SoK: Design Tools for Side-Channel-Aware Implementations

Ileana Buhan*, Lejla Batina*, Yuval Yarom†, and Patrick Schaumont‡

* Radboud University, Digital Security
† University of Adelaide and Data61
‡ Worcester Polytechnic Institute

***Abstract*—Side-channel attacks that leak sensitive information through a computing device's interaction with its physical environment have proven to be a severe threat to devices' security, particularly when adversaries have unfettered physical access to the device. Traditional approaches for leakage detection measure the physical properties of the device. Hence, they cannot be used during the design process and fail to provide root cause analysis. An alternative approach that is gaining traction is to automate leakage detection by modeling the device. The demand to understand the scope, benefits, and limitations of the proposed tools intensifies with the increase in the number of proposals.**

In this SoK, we classify approaches to automated leakage detection based on the model's source of truth. We classify the existing tools on two main parameters: whether the model includes measurements from a concrete device and the abstraction level of the device specification used for constructing the model. We survey the proposed tools to determine the current knowledge level across the domain and identify open problems. In particular, we highlight the absence of evaluation methodologies and metrics that would compare proposals' effectiveness from across the domain. We believe that our results help practitioners who want to use automated leakage detection and researchers interested in advancing the knowledge and improving automated leakage detection.

I. Introduction

When a computing device operates, it interacts with its physical environment. In his seminal work, Kocher [46] demonstrated that the power consumption of a device leaks information about the data it processes, allowing the recovery of cryptographic keys. Since then, research has demonstrated leakage of sensitive information via other *side-channels*, including electromagnetic emanations (EM) [34, 62], timing [10, 18, 60], micro-architectural components [12, 35, 51], and even acoustic and photonic emanations [36, 48].

In response to developments in attacks, several methodologies for leakage detection and assessment have been developed. Typically, such techniques emulate an attack. They involve collecting side-channel traces, e.g., power traces, from the device and analyzing these traces to demonstrate an attack or the existence of leaks. While effective, such methodologies require the physical device's presence for evaluation, and this demand poses significant challenges.

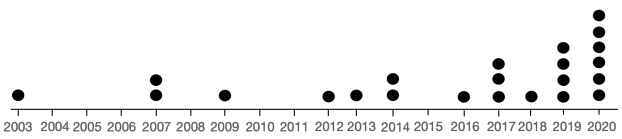

Figure 1. Distribution of papers which propose a new simulator per year (for multiple publications on the same tool, we cite the most recent one).

In the pre-silicon stage of the development, the device does not yet exist; hence it cannot be adequately assessed. Conversely, in the post-silicone stage, detailed design information may not be accessible, for example, when using third-party components. Consequently, it may be challenging to identify the root cause of leakage. Moreover, detecting, verifying, and mitigating side-channel leaks require expert knowledge and expensive equipment.

In recent years, we see the emergence of an alternative approach for evaluating device resilience to side-channel attacks. Instead of measuring the leakage from a physical device, *leakage emulators* aim to evaluate the device's model to reduce the effort required for leakage detection and potentially perform leakage detection early in the development process. The appeal of automation is proven by the early attempts of creating such tooling and the increased recent efforts directed to this purpose, as demonstrated in Figure 1. However, the abundance of proposed tools does not necessarily offer a solution for practitioners. Each tool aims to address a specific scenario, and with the increasing number of proposals, it may be complex to identify the best tool for each use case. Moreover, comparison of tools across the domain is lacking, preventing a straightforward assessment of the benefits that each of the tools may offer. A comprehensive study of automated tooling available for computer-aided cryptography was recently published [6] which covers design, functional verification, and implementation-level security of digital side-channels. The study covers tooling for side-channels such as execution time and side effects in shared resources (e.g., cache). Still, it excludes physical side-channels such as power consumption or EM radiation. This paper covers the gap and presents a taxonomy of state-of-the-art tooling for protecting against physical side-channel attacks. This work aims to chart the landscape and present a coherent view of current advances in leakage evaluation automation. Specifically, it aims to

- Give a comprehensive survey of the available tooling for leakage detection, verification, and mitigation and clarify the current capabilities and limitations.

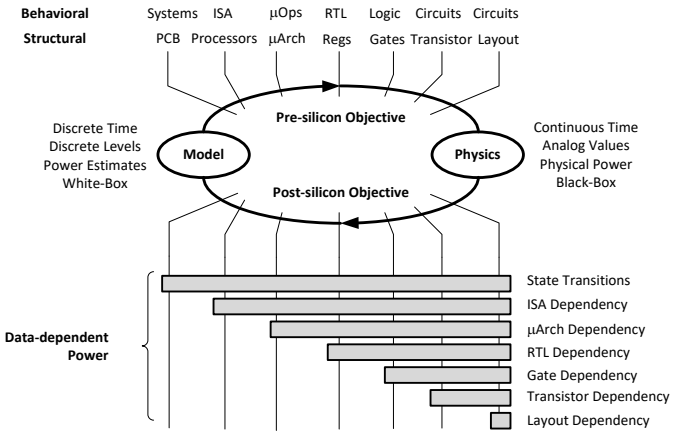

Figure 2. Side-channel leakage modeling requires abstraction from the true physical source of side-channel leakage, thereby also abstracting some sources of side-channel leakage. Pre- and post-silicon side-channel leakage modeling both aim at building a model that reflects the true physical source of side-channel leakage, but approach the problem from opposite sides.

- Present a taxonomy for the published tools, bringing forward their main innovations and potentials.
- Outline existing challenges and promising new research directions.

Our main classification of tools in pre-silicon and post-silicon for leakage detection, summarized in Figure 2, is based on the source of information for building the model which the tool uses. There are two primary sources for such information at a high level: measurements taken from the device and the device's design specifications. These two sources determine the two main axes along which we classify the tools.

The first axis specifies the relationship between the model and the hardware. In all tools, the model aims to predict the leakage from the device. However, *post-silicon* tools build the model, at least partially, based on measuring the device and using this measurement to predict future behaviour. In contrast, *pre-silicon* tools use information about the device's design to predict leakage without observing the actual device. This distinction is not arbitrary. Pre- and post-silicon tools serve different purposes and have different capabilities. Pre-silicon tools aim at detecting leakage early, before the production of the device, allowing the designer the opportunity to modify the design before investing in manufacturing. Post-silicon tools, in contrast, start from a given device and aim to determine the leakage that a change in condition, typically a new software, will cause. Pre-silicon tools typically operate under a white-box scenario, where the model is created from the device's design. They, therefore, include information typically only available to the manufacturer or trusted clients. On the other hand, post-silicon tools operate under a black- or gray-box scenario, where the tool operator does not have the entire device description. However, because these tools have access to actual measurements, they can detect leakage that is not apparent from the design documents.

The second axis we use is the level of abstraction of the model. Development of a hardware device typically proceeds along a sequence of refinements, commonly captured in the Gajski-Kuhn Y-chart [32]. With each refinement, the abstraction level decreases, and more details about the target device are generated. The level of abstraction used for building the model has significant implications on the tool's capabilities. The more refined the model is, the more leakage it can detect [3]. This is particularly relevant for pre-silicon tools, where modeling at one level cannot detect leakage caused by features that are only determined at more advanced levels of abstraction. For example, pre-silicon tools that model at the Register Transfer Level (RTL) cannot detect *glitches*, which depend on timing information only available at the logic level abstraction. Correspondingly, modeling at a high level of detail requires both access to the design documents and a considerable investment of computational and time resources. Figure 2 shows the primary abstraction levels in the design of hardware devices and the types of data dependencies apparent at each level. Finally, we note that because post-silicon tools also draw on information measured from the concrete device, such tools can detect leakage at a lower level of abstraction [61, 73].

Through the classification and analysis of the published tools, we identify the potential and the challenges we face when searching for appropriate solutions. There are significant differences between existing tools, and some problems cross-cut across the domain. In particular, most tool proposals include an evaluation of the effectiveness of the tool. However, many of these evaluations are not transferable across tools. Thus, it is impossible to compare the effectiveness of tools within the domain. The problem is exacerbated by the need to satisfy multiple aims, including simulation and detection accuracy, ease of use, and computational complexity.

We believe the work presented in this paper is of interest for two groups with distinct goals and challenges. The first group consists of hardware designers creating side-channel secure designs and side-channel hardened implementations at the pre-silicon stage. The second group are security researchers implementing a hardened side-channel cryptographic algorithm at the post-silicon stage. Arguably, it might be stated that such tooling could help an adversary interested in extracting a key from a specific device. However, we believe that the risk is somewhat limited. The tools we describe aim at assisting designers in identifying the root cause of leakage. Attackers are less interested in the cause of a leak and are more focused on recovering the key. In summary, the contributions of this work are:

- We investigate an emerging research area on automated tools for side-channel leakage detection and propose a system for classifying such tools. (Section III).
- We survey and analyze post-silicon (Section IV) and pre-silicon (Section V) tools, identifying both achievements and challenges.
- We explore the cross-cutting questions by evaluating tools across the domain. (Section VI)
- We identify open problems and directions for future

research on the design of automated tools for leakage detection. (Section VII)

II. Background

Throughout this paper, we use the term *simulated trace* to denote a time series generated with a commercially available tool as part of an EDA toolchain. We use the term *estimated trace* to refer to a time series derived through a custom-defined, so-called "selection function", which using a leakage model, maps sensitive values to specific predictions, typically used to estimate the power or the EM consumption in the context of side-channel evaluation. Finally, we use *measured trace* to denote the traces acquired from a physical device with the help of an oscilloscope and EM probes. The term *leakage simulator* stands for a device emulator connected to a leakage model, which creates a simulated or estimated trace.

