Coupling Leakage in Theory and Practice

Unveiling (Post-PnR) Security Flaws in Masked FPGA-Mapped Designs

Nicolai Müller[1], Daniel Lammers[2], Simon Osterheider[2] and Amir Moradi[1]

[1] Technische Universität Darmstadt, Darmstadt, Germany
{nicolai.mueller1,amir.moradi}@tu-darmstadt.de
[2] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
{daniel.lammers,simon.osterheider}@rub.de

Abstract. With the widespread adoption of Field Programmable Gate Arrays (FPGAs) in security-critical industries such as defense and telecommunications, ensuring the confidentiality of sensitive data processed by these devices has become paramount. Side-Channel Analysis (SCA) poses a significant threat, necessitating the protection of cryptographic primitives through effective and efficient countermeasures. Within the framework of well-established formal adversary models, Boolean masking offers provable resistance to SCA by randomizing sensitive intermediate computations using Shamir's secret sharing. However, the secure and efficient implementation of Boolean masking in hardware presents a complex and error-prone challenge. Physical effects such as glitches, transitions, and couplings can undermine essential security assumptions, potentially weakening the effectiveness of masking. Although robust masking schemes are designed to maintain their security in the presence of glitches and transitions, and their secure implementation can be verified at the gate level using a wide range of automated verification tools – albeit mostly for Application-Specific Integrated Circuits (ASICs) and not FPGAs – leakages caused by coupling effects remain undetectable by such tools, as they occur at a lower abstraction level.
In this work, we focus on detecting (low-level) vulnerabilities in masked FPGA designs through experimental and tool-assisted evaluation. The flaws that we target are not detectable at the Register Transfer Level (RTL) level, and in some cases, not even at the gate level, as they arise from optimizations introduced during synthesis or implementation, or from the specific outcomes of the Place and Route (PnR) process. We demonstrate that these flaws are not only theoretically concerning, but can lead to observable leakages in practical experiments. Furthermore, we show how to formally abstract, unveil, and mitigate such leakages, thereby enabling a security-aware FPGA design flow that spans from the behavioral to the physical level. As an example, we implement all the evaluation steps presented as an extension of `PROLEAD` and validate its accuracy and effectiveness through practical case studies.

Keywords: Coupling · Evaluation · FPGA · Hardware · Leakage · Masking · Probing Security · Side-Channel Analysis · Tools

1 Introduction

The masking countermeasure proposed by Chari et al. [CJRR99] offers provable security against SCA by encoding secret variables using multiple shares [Sha79]. However, securely and efficiently implementing masking in hardware, crucial for translating its theoretical security guarantees into practical protection, remains a challenge. In this context, proofs under formal adversary models provide evidence of a masked design's security by verifying

that different shares leak independently. Specifically, the d-probing model assesses the independence of the shares by requiring that all d-tuples of intermediate variables remain independent of any sensitive data [ISW03]. A masking scheme is considered secure under this model only if it satisfies this condition. Over the years, numerous hardware masking schemes have been developed and verified using the d-probing model, with the goal of gradually reducing the area and performance overhead. However, some have been deemed insecure in practice, primarily due to unforeseen physical defaults that compromise the independence of shares [MPG05, CGP+12, MMSS19, MKSM22, LMMS23, MLM24]. In particular, glitches have been shown to violate share independence, particularly when multiple shares are involved in the same combinational computations [MPG05]. Additionally, signal transitions are well known to compromise share independence, especially in iterative designs where a single gate evaluates multiple times across different clock cycles [CGP+12, MKSM22]. Since the impact of these physical defaults is often difficult to predict with precision, the research community addresses glitches and transitions in a more generic way through two distinct approaches. First, the robustness of security models has been strengthened by considering stronger adversaries who gain additional knowledge from leakages caused by physical effects, primarily glitches, transitions, and couplings. Specifically, the robust d-probing model extends the probing model by incorporating all additional signals revealed through those effects into each d-tuple [FGP+18]. This approach allows for capturing security violations caused by glitches and transitions in the worst-case scenario, but it also increases the complexity of the security model, making it more inconvenient to apply to complex masking schemes that process a high number of shares. Second, proving security under the robust probing model has been facilitated by focusing on proving the secure composability of small masked building blocks, known as gadgets. The simplicity of these gadgets allows for straightforward proofs of their composability and they can be combined to construct more complex masked circuits. The security of these circuits can then be reduced to the composability of the individual gadgets. Today, the impact of glitches and transitions is well understood in the design of masked ASICs. Many gadgets and custom masked designs have been extensively analyzed for their security under the robust d-probing model. To this end, various verification tools have been developed to automatically evaluate the security and composability of arbitrary designs, provided either as high-level descriptions or gate-level netlists, within the robust d-probing model [BBFG18, BGI+18, KSM20, BMRT22, ZCF25, BFG+25]. However, the security of masked designs on FPGAs has not been thoroughly investigated in the literature, and two primary threats can be identified. At a high level, the security of masking can be compromised by optimizations applied during the FPGA design flow such as Lookup Table (LUT) combining or register retiming. Furthermore, even designs whose concrete FPGA netlists (after implementation) can be formally proven secure with respect to glitches and transitions may still exhibit vulnerabilities, as demonstrated by Test Vector Leakage Assessment (TVLA) performed on measured power-consumption traces. The reason for this lies in previously overlooked physical defaults at lower abstraction levels, such as coupling leakages. The robust probing model can account for coupling leakages but they are often neglected in theoretical security analyses and tool-assisted verifications. This is because investigating coupling leakages requires additional placement and routing information, whereas verifying robust probing security through tools is typically performed at the gate level or higher abstraction levels. Consequently, leakages that can only be detected at lower levels of abstraction remain unnoticed and may ultimately manifest in deployed FPGA-based products.

1.1 Related Work

In 2017, De Cnudde et al. demonstrated that coupling effects can lead to potentially exploitable leakages on FPGAs by implementing a 32-bit KATAN cipher [CDK09] with

Threshold Implementations (TI) [NRR06] and deliberately forcing the placer to position gates operating on different shares in close proximity [CBG+17]. By means of physical experiments, i.e., TVLA, they were able to detect leakage even though the design is considered secure in the presence of glitches. This work effectively highlights the threat of coupling to the security of masked implementations but the placement and routing process was constrained specifically to make coupling leakage observable. The same applies to other studies that relied on external manipulations of the measurement setup to induce or amplify coupling effects [LBS19]. However, due to timing and placement constraints, coupling leakages may also occur unintentionally, as demonstrated in this work.

To the best of our knowledge, the only existing verification tool capable of assessing the probing security of an FPGA design, such as one represented as an FPGA netlist, is the Fenix tool proposed by Gigerl et al. [GPM24]. Fenix adopts the same approach as Rebecca [BGI+18], computing correlation sets for each circuit gate according to specific propagation rules. These sets are then encoded into a SAT formula, which is subsequently solved by a corresponding SAT solver. Fenix can detect leakages under the basic d-probing model (the stable mode in [BGI+18]) as well as under the robust d-probing model that accounts solely for glitches (the transient mode). This capability enables the detection of security violations introduced during the synthesis or implementation process, such as LUT combining or register retiming. However, the joint consideration of glitches and transitions is not supported, and Fenix is limited to evaluating relatively small designs, typically individual gadgets or small S-boxes, at low security orders.

1.2 Our Contributions

In this work, we propose a security-aware design flow for developing masked designs that are secure both in theory, validated through tool-assisted evaluations, and in practice, as demonstrated by physical measurements on the FPGA. Our contributions are as follows.

- We show that observable leakage can occur unintentionally on FPGAs by performing TVLA on a single Advanced Encryption Standard (AES) S-box instance protected with first-order Domain-Oriented Masking (DOM). In contrast to [CBG+17], we do not control or force the PnR process. The TVLA results indicate general information leakage, which we attribute to coupling effects.
- We extend `PROLEAD` to support the evaluation of masked FPGA netlists based on LUTs and Flip-Flops (FFs). Using this extension, we confirmed the $(1, 1, 0)$-robust 1-probing security of the previously analyzed DOM AES S-box on which the TVLA was performed. This evaluation is computationally infeasible for Fenix in transient mode and Fenix does not jointly account for glitches and transitions.
- We present a generic approach to incorporate coupling during tool-assisted evaluation of masked FPGA designs. Importantly, we emphasize that our method does not require access to the full physical design of the target FPGA, which is typically unavailable for proprietary FPGAs. Next, we extend `PROLEAD` with our approach and apply it to the design evaluated in our initial case study to identify the sources of coupling leakage.
- Based on the leakages reported by `PROLEAD`, we demonstrate how to mitigate coupling leakages by re-placing and re-routing vulnerable parts of the design. Finally, `PROLEAD` reports the resulting design as secure (including glitch, transition, and coupling-related leakages), and TVLA on the same FPGA does not show any leakages.

The extended version of `PROLEAD` is released as an open-source contribution and can be accessed publicly via GitHub[1].

[1] https://github.com/ChairImpSec/PROLEAD

2 Background

2.1 Boolean Masking

Boolean masking defines the masking countermeasure against SCA applied in the binary field $\mathbb{F}_2$ [CJRR99]. Boolean masking applies Shamir's secret sharing [Sha79] to encode any sensitive variable $x \in \mathbb{F}_2$ into a uniform $(d+1)$-sharing $\{x^0, \ldots, x^d\} \in \mathbb{F}_2^{d+1}$ such that $x = x^0 \oplus \ldots \oplus x^d$. Usually, all x^i with $i \in [0, d)$ are sampled uniformly at random, while x^d is computed as $x^d = x \oplus x^0 \oplus \ldots \oplus x^{d-1}$. Hence, no $X \subset \{x^0, \ldots, x^d\}$ reveals any information on x. To apply Boolean masking to a cryptographic primitive, all sensitive operations must be adjusted in order to operate on the sharing of each variable instead of the variable itself. Specifically, all computations must be performed on an incomplete subset of shares for every sensitive variable involved.

