a voltage swing S between the power and 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 the 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 rising edge of the voltage pulse generator supplying the EM probe. With the rising edge of S – caused by the second EMP generated through the falling edge of the pulse generator supplying the EM probe – the circuit recovers the original state.

Here, the authors of [DLM20] 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. Fig-

ure 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 [DLM19, DLM20].

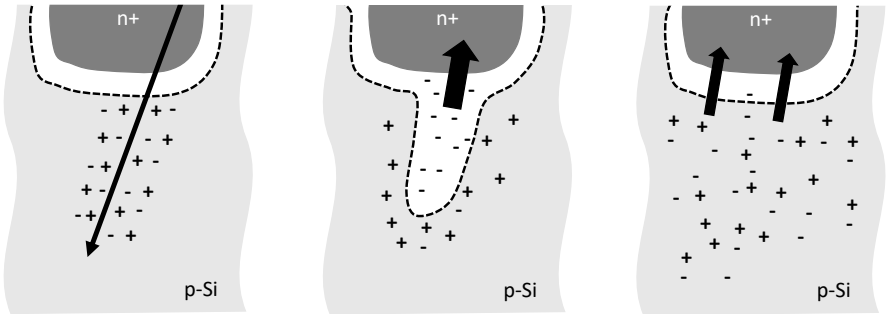
In summary, Dumont et *al.* showed that EMFI causes sampling faults which can also be modeled as set or reset faults in memory elements such as DFFs. Additionally, their most recent work in [DLM20] demonstrates a very fine spacial resolution of EMFI, surprisingly independent of the electromagnetic probe geometry.

Laser Fault Injection. Laser fault injection using focused laser beams was initially presented in 2002, in the seminal work of Skorobogatov and Anderson [SA02]. 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 [SA02] 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.* [RSDT13] already targeted a 250 nm technology and performed investigations on the underlying fault model. In the following year, Courbon et *al.* [CLMFT14] demonstrated the tremendous accuracy of laser fault injection and used it to characterize registers instantiated in a 90 nm technology. Similarly, Selmke et *al.* [SBHS15] investigated the accuracy of laser-induced faults for a 45 nm technology, but concluded that precise fault injections into memory cells become 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.* [SFG+16]. 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.* [DBC+18] 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 geometry size, the basic fundamentals and physical effects can be traced back to the same phenomena.

Physical Mechanism. Figure 4 exemplary 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 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



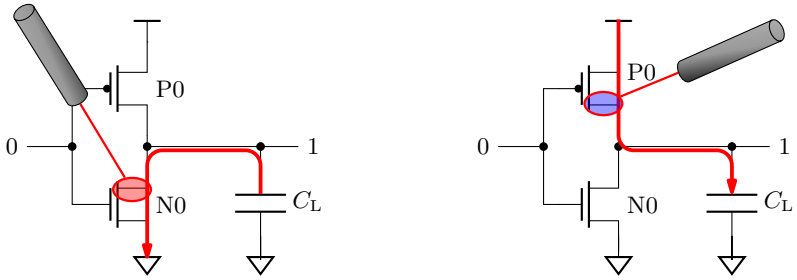
(a) Ionizing. (b) I_{drift}. (c) I_{diff}.

Fig. 4: Physical effect of LFI as introduced in [Bau04].

charge is compensated, resulting in a 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 more details we refer the interested reader to [Bau04, WA08].

As a consequence, the effect of producing a temporary drift current I_{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_{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_{L} is moved such that the output changes from one to zero. Note, 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_{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) [Pet11]). 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_{L} [RSDT13].

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 [Pet11], as this transient fault cannot be recovered while transient faults



(a) NMOS. (b) PMOS.

Fig. 5: Sensitive drain regions for laser fault injection [RSDT13].

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

Miscellaneous Mechanisms Besides the mechanisms introduced above, a few more techniques can be found in the literature, such as body biasing [MTOL12, O'F20], overpowering [CML+11], temperature [Sko09, HS13], and X-Ray beams [ABC+17]. However, they 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, although our versatile concept still allows modeling these mechanisms.

3 Concept

Given the broad range of physical fault injection mechanisms that we surveyed 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. 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 $i, o \geq 1$ where i defines the number of inputs and o the number of outputs of $\mathbf{C}$. 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 *memory gates*, as given in Definition 2.

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	u_0	u_1	u_2	u_3	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	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

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 input assignments only (without any dependency on the history of input assignments).*

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

Hence, *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 memory gates (also called *registers*) as clock-dependent synchronization points that store a single Boolean variable $x \in \mathbb{F}_2$ with $\mathcal{G}_m = \{\mathsf{reg}\}$. Additionally, given the set of combinational gates $\mathcal{G}_c$ and the set of memory gates $\mathcal{G}_m$, we define a set $\mathcal{G} = \mathcal{G}_c \cup \mathcal{G}_m$ to unite all valid gates of a digital circuit $\mathbf{C}$ in a single 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 gate, memory gate, or an input or output and a single edge* $e \in \mathcal{E}$ *represents a wire carrying a digital signal, modeled as an element from the finite field* $\mathbb{F}_2$*.*

Note that this work focuses on synchronous digital logic circuits only, hence, we always assume that inputs, outputs, and memory gates are synchronized to a common clock signal.

3.2 Fault Model

For the description of faults 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 $\mathcal{B} = \{\, \mathsf{b} \mid \mathsf{b} : \mathbb{F}_2^2 \to \mathbb{F}_2 \,\}$.