Leakage detection seeks evidence of sensitive data dependencies in the measured traces. The tools typically used for leakage detection are hypothesis testing. Due to hypothesis testing's intrinsic nature, it is only possible to confirm the leaks' presence (and not the absence). *Leakage verification* aspires to identify the cause of a leak. The most straightforward way to verify a leak is to exploit it using an attack-based evaluation. Alternatively, it is also possible to specify a set of rules that violate the algorithm's safe run assumptions, e.g., register reuse by mask shares from the same family. To determine the leak's cause, a careful investigation of the hardware and software's internal working is required. Finally, *leakage mitigation* will remove the cause of the leak.

A. Differential Power Analysis

The threat of Differential Power Analysis (DPA) attacks became evident already in the '90s by demonstrating how cryptographic keys can be extracted from embedded devices, e.g., smartcards, by merely observing a side-channel such as timing or power [46, 47]. A DPA attack that is using Pearson correlation for the key recovery is often called Correlation Power Analysis (CPA). Commonly used metrics to evaluate the DPA attack's performance are *guessing entropy* and *success rate*. The former represents the averaged key rank computed for all key candidates and the success rate of a side-channel the attack is defined as the probability that the secret target key is ranked first among all key guesses by a score vector. *Signal-to-Noise ratio* (SNR) [54] allows designers of cryptographic algorithms to verify that the combination of countermeasures they have chosen to implement in their device provides the required resistance against DPA attacks.

B. Profiled Attacks

There exists a difference in the approach taken by DPA attacks to another class, so-called *profiled attacks*. The latter, often referred to as a two-stage attack, assumes an "open" device (or a copy of it) for the profiling stage in which most of the attack work is done, while the critical recovery stage requires only a few measurements or, in some cases, a single measurement [84]. Examples include template attack [21], stochastic models [70], and recently machine learning-based attacks [50, 53].

C. Leakage Assessment methodology

Test Vector Leakage Assessment (TVLA) [39] is one of the most popular methods for leakage detection due to its simplicity and relative effectiveness. It is based on statistical hypothesis testing and comes in two flavors: *specific* and *non-specific*. The 'fixed-vs-random' is the most common non-specific test and compares a set of traces acquired with a fixed plaintext with another set of traces acquired with random plaintext. In the case of a specific test, the traces are divided according to a known intermediate value tested for leakage. In both cases, Welch's two-sample t-test for equality of means is applied for all samples in the measured trace set. A difference between two sets larger than a given threshold is taken as evidence for the presence of a leak. TVLA is creatively applied in many forms for both pre-silicon and post-silicon evaluation.

III. Pre- and Post-silicon Modeling of Side-channel Leakage

A fundamental property of power-based (or other) side-channel leakage is that its origin is a byproduct of the physical implementation of computations. While the leaked information may relate to any higher level form of computing – such as cryptographic software – the observation of power-based side-channel leakage requires access to the physical implementation of the cryptographic computations. Nevertheless, there are ample opportunities for such observations, including remote observation of power consumption. Recent works such as PlunderVolt [57] and Screaming Channels [20] have demonstrated that this remote access can be implemented in a multitude of ways.

The physical effects of computing are not harmful to computer system security by themselves; such effects only become side-channel leakage when an association can be made between the physical side-channel leakage and a high-level property in the computation stack. Hence, fundamental to every side-channel attack is the association of a high-level model with its physical implementation. The accuracy of this association determines the success of the attack. Therefore, side-channel leakage modeling is of great interest because it builds insight into the link between physical side-channel leakage and a high-level property.

The complexity of side-channel leakage modeling stems from the fact that computing infrastructures are implemented over many layers of design abstraction. The physical effects of computing are hidden by choice, for design efficiency and convenience. At higher levels of abstraction, it is difficult to understand or anticipate the physical effects of side-channel leakage. There are two distinct flavors of the side-channel leakage modeling problem, and we identify them as pre-silicon modeling and post-silicon modeling. Pre-silicon modeling predicts physical side-channel leakage based on high-level design information such as detailed hardware descriptions. Pre-silicon modeling is a task faced by the hardware designer of a secure

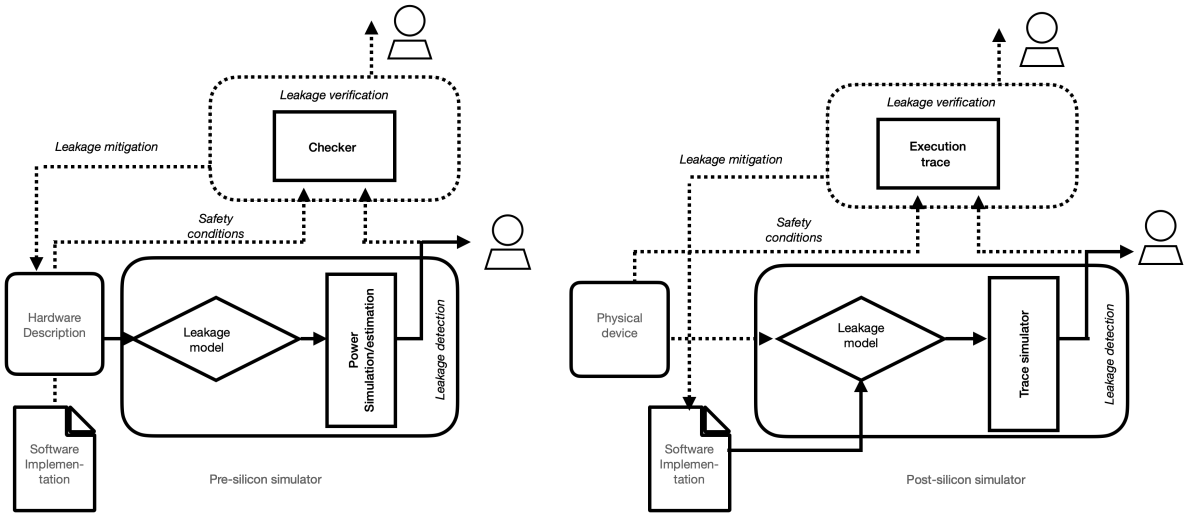

Figure 3. High-level architecture of a pre and post silicon simulator. With continuous line, we denote the essential components required for leakage detection and, with a dotted line, the optional but helpful functionality of verification and mitigation. One of the evident difference between the two is the object of the simulation. Post-silicon simulators, see Section IV, are used for securing software implementations; when present, the physical target generates a more precise leakage model. Pre-silicon simulators, see Section V, are used to secure a hardware target, which in some cases will be tested in combination with a software implementation.

chip. Post-silicon modeling estimates the properties of the higher abstraction levels in the computation stack based on the physical observation of side-channel leakage. Post-silicon modeling is the task faced by the software programmer who aims at writing side-channel secure software for an off-the-shelf processor.

Figure 2 is a synopsis that summarizes the ideas of the following paragraphs. Pre-silicon modeling typically operates under a white-box assumption, where the implementation details of the computing stack are known, even when the exact path towards physical implementation has not been completed. The Gajski-Kuhn Y-chart defines the typical abstraction levels in computer hardware and identifies a computing system's behaviour and structure. At the lowest, most detailed abstraction level, the structure is captured by the chip layout, which is then abstracted into less detailed structures such as a transistor netlist, gate netlist, register-transfer description, processor block diagram, up to system-level structural descriptions. Similarly, analogue continuous-time circuit semantics is captured at the most detailed level, which is then abstracted into logic levels and clock cycles, register transfers, processor instructions, up to system-level task descriptions. The computer chip under construction is decomposed into each of these behavioural and structural forms until a design is obtained at the most detailed level, ready for tape out. With pre-silicon side-channel leakage modeling, a designer tries to predict the physical computer chip's side-channel leakage using any design descriptions available.

Post-silicon modeling operates under a black-box assumption, meaning that the implementation details of the computing stack are only partially known, even as a physical artifact (such as a processor) is available. Post-silicon modeling establishes a high-level model that enables reasoning about side-channel leakage; this high-level model has to be expressed at the abstraction level required for side-channel analysis. Post-silicon modeling is helpful when designing programmable and configurable systems. Post-silicon modeling works along the same behavioral and structural abstractions as pre-silicon modeling.

The bottom of Figure 2 identifies some of the most important sources of data-dependent power dissipation. In the broadest sense, any data-dependent power dissipation can result in power-based side-channel leakage. An important observation in the abstraction levels of computation is that each level adds its own form of data-dependent power dissipation to the overall data-dependent power dissipation. The physical measurement of side-channel leakage is the sum of all these effects in addition to measurement noise. Models at higher design abstraction levels therefore lose some details of the side-channel leakage, and hence every design abstraction level deserves verification of side-channel leakage properties. Some examples of known sources of power-based side-channel leakage given below serve to illustrate this point.