2.1.1 Domain-Oriented Masking (DOM)

The DOM scheme employs a fixed number of $d+1$ shares to mask any Boolean function [GMK16]. In this approach, non-linear computations are performed using a dedicated DOM multiplier that computes a Boolean masked version of $z = x \cdot y$ over an arbitrary finite field $\mathbb{F}_q$, while all linear operations are performed share-wise. To illustrate, consider a DOM multiplier operating on two shares per variable, i.e., inputs $(x^0, x^1) \in \mathbb{F}_q^2$ and $(y^0, y^1) \in \mathbb{F}_q^2$, and producing the output $(z^0, z^1) \in \mathbb{F}_q^2$. Initially, the multiplier computes four intermediate product terms (s_0, s_1, s_2, s_3), where the cross-domain terms, those combining shares of different indices, are additionally masked using a randomly sampled variable $r \in \mathbb{F}_q$. The computations are defined as follows:

$$s_0 = x^0 y^0, \qquad s_1 = x^0 y^1 \oplus r, \qquad s_2 = x^1 y^0 \oplus r, \qquad s_3 = x^1 y^1. \tag{1}$$

All intermediate terms s_i, $i \in [0, 3]$, are synchronized using a register stage. Finally, the output shares are computed based on the synchronized intermediates as $z^0 = s_0 \oplus s_1$ and $z^1 = s_2 \oplus s_3$. Based on this concept, Gross et al. proposed a masked version of the Canright AES S-box [Can05], illustrated in its two-share variant in Figure 1. The

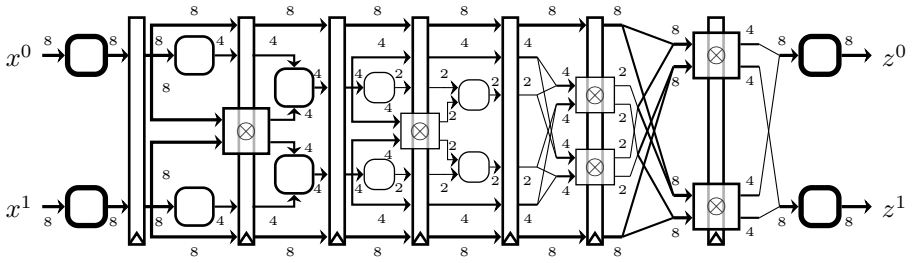


Figure 1: First-order DOM-masked AES S-box with pipelining registers.

S-box consists of several linear mappings, represented by rectangles with rounded corners, which are implemented separately for each share. Since the Canright S-box employs a tower-field decomposition to perform operations in smaller finite fields than $\mathbb{F}_{2^8}$, the non-linear computations are realized using DOM multipliers over the fields $\mathbb{F}_{2^2}$ and $\mathbb{F}_{2^4}$ highlighted by the $\otimes$-symbol while the field is indicated by the bus widths.

2.2 Robust Probing Security

The d-probing model defines an adversary capable of placing up to d probes on wires of the circuit [ISW03]. Each probe grants access to the stable signal carried by the probed

wire at a specific point in time. Then, a masked circuit is considered d-probing secure if no adversary can extract any sensitive information. Specifically, for any sensitive variable x, no adversary with access to at most d intermediates of the circuit can gain any knowledge about x. As the d-probing model assumes a circuit to be ideal, i.e., one in which no physical default can manifest, the (g, t, c)-robust d-probing model extends the d-probing model by incorporating physical defaults, namely glitches ($g \in \mathbb{F}_2$), transitions ($t \in \mathbb{F}_2$), and couplings ($c \in \mathbb{N}$) [FGP+18]. Specifically, this extension is achieved by substituting each probe with a set of additional probes based on the targeted physical effect. In the following, we discuss these effects in detail and outline the corresponding probe-extension procedures. The (g, t, c)-robust d-probing model can be specified depending on the physical effects which should be considered. For example, the $(1, 0, 0)$-robust d-probing model refers to the specification that considers only glitches, excluding transitions and couplings, whereas the $(1, 1, 0)$-robust d-probing model accounts for joint leakages caused by both glitches and transitions.

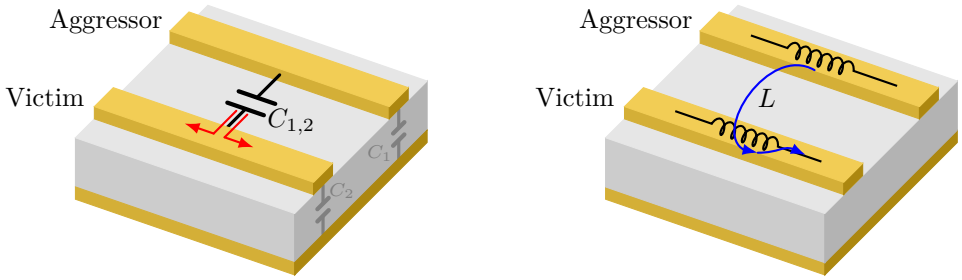
Glitches. Glitches arise from unequal data arrival times in combinational logic, leading certain parts of the circuit to process a new input before all other inputs have stabilized. In such cases, particular components of the combinational logic can produce a series of transient toggles before settling on their final stable values. We refer to these unexpected transient toggles as glitches. Glitches are data-dependent and can propagate through the entire combinational logic, as a glitchy signal can propagate to subsequent combinational logic blocks. Hence, glitches can violate the independent leakage assumption [CPR07], required for any secure masking scheme [MPG05]. The robust d-probing model accounts for glitches in a worst-case manner, assuming that a glitchy signal carries information about all intermediate values within the combinational logic required to compute that signal. As a result, a probe placed on a wire is glitch-extended by replacing it with probes on all inputs of the combinational logic that contribute to the probed wire, but only within the same clock cycle.

Transitions. When the state of a wire or memory element changes, the instantaneous power consumption depends on both the previous and the new state. Consequently, information about both states can leak to an adversary through so-called transitional leakage [CGP+12, BGG+14]. The robust d-probing model captures this behavior by substituting each probe with two probes, recording on the same wire for two consecutive clock cycles.

Couplings. Coupling refers to the physical interaction between two unconnected wires, where the voltage levels of one wire influence the voltage levels of the other due to various coupling effects. This interaction can lead to unintended information leakage between wires, even if they are not physically connected [CBG+17]. While multiple coupling effects exist, we will focus solely on crosstalk, as it is, to the best of our knowledge, the most relevant coupling effect influenced by wire proximity. There are two primary types of crosstalk relevant to our analysis: (1) capacitive coupling and (2) inductive coupling. In both cases, a change in the driven signal on an aggressor line, such as during a transition from zero to one or vice versa, induces an unwanted voltage change on a victim line.

Capacitive coupling arises from the electric field surrounding a wire. A voltage change on the aggressor line directly affects its electric field, which causes a displacement current due to the change in electric flux. These variations lead to unintended coupling between the two wires in the form of induced voltage pulses on the victim wire. In response, a small current flows along the victim line, which can be approximated and effectively modeled as a parasitic capacitance between the adjacent conductors, as illustrated in Figure 2a.

On the other hand, inductive coupling occurs due to changes of the magnetic field generated around a wire, as seen in Figure 2b. This magnetic field also follows from the current flowing through the aggressor line. When this current changes, the varying magnetic field induces a voltage in nearby (victim) conductors. This induced voltage, in turn, drives a current in the victim line, leading to inductive crosstalk between the two wires. This effect can be modeled as a parasitic mutual inductance between the aggressor and victim conductors. The robust d-probing model covers coupling leakages



(a) Capacitive Crosstalk.

(b) Inductive Crosstalk.

Figure 2: Overview of coupling mechanism types causing crosstalk.

by introducing a parameter c, where each probe is substituted with a set containing the original probe along with c additional probes placed on adjacent wires. These additional probes record signals during the same clock cycle as the original probe, capturing potential coupling effects. Hence, in contrast to glitches and transitions, couplings are not considered in a worst-case manner, but depending on c. This decision accounts for the fact that assuming coupling between all wires would render every masked design insecure.

2.3 Field Programmable Gate Arrays (FPGAs)

FPGAs are reconfigurable integrated circuits that can be repeatedly programmed to implement different hardware designs. Unlike ASICs, which are custom-designed for a certain use case and fixed after fabrication, FPGAs are designed to accommodate a wide range of digital circuits within the limits of their (physically) available logic, memory, I/O and routing resources. This allows reprogramming and updates, offering significantly greater flexibility at the cost of efficiency.

Architecture. Although specific FPGA platform architectures vary between vendors, they all share a similar fundamental layout approach and a common essential set of components, as illustrated in Figure 3. FPGAs are usually internally arranged in a grid-like structure of CLBs, each composed of several LUTs, FFs, and interconnection wires. LUTs, which essentially consist of multiplexers combined with SRAM cells that store output values for all possible input vectors, act as the fundamental computational units of FPGAs. They can be configured to compute arbitrary logical functions, while FFs serve as memory elements for storing state information. To enable connectivity between different CLBs, they are surrounded by programmable interconnects organized into so-called switch boxes. Each switch box consists of multiple PIPs, i.e., programmable multiplexers that can establish connections between wires passing through the switch box, allowing signals to be routed to other CLBs or switch boxes. A programmed PIP, illustrated by the dashed line in Figure 3, therefore represents an interconnection wire linking an input wire of the switch box to a corresponding output wire.