In general, for a 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 output bits, i.e., we have $|\mathcal{U}| = 4$ unary and $|\mathcal{B}| = 16$ binary functions. Further, the specific assignments of all possible functions for $\mathcal{U}$ and $\mathcal{B}$ are presented in Table 1.

As described before, we consider a limited set $\mathcal{G}_c$ of combinational gates. 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 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 memory gate by a Boolean function from the sets $\mathcal{U}$ or $\mathcal{B}$. For this, we define the following golden mapping τ_{golden} from gate to vertex and associated Boolean function f_g:

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

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

Definition 4 (Fault). *A* fault *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* f_g.

Definition 5 (Error). *In a digital circuit* $\mathbf{C}$ *an* error *occurs if a wrong value is visible at the output of the circuit. An error is always caused by a fault.*

Considering our limited set of combinational and memory gates, a fault 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 registers $g_\mathrm{r} \in \mathcal{G}_m$, faults will only manifest in synchronization to the provided clock signal such that $z' \neq \mathsf{f}_r(x)$.

Moreover, if a fault occurs in a gate of a digital circuit, i.e., the faulted gate evaluates incorrectly, this fault 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. In general, this effect is called *fault propagation*. 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 6 (Fault Scenario). *We define a* fault scenario *as the occurrence of a fault 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 in a target gate $g \in \mathcal{G}$ creates a unique fault scenario for each valid input $\mathbf{x} \in \mathbb{F}_2^i$. Therefore, the input size i, the amount of considered gates, and the number of valid faults for each gate (more details will be given in Section 3.3) determine the total amount of fault scenarios N_{scenario} for a given 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 routing information of the circuit in our fault model since these undermine our attempt to create a generic model. We work on a netlist level which can be perfectly mapped to DAGs as described in Section 3.1. Nevertheless, this abstraction level allows us to attach timing information, i.e., propagation delays of the gates, to each node in the DAG representing the physical logic gates. Third, we do not consider fault probabilities, hence, purely focusing on a quantitative rather than qualitative analysis. 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 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 faults through the entire circuit. Additionally, the structural level provides information about the placement of synchronization points, i.e., the memory gates. This information is important since faults ultimately will manifest in register stages which we will demonstrate in Section 4. However, the actual faults are injected directly in combinational gates $g_{\text{c}} \in \mathcal{G}_c$ or in memory gates $g_{\text{r}} \in \mathcal{G}_m$ where both types of gates describe the functional level of $\mathbf{C}$ through the associated Boolean functions given in τ_{golden} (cf. Section 3.2).

Modeling Faults. On a very abstract (and simplified) level, we model a single fault 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 memory gate $g_\mathrm{r} \in \mathcal{G}_m$ 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 τ_{golden} or a fault type is chosen from a *fault model* τ_{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 provides a generic approach to map various 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 b_{i_0} and b_{i_1} in case a corresponding gate in $\mathbf{C}$ is faulty, we denote the underlying mapping τ_j as $\tau_j : \mathcal{G}_\mathrm{sub} \mapsto \{\mathsf{b}_{i_0}, \mathsf{b}_{i_1}\}$ 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, the second parameter t defines the *type* of the faults. Finally, l limits the circuit *locations* where the faults can occur, i.e., the gates of the circuit that can be targeted by the adversary. In the following we present more details about the three parameters and their design rationals.

Number of Fault Events n: The parameter n sets the total number of faults that can occur at the same time and therefore it constrains 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 memory gates that are available in $\mathbf{C}$.

By 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 $n' \leq n$ faults. This assumption is well established in literature when evaluating countermeasure against fault injection attacks [SMG16, RBSBG20]. However, even if we select n as an upper bound, we still might observe more than n faults manifesting in

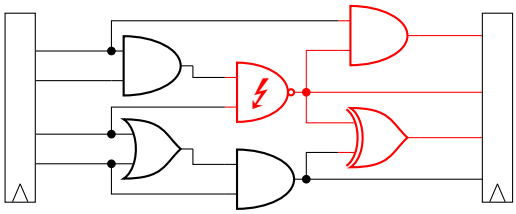


Fig. 6: Influence of a single fault on subsequent gates.

a register stage or primary output due to fault propagation. 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 nand gate in Figure 6. Although n is constrained by 1, the fault can propagate through the and and xor gate such that three errors would eventually manifest at the primary output register stage. Hence, n only indicates the number of faults an attacker is able to inject but it does not give any information about the total number of errors that will occur at the output of the circuit. This behavior was also mentioned by Aghaie et *al.* in [AMR+19].

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 τ_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 τ_{sr} as a fault model where each gate from $\mathcal{G}$ is mapped to the *set* or *reset* function. We decided to use the terms *set* and *reset* instead of *stuck-at-one* and *stuck-at-zero* since the physical fault mechanisms of electromagnetic pulses and laser fault injections cause bit-sets or bit-resets by charging or discharging nodes in the digital circuit. Even if in the presence of clock glitches the fault mechanism could be described by a stuck-at behavior, we stay with the terms *set* and *reset* in the remainder of this work to highlight that the faulty behavior is caused by an active attacker. Note, however, that the choice of terminology does not affect the underlying modeling procedure but both physical mechanisms can be modeled similarly. Particularly, the faulty behavior of a gate $g_u \in \mathcal{G}_u$ or a memory gate $g_r \in \mathcal{G}_m$ is modeled by the function $\mathsf{u}_2(x) = 0$ or by $\mathsf{u}_3(x) = 1$ with $x \in \mathbb{F}_2$ (cf. Table 1). Hence, we apply a mapping that is described by $\{\mathsf{not}, \mathsf{reg}\} \mapsto \{\mathsf{u}_2, \mathsf{u}_3\}$. 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 $\{\mathcal{G}_b\} \mapsto \{b_6, b_7\}$. In essence, the mapping τ_{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.

