SoK: Design Tools for Side-Channel-Aware Implementations

Ileana Buhan
Radboud University
The Netherlands
ileana.buhan@ru.nl

Lejla Batina
Radboud University
The Netherlands
lejla@cs.ru.nl

Yuval Yarom
University of Adelaide
Australia
yval@cs.adelaide.edu.au

Patrick Schaumont
Worcester Polytechnic Institute
USA
pschaumont@wpi.edu

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.

CCS CONCEPTS

• **Hardware** → *Power estimation and optimization*; • **Security and privacy** → **Side-channel analysis and countermeasures**.

KEYWORDS

Side-Channel, Emulators, Power Analysis

ACM Reference Format:
Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. 2022. SoK: Design Tools for Side-Channel-Aware Implementations. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30-June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3488932.3517415

ASIA CCS '22, May 30-June 3, 2022, Nagasaki, Japan

ACM ISBN 978-1-4503-9140-5/22/05.
https://doi.org/10.1145/3488932.3517415

1 INTRODUCTION

When a computing device operates, it interacts with its physical environment. In his seminal work, Kocher [53] showed that the power consumption of a device correlates with 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) [39, 74], timing [11, 19, 71], microarchitectural components [14, 41, 58], and even acoustic and photonic emanations [42, 55].

In response to developments in attacks, the community has developed several methodologies for leakage detection and assessment. They involve collecting side-channel 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. First, the physical implementation may not exist, for example, when verifying side-channel leakage while designing a novel chip. Second, the physical implementation details may be unknown when verifying side-channel leakage from a third-party chip. Moreover, detecting, verifying, and mitigating side-channel leaks require expert knowledge and expensive equipment.

In recent years, an alternative approach for evaluating device resilience to side-channel attacks has emerged. 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.

Early attempts of creating such tooling and the increased recent efforts directed to this purpose prove the appeal of automation as

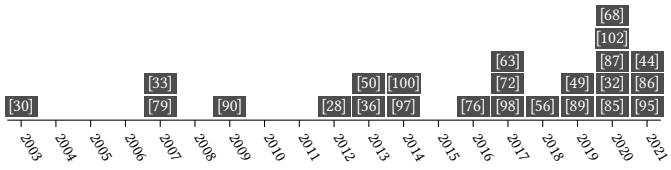

Figure 1: Distribution of papers which propose a new tool per year (for multiple publications on the same tool, we cite the most recent one).

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, there is no comparison of tools across the domain, preventing a straightforward assessment of the benefits of each tool. A comprehensive study of automated tooling available for computer-aided cryptography covering the design, functional verification, and implementation-level security of digital side channels was recently published [6]. While the study covers tooling for side channels in shared resources (e.g., cache), it excludes physical side channels, particularly power consumption and EM radiation. This work covers the gap and presents a taxonomy of state-of-the-art tooling for protecting against these physical side-channel attacks. We chart the landscape and present a coherent view of current advances in power and EM leakage evaluation automation. Specifically, we

- Give a comprehensive survey of the available tooling for leakage detection, verification, and mitigation while clarifying the current capabilities and limitations.
- Present a taxonomy for the published tools, bringing forward their main innovations and potentials.
- Outline challenges and promising new research directions.

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 the domain. In particular, most tool proposals include an evaluation of the tool's effectiveness. However, many of these evaluations are not transferable. Thus, it is impossible to compare the effectiveness of tools within the domain. The problem is intensified 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 of people with distinct goals and challenges. The first group consists of hardware designers creating side-channel secure designs and side-channel hardened implementations. The second group is security researchers implementing a hardened side-channel cryptographic algorithm on an existing third-party chip. One could argue that such tooling could help an adversary interested in extracting a key from a specific device. However, we believe that this risk is 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 power and EM side-channel leakage detection and propose a system for classifying such tools (Section 3).
- We survey and analyze post-silicon (Section 4) and pre-silicon (Section 5) tools, identifying achievements in this area.
- We explore the cross-cutting questions by evaluating tools across the domain (Section 6).
- We identify open problems and directions for future research on the design of automated tools for leakage detection (Section 7).

2 PRELIMINARIES

In this section, we summarize terminology and other essential concepts in side-channel leakage modeling.

2.1 Side-Channel Leakage

The power consumption and EM signals emitted from a device correlate with the computation steps and the data processed when performing computation. Consequently, monitoring the physical properties of devices can reveal information about the operations they carry and the data they process. To perform a side-channel attack, an attacker attempts to correlate the physical properties with secret values processed by the device. For example, the attacker might look for a statistical difference between the means of power consumption when processing two different secret values [45, 54].

Over the years, multiple sources of leakage have been identified. The amount of power required to maintain a signal depends on the logical state of the signal. Thus, signalling 1 requires a different power level than signalling 0. Consequently, the power consumption of a circuit directly correlates with the data the circuit processes. Other effects, such as differing power required for changing the logical state of a signal, the variations in signal propagation times over the circuit, or cross-capacitance effects, all contribute to the instantaneous power consumption of the device and therefore correlate with the data it processes. We further discuss these and other effects in Section 3.2.

Multiple approaches have been suggested for side-channel protection [22, 23, 35, 52, 64, 65, 92]. In particular, masking represents each secret value using multiple shares [22, 52] such that the leakage of any subset of the shares is independent of the secret value. However, theoretically secure defences often fail in practice, necessitating practical evaluations of devices and software to ensure protection against side-channel leakage [5, 67].