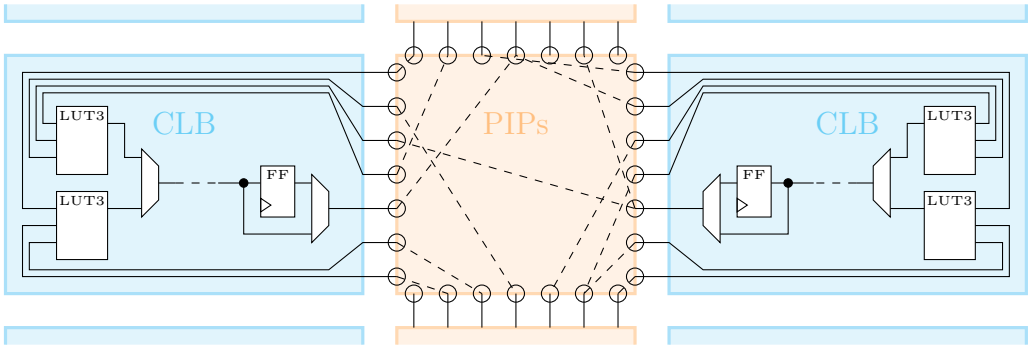


Figure 3: Excerpt of a generic FPGA layout with Configurable Logic Blocks (CLBs) in blue and switch boxes in orange, containing Programmable Interconnect Points (PIPs).

Design Flow. The FPGA design flow begins with writing the application logic in a Hardware Description Language (HDL). During logical synthesis, the HDL code is compiled into a netlist of LUTs and FFs, representing the desired functionality. These elements are then mapped to the available component types within the specific FPGA platform. The subsequent implementation phase consists of placement and routing. During placement, each component from the netlist is assigned a physical location on the FPGA. In the routing phase, the connections between the logical components are established, including the selection of appropriate PIPs. Finally, the bitstream is generated, which can be loaded onto the FPGA to configure it according to the specified design.

2.4 Automated Tools for Leakage Evaluation

Over the years, numerous tools have been proposed for the evaluation of masked hardware circuits [BBFG18, BGI+18, KSM20, BMRT22, GPM24, ZCF25, BFG+25]. These tools employ different approaches to evaluate specific security and composability notions under the robust d-probing model, but they differ in terms of capabilities, abstraction levels, and computational efficiency. However, all of them perform an exhaustive evaluation, analyzing every possible input vector of the design to eliminate the risk of incorrectly classifying an insecure design as secure. This approach is particularly useful for evaluating the security and composability of small circuits, such as gadgets, but it becomes computationally infeasible for larger designs, such as complex S-boxes or a complete cipher implementation.

`PROLEAD`. `PROLEAD` is an exception among the tools mentioned above, as it employs an incomplete evaluation approach rather than exhaustive verification [MM22]. By adjusting the number of input vectors considered, this approach allows the evaluation of larger masked designs at higher security orders, while maintaining a low and quantifiable – but nonzero – risk of incorrectly classifying an insecure design as secure. Specifically, `PROLEAD` follows the procedure outlined below:

1. `PROLEAD` represents a circuit $C = (V, E)$ as a directed graph, where the vertices in V correspond to combinational and sequential gates, and the edges in E represent wires that carry elements in $\mathbb{F}_2$. Additionally, every wire is annotated with a time instance $(e, t) \in E \times T$ while T denotes the set of clock cycles during which the circuit operates. Each tuple (e, t) is treated as a potential location where a robust d-probing adversary can place a probe. To account for this, `PROLEAD` precomputes all possible extensions of every probe on $(e, t) \in E \times T$ and stores it in two separate lists. The first list $\mathcal{G}$ stores all sets of probes obtained through glitch extensions on a specific wire. Thus, $\mathcal{G}[e, t] = \{(e_0, t), \dots (e_{n-1}, t)\}$ represents the set of wires

an adversary can probe due to glitches when placing a probe on (e, t). Similarly, $\mathcal{T}[e, t] = \{(e, t), (e, t-1)\}$ represents the tuple of wires an adversary can probe due to transitions when placing a probe on (e, t).

2. Next, `PROLEAD` constructs all robust d-probing adversaries by generating every possible d-tuple of probe locations and extending each probe through glitches and transitions. Specifically, for every probe placed on (e, t), the corresponding set $\mathcal{G}[e, t] = \{(e_0, t), \dots (e_{n-1}, t)\}$ is determined. The final extension set is then formed by merging all sets $\mathcal{T}[e_i, t] \forall i \in [0, n)$.

3. `PROLEAD` simulates the circuit using randomly sampled inputs from different groups, such as in a fixed-vs-fixed or fixed-vs-random setting, and extracts the observations made by various adversaries by collecting the signals recorded by the probes. The evaluation is then performed using a statistical hypothesis test to determine whether the distributions made by different groups differ. If a difference is detected, `PROLEAD` identifies and reports all adversaries that violate robust probing security.

3 Motivation

Initially, we demonstrate the threat of coupling leakage in FPGAs by showing that it can occur without manual adjustments to PnR, as done in [CBG+17]. This also highlights that coupling leakage can potentially affect many masked designs on FPGAs.

3.1 Target Design

We target a single first-order masked AES S-box design protected with DOM as discussed in [GMK16, Section 4.1]. The reference code is publicly available on GitHub[2] while we instantiate the 8-stage pipelined version of the S-box, visualized in Figure 1, which has a total latency of 9 clock cycles and requires 18 fresh random bits per execution to be $(1, 0, 0)$-robust 1-probing secure. However, in this version, the DOM AES S-box achieves robust 1-probing security only when considering glitches but not transitions. By refreshing the masks at every clock cycle, we ensure that all consecutively stored register values are mutually independent, which corresponds to the bubble-strategy as explained in [CS21]. Therefore, the design is $(1, 1, 0)$-robust 1-probing secure. The S-box receives a masked 8-bit input, represented as two shares. We synthesized and implemented the design using Vivado 2025.2. We note that the PnR of the design is not manually controlled; instead, all implementation settings are set to default. However, we adjusted the default synthesis settings to disable cross-boundary optimizations which is necessary to separate different share domains. Further, we disabled LUT combining and register retiming during the synthesis and implementation steps[3].

3.2 Measurement Setup

We measured the dynamic power consumption of our target design on the AMD (Xilinx) Kintex-7 FPGA of the SAKURA-X, a commercially available SCA evaluation board. Specifically, we continuously recorded the voltage drop across a 1Ω shunt resistor, inserted into the Vdd path of the target FPGA, using a PicoScope 6000 digital oscilloscope at $625\,MS/s$. The target design operated on a stable $6\,MHz$ clock signal, minimizing the

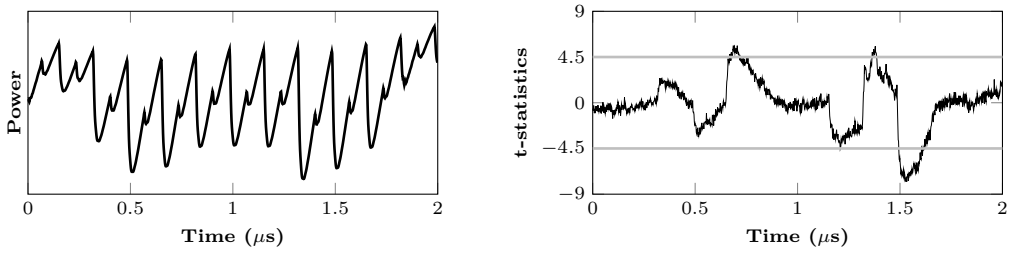
[2] https://github.com/hgrosz/aes-dom

[3] For AMD (Xilinx) FPGAs, cross-boundary optimizations are disabled by setting `flatten_hierarchy` to `none`, LUT combining is turned off with `-no_lc`, and register retiming is disabled via `-global_retiming off` in Vivado. Manual inspection of the resulting implemented design confirms that these settings take effect.

impact of adjacent clock cycles on each other through power traces. In addition, we used a Mini-Circuits ZFL-1000LN+ RF amplifier to increase the vertical resolution of the measured power traces. All masked inputs are generated on the AMD (Xilinx) Spartan-6 control FPGA on the SAKURA-X. Specifically, the control FPGA generates two uniformly random shares, which are then sent to the target FPGA. All additional randomness is generated on the target FPGA by independent and randomly-seeded Linear Feedback Shift Registers (LFSRs) as proposed in [WMM20]. For the analysis, we utilize the state-of-the-art TVLA technique [GJJR11] in a non-specific fixed-vs-random setting to investigate general information leakage. In this setup, the target FPGA receives the shared variant of either a fixed or a random input.

3.3 Results

The result of the leakage assessment performed over 100 million measurements is illustrated in Figure 4 and shows observable leakage in the first security order (cf. Figure 4b).



(a) The mean trace.

(b) 1st-order TVLA.

Figure 4: TVLA results of a DOM AES S-box over 100 million measurements, automatically placed and routed.