To allow more fine grained evaluations, we additionally define the mappings τ_s and τ_r which describe only set or only reset fault, respectively. Hence, the mapping τ_r defines $\{\mathsf{not}, \mathsf{reg}\} \mapsto \{\mathsf{u}_2\}$ for unary gates and $\mathcal{G}_b \mapsto \{\mathsf{b}_6\}$ for binary gates. Similarly, τ_s defines $\{\mathsf{not}, \mathsf{reg}\} \mapsto \{\mathsf{u}_3\}$ for unary gates and $\mathcal{G}_b \mapsto \{\mathsf{b}_7\}$ for binary gates. This distinction can be useful for specific primitives or technologies where a fault injection can either cause set or reset faults only. Examples for such primitives are NOR flash memories where only bit-set faults occur as shown and explained in [CMD+19].

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

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

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 τ_{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 τ_{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 $\mathcal{L} = \{\mathrm{c}_i, \mathrm{m}, \mathrm{mc}_i\}$ 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 solely affect combinational logic gates, i.e., gates from $\mathcal{G}_c$. Here, c_∞ considers all combinational gates available in the circuit under test as targets for fault injections. A more fine grained separation of combinational gates is also possible and denoted by the index i while the separation is performed based on the gates' *propagation delays*. More precisely, we exploit that each gate has a specific maximum propagation delay which can be extracted from a CMOS library or for the sake of simplicity be assumed to be the same for each gate. Based on the individual propagation delays of the single gates, we approximately compute the Data Arrival Time (DAT) for each combinational gate $g \in \mathcal{G}_c$ and access the corresponding value by $\mathsf{t}(g)$. Another approach could incorporate the notion of slack (i.e., the difference of the DAT and the Data Required Time (DRT)). However, we decided to rely our model just on the DAT since it is independent of the used clock frequency and can be better applied to actual circuits under test (see Example 3 and Section 4.1). In addition to the computation of the DAT for each gate $g \in \mathcal{G}_c$, we define a set $\mathcal{G}_{\text{regin}}$ that contains all combinational gates driving registers. The DAT of gates $g \in \mathcal{G}_{\text{regin}}$ are used to create an ordered set

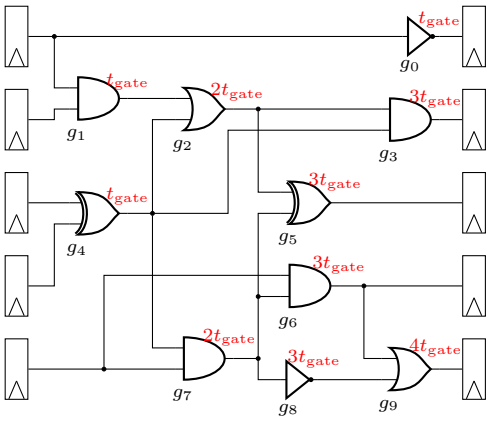


Fig. 7: Propagation delays of different data paths in an exemplary circuit.

$\mathcal{P} = \{t_0, t_1, ...t_{T-1}\}$ where $t_0 > t_1 > ... > t_{T-1}$ and $T \leq |\mathcal{G}_{\text{regin}}|$. This allows us to create clusters of gates defined by

$$\mathcal{G}_{\text{cluster},i} = \{\, g \in \mathcal{G}_{\text{regin}} \mid \mathsf{t}(g) \geq t_i, t_i \in \mathcal{P} \,\}\,. \tag{3}$$

Finally, setting the location parameter $l = \mathrm{c}_i$ corresponds to fault injections that are performed in the subset $\mathcal{G}_{\text{cluster},i}$ with $i < T$. The following example describes the process of generating $\mathcal{G}_{\text{cluster},i}$ based on the circuit given in Figure 7.

Example 3. First, for the sake of simplicity, we assume that each gate in Figure 7 has the same propagation delay t_{gate}. Second, we determine the set of gates whose outputs are connected to registers which is in our example $\mathcal{G}_{\text{regin}} = \{g_0, g_3, g_5, g_6, g_9\}$. The corresponding propagation delays are given by $\mathcal{P} = \{t_0, t_1, t_2\}$ with $t_0 = 4t_{\text{gate}}$, $t_1 = 3t_{\text{gate}}$, and $t_2 = t_{\text{gate}}$. Given this information, we construct the different subsets of $\mathcal{G}_{\text{regin}}$ representing the clusters and containing the following gates

$$\mathcal{G}_{\text{cluster},0} = \{g_9\} \qquad \mathcal{G}_{\text{cluster},1} = \{g_3, g_5, g_6\} \cup \{g_9\}$$
$$\mathcal{G}_{\text{cluster},2} = \{g_0\} \cup \{g_3, g_5, g_6\} \cup \{g_9\}.$$

Besides, setting $l = \mathrm{m}$, specifies fault injections where an attacker targets memory gates $g_r \in \mathcal{G}_m$ only. The location parameter $l = \mathrm{mc}_i$ models faults that can occur in both types of gates where combinational gates can still be separated into subgroups.

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

Table 2: Parameters to accurately model fault injections.