- State transitions, in the form of intermediate computation results, are the most traditional source of side-channel leakage, and they are often directly used in formulating a side-channel attack.
- When software executes on a processor, data-dependent power dissipation can occur because of dependencies in the instruction set - such as the dependency of power consumed by an instruction on similar operands.
- Similarly, the micro-architecture can add additional register transfer level dependencies when operands travel in the internal processor architecture.
- At gate level and below, detailed implementation effects lead to glitches, a non-linear source of data-dependent

power dissipation.

- Transistor circuits dissolve to convenient logic threshold levels from 0 and 1 into an analog range of voltages. Circuit effects such as static power leakage, which is invisible at the gate level, leads to data-dependent power dissipation.
- Finally, the physical geometries of the layout lead to interactions between neighboring wires and parasitic effects in the power distribution, both of which can cause data-dependent power dissipation.

To be useful, a leakage model must be "correct" to accurately reflect reality and be informative to be useful for key recovery. As in [61], we distinguish between *value* and *distance*-based leakage models. The leakage model is value-based if it takes as arguments the set of intermediate values of a cryptographic algorithm. Typical examples include the popular Hamming-weight (HW), Hamming-distance (HD), identity model (ID), or least-significant bit (LSB). A leakage model is *distance-based* if it takes as parameters any pairwise combination of the intermediate values [61]. Barthe et al. [7] give three examples for distance-based leakage, as follows: (1) *transition* leakage effect, such as a generalized Hamming distance leakage model; (2) the *revenant* leakage effect, where sensitive data from past executions may come back and influence the current instruction; and (3) the *neighbouring* leakage effect, which captures the event where accessing or processing of a data storage unit, may trigger a leak from a seemingly unrelated data storage unit [7]. In the rest of the paper, we will use the following categories for leakage models:

- BLACK-BOX typically value-based, there is no information about the placement of the circuit elements or the routing signal between them is determined using little or no specific information about the physical target on which the algorithm will run. This typically corresponds to ISA level information or architecture details - in the form of high-level code assembly instructions or low-level machine code. This leakage model is sometimes referred to as *behavioral level simulation*.
- GRAY-BOX combines the value with partial distance-based leakage models. Partial micro-architectural details of the physical target are used to derive the leakage model, obtained by either reverse engineering the micro-architecture or profiling the physical target using specialized equipment.
- WHITE-BOX Full knowledge of the target, access to RTL gates, or layout description of the target. We include in this model both *gate-level* simulations, where the instantaneous power consumption of a circuit is modeled as "toggle count", calculated as the (weighted) sum of the number of transitions in each gate, and the *transistor level* simulations where the power consumption is modeled as a set of differential equations.

IV. POST-SILICON TOOLING: STATE OF THE ART

The most basic functionality of a post-silicon simulator is that of leakage detection, Figure 3 (left). There are two main approaches for this task. The first, and by far the most common, mimics leakage detection in real traces, by first generating a set of simulated traces and then applying a leakage detection method (e.g. TVLA [39]). Generating realistic simulated traces depends to a great extent on the amount of information available about the target end device. The second approach uses *safety checks* to identify undesired interactions of sensitive variables. To verify the leakage of an implementation and to identify the location of the leakage, the structure or architecture of the target device must be known [61]. Leakage mitigation requires the ability to modify the target device by reprogramming or reconfiguring [73].

Table I presents a taxonomy of tools available for post-silicon side-channel evaluation, listed in chronological order. Based on their capabilities, we classify the tools in three categories: detection (D), verification (V) and mitigation (M), which we discuss subsequently in detail. The supported leakage model (LM) is mentioned explicitly as it has an important impact on the effectiveness of the tool. At one end of the spectrum, the common black-box leakage models, such as Hamming weight or Hamming distance, can be applied independently of the intended physical target. They only require a high-level description of the implementation and give a rough estimate of the actual power or EM consumption. This is enough to model the data dependencies during the execution and may give valuable insights into value-based leakage. At the other end of the spectrum, gray-box leakage models learn from the intended target's behavior by acquiring traces from the actual implementation, making the analysis specific to a particular sequence of instructions.

The amount of information and the degree of control of the end-device available when building the simulator (end-device), determines the capability of the tool (D,V, or M) and has an impact on how fined-grained is the leakage model of the end-device. On the down-side, the more information a simulator captures about the target, the less portable to other architectures it will be.

Some tools are designed without a physical end-target [63, 66]. As such, they are not necessarily useful for post-silicon evaluations *only*. We chose to list them in this category as these tools can be used for early design stages of software implementation, as the typical use case for post-silicon evaluation tooling. As masking is one of the key countermeasures for software implementations, we find it important to specify whether a tool has been demonstrated on a masked implementation (Masking).

A. Leakage detection at post-silicon stage

In the following subsections, we describe post-silicon tooling for leakage detection, leakage verification, and leakage mitigation. For each tool, we highlight achievements and challenges.

Achievement: Generic framework for modeling of micro-architectural details in a black-box model. Debande et al. [25], the first to point out the significance of deriving realistic leakage models, propose the first gray-box trace

Table I
TOOLS FOR POST-SILICON SIDE-CHANNEL EVALUATION (CHRONOLOGICAL ORDER)

Name	Year	LM	End-device	D	V	M	Masking	Side-channel	Open-Source
PINPAS [27]	2003	●	smartcards	✓	–	–	✗	power	✗
Inspector SCA [66]	2007	●	not relevant	✓	–	–	✗	power	✓ $
Oscar [76]	2009	●	AT90XX, ATmegaXX	✓	–	–	✗	power	✗
Debande [25]	2012	◐	not specified	✓	–	–	✗	power	✗
Gagnerot [31]	2013	●	Risc-V(not specified)	✓	–	–	✗	power	✗
SILK [80]	2014	●	ATmega328P	✓	–	–	✗	power	✓
SLEAK [82]	2014	●	*ARM Cortex A8*	✓	✓	–	✓	register access	✗
Reparaz [63]	2016	●	not relevant	✓	–	–	✓	power	✗
SAVRASCA [81]	2017	●	ATMega163	✓	✓	–	✓	power	✓
ASCOLD [61]	2017	◐	ATMega163	–	✓	✓	✓	ILA	✓
ELMO [55]	2017	◐	ARM Cortex M0	✓	–	–	✗	power	✓
ELMO* [73]	2019	◐	ARM Cortex M0	✓	–	–	✓	power	✓
ROSITA [73]	2019	ELMO*	ARM Cortex M0	✓	✓	✓	✓	power	✓
EMSIM [72]	2020	◐	Risc-V(custom)	✓	–	–	–	EM	✗

We use ● to represent a black-box leakage model (LM); ◐ to represent a gray-box model. We tick the box for masking for the tools that report a case study involving a masked algorithm.

simulator. The simulator uses stochastic modeling to fit a function of state bits and state transitions. It starts from a fixed model and estimates the state transitions for each bit in the target register.

We consider ELMO [55] to be the first truly gray-box simulator for the ARM-Cortex M0/M4 family. It brings two remarkable innovations. The first one is a portable framework for building a leakage model rather than estimating the coefficients for a fixed model, as is the case for stochastic modeling. ELMO achieves this by considering the contribution of a parameter before deciding to include it in the model. The second is the extension of the model to support *sequence dependency*. The key observation is that different instructions' power consumption depend on the instructions executed before that [78]. ELMO is instruction-accurate, which has the advantage of easily allowing the identification of a leaky instruction. When modeling the power consumption, the authors disregard the high registers (`r8-r15`) because those are only used for fast temporary storage. Following a cluster analysis to group "similar" instructions (i.e., which leak information in the same way), the authors identify five groups, which interestingly also correspond to the same processor component: ALU instructions in one group, shift instructions as another group, load and stores that interact with data as two or more groups, and multiply instruction with a distinct profile due to its single cycle implementation. The authors find a remarkable consistency in the data-dependent leakage of different physical boards. The only downside for extending the proposed framework to other architectures is the amount of human effort which has to be put into it. ELMO is open-sourced and publicly available.

ELMO* [73] improves the leakage model of ELMO by capturing interactions that span multiple cycles. ELMO [55] is augmented to account for the storage elements, which play a critical role for the security of masked implementations. A novel feature of ELMO* [73] is a battery of small code sequences which can be used to systematic highlight the interaction of instructions via storage elements. The idea for finding the hidden storage elements is generic and can be applied for any other architecture.

Achievement: EM simulation at post-silicon stage. While both EM and power are important for SCA evaluations, modern micro-controllers, with multiple power domains, can be immune to power side -channels but can leak in the EM domain. EMSIM [72] is the first EM simulator built for a custom 32-bit base Risc-V implementation. The simulator supports data and instruction dependent activities and micro-architecture effects such as pipeline stalls, cache miss, and misprediction. A comparison between the simulated and measured EM signal is performed to determine the simulated signal's quality. The result shows the two signals to be very close. To reduce the number of instructions that need to be fitted, the authors do perform clustering of the power consumption of the instructions and observe that the Risc-V ISA can be clustered into seven categories when the instructions have similar operands.