Based on the observed information leakage, the question arises as to its underlying source. One possible explanation would be that optimizations applied during synthesis or implementation introduced flaws into the masking scheme. However, since we disabled LUT combining and register retiming, these issues would only arise if Vivado failed to enforce the corresponding settings. Another explanation is information leakage caused by coupling, especially since the masked design is assumed to be $(1, 1, 0)$-robust 1-probing secure, which excludes potential leakages caused by coupling effects. However, the TVLA results in Figure 4 alone cannot confirm this hypothesis. Moreover, the TVLA results offer no insight into the nature or location of potential coupling-based leakage, such as which wires may have been routed in close physical proximity. However, we observe that the leakage coincides with the activity of the non-linear components during the last three clock cycles, where achieving secure routing is challenging due to the large number of wires that must be arranged in a complex and intertwined manner. Since evaluating such a complex design with Fenix is infeasible (see [GPM24, Table II]), the only viable way to assess the robust d-probing security of the design is to synthesize it for an ASIC and evaluate it using an ASIC verification tool. For this, we synthesized the design with Synopsys Design Compiler against the Nangate 45nm standard cell library and evaluated the resulting gate-level netlist by means of `PROLEAD`. We note that `PROLEAD` can estimate the required number of simulations using statistical power analysis [MM22]. For this, we used `PROLEAD`'s default statistical parameters, namely, a false-negative probability of $\beta = 10^{-5}$ and an effect size of $\varphi = 0.1$. In this setting, this resulted in approximately one million simulations. `PROLEAD` confirmed that the evaluated design is $(1, 1, 0)$-robust 1-probing secure, which strengthens the assumption that the observed leakage may be

coupling-related. However, we note that the ASIC design differs from the measured FPGA implementation, and the exact root cause of the leakage remains, so far, unidentified.

4 Security-Aware FPGA Design Flow

We propose a security-aware FPGA design flow that mitigates information leakages caused by physical defaults across multiple levels of abstraction. This flow also facilitates the early identification and mitigation of potential leakages. A schematic overview is provided in Figure 5. Our design flow combines the classical FPGA design flow, depicted in white, with

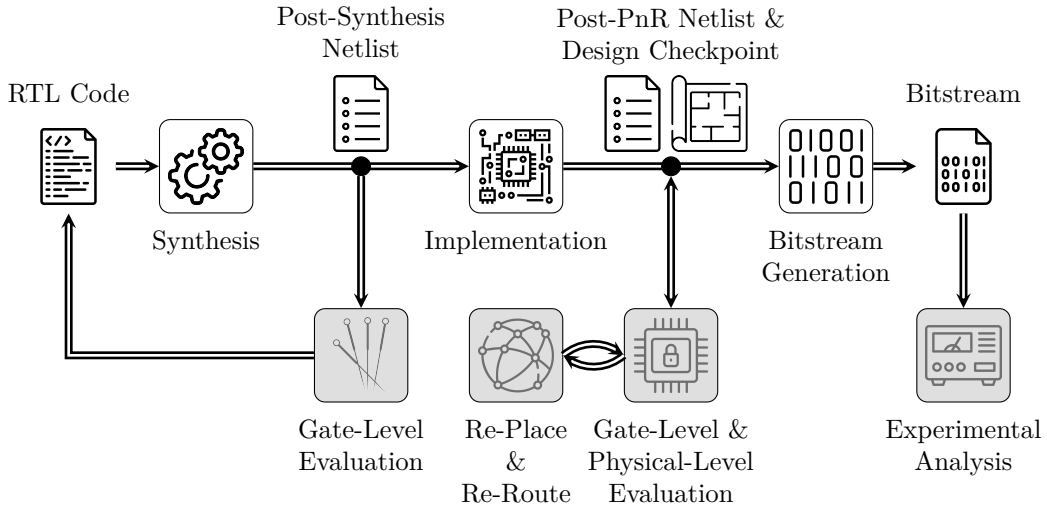


Figure 5: Our security-aware FPGA design flow, with all steps relevant to this work highlighted in gray.

additional verification and adjustment steps, highlighted in gray. Initially, the designer has to provide a masked design in a HDL which is then synthesized for FPGA, e.g. by commercial products, such as Vivado or Quartus Prime, or an open-source synthesizer, such as Yosys [Wol]. The outcome of this step is a gate-level netlist. In the following, the design is implemented, and the bitstream is generated, with an additional round of tool-based verification conducted after each step. This enables the detection of leakages at different abstraction levels and allows for mitigation at early design stages. In the following, we discuss all verification and mitigation steps.

4.1 Gate-Level Evaluation

Tool-assisted verification of masked circuits synthesized for ASIC is widely used, for example, to assess a circuit's robust d-probing security based on its gate-level netlist. However, only Fenix addresses the verification of designs synthesized for FPGA, even though FPGA-mapped designs pose unique challenges compared to ASICs. Specifically, two optimization techniques were identified in the literature [GPM24] to inadvertently recombine different shares, thereby violating the security guarantees of masking.

LUT Combining. Consider a first-order masked DOM multiplication in $\mathbb{F}_2$ such that $\mathsf{F} : \mathbb{F}_2^5 \to \mathbb{F}_2^2$ with $z^0 = x^0y^0 \oplus x^0y^1 \oplus r$ and $z^1 = x^1y^1 \oplus x^1y^0 \oplus r$. Both inputs are shared with two shares such that $x = x^0 \oplus x^1$ and $y = y^0 \oplus y^1$ and r denotes a fresh mask. For

ASICs, such a shared multiplication would be implemented using four dedicated AND2 gates and four XOR2 gates, depending on the standard-cell library used. However, it has been shown in [GPM24] that, during FPGA synthesis, the computations of $x^0y^1 \oplus r$ and $x^1y^0 \oplus r$ are merged into a single common LUT with two outputs, such as a LUT6_2[4] in case of AMD FPGAs. Although this optimization, referred to as LUT combining, decreases resource overhead by reducing the number of utilized LUTs, it unintentionally combines both shares of each variable within a single LUT, thereby introducing observable leakage in practice [GPM24]. We note that LUT combining can be completely prevented in Vivado by setting the `-no_lc`[5] synthesis flag, whereas Quartus Prime Pro for Intel (Altera) FPGAs does not provide an equivalent setting. Instead, it can be controlled with finer granularity by implementing parts that should not be recombined as separate modules and using the `keep_hierarchy`[6] or `keep`/`preserve`[7] attributes. However, the first approach completely deactivates the optimization, rendering it ineffective for designs that may benefit from it. The second approach, on the other hand, is manual and error-prone, placing the entire responsibility on the designer.

Register Retiming. Register retiming allows the synthesizer to modify the placement of FFs to improve the critical path delay of the design. However, this optimization can also affect the occurrence of glitches, which in turn may introduce glitch-related leakages. Similar to LUT combining, register retiming can be entirely disabled in Vivado by setting the `-global_retiming`[5] synthesis flag to off, which may result in less efficient designs. In Quartus Prime Pro, the same can be achieved via the `ALLOW_REGISTER_RETIMING`[8] option.

4.1.1 Evaluation Procedure

We propose a verification step operating on the gate-level netlist to guarantee that neither design flaws introduced by the designer nor optimizations performed during synthesis or implementation degrade the security of the masked implementation. As these optimizations may occur at different stages, the procedure should be applied after synthesis and again after implementation, based on the corresponding netlists. Consequently, leakage can be identified immediately after synthesis using the post-synthesis netlist, allowing the designer to address it at either the behavioral or gate level. If leakage is detected after implementation using the post-implementation netlist, the designer can mitigate the leakages by adjusting the implementation.

Currently, `PROLEAD` can only assess the robust d-probing security of ASIC designs provided as a gate-level netlist, along with a cell library that defines the behavior of each cell used in the netlist, i.e. its input and output ports along with the specific Boolean equation computed by a cell. Specifically, `PROLEAD` does not support attribute definitions in the cell library, which are commonly used for AMD design elements. To address this, we implemented attribute support, particularly for two common types.

- The `INIT` attribute is used to define the output function of a LUT or the initial state of a FF. For LUTs, `PROLEAD` converts the `INIT` string into a Conjunctive Normal Form (CNF), specifying the computed Boolean function of the gate. For FFs, `PROLEAD`'s internal simulator initializes all simulated FF states based on the attribute's value.

[4] https://docs.amd.com/r/en-US/ug974-vivado-ultrascale-libraries/LUT6_2
[5] https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Using-Synthesis-Settings
[6] https://docs.amd.com/r/en-US/ug912-vivado-properties/KEEP_HIERARCHY
[7] https://docs.altera.com/r/docs/683236/25.3/quartus-prime-pro-edition-user-guide-design-compilation/preserving-signals-for-monitoring-and-debugging
[8] https://docs.altera.com/r/docs/683236/25.3/quartus-prime-pro-edition-user-guide-design-compilation/allow-register-retiming

- For FFs, the optional inversion of an input signal x can be enabled using the `IS_X_INVERTED` attribute. This inversion is emulated by incorporating the attribute's value as an additional input to the Boolean function that represents the FF behavior in `PROLEAD`.

Consequently, the circuit model formalized in [MM22, Definition 3] and implemented in `PROLEAD` for evaluating gate-level ASIC netlists also remains applicable to FPGAs. Accordingly, we slightly adapt the definition for FPGA designs to better align the terminology with the FPGA context.