2.2 Leakage Models

Side-channel attacks and defences both rely on *leakage models*, which describe the salient features of the measurements, abstracting over the physical details of the device. To be useful, a leakage model must be "correct" to accurately reflect reality and be informative to be useful for key recovery. As in [72], 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), 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 [72]. 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 *neighboring* 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].

2.3 Traces

A measurement of a device creates a *trace*, which represents the measured physical quantity as a function of time. Throughout this

paper, we use the term *measured trace* to denote the traces acquired from a physical device with the help of an oscilloscope and EM probes. We use the term *simulated trace* to denote a time series generated with a commercially available tool as part of an EDA toolchain. Finally, the term *leakage emulator* stands for a device emulator, which is a program connected to a leakage model that creates a simulated trace.

2.4 Detection, Verification, and Mitigation

Leakage detection seeks evidence of sensitive data dependencies in the measured traces. The tools 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 (possible) assumptions required by the algorithm to run securely 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.

3 PRE- AND POST-SILICON MODELING OF SIDE-CHANNEL LEAKAGE

A fundamental property of side-channel leakage is that its origin is 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 the physical implementation of the cryptographic computations.

The physical effects of computing are not harmful to computer system security themselves; such effects only become side-channel leakage when an association between the physical side-channel leakage and a high-level property in the computation stack can be made. 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 probability of the attack.

3.1 Our Framework

Figure 2 summarizes the framework that is used to classify the tools for leakage detection. 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 behavior. 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 to detect leakage early, before the device's production, allowing the designer the opportunity to modify the design before investing in manufacturing. Post-silicon tools,

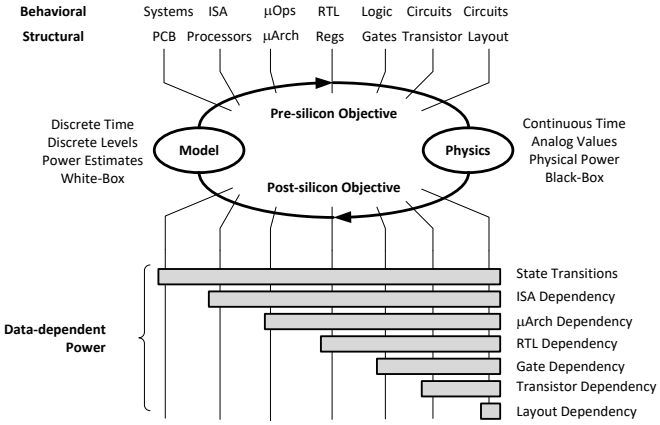

Figure 2: Side-channel leakage modeling requires abstraction from the true physical source of side-channel leakage, thereby also abstracting some sources of side-channel leakage. Pre- and post-silicon side-channel leakage modeling both aim at building a model that reflects the true physical source of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

in contrast, start from a given device and aim to determine the leakage that a change in condition will cause. Pre-silicon tools operate under a white-box scenario, where the model is created from the device's design. They, therefore, include information 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 full 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. The development of a hardware device typically proceeds along a sequence of refinements, commonly captured in the Gajski-Kuhn Y-chart [37]. With each refinement, the abstraction level decreases, and more details about the target device become available. 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 observation is particularly relevant for pre-silicon tools, where modeling at one level cannot detect leakage caused by features chosen at lower levels of abstraction. For example, pre-silicon tools at the Register Transfer Level (RTL) model time with cycle-accurate resolution. Such tools cannot detect leakage caused by sub-cycle timing effects. Correspondingly, modeling at a high level of detail requires 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 [72, 86].

3.2 Causes of Side-Channel Leakage

Power side-channel leakage is data-dependent power consumption during secure computation. Past research demonstrates that no single abstraction level can accurately capture all of the data-dependent power consumption. Data dependencies that cause power-based side-channel leakage may occur at *every* level of abstraction in the system stack. However, dependencies at lower abstraction levels are invisible at higher abstraction, which makes power-based side-channel leakage hard to predict at a system level. This section provides a bottom-up analysis of the causes of side-channel leakage, and we clarify why modeling at every level of abstraction is important. At several instances during the discussion, we mention the secret-sharing (or masking) countermeasure, a popular method to combat side-channel leakage in hardware. A secret bit is split in n shares using $n-1$ random bits to yield an order-$(n-1)$ countermeasure. In a d-probing model, an adversary can take d independent observations. To be secure, the number of shares in a secret-sharing countermeasure must fulfill $n > d$ [52].

The bottom half of Figure 2 lists the data dependencies that are known contributors to power-based side-channel leakage. The vertical lines indicate abstraction levels corresponding to the labels at the top of the figure. The highest abstraction level only captures dependencies from state transitions. On the other hand, the lowest, most detailed abstraction level captures all possible dependencies of power consumption to data, from state transitions down to layout dependencies, which leads to the staircase shape of Figure 2. In the following, we describe the nature of each dependency in a bottom-up manner, from layout-level to system-level.

In CMOS (Complement Metal Oxide Semiconductor) circuits, the mainstream technology for computers, power consumption results from electrical charges that flow when transistors switch. Moving charges requires effort (power) [34]. These charges flow over on-chip wires and through transistor channels, causing additional internal resistive losses that dissipate as heat [75]. We can observe the effects of moving charges in a circuit by measuring the flow (current) through voltage drop over a shunt resistor or using induction effects on a current probe. In addition, moving electrical charges create electromagnetic fields governed by Maxwell's laws. We can capture the electromagnetic fields generated by a circuit using E-field probes or H-field probes [8, 70]. Hence, EM-based side-channel leakage and power-based side-channel leakage have a single physical cause (moving electrical charges).