Param.	Options	Description
n	$\mathcal{N} = \{1, 2, ..., N\}$	Maximum number of fault events, N depends on the application
t	$\mathcal{T} = \{\tau_{sr}, \tau_s, \tau_r, \tau_{bf}, \tau_{fm}\}$	τ_{sr}: Fault model for set/reset fault τ_s: Fault model for set faults τ_r: Fault model for reset faults τ_{bf}: Fault model for bit-flips faults τ_{fm}: User-specified fault model
l	$\mathcal{L} = \{\mathrm{c}_i, \mathrm{m}, \mathrm{mc}_i\}$	c_i: Faults in comb. gates only m: Faults in memory gates only mc_i: Faults in all gates

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 ζ 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 Section 4, we evaluate a cryptographic circuit using our generic adversary model to transfer our definitions to a practical instantiation.

4 Practical Instantiation

After we introduced our generic adversary model expressed through the corresponding function ζ 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 ζ, we consider the ASCON S-box [DEMS16] 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, i.e., combinational gates from $\mathcal{G}_c$ and memory gates from $\mathcal{G}_m$. 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.

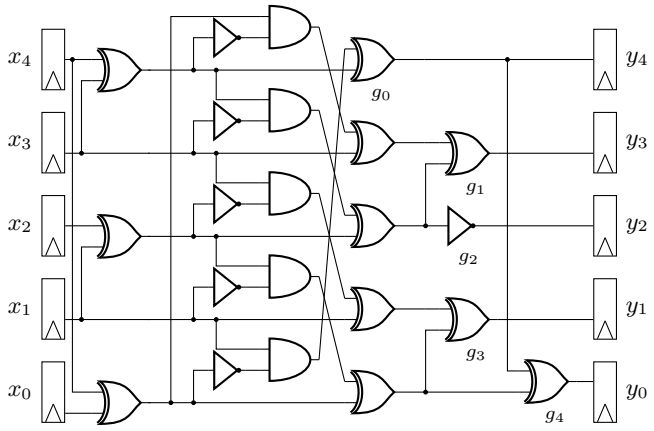


Fig. 8: ASCON S-box [DEMS16].

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

$$N_{\max} = o \cdot N_{\text{scenario}} = o \cdot 2^i \cdot \sum_{g \in \mathbf{C}} |\tau_j(g)| \tag{4}$$

under a considered fault model j.

As indicated in Equation 4, we consider each 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[3] for the considered example to allow a fairer comparison between the different instantiations of ζ 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 this work and still an open research question.

Please be aware, that the following analysis results only hold for an instantiation of the ASCON S-box as shown in Figure 8. Hence, in case the registers of the S-box are removed or any combinational logic is added to the circuit, the analysis has to be re-performed. Consequently, the following examples only demonstrate the functionality of our proposed fault model and the mapping between theory and practice without any claims for generality.

[3] The corresponding source-code can be accessed at `https://nextcloud.seceng.rub.de/index.php/s/7ZmqynqYNnfPLw8`

4.1 Clock Glitches

As first step, in order to model clock glitches and the underlying fault mechanism explained in Section 2.2, we instantiate our adversary model as $\zeta(n, \tau_{sr}, \mathrm{c}_i)$ with $n \in \mathcal{N} = \{1, 2, ..., |\mathcal{G}_{\mathrm{cluster},i}|\}$. By setting $t = \tau_{sr}$ and $l = \mathrm{c}_i$, we consider *set* and *reset* faults occurring in combinational gates g of the cluster $\mathcal{G}_{\mathrm{cluster},i}$ with $0 \leq i \leq T-1$.

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 (negative slack) 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 short clock glitches[4] with $T_{\mathrm{clk}} + \delta < t_{T-1}$ the propagation of the data could be stopped before it stabilizes at any of the outputs of the gate $g \in \mathcal{G}_{\mathrm{regin}}$ such that in a worst case scenario all registers connected to these gates sample wrong results. This corresponds to a fault model instantiated as $\zeta(n, \tau_{sr}, c_{T-1})$ with $n \in \mathcal{N} = \{1, 2, ..., |\mathcal{G}_{\mathrm{cluster},T-1}|\}$. We decided to allow n to be an element from $\mathcal{N}$ and not to fix $n = |\mathcal{G}_{\mathrm{cluster},T-1}|$ in order to provide more fine grained analysis possibilities. Note, however, that the number of faults occurring in one cluster cannot be controlled by an attacker using clock glitches since they solely depend on the previous data processed by the circuit. Due to this behavior, the faults can be modeled best by set and reset faults and assuming that the previous state of each bit was either one or zero. However, the attacker can precisely control the clusters where the faults should occur by adjusting the duration of the clock glitch. This capability is modeled by selecting different c_i which corresponds to fault injections in combinational gates that are included in $\mathcal{G}_{\mathrm{cluster},i}$ with $0 \leq i \leq T-1$. The least number of combinational gates is affected by setting $l = \mathrm{c}_0$ while the maximum number of affected combinational gates is given by $l = \mathrm{c}_\infty$.