Definition 1 (FPGA Design)**.** We model any FPGA design as a directed graph $G = (V, E)$. Each vertex $v \in V$ models a cell, such as a LUT or FF while each $e \in E$ models an interconnection between cells in V. Each $v \in V$ is a tuple $v = (I_v, O_v, \mathsf{f}_v)$ where $I_v \subseteq E$ (resp. $O_v \subseteq E$) is a set of input (resp. output) wires of v and $\mathsf{f}_v : \mathbb{F}_2^{|I_v|} \to \mathbb{F}_2^{|O_v|}$ is the evaluation function. For each $v \in V$, we define $\mathsf{isFF}(v) \in \{\mathtt{true}, \mathtt{false}\}$ with $\mathsf{isFF}(v) = \mathtt{true}$ if v is a FF and $\mathsf{isFF}(v) = \mathtt{false}$ otherwise. Each $e \in E$ is a tuple $e = ((r, n), L)$ where (r, n) specifies the output port driving the wire, i.e., its root. In particular, $0 \leq n < |O_r|$ specifies the output port of cell $r \in V$ by its index. In addition, L is a set of tuples such that every $(l, m) \in L$ specifies an input port to which e is connected. Here, $l \in V$ denotes the cell reached by e and $0 \leq m < |I_l|$ denotes the port index. We refer to elements in L as leaves.

Next, we define a restricted cell library based on the specification of the SoC library guides[9] provided by AMD, which `PROLEAD` can use for simulation. Specifically, our library includes LUTs such as LUT1–LUT6_2 and FFs like FDRE, FDCE, FDSE, and FDPE. Other FPGA-specific primitives are not considered in this work but can be integrated into the library in the same way. As only basic design elements are supported, this dedicated library can be used to analyze designs synthesized (with the respective design elements only) for 7-series architectures (Artix-7, Kintex-7, Virtex-7, Zynq-7000) as well as AMD UltraScale architecture-based devices including the AMD UltraScale and AMD UltraScale+ families. However, using the same approach, equivalent libraries can be developed to support FPGAs from other vendors. For verification under the robust d-probing model, no further modifications to `PROLEAD` are required. `PROLEAD` automatically extends a probe on a LUT-output to all its inputs, while the glitch extension stops if the probe is placed on a FF output.

Overall, the gate-level evaluation process is inherently iterative. If `PROLEAD` detects any leakage in the gate-level netlist, the identified vulnerabilities must be mitigated, either by modifying the RTL code or by adjusting the netlist itself, until no further leakages are detected. When completed successfully, this step yields a gate-level netlist for which `PROLEAD`, under its current evaluation settings, detects no remaining violations of robust d-probing security, accounting for both glitches and transitions.

4.2 Physical-Level Evaluation

Once the gate-level evaluation is completed, the analysis can focus on coupling leakages introduced by the specific PnR of the design. To forward PnR information to `PROLEAD`, we implemented the design using Vivado and exported its resulting implementation as a design checkpoint. A design checkpoint represents a complete snapshot of the design at its current stage in the design flow, for example, after implementation, and includes the verified netlist, specific design constraints, and the corresponding implementation results. The checkpoint is then processed by RapidWright[10], an open-source framework developed

[9] https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries, https://docs.amd.com/r/en-US/ug974-vivado-ultrascale-libraries

[10] https://www.rapidwright.io/

by AMD Research and Advanced Development (previously Xilinx Research Labs) that seamlessly integrates with Vivado. Specifically, RapidWright allows for the analysis and modification of a checkpoint at any stage of the design flow, with the modified snapshot being reintegrated into Vivado. We use RapidWright to convert the design checkpoint into the FPGA Interchange Format (FPGAIF), a vendor-agnostic format that facilitates the exchange of FPGA designs at both the gate and physical levels, and enables customized PnR by third-party tools. We note that the outlined procedure is specific to AMD FPGAs and their associated toolchain, involving Vivado and RapidWright. At present, other vendors do not offer complete support for generating the FPGAIF from their designs. We hope that such support will be made available in the future. In the FPGAIF, the implementation is represented as a tuple comprising a logical and a physical netlist. Since both files are not human-readable, and given that `PROLEAD` already supports parsing JSON files, we further convert the FPGAIF file into a human-readable JSON format using a Python framework[11] provided by CHIPS Alliance. The resulting JSON file is then provided to `PROLEAD` as an additional input file.

4.2.1 Modeling Coupling-Extended Probes

The probe-extension procedure proposed in [FGP+18] introduces a parameter c that limits the extension of a probe to its c nearest neighboring wires. Hence, if information about the physical proximity of wires can be obtained from the hardware layout, as is typically available for ASICs through a Graphic Design System (GDS)II file, the above approach can be applied straightforwardly. However, for many commercial FPGAs, the physical design is proprietary and therefore unavailable for evaluation purposes. Consequently, the coupling-based probe-extension procedure proposed in [FGP+18] cannot be directly applied to FPGAs. Instead, the resulting JSON file abstracts each physical wire as a sequence of PIPs (cf. Definition 2), which define its route from the source cell to its destination cells.

Definition 2 (Programmable Interconnect Point (PIP))**.** A PIP $\theta = (i, o, s)$ is a directed connection from the entry point i to the exit point o within the switch box s.

For simplicity, we assume that a wire is routed from $\theta = (i, o, s)$ to $\theta' = (i', o', s')$ if $o = i'$. We remark that a large number of such PIPs is available within a single switch box, which makes it necessary to concentrate many wires within a limited physical area. This high routing density is a direct consequence of the architectural flexibility and efficiency required in modern FPGAs, where each logic block must be connectable to multiple routing channels to enable short critical paths. Hence, even though there is no direct electrical connection between different wires inside a switch box, they may still leak jointly due to capacitive or inductive coupling effects. Therefore, we propose a probe-extension procedure in which the extension is not limited by a fixed threshold to adjacent wires but rather constrained by the switch boxes through which multiple wires are jointly routed. Specifically, we assume that a coupling-extended probe on a given wire passing through a switch box is extended to probes on all other wires traversing the same switch box. The main advantage of this approach is that it does not require detailed architectural knowledge, as it inherently follows a worst-case assumption when considering a single switch box. For instance, even if wires within the same switch box are directly connected, e.g., through dedicated internal logic, the resulting leakages are still captured by our method, regardless of whether they originate from actual coupling effects. In practice, all the information required to perform this probe-extension procedure is contained within the JSON file and can be combined with the well-known extension procedures for glitches and transitions. However, this worst-case approach may also reveal coupling effects that

[11] https://github.com/chipsalliance/python-fpga-interchange

are inherent to certain FPGA architectures, thereby implying that any implementation of a given masking scheme would be insecure on those specific FPGAs. We visualize these couplings in Figure 6. Consider a DOM multiplier computing $z = x \cdot y$ in $\mathbb{F}_2$ based on

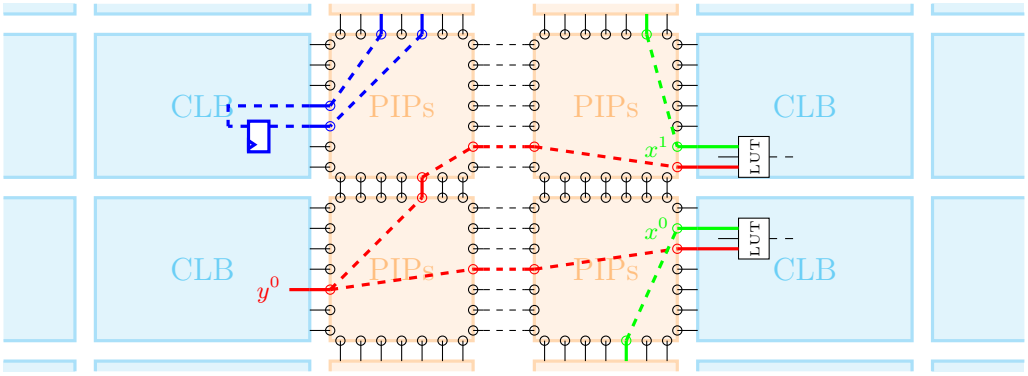


Figure 6: Unavoidable coupling leakages according to our approach.

two-shared input variables x^0, x^1 and y^0, y^1. As described in Section 2.1.1, the DOM multiplier computes the product terms x^0y^0 and x^1y^0. Now, consider the routing depicted in red, which forwards y^0 to two different LUTs to realize these product terms. Since x^0, y^0, and x^1 are routed to the same LUTs, they also pass through the same switch boxes, where x^0 and x^1, depicted in green, are assumed to couple with one of the routing paths of y^0. Consequently, if an adversary were to place a probe on y^0, it would be coupling-extended to x^0 and x^1, leading the analysis to classify this implementation as insecure. We remark that there is usually no way to route x^0, y^0, and x^1 such that they avoid passing through the same switch box, as each wire serving as an input or output to a CLB can only be routed through a single switch box or a specific, unique sequence of switch boxes. This also leads to another issue which is illustrated by the blue wire. If a signal is forwarded to a FF the input and output wire of the FF must be routed through the same switch box implying that every FF output couples with its own input. Hence, a probe on the FF output would always be extended to a probe on the FF input, which must then be further extended to account for glitches and transitions. However, since FFs are essential to prevent the propagation of glitches in a masking scheme, the assumption that both wires of a FF couple would imply that every glitch-resistant masking scheme is inherently insecure, as the glitch extension would continue regardless of the presence of FFs.

4.2.2 Refined Probe-Extension

To keep our approach constructive, meaning that it can cover couplings that can be mitigated by adjusting PnR while ignoring unavoidable couplings, we adopt and refine the notion of a wire as defined in Definition 1, modeling its routing through a sequence of switch boxes. Definition 3 formalizes the refined notion of a so-called physical wire.

Definition 3 (Physical Wire)**.** Let $G = (V, E)$ be an FPGA design. We model a physical wire as a tuple $e^* = (e, \theta)$ where $e \in E$ and θ models a PIP which routes e.