Interestingly, the same phenomena was identified and used by ELMO [55] to simplify the modeling of the target. The manufacturing variability (same manufacturer, different physical boards) on the model accuracy was investigated. Although the authors detected a slight shift in the clock frequency for different boards, the conclusion is that this shift has no impact on the accuracy of the simulator. Furthermore, the model accuracy was explored as a function of board variability (same core, different manufacturer). The conclusion was that for different designs, only the baseline amplitude and activity factors should be retrained.

Challenge: EM modeling requires access to design details. The authors of EMSIM [72] show that having access to micro-architectural details is critical for achieving good accuracy for EM simulations.

Challenge: Access to tools. Although in recent years the stan-

dard practise is to open-source tools (as the authors are mostly from academia), for many of the earlier tools [27, 31, 76] we only have a description from the authors about the tools capability and innovations, which in many cases provides limited information.

Challenge: The existing simulators target relatively simple architectures. As can be seen from Table I the most commonly targeted end-devices are ATMegaXX or ARM-Cortex M0, which are simple, in-order, single-core CPUs. In addition, the simulators which target the micro-architecture of the design require significant effort to port to other architectures. With no access to design information, the task of the designer is to reverse-engineer the micro-architecture details. The most notable exception is the tool called SLEAK [82] which is showcased on the ARMCortex A8, a modern and complex processor. To access the values of intermediate state, SLEAK uses Gem5 as an open source- full system simulator. Power consumption is modelled with a black-box leakage model.

Challenge: Lack of integration with formal verification tools. Oscar [76] is a power simulator tool, designed for 8-bit Atmel AVR micro controllers, constructed in a pure functional style (`OCaml` language) with the intention of integrating the power simulator tool into formal proofs of resistance. The default supported leakage models are the black-box leakage models, but the tool allows the monitoring of successive microprocessor states or partial states (e.g., a register or memory). Oscar is the only leakage detection tool in the post-silicon category which aims to bridge the gap between physical security and formal verification.

B. Leakage verification at post-silicon stage

Achievement: Verification at high abstractions levels is effective. After detecting the presence of a leak in an implementation, mitigating the leak requires discovering the cause of the offending instruction. The tools which are capable of locating the leak must possess the capability of mapping a time sample in the power trace to the precise instruction responsible corresponding to that time sample. The alternative to the tools which identify the cause of the leak is either an educated guess or trial and error.

The tool proposed by Reparaz [63] can detect leakage in masked implementations of high-level code. The tool has a trace generation feature that uses a black-box leakage model. For each time sample, the value of the processed variable is also recorded. A fixed-vs-fixed test is used for leakage detection. As the tool records which variables correspond to the leaky sample, it is possible to locate the source of the leakage.

SAVRASCA [81] uses the tracing feature of the SimulAVR tool and is suitable for the analysis of code for the AtmelAVR family. The simulator can produce both power and execution traces. To create power traces, the leakage model is computed during each memory unit access (available via the tracing feature of SimulAVR), making a difference between a write and read access. The separation allows for different leakage functions depending on the type of access (Hamming weight for reading and Hamming distance for writing). The simulator produces one power sample per executed instruction and does not consider the memory unit's address.

Challenge: Mapping a time sample in a measured trace to the corresponding executed instruction is difficult. The tools which can map a time sample in a measured trace to the corresponding instruction of the executed code are limited and typically fall in two categories: either a machine emulator such as Qemu [9], Gem5 [16] or SimulAVR [68] or specialized hardware, such as JTrace Pro[1] for hardware which provides advanced debug probe supports the tracing features of ARM Cortex Cores. Therefore, verification depends on whether a machine emulator supports the board, or the board has a tracing pin availa

C. Leakage mitigation at post-silicon stage

While verification tools still rely on a human expert to remove leakage, mitigation tools aim to apply the fixes automatically.

Achievement: Generic code-rewrite for trace simulators. ROSITA [73] is a rule-driven code rewrite engine that patches the code automatically once leakage is detected. ROSITA starts with a (masked) implementation of a cryptographic algorithm, cross-compiled to produce both the assembly and the binary executable. A very compelling feature of ROSITA is that it extends an existing leakage detection tool, ELMO [55] to report instructions that leak secret information. The new detection framework (ELMO*), uses the binary file to detect leakage and identify the offending machine instruction, ROSITA then applies a set of rules that replace the leaky instruction with an equivalent one (functionally) that does not leak. The process is repeated until no more leakage is detected. The ROSITA concept can be extended to other architectures.

Challenge: Distance-based leakage is platform specific. ASCOLD [61] can verify code and mitigate leakage by checking violations of the independent leakage assumption (ILA), responsible for reducing the actual security order of an implementation. The tool takes as input the assembly file of a masked implementation and a configuration file which describes the initial state of the system, e.g. the registers or addresses in the memory which contain the secret shares or sensitive variables. The output of the simulator is the line number and the rule that was violated by the program. The algorithm verifies for *overwrite effects*, so that shares from the same family are not written in the same register, it checks the *memory remenant effect* of the load/store instructions and *neighbouring leakage* where an access of a register will cause a leakage in a unit elsewhere. Unfortunately, these effects depend on the architecture and are only observable throughout an implementation. As in the case of ROSITA [73], ASCOLD assumes a detailed description of the micro-architectural effects of the target board, which requires intensive effort to determine.

[1] https://www.segger.com/products/debug-probes/j-trace/models/j-trace/

V. Pre-silicon tooling

The world of pre-silicon side-channel leakage verification tooling, while at first glance relatively rich (see Table II), is limited by the fact that these tools are not public and the results cannot be reproduced. Additionally, the reported results consider a prototype chip design for reporting, which might significantly adapt to a complex chip design. While any measurable data-dependent power dissipation may be a source of side-channel leakage, there is a trade-off between the precision and the simulation speed. Higher abstraction levels (ISA, RTL) will offer quick power estimates. Still, they will miss SCA leakage sources, while lower abstraction levels (gate layout) consume more simulation time but are more precise. In the following, we summarize the simplifications made by the different tools to estimate the power or the EM consumption and the target used for modeling. Figure 3 (right), a conceptualized architecture of a pre-silicon tool, will guide our narrative.

A. Leakage detection at pre-silicon stage

Achievement: Power estimates at different abstraction levels speed-up the generation of power signals. NCSIM [30] is the first white-box simulator to estimate DPA resistance at the gate level. It neglects the static power consumption, but it can model glitches and early propagation when timing information is added to the model. The simulator supports several power estimation functions. The simplest one is *transition counting* (each time the signal changes its logical state, the power consumption, at the current point, is increased by one). This information is present in the VCD files, and a Matlab toolbox was used to estimate the power trace. A more refined power model is the *random transition weighting*, which captures the fact that the load capacitances are not identical for every gate (the experiments are performed on a dual-rail precharged logic style) implemented by adding a random weighting to each transition. Finally, *back-annotation of the transition weighting* can also be added by extracting the full-chip layout's parasitic information. In terms of speed, NCSIM reports that a transistor level simulation of an internal MOV operation including the initialization phase of the core, has taken about 10 hours vs, the logic simulation that finishes in a few minutes.

PLAN/PARAM [29] estimates the power consumed by a module as an aggregation of the power consumed by all signals present in the module. The assumption to support this choice is that the power consumption of a k-bit signal is proportional to its Hamming weight. The benefit of this approach is that the whole Shakti-C processor's evaluation takes about 5 hours compared to a post-and-place route simulation that would take a complete month.

RTL-PSC [43] estimates the power profile of a hardware design using functional simulation at the RTL level. To ensure a fast framework, the power profile is estimated based on the number of transitions using the Synopsys VCS tool. Compared to state-of-art, RTL-PSC claims two advantages. The first is the ability to quantitatively and accurately assess power side-channel leakage and the second is speed. The evaluation time for AES-GF is 43.6 minutes and for AES-LUT for about 24 minutes. The same evaluation at the gate level would take about 31 hours, while the authors estimate it would take more than 1 month at the layout level.

ACA [85] uses a gate-level model for a target design, which is typically available after logic synthesis, as well as a side-channel leakage model. The latter leakage model is common in DPA attacks. The objective of ACA is to identify the gates in the design that are contributing the most to the selected side-channel leakage model. ACA introduces the Leakage Impact Factor (LIF), a numerical score that reflects the relative contribution of a single gate to side-channel leakage. The authors demonstrate for several different application scenarios (an AES engine, a Sparc-V CPU) that only a handful of gates can be identified as majority contributors to side-channel leakage. This leads to the mitigation strategy of selective replacement, in which only those gates with high LIF are substituted and protected by side-channel resistant versions.

CASCADE [74], a white-box simulator that aims to speed up the time to market and reliability of the secure design uses an extended version of the Hamming Distance model, named the *Marching-Stick Model (MSM)* to model power consumption. MSM is a generic model that captures the asymmetry between rising and falling edges, unlike the simple toggle counting. When the tool is used to check second-order security using both the marching stick leakage model and the PrimeTime with PX, we see that both results indicate the presence of second-order leakage, as expected. The second use case is the Boyar-Peralta AES S-box [24, 37]which was found to be leaky [83]. To demonstrate the presence of the mentioned vulnerability, the setup used 10 million traces, but it takes CASCADE only 30 min to find the indicated vulnerability in the gate-level netlist. The third analysed S-box implementation is an in-house implementation of a PRESENT S-layer in WDDL, [77] a dual-rail logic style. The analysis is done at both gate-level netlist and place and route, using both the simulated power model with PrimeTime and estimated power with the marching stick model. In all cases, no leakage is detected.