For clarification of these decisions, we transfer this model to the ASCON S-box depicted in Figure 8. Instead of assuming the same propagation delay for each gate, we now extract the propagation delays from the Open-Cell Library[5]. Here, we use the slowest propagation delays for an input transition of 0.198 5 ns and a load capacity of 26.016 2 fF for the gates with a driving strength of one. In our example, we need the propagation delays of `xor`-gates, `not`-gates, and `and`-gates which are 0.24 ns, 0.26 ns, and 0.20 ns, respectively. Applying these propagation delays to the ASCON S-box, results in three subsets $\mathcal{G}_0 = \{g_1, g_3, g_4\}$, $\mathcal{G}_1 = \{g_1, g_2, g_3, g_4\}$, and $\mathcal{G}_2 = \{g_0, g_1, g_2, g_3, g_4\} = \mathcal{G}_{\mathrm{regin}}$ with corresponding DATs of $t_0 = 0.118$ ns, $t_1 = 0.96$ ns, and $t_2 = 0.94$ ns that can be considered as targets for fault injections. Hence, considering $l = \mathrm{c}_0$ could lead to

[4] By *short clock glitches* we refer to a clock cycle with a drastically decreased duration (i.e., the time between two rising edges) compared to the cycle duration of the undisturbed clock.

[5] `https://www.cs.upc.edu/~jpetit/CellRouting/nangate/Front_End/Doc/Databook/CornerList.html`

wrongly sampled bits in the registers y_0, y_1, and y_3 while $l = c_2$ could lead to a total of five effective faults manifesting in the registers. Note, however, that any designer should use these values with caution since t_1 and t_2 are very close to each other. Due to process variations in chip manufacturing processes, these values could easily change such that faults in all corresponding gates could occur with the same likelihood. Hence, our model just considers an abstraction of the circuit under test.

Applying the specific adversary model $\zeta(1, \tau_{sr}, \mathrm{c}_0)$ to the ASCON S-box, we consider 2^5 input combinations, three available gates to inject a set or reset fault, and five output registers to observe an error, resulting in a maximum number of $N_{\max} = 960$ faults. 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 $N_{\text{effective}} = 96$ effective faults appearing at the output. Additionally, we instantiate our adversary model as $\zeta(5, \mathrm{sr}, \mathrm{c}_2)$ modeling a worst case scenario where the clock glitch prevents data propagation in all gates $g \in \mathcal{G}_{\text{regin}}$. In this case, there are $N_{\max} = 38\,720$ faults where $N_{\text{effective}} = 13\,824$ are effective.

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 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 $\zeta(n, \tau_{sr}, \mathrm{c}_i)$ adversary with $n \in \mathcal{N} = \{1, 2, ..., |\mathcal{G}_{\text{cluster},i}|\}$.

Obviously, applying the voltage glitch adversary model to our example generates the same results as described in Section 4.1.

4.3 Electromagnetic Pulses

The modeling of fault injections caused by EMPs can be conducted by instantiating the adversary model function as $\zeta(n, \tau_{sr}, \mathrm{m})$ 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 respectively. Hence, setting $t = \tau_{sr}$ perfectly models the physical mechanism of EMFI since the target gate function are mapped to the set or reset function. Additionally, the choice for selecting memory gates as fault location l, matches the recently published results by Dumont et *al.* [DLM19, DLM20] 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

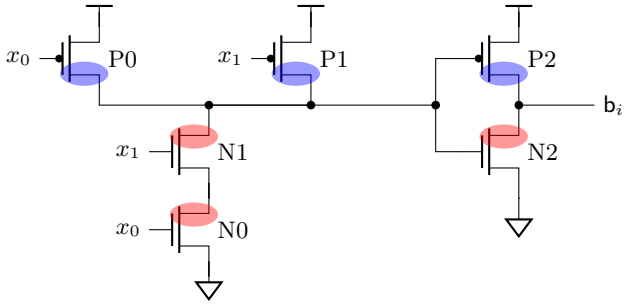


Fig. 9: AND gate from the 15 nm Open-Cell Library. Blue areas mark drain regions of PMOS transistors, red areas mark drain regions of NMOS transistors.

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 $\zeta(1, \tau_{sr}, \text{m})$ adversary model.

4.4 Laser Fault Injection

At last, considering the mechanism of optical fault injections, an appropriate modeling is possible by defining the adversary model function as $\zeta(n, \tau_{fm}, \text{mc}_\infty)$. This selection covers fault injections into combinational and 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 memory gates, the value of a register can either be set or reset which needs to be covered in the custom fault mapping τ_{fm} defining $\{\mathsf{reg}\} \mapsto \{\mathsf{u}_2, \mathsf{u}_3\}$.

Additionally, the instantiation of the adversary function covers faults that directly occur in the combinational logic. Here, we define specified mappings for τ_{fm} between the instantiated gates $g \in \mathcal{G}_c$ and the defined functions in $\mathcal{U}$ and $\mathcal{B}$. 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 and gate designed in the 15 nm Open-Cell Library[6] which is depicted in Figure 9. Therefore, we will call the custom defined mapping τ_{fm} in the following τ_{nang15} as it is tailored to the given example.

The 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 faults on gate level but does not distinguish the number of charged or discharged nodes. However, in case of the considered 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_{drift} into the transistors N0 and N1 to force the gate to behave as an or gate. This is possible if I_{drift} is larger than the current delivered by one of

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

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 τ_{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 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 nand gate or as an 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 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 LFI for the Nangate 15 nm technology, in the appendix in Table 4.