Consequently, probing wires with multiple destinations, i.e., wires replicated during routing, depends on the exact probe location along the wire, meaning that multiple routing paths are assumed to leak independently with respect to coupling effects. Naturally, Definition 1 can be extended to incorporate physical wires, as formalized in Definition 4.

Definition 4 (Physical FPGA Design)**.** A physical FPGA design (G^*, Θ) is a tuple where

- $G^* = (V, E^*)$ is a directed graph whose vertices are cells and the edges are physical wires according to Definition 3. Each $v \in V$ is a tuple $v = (I_v, O_v, \mathsf{f}_v)$ where $I_v \subseteq E^*$ (resp. $O_v \subseteq E^*$) is a set of physical input (resp. output) wires of v and $\mathsf{f}_v : \mathbb{F}_2^{|I_v|} \to \mathbb{F}_2^{|O_v|}$ is the evaluation function.
- Θ is a set of PIPs. For all $\theta \in \Theta$, we define $\mathsf{pred}(\theta) \in \Theta \cup \{\perp\}$ with $\mathsf{pred}(\theta) = \theta'$ if $\theta' = (i', o', s')$ is the parent of $\theta = (i, o, s)$, i.e. $i = o'$, and $\mathsf{pred}(\theta) = \perp$ if θ is the root, i.e., the first PIP through which a wire is routed from its output port.

To consider independent couplings on different paths taken by the wire, we incorporate the PIP which is currently probed into the probe definition. Hence, according to Definition 5 a physical probe is placed on a physical wire, i.e. a specific PIP through which a wire is routed, and its extension starts from this location.

Definition 5 (Physical Probe)**.** Let (G^*, Θ) be a physical FPGA design and T be a set of clock cycles. A physical probe $p = (e^*, t)$ is a tuple where $e^* = (e, \theta) \in E^*$ and $t \in T$. It records a signal on wire e during clock cycle t.

Finally, the recursive procedure for extending a probe considering glitches, transitions, and couplings is outlined in Algorithm 1. Initially, p is extended to account for glitches and transitions (cf. Lines 4-18). Specifically, if p is placed on a PIP that is not the root PIP of the probed wire, it is further extended (Line 17) to an additional probe p' on the predecessor PIP, and this extension continues until the root PIP is reached. For probes which are placed on a root PIP the glitch-extension is done as usual. If the probed wire is the output of a LUT, the probe gets glitch-extended to additional probes on all inputs of the LUT (cf. Lines 12-14) while the glitch-extension stops at root PIPs of primary inputs and FF outputs. If the glitch-extension stops, p gets transition-extended (cf. Line 8) and the resulting tuple is added to the final set of probes Q. The coupling-extension is performed in Lines 19-28 and is performed independently of the glitch- and transition-extension. In general, the coupling-extension extends p to additional probes on all other physical wires that are routed through the same switch box (cf. Lines 25-27). Further, it holds that every probe gets extended according to all physical effects. Specifically, every probe received through a coupling-extension gets further extended by glitches and transitions. However, while this approach addresses the issue of multi-fanout wires, FF inputs and outputs still exhibit coupling. We discovered two constraints on the coupling-extension in order to tackle this issue.

- We can prohibit the coupling extension from a FF output back to its own input, while allowing it from the FF output to probes on all other wires routed through the same switch box.
- We can prohibit the entire coupling extension from a FF output in the switch boxes that are the only access point to the CLB in which the FF is placed.

The first approach is reasonable and preserves the conservativeness of the original model. However, although a probe placed on the output of a FF can no longer be coupling-extended to its own input, it can still reach the inputs of all other FFs within the same CLB. This scenario would violate the probing security of nearly every masking scheme. Maintaining security would therefore require that no two FFs be placed within the same CLB, resulting in extremely poor FPGA resource utilization and, consequently, impractical designs. Therefore, we proceed with the second approach, as implemented in Lines 19-24. Specifically, we introduce an additional parameter, δ, which denotes the number of consecutive switch boxes that must be traversed to route a wire to or from a particular CLB. We then verify whether p is located in a switch box at a distance of δ from the root,

Algorithm 1 Probe Extension

Input: $(G^*, \Theta), \delta$ ▷ Physical FPGA design and depth parameter
Input: $p = (e^*, t)$ ▷ A single physical probe to extend
Output: Q ▷ The set of physical probe extensions

$(e, \theta) \leftarrow e^*, (i, o, s) \leftarrow \theta, \theta' \leftarrow \theta, Q \leftarrow \emptyset$
$w \leftarrow$ **if** $\exists! v \in V, e^* \in O_v$ **then** v **else** $\bot$ ▷ Get the cell w which outputs e^*
$b \leftarrow w \neq \bot \wedge \mathsf{isFF}(w) = \texttt{true}$ ▷ $b = \texttt{true}$ if e^* is the output of a FF
if $\mathsf{pred}(\theta) = \bot$ **then** ▷ Check if θ is root, i.e. the initial PIP on the path
  if $w = \bot \vee b = \texttt{true}$ **then** ▷ Check if e^* is a primary input or FF output
    $Q \leftarrow Q \cup \{p\}$
    if $t > 1$ **then** ▷ Perform the transition-extension on p
      $p' \leftarrow (e^*, t-1)$
      $Q \leftarrow Q \cup \{p'\}$
    end if
  else ▷ e^* is neither a primary input nor a FF output
    for $\forall h^* \in I_w$ **do** ▷ Glitch-extend p to all inputs of w
      $Q \leftarrow Q \cup$ Algorithm 1$((G^*, \Theta), \delta, p' = (h^*, t))$
    end for
  end if
else ▷ If θ is not the root, glitch-extend p to the predecessor PIP
  $Q \leftarrow Q \cup$ Algorithm 1$((G^*, \Theta), \delta, p' = ((e, \mathsf{pred}(\theta)), t))$
end if
if $b = \texttt{true}$ **then** ▷ Restrict the coupling-extension of FF outputs
  for $\forall x \in [0, \delta)$ **do**
    $\theta' \leftarrow \mathsf{pred}(\theta')$ **if** $\theta' \neq \bot$ ▷ Get the predecessor PIP until root is reached
  end for
end if
if $\theta' \neq \bot$ **then** ▷ Coupling-extend p to all physical wires in the same switch box
  for $\forall h^* = (e', (i', o', s')) \in E^*, s' = s \wedge h^* \neq e^*$ **do**
    $Q \leftarrow Q \cup$ Algorithm 1$((G^*, \Theta), \delta, p' = (h^*, t))$
  end for
end if

and proceed with the coupling extension only if the switch box containing p is sufficiently distant from the root. Note that this restriction can be fully disabled by setting $\delta = 0$, provided that the specific FPGA architecture allows access to a CLB through multiple switch boxes.

4.2.3 Evaluation Procedure

Based on the adjusted probe-extension from Algorithm 1, `PROLEAD` applies the same steps as discussed in Section 2.4. To this end, Algorithm 1 is precomputed to obtain a mapping

$$\mathcal{Q}[e^*, t] = \{(e_0^*, t), \ldots, (e_{n-1}^*, t), (e_0^*, t-1), \ldots, (e_{n-1}^*, t-1)\},$$

which defines the set of physical probes to which a physical probe on (e^*, t) is extended. To construct all relevant d-tuples of physical probes, it suffices to consider all (e, θ, t) with $\theta = (i, o, s)$ such that there exists no $\theta' = (i', o', s')$ with $o = i'$, i.e., θ is the last PIP on the path. A physical probe on (e, θ, t) is strictly less informative than a probe on (e, θ', t) with $o = i'$, since a probe on (e, θ', t) will always extend to (e, θ, t). This optimization slightly improves the evaluation time, as the number of considered d-tuples is reduced. Finally, Step 3 described in Section 2.4 can be performed without any modification.

4.3 Re-Place and Re-Route

If PROLEAD detects leakage in the physical implementation of the design, given as a list of probe sets for which the circuit is not robust d-probing secure, an iterative process of re-placement and re-routing of the identified leakage sources is performed until no further leakage is observed. For AMD FPGAs, this step can be carried out directly in Vivado by modifying the design snapshot, from which the final bitstream is subsequently generated. In the following, we discuss Vivado's auxiliary methods for adjusting PnR according to our requirements. For FPGAs from other vendors methods provided by their respective toolchains or by third-party tools can be used to achieve similar results[12].

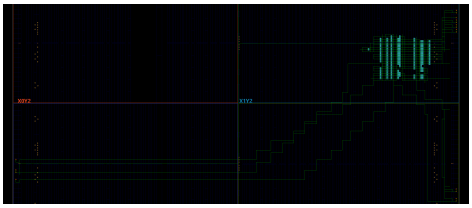
Physical Blocks (Pblocks). A Pblock defines a group of cells that Vivado attempts to place within one or more predefined rectangular regions. This guides the placer to position a selected set of cells in a specific area without requiring each cell to be placed manually. The placer can be instructed to keep cells not assigned to the Pblock out of its region and to restrict the routing of cells inside the Pblock to the routing resources available within its boundaries. Hence, all cells belonging to the same share domain should be assigned to the same Pblock, and the Pblocks should be physically separated, i.e., they must not overlap. We note that Vivado does not always follow the intended PnR constraints defined by the Pblocks, particularly when a Pblock does not provide sufficient resources. However, if done correctly, wires routed entirely within a single Pblock should not exhibit coupling that could lead to information leakage. Consequently, the manual routing effort is limited to the wires connecting different Pblocks, i.e., the input and output connections between Pblocks and to wires that Vivado routed outside their associated Pblocks.