The power consumption caused by CMOS transistors is proportional to the number of switching events (≈ the amount of charge moved) per transistor per unit of time. However, the size of each electrical charge, and therefore the resulting power consumption depends on the physical topology and size of the transistors and on-chip wires [75]. In addition, second-order parasitic effects, such as the capacitive coupling between neighboring on-chip wires, may cause the amount of charge moved by a switching event to depend on the *joint* value of two neighboring wires. Parasitic coupling can directly degrade the secrecy of secret-sharing schemes when two or more shares become electrically coupled [24].

At gate-level, analog voltage levels become bits, and logic gates provide elementary boolean computations using those bits. Digital gates consume power when they switch, similarly to their atomic component, the transistor. The number of gates driven by a digital gate is called the *fan-out*, and it is a measure of how hard a gate must work during a switching event: gates with a higher fan-out will consume more power during a switching event. The majority of logic circuits operate according to the synchronous paradigm, where clock cycles drive the execution of sequential computations. However, logic gates do not switch exclusively during clock-edge events and each logic gate computes its output with a slight delay called the transition delay. Logic gates deep into a logic network may switch multiple times per clock cycle when their inputs do not arrive at the exact moment. These transient switching events are called *glitches*, and they make up 15–20% of the dynamic power consumption [88]. Glitches depend on the *joint* value of network inputs, and for this reason, they may also potentially degrade the secrecy of secret-sharing schemes [61].

Gate-level logic circuits are abstracted into functional expressions on registers at RTL level and the closely related microarchitecture level. The switching events model power consumption at this level in a record. The Hamming distance between successive values stored in a register approximates these switching events. Storing a secret value into a register will cause side-channel leakage twice, first when loaded and then when a new value overwrites it. To prevent side-channel leakage, the predecessor and successor values of the secret must be uniformly random and independent of the secret, which is a requirement that is hard to achieve in a general-purpose environment. The leakage effect also degrades secret-sharing schemes when two shares occupy the same register during different clock cycles.

At the Instruction-Set Architecture (ISA) level, the dependencies of the microarchitecture disappear, and power is modeled in terms of the visible machine state only, such as programmer-visible registers and memory. The dependencies are similar to those at the microarchitecture level: every store that contains a secret value can leak two times, at the start and the end of the value's lifetime in that store. At System-level, all implementation details have disappeared, and the power consumption is only visible as system-level state transitions, such as in the overall control flow of a software program. At this level, power-based side-channel leakage is a mechanism to implement timing attacks.

Data dependencies on power consumption are a byproduct of the pre-silicon design process. Therefore, it is generally impossible to guarantee that a partially completed design (say, the RTL design of a RISC-V processor) will be free of side-channel leakage later, at a more refined design stage. Instead, in the pre-silicon setting, the design verification for side-channel leakage must, in principle, be repeated at every abstraction level to ensure no new dependencies appear. In the post-silicon stage, all data dependencies on power are present from the start of the design process, but they may be unknown to the application programmer.

We traditionally measure power consumption near the source of consumption, perform side-channel attacks on lab setups with ideal observation conditions. However, that is not a requirement for a successful side-channel attack. Recent work has demonstrated the use of indirect effects of power side-channel leakage when the attacker is co-located on the same chip [83, 104], on the same machine [57], or even within the range of a wireless transmission [21].