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

4.5 Comparison of Fault Injection Mechanisms

The previous sections illustrate 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 ζ is easily possible. It further demonstrates that the laser fault model $\zeta(n, \tau_{nang15}, \mathrm{mc}_\infty)$ is the most precise but also the most complex instantiation. Assuming that the same n is selected, a design which is evaluated in $\zeta(n, \tau_{nang15}, \mathrm{mc}_\infty)$ and reports no effective faults, will also report no effective faults in the remaining adversary models. The adversary model $\zeta(n, \tau_{nang15}, \mathrm{mc}_\infty)$ covers all set and reset fault in combinational and memory gates which automatically includes all faults modeled by the adversary models for clock glitches, voltage glitches, and EMPs (again assuming the same n). However, this does not hold vice versa so that a design, which for example is evaluated and secure under the EM fault model $\zeta(n, \tau_{sr}, \mathrm{m})$, is not necessarily secure against attackers using optical based fault injection mechanisms.

5 Case Study: Integration into 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 [AWMN20].

VerFI. VerFI is an open-source tool to verify hardware countermeasures against fault injection attacks presented at HOST in 2019 [AWMN20]. 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, stuck-at). Based on this 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[7]. 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 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 [GPPR11] taken from Impeccable Circuits [AMR+19]. The authors published a list of hardware implementations of common block ciphers where each cipher is implemented with different levels of protection[8]. 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

[7] The adapted version of VerFI can be found at `https://nextcloud.seceng.rub.de/index.php/s/7ZmqynqYNnfPLw8`.

[8] `https://github.com/emsec/ImpeccableCircuits`

Table 3: 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)
$\zeta(4, \tau_{bf}, \mathrm{mc}_\infty)$	97 428	3 598	1 064	102 090
$\zeta(4, \tau_{s}, \mathrm{mc}_\infty)$	96 660	497	4 933	102 090
$\zeta(4, \tau_{r}, \mathrm{mc}_\infty)$	87 372	49	14 669	102 090
$\zeta(4, \tau_{sr}, \mathrm{c}_7)$	1 520	14	162	1 696
$\zeta(4, \tau_{sr}, \mathrm{m})$	1 520	14	162	1 696
$\zeta(4, \tau_{nang15}, \mathrm{mc}_\infty)$	14 383 842	72 462	1 245 232	15 701 536

$$p = \texttt{0x0123456789ABCDEF} \qquad k = \texttt{0xDEADBEEFDEADBEEF}$$

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

Table 3 summarizes the evaluation results provided by the adjusted version of VerFI[9]. 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 $\sum_{k=1}^{4} \binom{8}{k} \cdot 2^k$ because we selected for the location parameter $l = \mathrm{c}_7$ which includes all combinational gates with connected outputs to the considered registers. For each combination of k target gates there are 2^k possibilities to combine set and reset faults. Switching to $\zeta(4, \tau_{sr}, \mathrm{m})$ (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 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 registers 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.

[9] Detailed results (for $n < 4$) can be found in Table 5.

We showed that our approach provides the possibility of instantiating a more fine-grained 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 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 considers the physical behavior of 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.

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 τ 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 a user of a fault verification tool would still cover all worst-case fault scenarios by applying a bit-flip model resulting in a lower number 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 NOR flash memory mentioned in Section 3.3. 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 simplified assumptions for e.g., critical paths.

Practical Application Eventually, we discuss the practical application of the proposed adversary model. Particularly, our model is applicable for robustness evaluation, i.e., evaluating the correctness and effectiveness of countermeasures against FIA. Additionally, our approach enables analysis of circuits in a setting with precise adversarial control over the fault injection, i.e., we do not consider random faults and hazards due to environmental effects and conditions (usually considered during safety evaluation in contrast to security analysis). Hence, our presented fault model first and foremost supports the constructive development of robust FIA countermeasures. In particular, the integration into verification tools can assist in correctly, effectively, and efficiently designing protected designs (e.g., [RSS+21]).

The benefits of this approach are manifold. First, analyzing a design requires no setup or expensive equipment (see for example the fault injection setups from Riscure [Ris21] or NewAE [Inc]). Second, no prototyping is required since the circuit under test can be evaluated on a gate level netlist. Third, the development process is faster, cheaper and less error-prone. Fourth, the designer does not need deep expertise in fault injection setups. All together, the application of our proposed adversary model can assist designers in early stages when implementing hardware countermeasures against fault injection attacks.

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 faults, 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.

Acknowledgments

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

References

ABC+17. Stéphanie Anceau, Pierre Bleuet, Jessy Clédière, Laurent Maingault, Jean-Luc Rainard, and Rémi Tucoulou. Nanofocused X-Ray Beam to Reprogram Secure Circuits. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2017.

ADN+10. Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail: On Critical Paths and Clock Faults. In *CARDIS*, pages 182–193. Springer, 2010.

AMR+19. Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Impeccable Circuits. *IEEE Trans. Computers*, 69(3):361–376, 2019.

AWMN20. Victor Arribas, Felix Wegener, Amir Moradi, and Svetla Nikova. Cryptographic Fault Diagnosis using VerFi. In *HOST*, pages 229–240. IEEE, 2020.

Bau04. Robert C Baumann. Soft Errors in Commercial Integrated Circuits. *International Journal of High Speed Electronics and Systems*, 14(02):299–309, 2004.

BKH+19. Arthur Beckers, Masahiro Kinugawa, Yuichi Hayashi, Daisuke Fujimoto, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. Design Considerations for EM Pulse Fault Injection. In *CARDIS*, pages 176–192. Springer, 2019.

BS97. Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, pages 513–525, 1997.

Cla07. Christophe Clavier. Secret External Encodings do not Prevent Transient Fault Analysis. In *CHES*, pages 181–194. Springer, 2007.