Fixed Routing Constraints. Vivado allows the modification of the automated wire routing. Therefore, if PROLEAD reports leakage, we must identify its source, specifically, the switch box through which wires from different Pblocks are jointly routed, and manually re-route the affected wires to ensure they no longer share the same switch box. This process involves first unrouting the relevant wires and then rerouting them manually through different switch boxes. The final routing can then be fixed to ensure that it is preserved when re-routing the entire design.

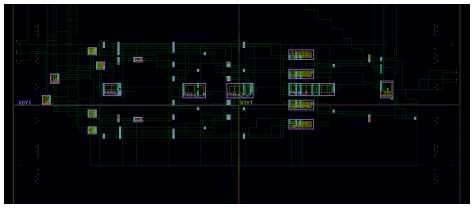
5 Case Studies

In the following case study, we fix the leaking design that we previously analyzed by physical experiments in Section 3. To this end, we first evaluate the design using our extended version of PROLEAD, then manually re-place and re-route critical components to eliminate all detected leakages. This process is repeated until no further leakages are found. Finally, we confirm the security of the adjusted design by rerunning the experimental analysis described in Section 3. To challenge the assumptions made in Section 4, we conducted four experiments, depicted in Figure 7, that demonstrate how specific design adjustments influence the observed leakage. Initially, we refined the design evaluated in Section 3.3 by physically separating the computations of each intermediate product term (see Section 2.1.1). For this, we assigned them to separate Pblocks, which were then placed with spatial separation. However, when multiple non-linear computations were performed in parallel, as occurs during the last two clock cycles (cf. Figure 1), we merged the Pblocks corresponding to identical product terms. For all linear terms, we isolated the computations of different share domains by placing them in distinct Pblocks. We note that

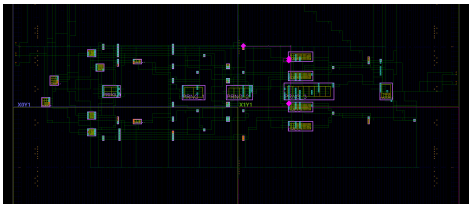
[12] In Quartus Prime, the concept of design partitions can be used to isolate the placement of sensitive logic: https://docs.altera.com/r/docs/683641/25.1/quartus-prime-pro-edition-user-guide-design-optimization/using-logic-lock-regions-in-combination-with-design-partitions



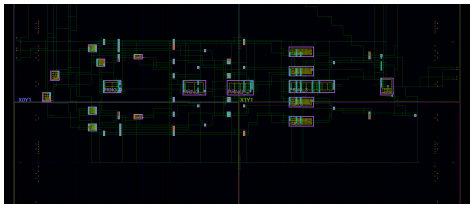
(a) Auto PnR; $f_{\max} = 821\,\text{MHz}$.



(b) Manual place, auto route; $f_{\max} = 423\,\text{MHz}$.



(c) Manual PnR (one sensitive wire auto-routed (pink)); $f_{\max} = 406\,\text{MHz}$.



(d) Manual PnR (all sensitive wires); $f_{\max} = 406\,\text{MHz}$.

Figure 7: Implementation views of the DOM AES S-box design on an AMD Kintex-7 FPGA (xc7k160tfbg676-1) as visualized by Vivado 2025.2. Total FPGA resource utilization: 554 LUTs and 907 FFs, including measurement circuitry.

these modifications are made solely to facilitate the subsequent routing adjustments. The original design, implemented with unconstrained PnR and shown in Figure 7a, is densely placed, making manual re-routing impractical. Grouping related components into the same Pblock reduces the manual effort largely to wires crossing between Pblocks, since internal wires can couple without causing leakage. The resulting design is shown in Figure 7b. We again note that the adjusted designs (cf. Figure 7) were constructed to facilitate manual modifications rather than to achieve maximum performance. In fact, the re-placement reduces the maximum clock frequency $f_{\max}$ by 48.5%. Afterwards, we manually re-routed all wires that shared a switch box with any wire leading to a different Pblock. Specifically, if a wire is routed to Pblock a, it must not pass through the same switch box as a wire routed to Pblock b. We emphasize that the re-routing was intentionally performed in a way that challenges the assumptions made in Section 4. In particular, sensitive wires are routed through switch boxes in close proximity, i.e., neighboring switch boxes, and we allow wires to be routed close to each other as long as they do not share the same switch box. If our coupling-leakage model is too weak, these conditions should reveal leakage in the practical measurements. The resulting implementations are shown in Figure 7c and Figure 7d. The only difference is that in Figure 7c we keep one sensitive wire (highlighted in pink) with its automatically assigned routing, which violates our security model. In contrast, Figure 7d shows the version in which all relevant wires have been manually re-routed.

5.1 Tool-Assisted Evaluation

Initially, we conducted a gate-level evaluation to verify that the synthesis and implementation processes did not introduce vulnerabilities as a result of optimization techniques. To this end, we exported the post-implementation netlist of the design discussed in Section 3 and illustrated in Figure 7a (but not the design checkpoint) and fed it to `PROLEAD`. As in Section 3.3, we applied `PROLEAD`'s standard statistical parameters as proposed in [MM22] and considered the joint leakages of glitches and transitions. We ran this experiment on a standard laptop equipped with 16 GB of RAM and an Intel Core i7-10610U CPU with 4 cores at 1.80 GHz, running Ubuntu 22.04 under WSL 2. We note that a similar setup, with

a slightly more powerful CPU, was used for the benchmarks of Fenix [GPM24]. However, such a comparison should be taken with caution due to the following reasons.

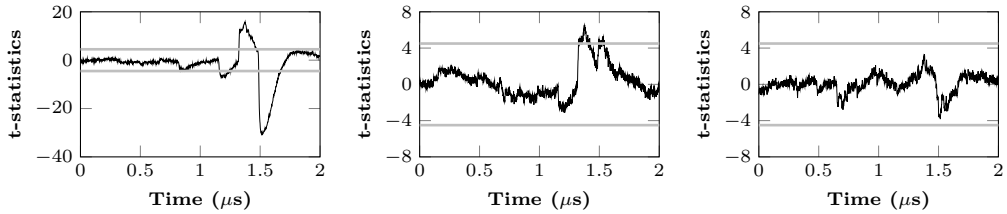
- PROLEAD delivers an incomplete yet still meaningful evaluation, whereas Fenix performs an exhaustive analysis.
- The DOM AES S-box designs analyzed with Fenix and PROLEAD are not the same. We targeted the DOM AES S-box that includes pipeline registers, while the Fenix benchmark considers the version without them. Pipeline registers restrict the probe-extension and, therefore, facilitate manual re-routing and tool-assisted verification.
- Fenix runs in a single thread, while PROLEAD supports multi-threaded execution.

Despite this, PROLEAD completes its evaluation in approximately 46 minutes, while [GPM24] reports that Fenix did not finish evaluating a DOM AES S-box even after one week. PROLEAD confirms the $(1, 1, 0)$-robust 1-probing security of the design, meaning that no vulnerabilities were introduced during synthesis. This aligns with our expectations, as we disabled security-critical optimizations (see Section 3.1). Next, we evaluate the same design but include its PnR in the evaluation. As considering more effects leads to an overhead in evaluation time, we accelerate the evaluation by switching to an Ubuntu 22.04 server with 2 TB of RAM and an Intel(R) Xeon(R) Gold 5320 CPU with 52 used cores at 2.2 GHz. However, the configuration of PROLEAD remains unchanged from the previous evaluation except that we set $\delta = 2$. This is a specific design choice for our target FPGA, as every wire routed to a given CLB must pass through both a switch box inside the CLB[13] and the switch box located in the adjacent switch box tile. In other words, each wire is required to traverse these two switch boxes. Under these conditions, the tool identifies several leakages across all analyzed clock cycles, which we attribute to potential coupling effects. However, the t-statistic values shown in Figure 4b exceed the 4.5 threshold only in some of the clock cycles. This suggests that either not all of the considered coupling effects manifest in practice or that the corresponding leakage is too small to be detected experimentally. In the former case, this behavior can be explained by the conservative nature of the model, which is consistent with the conservative treatment of glitches and transitions [MM24]. We continued with the designs depicted in Figure 7b and Figure 7c and evaluated them including their PnR information. PROLEAD detects leakage for both designs, as only restricting the placement is not enough to ensure security in our model. However, for the design shown in Figure 7c, PROLEAD detects leakage only on a single wire, the one that was not manually rerouted. Finally, after iteratively refining the design and repeatedly verifying it with PROLEAD, we obtained the implementation shown in Figure 7d, for which PROLEAD no longer detects any leakages. The final verification took around 8 minutes and consumed 450 GB of RAM on the described server.

5.2 Experimental Analysis

The experimental analysis of the adjusted designs was performed with the same setup and parameters as outlined in Section 3.2. The TVLA results shown in Figure 8 confirm our assumptions and the tool-assisted evaluations, indicating that the leakage observed in Section 3 has been effectively mitigated in the manually re-placed and re-routed design (cf. Figure 8c). Specifically, for the design reported as secure by PROLEAD, the results never exceed the 4.5 threshold at any point. However, for the designs with no or incomplete re-routing the TVLA results confirm the results of PROLEAD. Specifically, even the design where only a single wire is not correctly re-routed shows leakage as shown in Figure 8b. We attribute this to the fact that the wire in question is unusually long due to the placement separation and is routed directly through another Pblock, making it particularly susceptible

[13] https://docs.amd.com/r/en-US/ug474_7Series_CLB/CLB-Overview



(a) Automatically routed.

(b) Manually re-routed, except for one sensitive wire.