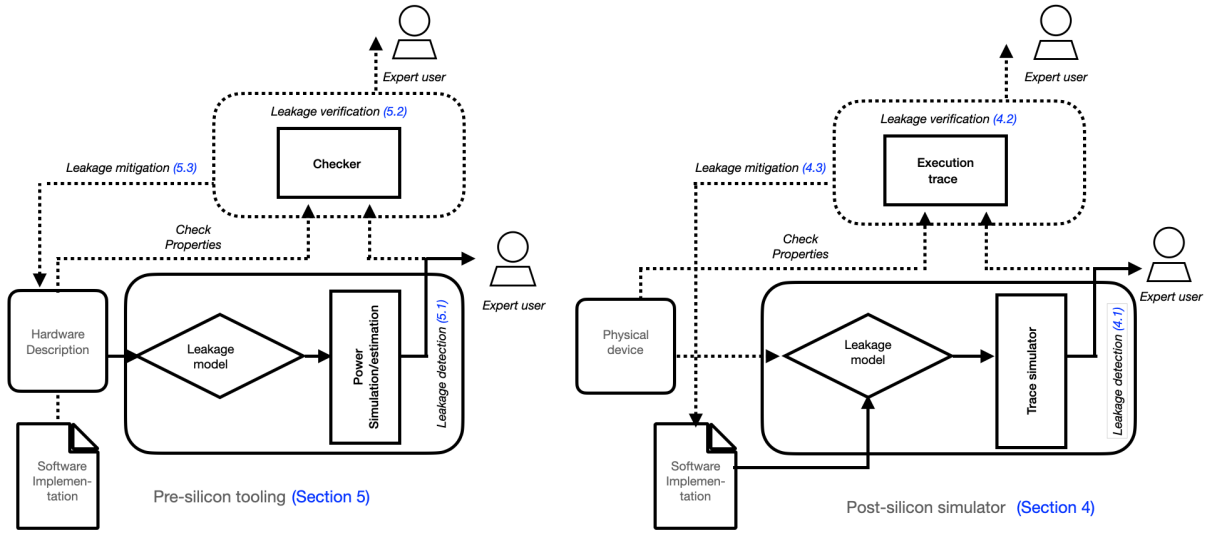

Figure 3: High-level architecture of a pre and post-silicon emulator. With continuous line, we denote the essential components required for leakage detection and, with a dotted line, the optional but helpful functionality of verification and mitigation. One of the evident differences between the two is the object of the simulation. Post-silicon emulators, see Section 4, are used for securing software implementations; when present, the physical target generates a more precise leakage model. Pre-silicon emulator, see Section 5, are used to secure a hardware target, which in some cases are tested in combination with a software implementation.

3.3 Pre- and post-silicon tools for Side-Channel Leakage

Following the two scenarios above, we divide existing tooling into two categories. The first category is *post-silicon*, which given a software implementation of a cryptographic algorithm and (possibly) the end device, aim to produce a power or an EM trace that closely resembles a laboratory measurement performed by expert security analysts. The second category is the *pre-silicon* tooling which can find and remove leaks in crypto cores (hardware implementations) and cryptographic algorithms.

Albeit a generic architecture, which covers all corner cases, is difficult to specify, we attempt in Figure 3 to capture a high-level view of the salient components for both pre-and post-silicon emulator[1]. We observe that tools in both scenarios share the same goals and are used to detect, verify, and mitigate side-channel leaks. During post-silicon evaluation, no changes are possible to the end-device (in most cases, a general-purpose microcontroller). Hence, the typical evaluation target is a software implementation of a cryptographic algorithm. The most basic functionality of a post-silicon emulator is that of leakage detection, Figure 3 (left). There are two main approaches to 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. Test Vector Leakage Assessment i.e. TVLA [45]). Generating realistic simulated traces depends significantly on the amount of information available about the target end device. The second approach is to *check properties* to identify undesired interactions between sensitive variables. To verify leakage and pinpoint the source of the leak, the structure or architecture of the target device must be known [72]. In case of leakage verification *execution traces*, which represent a set of events (such as instruction, register access or other) ordered by the algorithm execution, are used to annotate the traces. Using annotation, we can link the timing of side-channel leakage to algorithmic events, thereby identifying the leakage source. Leakage mitigation requires additionally the ability to modify the target device by reprogramming or reconfiguring [86].

There is more flexibility in making changes to the end-device during a pre-silicon evaluation, so both hardware and software implementations of cryptographic algorithms are in scope, Figure 3 (right). One significant difference between the pre and post-silicon tooling is the amount of information available for deriving a suitable leakage model. Both estimation and simulation of power consumption using commercial tooling are possible.

4 POST-SILICON TOOLING

Table 1 presents a taxonomy of tools available for post-silicon side-channel evaluation, listed in chronological order. Based on their capabilities, we classify the tools into three categories: detection, verification, and mitigation, which we discuss subsequently in detail. We explicitly mention the supported leakage model (LM), which impacts the tool's effectiveness. At one end of the spectrum, standard black-box leakage models, such as Hamming weight or Hamming distance, can be applied independently of the intended physical target. Such black-box leakage models only require a high-level description of the implementation and a rough estimate of the actual power or EM consumption, which is enough to model

[1] we use the term leakage emulator and emulator interchangeably

the data dependencies during the execution and 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 emulator determines the tool's capability (detect, verify, or mitigate) and impacts the granularity of the model of leakage from the end-device. On the downside, the more information an emulator captures about the target, the less portable to other architectures it becomes.

Some tools do not model a physical end-target [76, 79]. As such, they are beneficial for both pre-and post-silicon evaluations.

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 fundamental countermeasures for software implementations, we find it necessary to specify whether a tool was applied on a masked implementation (Masking).

In the rest of this section, we describe post-silicon tooling for leakage detection, verification, and mitigation, highlighting current achievements in each category.

4.1 Leakage detection at post-silicon stage

Generic framework for modeling microarchitectural details. Debande et al. [28] are the first to point out the significance of realistic leakage models and to propose a gray-box trace emulator. The tool 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. The formula for deriving the power trace would remain the same for different physical targets, but the coefficients would be different. We consider ELMO [63] to be the first genuinely gray-box emulator for the ARM-Cortex M0/M4 family, and it brings two remarkable innovations. The first 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. In contrast to the approach used by Debande et al., the formula for deriving power traces depends on the level of details available about the microarchitecture implementation available and the contribution of each (group) of variables to the overall model prediction capability. The second innovation that ELMO brings is the extension of the model to support *sequence dependency*. The critical observation is that the power consumption of different instructions depends on preceding instructions [93]. ELMO is instruction-accurate, which has the advantage of quickly allowing the identification of a leaky instruction. 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 the memory 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 to put into it.

ELMO* [86] improves the leakage model of ELMO by capturing interactions that span multiple cycles. ELMO [63] is augmented to account for the storage elements, which play a critical role in the security of masked implementations. A novel feature of ELMO* [86] is a battery of small code sequences which can highlight the interaction of instructions via storage elements systematically. Marshall et al. [62] extend the idea of discovering sources of leakage by intentionally triggering leakage effects with specially crafted code sequences and put forward 23 new benchmarks covering microarchitecture implementation related to memory such as hidden memory states, sequential access or data-bus width, pipeline registers, control-flow instructions, and the impact of speculative execution in short pipelines.