CLMFT14. Franck Courbon, Philippe Loubet-Moundi, Jacques JA Fournier, and Assia Tria. Adjusting Laser Injections for Fully Controlled Faults. In *COSADE*, pages 229–242. Springer, 2014.

CMD+19. Brice Colombier, Alexandre Menu, Jean-Max Dutertre, Pierre-Alain Moëllic, Jean-Baptiste Rigaud, and Jean-Luc Danger. Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller. In *HOST*, pages 1–10. IEEE, 2019.

CML+11. Gaetan Canivet, Paolo Maistri, Régis Leveugle, Jessy Clédière, Florent Valette, and Marc Renaudin. Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA. *J. Cryptol.*, 24(2):247–268, 2011.

DBC+18. Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hely, Régis Leveugle, Paolo Maistri, et al. Laser Fault Injection at the CMOS 28 nm Technology Node: an Analysis of the Fault Model. In *FDTC*, pages 1–6. IEEE, 2018.

DDRT12. Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AEs. In *FDTC*, pages 7–15. IEEE, 2012.

DEG+18. Christoph Dobraunig, Maria Eichlseder, Hannes Gross, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical Ineffective Fault Attacks on Masked AES with Fault Countermeasures. In *ASIACRYPT*, pages 315–342. Springer, 2018.

DEK+18. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: Exploiting Ineffective Fault Inductions on Symmetric Cryptography. *CHES*, pages 547–572, 2018.

DEMS16. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. ASCON v1. 2, 2016. `https://competitions.cr.yp.to/round3/asconv12.pdf`.

DLM19. Mathieu Dumont, Mathieu Lisart, and Philippe Maurine. Electromagnetic Fault Injection: How Faults Occur. In *FDTC*, pages 9–16. IEEE, 2019.

DLM20. M Dumont, M Lisart, and P Maurine. Modeling and Simulating Electromagnetic Fault Injection. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

ESH+11. Sho Endo, Takeshi Sugawara, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. An on-chip glitchy-clock generator for testing fault injection attacks. *Journal of Cryptographic Engineering*, 1(4):265, 2011.

FJLT13. Thomas Fuhr, Eliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *FDTC*, pages 108–118. IEEE, 2013.

GPPR11. Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

GST12. Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In *LATINCRYPT*, pages 305–321, 2012.

HS13. Michael Hutter and Jörn-Marc Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS*, pages 219–235. Springer, 2013.

Inc. NewAE Technology Inc. ChipSHOUTER Kit.

KHEB14. Thomas Korak, Michael Hutter, Baris Ege, and Lejla Batina. Clock Glitch Attacks in the Presence of Heating. In *FDTC*, pages 104–114. IEEE, 2014.

KR11. Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011.

MTOL12. Philippe Maurine, Karim Tobich, Thomas Ordas, and Pierre Yvan Liardet. Yet another fault injection technique: by forward body biasing injection. In *YACC'2012: Yet Another Conference on Cryptography*, 2012.

O'F20. Colin O'Flynn. Low-Cost Body Biasing Injection (BBI) Attacks on WLCSP Devices. In *CARDIS*, volume 12609 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2020.

OGSM15. Sébastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine. EM Injection: Fault Model and Locality. In *FDTC*, pages 3–13. IEEE, 2015.

OGSM17. Sébastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine. Electromagnetic Fault Injection: the Curse of Flip-flops. *Journal of Cryptographic Engineering*, 7(3):183–197, 2017.

OGST+14. Sébastien Ordas, Ludovic Guillaume-Sage, Karim Tobich, J-M Dutertre, and Philippe Maurine. Evidence of a Larger EM-induced Fault Model. In *CARDIS*, pages 245–259. Springer, 2014.

Pet11. Edward Petersen. *Single Event Effects in Aerospace*. John Wiley & Sons, 2011.

Raz08. Behzad Razavi. *Fundamentals of Microelectronics*. Wiley, 2008.

RBSBG20. Jan Richter-Brockmann, Pascal Sasdrich, Florian Bache, and Tim Güneysu. Concurrent Error Detection Revisited: Hardware Protection against Fault and Side-Channel Attacks. In *ARES*, pages 1–11, 2020.

Ris21. Riscure. Inspector Fault Injection, Oct 2021.

RSDT13. Cyril Roscian, Alexandre Sarafianos, Jean-Max Dutertre, and Assia Tria. Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells. In *FDTC*, pages 89–98. IEEE, 2013.

RSS+21. Jan Richter-Brockmann, Aein Rezaei Shahmirzadi, Pascal Sasdrich, Amir Moradi, and Tim Güneysu. FIVER - Robust Verification of Countermeasures against Fault Injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):447–473, 2021.

SA02. Sergei P Skorobogatov and Ross J Anderson. Optical Fault Induction Attacks. In *CHES*, pages 2–12. Springer, 2002.

SBHS15. Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise Laser Fault Injections into 90 nm and 45 nm SRAM-Cells. In *CARDIS*, pages 193–205. Springer, 2015.

SFG+16. Falk Schellenberg, Markus Finkeldey, Nils Gerhardt, Martin Hofmann, Amir Moradi, and Christof Paar. Large Laser Spots and Fault Sensitivity Analysis. In *HOST*, pages 203–208. IEEE, 2016.

SGD08. Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. Practical Setup Time Violation Attacks on AEs. In *7th European Dependable Computing Conference*, pages 91–96. IEEE, 2008.

SHO19. Bodo Selmke, Florian Hauschild, and Johannes Obermaier. Peak Clock: Fault Injection into PLL-Based Systems via Clock Manipulation. In *Workshop on Attacks and Solutions in Hardware Security Workshop*, pages 85–94, 2019.