B. Leakage verification at pre-silicon stage

Achievement: Formal verification at gate level. Both SCRIPT [58] and COCO [38] allow formal verification at gate level. However while SCRIPT targets crypto cores, COCO is aimed at formally verfying a masked software implementation on a given hardware platform. The approach used for the formal proof, described below, is also different. SCRIPT [58] takes as input a gate-level description of a crypto core, and a *target function*, which can be a potential target for side-channel attacks if it satisfies the following four properties:

- a function of the secret,
- a function of controllable inputs,
- a function with confusion property,
- and functions with the divide-and-conquer property.

The target registers (which store the target functions' output values) are identified using information flow tracking. To

Table II
TOOLS FOR PRE-SILICON SIDE-CHANNEL VERIFICATION (CHRONOLOGICAL ORDER)

Name	Year	Input	End-device (description)	LM	D	V	M	Masking	Target	Side-Channel	Open-Source
NCSIM [30]	2007	gate	SCARD [40]	S	✓	–	–	✗	CC	power	✗
AMASIVE [44]	2013	RTL	–	E	✓	–	–	✗	CA	power	✗
MAPS [49]	2018	ISA	ARM CortexM3	E	✓	✓	–	✓	CA	power	✓
KARNA [75]	2019	layout	AES [11], SIMON [2]	S	✓	✓	✓	✗	CC	power	✗
RTL-PSC [43]	2019	RTL	AES-GF [1], AES-LUT [69]	S	✓	–	–	✗	CC	power	✗
PARAM [29]	2020	gate	Risc-V(ShaktiC)	E	✓	✓	–	✗	ED	power	✗
COCO [38]	2020	gate	Risc-V	–	✓	✓	–	✓	CA+ED	power	✓
ACA [85]	2020	gate	Risc-V(LEON3)	S	✓	–	–	✗	CC	power	✗
SCRIPT [58]	2020	gate	AES-GF [1], AES-LUT [69]	–	✓	✓	–	✗	CC	power	✗
CASCADE [74]	2020	gate	ASIC (custom)	E	✓	–	–	✗	CC	power	✓

Input specifies the abstraction level for the input to the simulator, for the end-device specified by the column *End-device*. For the leakage model*(LM)* we have two options: simulated power (S) or estimated power(E.). Columns (D,V,M and *Masking*) are defined as in Table I
The *Target* column describes what is being simulated: the crypto core (CC), the cryptographic algorithm(CA) or the end-device(ED).

estimate the power consumption, SCRIPT uses a vectorless power estimation technique, which requires the verification engineer to define the signal probability (the percentage of the analysis when the input is driven at high logic levels) and the toggle rate (the rate at which the net or logic element switches compared to its input) of the primary input ports. The vectorless power analysis can be performed using PrimeTool (Synopsys) or XPE (Xilinx), which returns the total estimated power for the design. COCO [38] allows security proofs at gate level for the execution of masked implementations. The proofs are done in the time-constrained probing model (proposed in the same paper), which simulates the hardware of pipelined circuits. As a first step, the masked assembly implementation is executed on a given CPU hardware design, the result being a *trace execution* which contains the concrete values for all CPU control signals in each clock cycle. The location of the registers and memory cells which contain shares of sensitive values are annotated. Next, COCO uses correlation sets and a SAT solver to find the exact gate and execution cycle where the implementation leaks. In essence, COCO verifies adherence to the following two design principles: first, that shares of the same secret must not be accessed within two successive instructions and second, that a register or memory location which contains one share must not be overwritten by its counterpart.

Achievement: Open-source tooling. The first open-source verification tool is MAPS [49], a power simulator for the ARM Cortex M3 series. It takes in assembly code and targets pipeline leakage, as they combine operand values from consecutive instructions. For identifying power leakage, it uses the fixed vs-random t-test [39]. Using information from an ARM Cortex M3 HDL file, the cause of a leak, the registers related to the data path are isolated and traced. To simplify the tracing and reduce the resulting power trace, the operation is restricted to registers which deal with sensitive values; as such, the `r15` which holds the program counter is ignored. Furthermore, the three ALU registers are discarded as they are used for multicycle instructions, which are not commonly used during crypto algorithms. MAPS is not cycle-accurate and traces only registers of the ARM Cortex M3 core, other registers located outside the core are not considered. While both MAPS and COCO [38], the two open-source verification tools, can be used to verify masked software implementations, MAPS supports only the ARM Cortex M3 platform, while COCO can handle any given netlist.

Achievement: SCA resilience for non-cryptographic designs. PARAM [29] is a microprocessor design hardened for side-channel resistance. It is a trace simulator, but it features a Power attack Leakage Analyzer (PLAN) module, which works on the RTL source code to identify the target microprocessor's leaking module. The running example is the open source of the Shakti-C Risc-V processor. The processor is represented as a netlist of functional modules such as the main pipeline, the ALU unit, data cache, instruction cache, etc. For a given module, leakage is estimated from the signals (wires and registers) associated with the module. Once the power consumption is estimated, SVF (Side-Channel Vulnerability Factor) [26], is used to calculate the leakage. The authors do mention among the caveats that PLAN can only capture linear leakage and leakage due to dynamic power consumption (also the most exploited one in side-channel attacks).

Challenge: Differentiating the merits of the tools is difficult. If we compare the architectures of the tools, Figure 3 (right), we notice that SCRIPT and COCO use the user input to define safety conditions for the underlying architecture. To determine the presence of a leak, MAPS and PLAN/PARAM employ empirical leakage detection strategies, t-tests [39] and SVF [26] respectively. Furthermore, if we compare the input of the simulator, we observe that while MAPS and COCO target, a masked software implementation, SCRIPT aims at the verification of crypto cores and PLAN/PARAM aims to secure the end-device or non-cryptographic implementation. If we explore the dimension of security guarantees, SCRIPT and COCO aim for a formal proof, while MAPS and PLAN/PARAM take an empirical testing approach.

C. Leakage mitigation at pre-silicon stage

Achievement: Leakage mitigation tool at layout level. KARNA [75] identifies vulnerable gates in the design and then

re-configures them. The chip is partitioned into small cells, and a TVLA assessment is done for each cell. To estimate the power consumption of a gate, KARNA uses commercial tooling, and the tool reveals leakage specific to a given area. ***Challenge: Removal of leaks are done empirically.*** KARNA [75] identifies side-channel security from the netlist by computing TVLA scores at the gate level. The assumption is that not all gates on the netlist contribute equally to the side-channel leakage. To remove a vulnerable region, KARNA first determines if the gate is critical (if it can not undergo any more configurations). If not, it will replace the vulnerable gate with its next low-power configuration. KARNA will also optimize the gate parameters such that the overall security of the design improves while keeping the design requirements such as area, power, and delay.

VI. Evaluation criteria

All research contributions to put forward a tool for leakage detection, evaluation, or mitigation typically contain a part dedicated to the tool's validation and experimental results. This is important as it demonstrates the practical value of the tool in identifying and removing side-channel leaks. In this section, we examine the different metrics used to determine how the leakage simulator's output can be used for developing a side-channel hardened target. Among the presented evaluation techniques, we identify four distinct groups:

1) Comparison between simulated/estimated and reference traces, relevant mostly for trace simulators.
2) Evaluations of *leakage model's quality,* most often through comparing it to a simpler model.
3) Evaluation by case studies, where the simulator is used to find and/or fix side-channel leaks.
4) *Usability measures* explore related benefits for using a simulator, typically by comparing the performance of the tool with either a measurement setup for the post-silicon simulators or the power simulation techniques for the pre-silicon simulators.

It is important to mention that most simulators are evaluated using a subset of the groups mentioned above.

A. Metrics for evaluating the output of leakage simulators

The question answered by the metrics we place in this group can be summarized as: *how close is the output of a leakage simulator to the reference traces?* The implicit assumption is that the closer the leakage simulator's output to the reference traces, the more it can be trusted. Different boards exhibit different behaviours [14, 65] that may cause slight variations between traces measured from different boards. This difference is relevant when matching simulated traces with measured traces. However, we found no reference which evaluates the differences in traces between different sets of traces from different boards.