Reverse Engineering of microarchitectural implementations. While the importance of microarchitecture details in a security analysis has been established [62], access to its implementation is typically not available. The current state of the art allows reverse engineering a commercial ARM-Cortex M3 microprocessor [40]. The authors note that the current methodology involves intensive manual effort. However, it is worthwhile as it shows the importance of capturing microarchitectural effects.

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 [85] is the first EM emulator built for a custom 32-bit base RISC-V implementation. EMSIM supports data- and instruction-dependent activities and microarchitecture effects such as pipeline stalls, cache miss, and misprediction. The designers of the emulator have white-box access to the design details of the target microprocessor. To reduce the number of instructions to be fitted, the authors 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, ELMO [63] used instruction clustering to simplify the modeling of the target. The authors also investigate the model accuracy as a function of manufacturing variability (same manufacturer, different physical boards). 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 emulator's accuracy.

4.2 Leakage verification at post-silicon stage

Verification at high abstractions levels is effective. After detecting a leak in an implementation, mitigating the leak requires discovering the source of the leak. Tools capable of locating the source of a leak can map a time sample in the power trace to the precise instruction corresponding to that time sample.

The tool proposed by Reparaz [76] 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 tool records the value of the processed variable. It then applies a fixed-vs-fixed test to detect leakage. As the tool records which variables correspond to the leaky sample, it can locate the source of the leakage. SAVRASCA [98] uses the tracing feature of the SimulAVR

Table 1: Tools for post-silicon side-channel evaluation (chronological order)

Name	Year	LM	End-device	Detect	Verify	Mitigate	Masking	SC	Open-Source
PINPAS [30]	2003	●	smartcards	✓	–	–	✗	power	✗
Inspector SCA [79]	2007	●	not relevant	✓	–	–	✗	power	✓ $
Oscar [90]	2009	●	AT90XX, ATmegaXX	✓	–	–	✗	power	✗
Debande [28]	2012	◐	not specified	✓	–	–	✗	power	✗
Gagnerot [36]	2013	●	RISC-V(not specified)	✓	–	–	✗	power	✗
SILK [97]	2014	●	ATmega328P	✓	–	–	✗	power	✓
SLEAK [100]	2014	●	ARM Cortex A8	✓	✓	–	✓	register access	✗
Reparaz [76]	2016	●	not relevant	✓	–	–	✓	power	✗
SAVRASCA [98]	2017	●	ATMega163	✓	✓	–	✓	power	✓
ASCOLD [72]	2017	◐	ATMega163	–	✓	✓	✓	ILA	✓
ELMO [63]	2017	◐	ARM Cortex M0	✓	–	–	✗	power	✓
EMSIM [85]	2020	◐	RISC-V(custom)	✓	–	–	–	EM	✗
ELMO* [86]	2021	◐	ARM Cortex M0	✓	–	–	✓	power	✓
ROSITA [86]	2021	ELMO*	ARM Cortex M0	✓	✓	✓	✓	power	✓

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.

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

SLEAK [100] models an ARMCortex A8, a complex processor. To access the values of intermediate states, SLEAK uses Gem5 as an open source-full system simulator. However, the Gem5 model for this CPU has an average 7% of errors stemming from the microarchitecture events and execution time. [99]

4.3 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.

Generic code-rewrite for trace emulators. ROSITA [86] 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 [63] 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. ROSITA repeats the process until no more leakage is detected.

5 PRE-SILICON TOOLING

The world of pre-silicon side-channel leakage verification tooling, while at first glance relatively rich (see Table 2), is limited by the fact that these tools are not public and the results are difficult to reproduce. Additionally, the reported results consider prototype chip designs, which might require effort to 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 achievements of tools in this category.

5.1 Leakage detection at pre-silicon stage

Power estimates at different abstraction levels speed-up the generation of power signals. NCSIM [33] is the first white-box emulator to estimate the resistance to differential power analysis (DPA) [54] at the gate level. It neglects static power consumption, but it can model glitches and early propagation when timing information is available. The emulator supports several power estimation functions. The simplest is *transition counting* (each time the signal changes its logical state, the power consumption at the current point is increased by one). A more refined power model is the *random transition weighting*, which captures the variations in load capacitances across gates. 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 [32] 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 an entire month.

RTL-PSC [49] estimates the power profile of a hardware design using functional simulation at the RTL level. The power profile is estimated based on the number of transitions using the Synopsys VCS tool to ensure a fast framework. Compared to state-of-the-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 paper reports evaluation times for AES circuits in the order of dozens of minutes, compared with over 30 hours for gate-level evaluations of the same circuits. The authors

Table 2: Tools for pre-silicon side-channel verification (chronological order)