(c) Manually re-routed for all sensitive wires.

Figure 8: TVLA results over 100 million traces of a DOM AES S-box with different PnR.

to coupling. We believe that the same factors also explain why the re-placed design exhibits higher leakage than the design without any PnR modifications (cf. Figure 8a).

6 Limitations and Future Work

We regard this work as a promising starting point for unveiling coupling leakages in masked FPGA-based designs, even though such leakages can never be assessed exhaustively. However, several research questions remain only partially addressed or entirely open, which we highlight as directions for future work.

Precise Modeling of Physical Effects on Proprietary Hardware Platforms. We model coupling leakage in an abstract manner, assuming that all wires accessing the same switch box leak jointly. This approach has two limitations. First, it treats all wires routed within a switch box equally, without distinguishing between coupling and non-coupling wires within the same switch box. Second, it abstracts routing as a path through multiple switch boxes, without accounting for wires routed in close proximity that do not access the same switch box. Both limitations make it difficult to derive implications from our model for coupling-extended probes as defined in [FGP+18]. For example, guaranteeing $(1, 1, 1)$-robust 1-probing security for the designs discussed in Section 5 would require knowing, for every wire, its closest neighbor. Consequently, our model may incorrectly classify an insecure design as $(1, 1, 1)$-robust 1-probing secure if the closest neighbor of a wire is not routed through the same switch box. Conversely, it may classify a $(1, 1, 1)$-robust 1-probing secure design as insecure if the observed leakage originates from a wire that is not the closest neighbor, but happens to be routed through the same switch box. Our approach is motivated by the fact that addressing these issues would require access to the physical FPGA design, such as a GDSII file, which is not publicly available for commercial (closed-source) FPGAs. Therefore, we welcome the recent development of a reconfigurable publicly available platform with inherent protection through masking [MLTM25]. On such platforms, the coupling between wires can be estimated using proximity information from the GDSII file, which makes leakage models more accurate in terms of coupling and naturally aligns with the robust probing model.

Investigation of Coupling Leakage in Alternative Verification Tools and Models. We implemented our approach as an extension of `PROLEAD` to assess the probing security of a masked FPGA design. However, our method is generic and can likewise be integrated into other verification frameworks operating under the robust d-probing model. This includes tools that perform exhaustive verification, as proposed in [BBFG18, BGI+18, KSM20, BMRT22, ZCF25]. Moreover, since `PROLEAD` supports the evaluation of security

notions beyond probing security, such as Non-Interference (NI) [BBD+15], Strong Non-Interference (SNI) [BBD+16], and Probe-Isolating Non-Interference (PINI) [CS20], our approach naturally extends to the analysis of composability notions in the presence of coupling effects, as highlighted for future work in [GPM24]. This property would also be preserved if our method were integrated into other tools that support composability notions, such as SILVER [KSM20]. However, we remain cautious about whether analyzing composability notions concerning coupling is meaningful, as the result depends on the PnR of the composition rather than on a single gadget. More precisely, a gadget that is instantiated multiple times within a design can experience coupling in different ways: either internally, within the gadget itself, or externally, between different gadgets. Internal coupling can be mitigated by placing each gadget in a dedicated CLB and ensuring identical PnR within its respective CLBs, but external coupling cannot be addressed by analyzing the composability of a single gadget. Therefore, additional routing constraints would be required to restrict coupling leakages between gadgets.

Automated and Security-Aware Routing. We focused on detecting coupling leakages through tool-assisted verification and practical experiments, while manually patching all leakage sources. However, with increasing design complexity, manually patching all these issues becomes infeasible. Therefore, a natural follow-up to this work would be the development of a security-aware router capable of routing wires carrying shares from the same share domain in close proximity while isolating them from wires carrying shares from different share domains. This router could operate incrementally alongside a verification tool that identifies coupling-related leakage. Upon receiving a report, the router reroutes the affected wires efficiently, minimizing resource overhead, and then updates the verification tool with the revised design. This process would repeat until the verification tool confirms that the design is secure. A similar router, targeting Xilinx FPGAs, already exists in the context of Dual-Rail Pre-charge (DRP) logic to balance routing delays [MI14].

Scalability. We demonstrated the evaluation of masked designs using `PROLEAD` while explicitly accounting for coupling effects, focusing on a single DOM AES S-box. We selected this target because isolating the different share domains in a masking scheme with $d + 1$ shares is considerably more challenging than separating the individual component functions in a TI-protected design as done in [CBG+17]. Moreover, we chose a relatively small design, specifically a single S-box, to keep the manual PnR process feasible. Nevertheless, our extended version of `PROLEAD` is capable of evaluating substantially larger designs, including fully masked, round-based AES implementations, with respect to coupling effects.

7 Conclusion

This work presents a security-aware FPGA design flow that enables the automated evaluation of masked FPGA designs across multiple abstraction levels using standard evaluation tools. Specifically, we extended `PROLEAD` to (1) evaluate the security of a given FPGA netlist under the $(1, 1, 0)$-robust d-probing model, and (2) assess security while accounting for coupling leakages by incorporating PnR information during the evaluation. To support this, we introduced a novel approach for modeling coupling-based leakages on FPGAs, even when the physical design of the device is not publicly available. We highlighted the relevance of our contributions with a case study based on a single DOM AES S-box design. We demonstrated that such a design which is provably secure under the $(1, 1, 0)$-robust d-probing model may still exhibit leakage in experimental analyses. In such cases, the extended version of `PROLEAD` provides valuable guidance for identifying and mitigating these leakages, enabling the creation of a re-placed and re-routed design that ensures security both in theory and in practice.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) through the projects 456967092 (SecFShare) and 549340884 (MatSec), and by the National Research Center for Applied Cybersecurity ATHENE.

References

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 457–485. Springer, 2015.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *ACM SIGSAC 2016*, pages 116–129. ACM, 2016.

[BBFG18] Gilles Barthe, Sonia Belaïd, Pierre-Alain Fouque, and Benjamin Grégoire. maskverif: a formal tool for analyzing software and hardware masked implementations. *IACR Cryptol. ePrint Arch.*, page 562, 2018.

[BFG+25] Christof Beierle, Jakob Feldtkeller, Anna Guinet, Tim Güneysu, Gregor Leander, Jan Richter-Brockmann, and Pascal Sasdrich. INDIANA - verifying (random) probing security through indistinguishability analysis. In *EUROCRYPT 2025*, volume 15608 of *LNCS*, pages 33–63. Springer, 2025.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS 2014*, LNCS, pages 64–81. Springer, 2014.

[BGI+18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In *EUROCRYPT 2018*, volume 10821 of *LNCS*, pages 321–353. Springer, 2018.

[BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. Ironmask: Versatile verification of masking security. In *IEEE SP*, pages 142–160. IEEE, 2022.

[Can05] David Canright. A very compact s-box for AES. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005.

[CBG+17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In *COSADE 2017*, volume 10348 of *LNCS*, pages 1–18. Springer, 2017.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[CGP+12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In *COSADE 2012*, volume 7275 of *LNCS*, pages 69–81. Springer, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In *CHES 2007*, LNCS, pages 28–44. Springer, 2007.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE TIFS*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR TCHES*, 2021(2):136–158, 2021.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *TIS@CCS 2016*, page 3. ACM, 2016.

[GPM24] Barbara Gigerl, Kevin Pretterhofer, and Stefan Mangard. Security aspects of masking on fpgas. In *IEEE HOST*, pages 199–210. IEEE, 2024.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 787–816. Springer, 2020.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations and an explanation based on externally-amplified couplings. *IACR TCHES*, 2019(2):293–317, 2019.

[LMMS23] Daniel Lammers, Amir Moradi, Nicolai Müller, and Aein Rezaei Shahmirzadi. A thorough evaluation of RAMBAM. In *ACM SIGSAC CCS*, pages 1705–1717. ACM, 2023.

[MI14] Amir Moradi and Vincent Immler. Early propagation and imbalanced routing, how to diminish in fpgas. In *CHES 2014*, volume 8731 of *LNCS*, pages 598–615. Springer, 2014.

[MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice - unveiling security flaws in masked circuits. *IACR TCHES*, 2022(2):266–288, 2022.

[MLM24] Nicolai Müller, Daniel Lammers, and Amir Moradi. A deep analysis of two glitch-free hardware masking schemes SESYM and LMDPL. *IACR TCHES*, 2024(3):76–98, 2024.

[MLTM25] Sergej Meschkov, Daniel Lammers, Mehdi B. Tahoori, and Amir Moradi. Design and implementation of a physically secure open-source FPGA and toolchain. *IACR TCHES*, 2025(3):542–582, 2025.

[MM22] Nicolai Müller and Amir Moradi. PROLEAD - A probing-based hardware leakage detection tool. *IACR TCHES*, 2022(4):311–348, 2022.

[MM24] Nicolai Müller and Amir Moradi. Robust but relaxed probing model. *IACR TCHES*, 2024(4):451–482, 2024.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR TCHES*, 2019(2):256–292, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *ICICS 2006*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.

[Sha79] Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[WMM20] Felix Wegener, Lauren De Meyer, and Amir Moradi. Spin me right round rotational symmetry for fpga-specific AES: extended version. *J. Cryptol.*, 33(3):1114–1155, 2020.

[Wol] Claire Wolf. Yosys open synthesis suite. https://yosyshq.net/yosys/.

[ZCF25] Feng Zhou, Hua Chen, and Limin Fan. Prover - toward more efficient formal verification of masking in probing model. *IACR TCHES*, 2025(1):552–585, 2025.