Challenge: Many of the measures provide evidence based on visual comparison. As the number of samples between the simulator output and the reference traces is different, it is difficult for the two sets of data to be compared directly. Here we give some examples:

- *Dynamic Time Warping.* To evaluate SILK [80], the distance between a set of measured traces and the simulator's output is computed. Dynamic Time Warping (DTW), computes the distance between two temporal series, even when they have different elements. Several leakage models are used to generate simulated traces. DTW is instantiated with two different distance metrics, Euclidean and correlation-based distance. For a fair comparison, noise is added to the simulated traces. Notably, the two used distance metrics give different results regarding what constitutes a more realistic trace set.
- *Power correlation.* Although the term is defined in [85], this is a well-known measure used when analyzing side-channel leakage. Here, it assumes the knowledge of the key and requires the explicit choice of a target variable and leakage model. The authors of ELMO [55] compare (visually) the correlation traces produced by predicting the ELMO leakage model on the measured traces with the same model's prediction on the traces produced by the simulator. The approach used then is to apply the same measure to both sets and show how the trend matches.
- *Leakage detection comparison.* The de facto technique for leakage detection is TVLA [39]. It comes then as no surprise that the practise of comparing a t-test trace produced by a leakage simulator with the t-test trace produced by the reference traces, using either a fixed-vs random test [55, 73] or a fixed-vs-fixed test [63] is widespread. For computing the t-test traces, the datasets for both the simulator and the reference traces are prepared in advance by feeding the cryptographic algorithm with a fixed or a random plaintext. Next to its simplicity, this test is non-specific, meaning that it does not target one specific variable. The classical application is a visual check to ensure that the simulated and reference t-traces match the identified leaking points.
- *DPA performance.* To demonstrate the merit of a profiled simulator, Debande et al. [25] compares the evolution of a DPA attack, in terms of guessing entropy, between a set of measured traces, a set of traces generated by a profiled leakage model, and a set of traces generated by a non-profiled model. Although the attack performance of the non-profiled leakage model is superior to the that of the measured traces (and to that of the traces produced by the profiled leakage models), the conclusion is that the profiled models which closely follow the behaviour of the measured traces are preferred. The main indicator for a desirable output is to match the trend of guessing entropy and the simulated traces' success rate with one of the real traces. We note that DPA is capable of tracking the performance for one target intermediate value. The same metric is used for evaluating the performance of NC-SIM [30] where the similarity of a DPA attack performed on the internal MOV operation is used to demonstrate

the advantage of using a simulator for the design of a secure chip. The authors of PARAM [29] also use DPA to compare the reference architecture's resistance before and after applying hardening countermeasures.

Aside the fact that visual comparison is a subjective measure, typically due to space limitations, we only see one example of how these match. It is fair to point out that quantifiable measures for assessing leaks are sparse and not widely accepted in the side-channel community.

Challenge: No consensus for what is a good procedure in comparing simulated vs reference traces. An open question is whether the leakage model should be included in the evaluation. While most of the measures mentioned above do include a leakage model, the authors of EMSIM [72] use *normalized cross-correlation* to show how well the simulated traces match the reference traces without relying on a specific leakage model. As it computes the average cross-correlation between individual clock cycles, this metric's output is a quantifiable measure, EMSIM reports an impressive 94.1% accuracy in simulating side-channel signals across all possible instruction combinations. The requirement to use this measure is to precisely identify the correct clock cycles, which requires knowledge of the design details. Another open questions, is whether the existing measures are enough or new ones are needed and some contributions propose new metrics for evaluating the two sets. Although initially a measure of the side-channel created by a single instruction the *Signal Available to Attacker*, (SAVAT) [19] is used in EMSIM [72] to measure the similarity between the simulated and reference traces. For a set of six instructions, the pairwise score SAVAT is computed, and the results between the simulated and reference traces are compared and found to be very close. SCRIPT [58] uses *side-channel vulnerability* (SCV), which is the equivalent of SNR [54] at the pre-silicon stage, as it requires a small number of traces to compute and differs from SNR, according to its authors by a scaling factor. RTL-PSC [43] combines KL-divergence with SNR to identify vulnerable design blocks, while SLEAK [82] uses mutual information between the sensitive values processed by the algorithm and the value or state of a system component during the execution binary.

B. Metrics for evaluating the quality of the leakage model

It is common for the post-silicon evaluation tools [25, 55, 73] which propose a complex leakage model to compare its performance with simpler or previously known leakage models. In pre-silicon simulators, we count in this category the metrics which quantify the leakage identified by the simulator, compared to an ideal case, where the target does not leak information.

Achievement: Empirical evidence that gray-level leakage models are superior to black-box leakage model. Although the statement above might seem naive, the question if *it is worth to invest time and effort in creating sophisticated gray leakage models* is valid. To prove the merit of the ELMO leakage model [55], the authors use *power correlation*, to compare the predictions of a simple leakage model (Hamming weigh) with the prediction of leakage produced by the ELMO model. The comparison is made by computing the correlation traces produced by both leakage models on *the same* reference traces. The result, Figure 4 in [55] shows that the peaks in the correlation trace generated by the ELMO model are more clearly defined compared to those produced by the simple model. Additionally, the ELMO leakage model generates more peaks. The conclusion drawn by the authors is that the simple leakage model captures only a portion of the true leakage and should not be relied upon when protecting sensitive data. The same metric is used by ELMO* [73] to shows it superiority to ELMO [55] leakage model. Debande et al. [25] use guessing entropy to compare the performance of a simple black-box leakage model with a profiled leakage model.

Achievement: Metrics to quantify leakage of hardware components. *Side Channel Vulnerability Factor* (SVF) [26] quantifies the correlation between attacker observation patterns and patterns in victim execution. The insight is that side-channel attacks rely on recognizing leaked execution patterns. SVF quantifies the patterns in attackers' observations and measures the correlation with the victim's actual execution patterns and captures systems' vulnerability to side-channel attacks. SVF quantifies the overall 'leakiness' of a particular system but does not provide insight into the cause. . In PARAM [29], SVF is used to quantify the amount of leakage in the target processor's different components.

Leakage Impact Factor (LIF) is used by the authors of ACA [85] to quantify the similarity of the activity profile of a single gate or cell to a high-level leakage model used by DPA. The LIF is further weighted by the relative power consumption of the cell in the overall design. The LIF directly quantifies the contribution of a single gate or cell to the side-channel leakage, and it is used in ACA to rank the gates of the design from leaky to least leaky.

C. Case studies

In this section, we explore the answers to the question: *How effective are the existing tools at verifying and eliminating SCA vulnerabilities*? To answer this question, most case studies will showcase the tool's ability to verify leakage (find the cause for producing leakage) and mitigate the leakage (eliminate it manually or automatically). The depth and breadth of the presented cases vary greatly between the different contributions. While some verification tools showcase a toy example [27], others explore a wide variety of scenarios. For a leakage verification tool, the case study will reproduce a known flaw, introduce one in an otherwise secure design, or seldom use the tool to find a new unknown vulnerability. In the following, we present a representative selection of use cases.

1) Software implementation of cryptographic algorithms: To show its effectiveness, the tool of Reparaz [63] is used to test the security of six high-order implementations. The first is a "smoke test" where the aim is to reproduce the flaw found by [67] for the first-order masking scheme proposed by [4]. The simulator performs six fixed-vs-fixed TVLA tests and reports that five of the six tests show leakage. The same

test is applied to the first-order secure table recomputation scheme proposed by [22] which, as expected, "on the strength of the found evidence" is reported to be secure. Next, the tool is used to reproduce the second-order flaw spotted by [23] in the masking scheme proposed by [67]. The authors report that it takes the tool 5 seconds to find the flaw, including the time to spot the cause. Next, the tool is used to reproduce the third-order vulnerability spotted by [22] in the technique used for refreshing the mask in the scheme proposed by [67]. The authors report that the flaw was found in less than one second. A more difficult case for the tool (200 million traces and eight hours of simulation) is to reproduce the observations from [64] on higher-order implementations [15]. The authors also report a new second-order flaw found in [71], which, once found, is easily proved.

2) Hardware implementation of cryptographic algorithms: KARNA [75] is put to the test of securing three open-source cores. The first is a bit-serial implementation of the Simon block cipher [2] [2]. The tool performs three iterations, using TVLA with 8000 inputs and removes the leaking gate. The netlist is synthesized at 28 nm cell size. The second is a PRESENT core[3], a minimal design which after one iteration achieves side-channel resilience. The third is optimizing the AES core [11][4] after which a DPA attack is performed after place and route stage, and the design is shown not to leak information with 100 000 traces. Compared with the unoptimized AES synthesized design, the result is shown to reveal the correct key byte at approx 2k traces. Karna can achieve a user-specified security level in all three designs with no impact on the delay or the number of gates and a 20% increase in the utilization area.

RTL-PSC [43] is evaluated on two AES designs based on Galois Field (GF) [1] and Look-up Table(LUT) [69]. The tool is used to identify the leaky modules in the design, using a combination of Kullback-Leibler divergence and success rate. The validation of the tool is done on both gate-level netlist simulation and FPGA simulation. The comparison is made by computing the Pearson correlation of the RTL simulations (produced by the tool) with the gate-level netlist's KL-divergence trace.

AMASIVE [44] is showcased against an unprotected hardware implementation of the PRESENT cipher [17], for the first and the last round. The tool identifies hypothesis functions for the HW and HD leakage model. To confirm the tool's attack vector, a CPA attack is mounted, and the key is recovered within 10 000 traces.

3) Hardening of non-cryptographic hardware: PARAM [29] is used to produce a hardened implementation of a Shakti-C [33] core. The approach used to secure the software AES implementation deviates from the classical application of countermeasures. The authors identify and remove the leakage from each hardware component of the microprocessor. The DPA results in terms number of traces vs correlation score are used to show the hardened microprocessor's resilience.