Name	Year	Input	End-device (description)	LM	Detect	Verify	Mitigate	Masking	Target	SC	Open-Source
NCSIM [33]	2007	gate	SCARD [46]	S	✓	–	–	✗	CC	power	✗
AMASIVE [50]	2013	RTL	–	E	✓	–	–	✗	CA	power	✗
MAPS [56]	2018	ISA	ARM CortexM3	E	✓	✓	–	✓	CA	power	✓
KARNA [89]	2019	layout	AES [12], SIMON [2]	S	✓	✓	✓	✗	CC	power	✗
RTL-PSC [49]	2019	RTL	AES-GF [1], AES-LUT [82]	S	✓	–	–	✗	CC	power	✗
PARAM [32]	2020	gate	RISC-V(ShaktiC)	E	✓	✓	–	✗	ED	power	✗
ACA [102]	2020	gate	RISC-V(LEON3)	S	✓	–	–	✗	CC	power	✗
SCRIPT [68]	2020	gate	AES-GF [1], AES-LUT [82]	–	✓	✓	–	✗	CC	power	✗
CASCADE [87]	2020	gate	ASIC (custom)	E	✓	–	–	✗	CC	power	✓
PATCH [95]	2021	gate	AES [96]	E	✓	✓	✓	✗	CC	power	✗
COCO [44]	2021	gate	RISC-V	–	✓	✓	–	✓	CA+ED	power	✓

Input specifies the abstraction level for the input to the emulator, the supported end-device specified by the column *End-device*. For the leakage model *(LM)* we have two options: simulated power (S) or estimated power (E) The *Target* column describes what is being simulated: the cryptographic core (CC), the cryptographic algorithm (CA) or the end-device (ED).

further estimate that the same circuit's evaluation time at the layout level would take more than one month.

ACA [102] 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 standard in DPA attacks. The objective of ACA is to identify the gates in the design that contribute 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 that only a handful of gates are the majority contributors to side-channel leakage for several different application scenarios (an AES engine, a Sparc-V CPU). This finding 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 [87], a white-box emulator 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 simple toggle counting. When the tool checks a masked implementation 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 [27, 43], which [101] showed to be leaky. 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 analyzed S-box implementation is an in-house implementation of a PRESENT S-layer in WDDL [91], a dual-rail [23] logic style. The analysis is done at gate-level netlist and place and route, using the simulated (with PrimeTime) and estimated power with the marching stick model. In all cases, the tool detects no leakage.

5.2 Leakage verification at pre-silicon stage

Formal verification at gate level. Both SCRIPT [68] and COCO [44] allow formal verification at gate level. However, while SCRIPT targets cryptographic cores, COCO aims to formally verify a masked software implementation on a given hardware platform. The approach used for the formal proof, described below, is also different.

SCRIPT [68] takes as input a gate-level description of a cryptographic 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. 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. PrimeTool (Synopsys) or XPE (Xilinx) provide the vectorless power analysis, which returns the total estimated power for the design.

COCO [44] allows security proofs at the gate level for the execution of masked implementations. The proofs use the time-constrained probing model, which simulates the hardware of pipelined circuits. As a first step, the tool executes the masked assembly implementation 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 that contain shares of sensitive values are annotated. COCO uses correlation sets and an SAT solver to find the gate and execution cycle where the implementation leaks. 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 its counterpart must not overwrite a register or memory location which contains one share.

Open-source tooling. MAPS [56], is the first open-source verification tool, a power emulator 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 [45]. 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. MAPS restricts its operation to registers which deal with sensitive values to simplify the tracing and reduce overhead.

MAPS is not cycle-accurate. While both MAPS and COCO [44], the two open-source verification tools, can be used to verify masked software implementations, MAPS supports only the ARM Cortex M3 platform. At the same time, COCO can handle any given netlist.

SCA resilience for non-cryptographic designs. PARAM [32] is a microprocessor design hardened for side-channel resistance. It is a trace emulator, 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. PARAM sees the processor as a netlist of functional modules such as the main pipeline, the ALU, data cache, or the instruction cache. The signals (wires and registers) associated with the module are used as input to estimate leakage for a given module. Once the power consumption is estimated, SVF (Side-Channel Vulnerability Factor) [29] 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 in side-channel attacks).

5.3 Leakage mitigation at pre-silicon stage

Leakage mitigation tool at the layout level. KARNA [89] identifies vulnerable gates in the design and then re-configures them. KARNA partitions the chip into small cells and performs TVLA assessment for each cell. To estimate the power consumption, KARNA uses commercial tooling, and the tool reveals leakage specific to a given area. PATCH [95] detects leakage at gate-level and offers the option of injecting hiding countermeasures. Although it currently uses a t-test for leakage detection, its design is modular and allows using other metrics.

6 METRICS AND EVALUATION

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. Validation is crucial as it demonstrates the practical value of the tool in identifying and removing side-channel leaks. This section examines the different metrics used to determine if the leakage emulator's output helps develop 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 emulators.
(2) Evaluations of *leakage model's quality* most often through comparing it to a simpler model.
(3) Evaluation by case studies, where the emulator will find or fix side-channel leaks (listed in Appendix B).
(4) *Usability measures* explore the benefits of using an emulator, typically by comparing the performance of the tool with either a measurement setup for the post-silicon emulator or the power simulation techniques for the pre-silicon emulators.

6.1 Metrics for evaluating the output of leakage emulator's

We can summarize the question answered by the metrics we place in this group as follows: *how close is the output of a leakage emulator to the reference traces?* The implicit assumption is that the closer the leakage emulator's output to the reference traces, the more the output of the tool can be trusted. As the number of samples in the emulator output differs from that in reference traces, it is not straightforward to compare the two data sets (Appendix A lists common choices). Most of these measures provide visual evidence of similarity. Different boards exhibit different behaviors [15, 78] that may cause slight variations between traces measured from various devices. This difference is relevant when matching simulated traces with measured traces. However, we found no reference that evaluates differences between sets of traces from different boards.