Sko09. Sergei Skorobogatov. Local Heating Attacks on Flash Memory Devices. In *HOST*, pages 1–6. IEEE, 2009.

SMG16. Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In *CRYPTO*, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2016.

SRM20. Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. Impeccable Circuits II. *DAC*, 2020, 2020.

WA08. Fan Wang and Vishwani D Agrawal. Single Event Upset: An Embedded Tutorial. In *21st International Conference on VLSI Design*, pages 429–434. IEEE, 2008.

ZDCT13. Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Assia Tria. Power Supply Glitch Induced Faults on FPGA: An In-Depth Analysis of the Injection Mechanism. In *On-Line Testing Symposium (IOLTS)*, 2013.

A Additional Schematic

Figure 10 shows the schematic of a xor gate designed in the 15 nm Open-Cell Library. The gate consists of four NMOS transistors and four PMOS transistors where the drain regions are marked by red and blue ellipses, respectively.

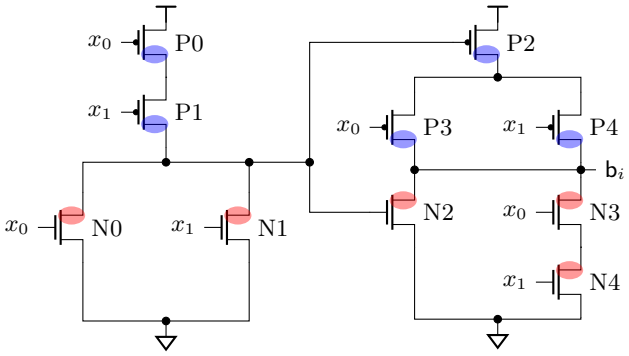


Fig. 10: XOR gate from the Open Nangate 15 technology.

B Mappings for Nangate 15 Technology

Table 4 states the fault types for the 15 nm Open-Cell Library 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 $\mathcal{B}$ while the last column states the corresponding Boolean functions.

Table 4: Fault types for LFI on a Nangate 15 technology.

	Mapped Functions	
Gate	*Functions from $\mathcal{U}$ and $\mathcal{B}$*	*Description*
not	$\{u_2, u_3\}$	{set, reset}
and	$\{b_2, b_6, b_7\}$	{or, set, reset}
nand	$\{b_3, b_6, b_7\}$	{nor, set, reset}
or	$\{b_0, b_6, b_7\}$	{and, set, reset}
nor	$\{b_1, b_6, b_7\}$	{nand, set, reset}
xor	$\{b_1, b_2, b_6, b_7\}$	{nand, or, set, reset}
xnor	$\{b_1, b_2, b_6, b_7\}$	{nand, or, set, reset}

C Detailed Reports from VerFI Case Study

Table 5: Detailed results of the VerFI case study.

Fault Model	Detected	Non-detected	Ineffective	Scenarios (sum)
$\zeta(1, \tau_{bf}, \mathrm{mc}_\infty)$	40	0	0	40
$\zeta(2, \tau_{bf}, \mathrm{mc}_\infty)$	772	0	48	820
$\zeta(3, \tau_{bf}, \mathrm{mc}_\infty)$	10 652	0	48	10 700
$\zeta(4, \tau_{bf}, \mathrm{mc}_\infty)$	97 428	3 598	1 064	102 090
$\zeta(1, \tau_{s}, \mathrm{mc}_\infty)$	24	0	16	40
$\zeta(2, \tau_{s}, \mathrm{mc}_\infty)$	666	0	154	820
$\zeta(3, \tau_{s}, \mathrm{mc}_\infty)$	9 710	0	990	10 700
$\zeta(4, \tau_{s}, \mathrm{mc}_\infty)$	96 660	497	4 933	102 090
$\zeta(1, \tau_{r}, \mathrm{mc}_\infty)$	16	0	24	40
$\zeta(2, \tau_{r}, \mathrm{mc}_\infty)$	514	0	306	820
$\zeta(3, \tau_{r}, \mathrm{mc}_\infty)$	8 230	0	2 470	10 700
$\zeta(4, \tau_{r}, \mathrm{mc}_\infty)$	87 372	49	14 669	102 090
$\zeta(1, \tau_{sr}, \mathrm{c}_7)$	8	0	8	16
$\zeta(2, \tau_{sr}, \mathrm{c}_7)$	92	0	36	128
$\zeta(3, \tau_{sr}, \mathrm{c}_7)$	484	0	92	576
$\zeta(4, \tau_{sr}, \mathrm{c}_7)$	1 520	14	162	1 696
$\zeta(1, \tau_{sr}, \mathrm{m})$	8	0	8	16
$\zeta(2, \tau_{sr}, \mathrm{m})$	92	0	36	128
$\zeta(3, \tau_{sr}, \mathrm{m})$	484	0	92	576
$\zeta(4, \tau_{sr}, \mathrm{m})$	1 520	14	162	1 696
$\zeta(1, \tau_{nang15}, \mathrm{mc}_\infty)$	76	0	68	144
$\zeta(2, \tau_{nang15}, \mathrm{mc}_\infty)$	7 720	0	2 520	10 240
$\zeta(3, \tau_{nang15}, \mathrm{mc}_\infty)$	405 616	0	63 824	469 440
$\zeta(4, \tau_{nang15}, \mathrm{mc}_\infty)$	14 383 842	72 462	1 245 232	15 701 536