4) Combination of software implementations running on a physical target: The case study for MAPS focuses on showcasing the design flow with a tool such as MAPS. An example is a naive implementation of SIMON [8] protected with Trichina AND gate [79], which aims to minimize the number of execution cycles. The authors simulate the implementation of this cipher with and without pipeline leakage. Using a random vs fixed t-test, it is shown that both instances leak information. In the next iteration, the leakage is due to the reuse of register registers. In this version, the remaining leakage comes from the two pipeline registers. After fixing the two pipeline registers' leakage, the t-test traces obtained from the simulated traces show no leaky points as a final step t-test is also performed on a set of reference traces measured from a physical implementation, which show a few remaining leaky points.

SAVRASCA [81] is used to find a flaw in the AES implementation used for version 4 of the DPA contest[5]. Using the tool, the authors noted that the simulated traces' size depended on the value manipulated by the microcontroller, even though the implementation was running in a constant number of cycles. Analysing the implementation, the authors found that the number of register access depended on the manipulated value. As a response to this finding, the new version of the DPA contest v4.2 fixed the implementation and released a new set of traces.

ASCOLD [61] is used to develop a hardened 1st-order, ISW-based [45] S-box with a bit-sliced RECTANGLE implementation [86]. The performance of the hardened implementation is investigated for two different security objectives. The first is an *efficient* implementation where the registers are cleared on a need-to basis to avoid overwrite and remnant effects. The second is *conservative* implementation, which adds to the efficient implementation of dummy instructions' insertion through register/memory clearing. The strengths of hardening were evaluated using non-specific TVLA.

COCO uses the open-source IBEX core[6], part of the PULP platform [28] and the OpenTitan [52] project. The main application of COCO is the verification of a masked software implementation running on hardware specified at gate-level netlist. A considerable selection of masked circuits, which cover domain-oriented masking (DOM) AND gate [41], Ishai-Sahai-Wagner (ISW) AND [64], Threshold implementation(TI) AND [59] and larger implementations DOM Keccak S-box [13, 42], DOM AES S-box [37] and the Trichina AND gate [79] are presented to demonstrate the effectiveness of the tool. The scenarios cover two case studies with intentionally injected vulnerabilites [41, 42]. The implementations also cover second-order security [41, 42] and third-order security [41] (for a complete overview see Table 3

[2]reference implementation, https://opencores.org/projects/simon_core
[3]reference implementation https://opencores.org/projects/present
[4]reference implementation https://opencores.org/projects/tiny_aes
[5]http://www.dpacontest.org/v4/rsm_doc.php
[6]reference implementation https://github.com/lowRISC/ibex

in [38]). To show that COCO's output leads to practical secure implementations, a sample of the verified netlist of IBEX cores and the DOM Keccak S-box [42] is mapped onto a Xilinx Spartan-6 FPGA. The design is then evaluated using TVLA and shown to not leak information at 100000 traces.

D. Evaluating Usability

Next to security-related advantages, emulators also offer important *usability* advantages for the implementation and testing of a (masked) cryptografic primitive. The use of a simulator encourages testing at different development stages, allowing the removal of vulnerabilties as early as possible in the development cycle. For post-silicon, the cryptographic implementation can be tested at source- code level [63], assembly level [55] or compiled binary. For pre-silicon, the design can be tested at RTL, gate or transistor level.

- *Ease of use and convenience.* Building a setup for side-channel measurement is costly as it requires time, equipment, and expertise for preparing the target. Furthermore, the implementation and testing of a cryptographic primitive requires advanced skills in cryptographic engineering. The simulator is easy to use and reduces the effort of a task that is highly iterative and requires (manual) effort.
  Application: post-silicon.
- *Fast(er) Development Cycles.* The speed of simulating the power consumption of a design is increasing as we progress with the design stages. As the complexity of the design increases, [73] mentions a 4.5–7 speed increase compared to real hardware. As an extreme example [74] mentiones that for a fully-unrolled AES (which exceeds the security budget of most embedded device), simmulation and analysis of one million traces, can be done in 4 hours on a 8-thread worskstation. At the same time we have ample evidence (Section VI-C) that leakage at early design stages does have a positive effect. The flexibility in making design changes and the leakage assessment time depends on the design stage [43]. In other words, while it is relatively easy to make changes at RTL-level, only small changes are possible at the layout level, while at post-silicon level, no changes in the design can be made.
  Application: pre and post-silicon.
- *Cost.* Faster development cycles, and assurance in the final product ultimately increase the time to market of the product, which save costs or give a significant competitive advantage. For the post-silicon development stage, the idea of performing side-channel evaluation without the need of a lab and a team of experts available for assistance will make side-channel evaluation more accessible.
  Application: pre and post-silicon.

VII. Open problems in designing side-channel simulators

Open problem: Fine grained leakage models for complex architectures, with no design information. For post-silicon simulators, the main challenge for developing sophisticated gray-box leakage models is the lack of micro-architectural details. For an accurate simulator at the post-silicon stage, the techniques and methods available aim *to find and add*, the elements which capture leaks to gain precision. Today, we know how to model simple micro-architectural features, such instruction-dependent activities in different pipeline stages, add support for sequence dependency (the power consumed by an instruction depends on the other instructions in the pipeline), or find hidden storage elements. However, creating a model such as ELMO [55] is prohibitively effort-intensive, even for relatively simple processors (in-order, no cache). Furthermore, it is unclear how to capture micro-architecture events characteristic for more complex processors, e.g. pipeline stalls, misprediction or cache miss.

Open problem: EM simulation. The one EM simulator we have, EMSIM [72], a trace simulator, was constructed for a relatively simple, custom-made Risc-V processor. To build the simulator, EM measurements of the physical device are required. Based on the experimental results aimed to assess how the simulated signal degrades when key micro-architectural features are omitted, the authors conclude that it is not possible to build an EM simulator without access to micro-architectural information. This means that we may only be able to build EM simulators for open-source hardware. If the presence of a physical target is a must for constructing EM simulators, it could explain the fact that there are no EM simulators at pre-silicon stage.

Open problem: Metrics for quantifying potential side-channel leaks at micro-architectural level. While the importance of micro-architecture features on the security of masked implementation has been shown to be crucial [5, 56] most contributions focus on one platform and zoom in on the feature which leaks side-channel information on the studied platform. Even if we would know how to model every micro-architecture event, there is no widely accepted measure in the side-channel community to quantify leaks.

Open problem: Benchmark existing simulators. While some simulators are being open-sourced, the specific scenarios for which the tools are intended for, are hard to compare. An alternative, is to agree on representative, public data-sets and could be used to evaluate the potential of a tool, even when the tool is not open-sourced.

Open problem: Lacking case studies for asymmetric cryptographic implementations. All case studies we encountered in the existing literature are focused on the implementation of symmetric algorithms.

Open problem: Composability of pre-silicon simulators. While power simulation techniques are known and used, the primary application is heat dissipation and battery life. The goal of power estimation applied for side-channel evaluation is to capture the instantaneous power consumption. One of the important requirements is to process large quantities of data-dependent simulations. This category's main challenge is to *find and remove* the design specifications that do not contribute to leaks, such as to gain speed. Today we have SCA-aware design-tools for every design stage. It has also

been proven that removing vulnerabilities as early as possible in the design stage, is a sound engineering practice. Although creating a pre-silicon trace simulator requires significant effort, the existing tooling is fragmented and not reusable.

Acknowledgments

This research was supported by The Australian Research Council Discovery Early Career Award DE200101577 and Discovery Project DP210102670, the Blavatnik ICRC at Tel-Aviv University, a gift from Intel Corporation, and the National Science Foundation Grant 1931639.

References

[1] Aoki Laboratory. Galois field based AES Verilog design. http://www.aoki.ecei.tohoku.ac.jp/crypto/web/cores.html, 2007. (accessed: 15.03.2021).

[2] Aydin Aysu, Ege Gulcan, and Patrick Schaumont. SIMON says: Break area records of block ciphers on FPGAs. *IEEE Embed. Syst. Lett.*, 6(2):37–40, 2014.

[3] Melissa Azouaoui, Davide Bellizia, Ileana Buhan, Nicolas Debande, Sébastien Duval, Christophe Giraud, Éliane Jaulmes, François Koeune, Elisabeth Oswald, François-Xavier Standaert, and Carolyn Whitnall. A systematic appraisal of side channel evaluation strategies. In *SSR*, pages 46–66, 2020.

[4] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In *Asiacrypt*, pages 758–775, 2012.

[5] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS*, pages 64–81, 2015.

[6] Manuel Barbosa, Gilles Barthe, Karthikeyan Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *IEEE SP*, pages 123–141, 2021.

[7] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):189–228, 2021.

[8] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC*, 2015.

[9] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *USENIX ATC*, 2005.

[10] Daniel J. Bernstein. Cache-timing attacks on AES, 2005. Preprint available at http://cr.yp.to/papers.html#cachetiming.

[11] Daniel J. Bernstein and Peter Schwabe. New AES software speed records. In *Indocrypt*, pages 322–336, 2008.

[12] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES power attack based on induced cache miss and countermeasure. In *ITCC*, pages 586–591, 2005.

[13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The making of KECCAK. *Cryptologia*, 38(1):26–60, 2014.

[14] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS*, 2020.

[15] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *Asiacrypt*, pages 326–343, 2014.

[16] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, August 2011.