Aside from 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.

6.2 Metrics for evaluating the quality of the leakage model

It is typical for the post-silicon evaluation tools [28, 63, 86], which propose a complex leakage model to compare their performance with simpler or previously known leakage models. In pre-silicon emulators, we count in this category the metrics which quantify the leakage identified by the emulator, compared to an ideal case where the target does not leak information. In the following, we list achievements in this category.

Empirical evidence that gray-level leakage models are superior to black-box leakage model. Although the statement above might seem naive, the question of whether *is worth investing time and effort in creating sophisticated gray leakage models* is valid. To prove the merit of the ELMO leakage model [63], the authors use *power correlation* to compare the predictions of a simple leakage model (Hamming weight) with the prediction of leakage produced by the ELMO model. The comparison consists of computing the correlation traces produced by both leakage models on *the same* reference traces. The result [63, Figure 4] 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 authors conclude that the simple leakage model captures only a portion of the actual leakage and should not be relied upon when protecting sensitive data. ELMO* [86] uses the same metric to show its superiority over ELMO [63]. Debande et al. [28] use guessing entropy to compare the performance of a simple black-box leakage model with a profiled leakage model.

Metrics to quantify leakage of hardware components. *Side-Channel Vulnerability Factor* (SVF) [29] 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, thus capturing the system vulnerability to side-channel attacks. SVF quantifies a particular system's overall 'leakiness' but does not provide any insight into the cause. PARAM [32] uses SVF to quantify the level of leakage in the different components of the target processor.

Leakage Impact Factor (LIF), proposed in ACA [102], quantifies the similarity of the activity profile of a single gate or cell to a high-level leakage model used by DPA. The relative power consumption of the cell is used as a weight coefficient for the LIF score. LIF directly quantifies the contribution of a single gate or cell to the side-channel leakage. It is used in ACA to rank the gates of the design from leaky to least leaky.

6.3 Evaluating Usability

Next to security-related advantages, emulators also offer important *usability* advantages for the implementation and testing of a (masked) cryptographic primitive. The use of an emulator encourages testing at different development stages, allowing the removal of vulnerabilities as early as possible in the development cycle. For post-silicon, the cryptographic implementation can be tested at source-code level [76], assembly level [63], or compiled binary. It is possible (and advisable) to test the design at RTL, gate, or transistor level for pre-silicon.

- *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 require advanced skills in cryptographic engineering. An emulator is easy to use and reduces a highly iterative task that 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, [86] mentions a factor of 4.5–7-speed increase compared to real hardware. As an extreme example [87] mentioned that for a fully-unrolled AES (which exceeds the security budget of most embedded devices), simulation and analysis of one million traces might take about 4 hours on an 8-thread workstation. At the same time, we have ample evidence (Appendix B) 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 [49]. In other words, while it is relatively easy to make changes at the RTL level, only small changes are possible at the layout level. In contrast, at the post-silicon level, no changes in the design are possible.
  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 saves 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 and as a result increase the security of cryptographic implementations.
  Application: pre-and post-silicon.

7 OPEN PROBLEMS IN DESIGNING SIDE-CHANNEL LEAKAGE EMULATORS

Post-silicon: How can we efficiently create fine-grained leakage models? We now know that it is insufficient to make security claims for an implementation based only on architecture [62] level information. We also know that existing emulators target relatively simple architectures. As can be seen from Table 1 the most commonly targeted end-devices are ATMegaXX or ARM-Cortex M0, which are simple, in-order, single-core CPUs. With no access to design information, the task of the designer is to reverse-engineer the microarchitecture details. Today we know how to model simple microarchitectural features, such as 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 [63] or reverse-engineering the ARM-Cortex M3 [40] is still based on manual work and prohibitively effort-intensive. Capturing microarchitecture events characteristic to complex processors, such as pipeline stalls or misprediction is an open problem.

Post-silicon: Going beyond leakage detection is challenging. To verify the source of a leak, we must have the ability to know the instruction executed at that specific time sample in the power trace. 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 [10], Gem5 [17] or SimulAVR [81] or specialized hardware, such as JTrace Pro[2] for boards with advanced debug support such as the tracing features of ARM Cortex cores. Therefore, verification depends on whether a machine emulator supports the board or a tracing pin is available.

Post-silicon: Procedure for comparing simulated vs reference traces. An open question is whether we should include the leakage model in the evaluation. While most of the measures mentioned do include a leakage model, the authors of EMSIM [85] use *normalized cross-correlation* to show how well the simulated traces match the reference traces without relying on a specific leakage model. This metric's output is a quantifiable measure because it computes the average cross-correlation between individual clock cycles. 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. Although initially a measure of the side-channel created by a single instruction, the *Signal Available Attacker*, (SAVAT) [20] is used in EMSIM [85] to measure the similarity between the simulated and reference traces. SCRIPT [68] uses *side-channel vulnerability* (SCV), which is the equivalent of SNR [60] 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 [49] combines KL-divergence with SNR to identify vulnerable design blocks. At the same time, SLEAK [100] uses mutual information between the sensitive values processed by the algorithm and the value or state of a system component during the execution binary.