[17] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *CHES*, pages 450–466, 2007.

[18] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *ESORICS*, pages 355–371, 2011.

[19] Robert Locke Callan, Alenka G. Zajic, and Milos Prvulovic. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *MICRO*, pages 242–254, 2014.

[20] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *CCS*, pages 163–177, 2018.

[21] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, pages 13–28, 2002.

[22] Jean-Sébastien Coron. Higher order masking of look-up tables. In *Eurocrypt*, pages 441–458, 2014.

[23] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *Fast Software Encryption*, pages 410–424, 2014.

[24] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020. ISBN 978-3-662-60768-8.

[25] Nicolas Debande, Maël Berthier, Yves Bocktaels, and Thanh-Ha Le. Profiled model based power simulator for side channel evaluation. Cryptology ePrint Archive, Report 2012/703, 2012. https://eprint.iacr.org/2012/703.

[26] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. In

ISCA, pages 106–117, 2012.

[27] Jerry den Hartog, Jan Verschuren, Erik P. de Vink, Jaap de Vos, and W. Wiersma. PINPAS: a tool for power analysis of smartcards. In *SEC*, pages 453–457, 2003.

[28] ETH Zurich. Pulp platform. https://pulp-platform.org/. (accessed: 15.03.2021).

[29] Muhammad Arsath K. F, Vinod Ganesan, Rahul Bodduna, and Chester Rebeiro. PARAM: a microprocessor hardened for power side-channel attack resistance. In *HOST*, pages 23–34, 2020.

[30] Omnia S. Fadl, Mohamed F. Abu-Elyazeed, Mohamed B. Abdelhalim, Hassanein H. Amer, and Ahmed H. Madian. Accurate dynamic power estimation for cmos combinational logic circuits with real gate delay model. *Journal of Advanced Research*, 7(1):89–94, 2016.

[31] Georges Gagnerot. *Étude des attaques et des contre-mesures associées sur composants embarqués*. PhD thesis, Université de Limoges, 2013.

[32] Daniel Gajski and Robert H. Kuhn. New VLSI tools - guest editors' introduction. *Computer*, 16(12):11–14, 1983.

[33] Neel Gala, Arjun Menon, Rahul Bodduna, G. S. Madhusudan, and V. Kamakoti. SHAKTI processors: An open-source hardware initiative. In *VLSI Design*, pages 7–8. IEEE Computer Society, 2016.

[34] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[35] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptographic Engineering*, 8(1):1–27, 2018.

[36] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO*, pages 444–461, 2014.

[37] Ashrujit Ghoshal and Thomas De Cnudde. Several masked implementations of the Boyar-Peralta AES S-Box. In *Indocrypt*, pages 384–402, 2017.

[38] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. Cryptology ePrint Archive, Report 2020/1294, 2020. https://eprint.iacr.org/2020/1294.

[39] G. Goodwill, J.J.B. Jun, and P.Rohatgi. A testing methodology for side channel resistance validation. *NIST non-invasive attack testing workshop*, 2018.

[40] Grant agreement ID: 507270. Side channel analysis resistant design flow. https://cordis.europa.eu/project/id/507270, 2004. [Online; accessed 3-Apr-2021].

[41] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *TIS*, 2016.

[42] Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In *DSD*, pages 205–212, 2017.

[43] Miao Tony He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark Mohammad Tehranipoor. RTL-PSC: automated power side-channel leakage assessment at register-transfer level. In *VTS*, pages 1–6, 2019.

[44] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. *AMASIVE: An Adaptable and Modular Autonomous Side-Channel Vulnerability Evaluation Framework*, volume 8260 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2013.

[45] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[46] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[47] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[48] Juliane Krämer, Dmitry Nedospasov, Alexander Schlösser, and Jean-Pierre Seifert. Differential photonic emission analysis. In *COSADE*, pages 1–16, 2013.

[49] Yann Le Corre, Johann Großschädl, and Daniel Dinu. Micro-architectural power simulator for leakage assessment of cryptographic software on ARM Cortex-M3 processors. In *COSADE*, pages 82–98, 2018.

[50] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *COSADE*, pages 20–33, 2015.

[51] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. A survey of microarchitectural side-channel vulnerabilities, attacks and defenses in cryptography. *CoRR*, abs/2103.14244, 2021.

[52] lowRISC contributors. Open titan. https://opentitan.org/. (accessed: 15.03.2021).

[53] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *SPACE*, pages 3–26, 2016.

[54] Stefan Mangard. Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In *CT-RSA*, pages 222–235, 2004.

[55] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *USENIX Security Symposium*, pages 199–216, 2017.

[56] Lauren De Meyer, Elke De Mulder, and Michael Tunstall. On the effect of the (micro)architecture on the development of side-channel resistant software. Cryptology ePrint Archive, Report 2020/1297, 2020.

[57] Kit Murdock, David F. Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *IEEE S&P*, pages 1466–1482, 2020.

[58] Adib Nahiyan, Jungmin Park, Miao Tony He, Yousef

Iskander, Farimah Farahmandi, Domenic Forte, and Mark Mohammad Tehranipoor. SCRIPT: a CAD framework for power side-channel vulnerability assessment using information flow tracking and pattern generation. *ACM Trans. Design Autom. Electr. Syst.*, 25(3):26:1–26:27, 2020.

[59] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *Information and Communications Security*, pages 529–545, 2006.

[60] Dan Page. Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169, 2002. http://eprint.iacr.org/2002/169/.

[61] Kostas Papagiannopoulos and Nikita Veshchikov. Mind the gap: Towards secure 1st-order masking in software. In *COSADE*, pages 282–297, 2017.

[62] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. In *Smart Card Programming and Security*, pages 200–210, 2001.

[63] Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. In *FSE*, pages 204–222, 2016.

[64] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *CRYPTO*, pages 764–783, 2015.

[65] Unai Rioja, Lejla Batina, and Igor Armendariz. When similarities among devices are taken for granted: Another look at portability. In *Africacrypt*, pages 337–357, 2020.

[66] Riscure. Inspector sca. https://www.riscure.com/security-tools/inspector-sca, 2002. [Online; accessed 7-Apr-2021].

[67] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES*, pages 413–427, 2010.

[68] Theodore Roth and Klaus Rudolph. SimulAVR. https://www.nongnu.org/simulavr/, 2001. [Online; accessed 13-Apr-2021].

[69] Satoh Lab. Lookup table based AES Verilog design. http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html, 2017. (accessed: 17.03.2021).

[70] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES 2005*, pages 30–46, 2005.

[71] Kai Schramm and Christof Paar. Higher order masking of the AES. In *CT-RSA*, pages 208–225, 2006.

[72] Nader Sehatbakhsh, Baki Berkay Yilmaz, Alenka G. Zajic, and Milos Prvulovic. EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals. In *HPCA*, pages 71–85, 2020.

[73] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. Rosita: Towards automatic elimination of power-analysis leakage in ciphers. In *NDSS*, 2021.

[74] Danilo Sijacic, Josep Balasch, Bohan Yang, Santosh Ghosh, and Ingrid Verbauwhede. Towards efficient and automated side-channel evaluations at design time. *J. Cryptogr. Eng.*, 10(4):305–319, 2020.

[75] Patanjali SLPSK, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. Karna: A gate-sizing based security aware EDA flow for improved power side-channel attack protection. In *ICCAD*, pages 1–8, 2019.

[76] Céline Thuillet, Philippe Andouard, and Olivier Ly. A smart card power analysis simulator. In *CSE (2)*, pages 847–852, 2009.

[77] Kris Tiri, David D. Hwang, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. Prototype IC with WDDL and differential routing - DPA resistance assessment. In *CHES*, pages 354–365, 2005.

[78] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *VLSI Design*, pages 326–328, 1996.

[79] Elena Trichina, Tymur Korkishko, and Kyung Hee Lee. Small size, low power, side channel-immune AES coprocessor: Design and synthesis results. In *Advanced Encryption Standard – AES*, pages 113–127, 2005.

[80] Nikita Veshchikov. SILK: high level of abstraction leakage simulator for side channel analysis. In *PPREW@ACSAC*, pages 3:1–3:11, 2014.

[81] Nikita Veshchikov and Sylvain Guilley. Use of simulators for side-channel analysis. In *EuroS&P Workshops*, pages 104–112, 2017.

[82] Dan Walters, Andrew Hagen, and Eric Kedaigle. SLEAK: A side-channel leakage evaluator and analysis kit. Technical report, The MITRE Corporation, November 2014. [Online; accessed 8-Apr-2021].

[83] Felix Wegener and Amir Moradi. A first-order SCA resistant AES without fresh randomness. In *COSADE*, pages 245–262, 2018.

[84] Leo Weissbart, Lukasz Chmielewski, Stjepan Picek, and Lejla Batina. Systematic side-channel analysis of Curve25519 with machine learning. *J. Hardw. Syst. Secur.*, 4(4):314–328, 2020.

[85] Yuan Yao, Tarun Kathuria, Baris Ege, and Patrick Schaumont. Architecture correlation analysis (ACA): identifying the source of side-channel leakage at gate-level. In *HOST*, pages 188–196, 2020.

[86] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. *Sci. China Inf. Sci.*, 58(12):1–15, 2015.