Pre& Post-silicon: Quantitative measures for SCA resilience. Most side-channel countermeasures come at the price of slower performance or more area. On the post-silicon side, we know [5, 66] the importance of including microarchitecture features when evaluating the security of masked implementations. However, most studies focus on one platform and zoom in on the feature which

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

leaks side-channel information on the studied platform. Even if we knew how to model every microarchitecture event, there is no widely accepted measure in the side-channel community to quantify leaks. We do not know how to measure the "gain" in security when applying SCA countermeasures.

Pre-silicon: Common evaluation criteria for tools. The objective criteria for differentiating the metrics used by the different tools would be to confirm the predictions with the results of the leakage present on the physical device. The practical challenge is that there are no open-source ASIC devices (i.e., we have access to the design details) for testing. The current best practice approach is to use FPGAs to confirm that the results produced by the tool are accurate. However, the fundamentally different structure of an ASIC gate and an FPGA configurable block leaves room for unexpected leaks in the final product.

Pre-silicon: Composability of pre-silicon emulators. 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 essential 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. We also know that removing vulnerabilities as early as possible in the design stage is a sound engineering practice. Although creating a pre-silicon trace emulator requires significant effort, the existing tooling is fragmented and not reusable.

Pre& Post-silicon: EM simulation. The only EM emulator we have, EMSIM [85], was constructed for a relatively simple, custom-made RISC-V processor. To build EMSIM, EM measurements of the physical device are required. The experimental results aimed to assess how the simulated signal degrades when key microarchitectural features are removed, show that building an *accurate* EM emulator is impossible without access to microarchitectural information. Therefore we may only be able to build EM emulator for open-source hardware. If the presence of a physical target is a must for constructing EM emulator, it could explain the fact that there are no EM emulators at the pre-silicon stage. To build a power estimator, how have to estimate 'only' how much electrical charge is being moved around (Ohm's Law). The build an EM emulator, we have to estimate not only the quantity of charge, but also its precise orientation in 3D space (Maxwell's 4 Laws). It is unclear how to port the approach used in EMSIM [85] to more mainstream processors. There is no CAD tool for pre-silicon to compute the magnetic field radiated by ICs and hence no methodology to detect hotspots at the design stages. Some hope for creating emulators for the pre-silicon stage comes [73], who introduces a workflow for predicting the EM radiations of an IC.

Pre& Post-silicon: Benchmark existing emulators. While some emulators are available and open-source, the specific use-case for the tools makes it hard to compare. An alternative is to agree on a representative, public data set that covers different use-cases to evaluate the potential of a tool. Differentiating the merits of the tools is challenging. 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 [45] and SVF [29] respectively. Also, let's compare the emulator's input. We observe that while MAPS and COCO target a masked software implementation, SCRIPT aims to verify 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 formal proof, while MAPS and PLAN/PARAM take an empirical testing approach.

Although in recent years the standard practice is to open-source tools (as the authors are primarily from academia), for many of the earlier tools [30, 36, 90] we only have a description about the capability and innovations of the tool, which in many cases provides limited information.

Pre& Post-silicon: Lacking case studies for asymmetric cryptographic implementations. All case studies we encountered in the existing literature are focused on the implementation of symmetric algorithms.

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, the National Science Foundation award CNS-1931639, a gift from Intel Corporation, and received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

REFERENCES

[1] Aoki Laboratory. 2007. Galois field based AES Verilog design. http://www.aoki.ecei.tohoku.ac.jp/crypto/web/cores.html. (accessed: 20-Nov-2021).
[2] Aydin Aysu, Ege Gulcan, and Patrick Schaumont. 2014. SIMON Says: Break Area Records of Block Ciphers on FPGAs. *IEEE Embed. Syst. Lett.* 6, 2 (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. 2020. A Systematic Appraisal of Side Channel Evaluation Strategies. In *SSR*.
[4] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. 2012. Theory and Practice of a Leakage Resilient Masking Scheme. In *Asiacrypt*.
[5] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. 2015. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS*.
[6] Manuel Barbosa, Gilles Barthe, Karthikeyan Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *IEEE SP*.
[7] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. 2021. Masking in Fine-Grained Leakage Models: Construction, Implementation and Verification. *TCHES* 2021, 2 (2021).
[8] H. Bassen and G. Smith. 1983. Electric field probes–A review. *TAP* 31, 5 (1983).
[9] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. 2015. The SIMON and SPECK Lightweight Block Ciphers. In *DAC*.
[10] Fabrice Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *USENIX ATC*.
[11] Daniel J. Bernstein. 2005. Cache-timing attacks on AES. Preprint available at http://cr.yp.to/papers.html#cachetiming.
[12] Daniel J. Bernstein and Peter Schwabe. 2008. New AES Software Speed Records. In *Indocrypt*.
[13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2014. The Making of KECCAK. *Cryptologia* 38, 1 (2014).
[14] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. 2005. AES Power Attack Based on Induced Cache Miss and Countermeasure. In *ITCC*.
[15] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. 2020. Mind the Portability: A Warriors Guide

through Realistic Profiled Side-channel Analysis. In *NDSS*.
[16] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2014. Higher-Order Threshold Implementations. In *Asiacrypt*.
[17] 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. 2011. The Gem5 Simulator. *SIGARCH CAN* 39, 2 (2011).
[18] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. 2007. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*.
[19] Billy Bob Brumley and Nicola Tuveri. 2011. Remote Timing Attacks Are Still Practical. In *ESORICS*.
[20] Robert Locke Callan, Alenka G. Zajic, and Milos Prvulovic. 2014. A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events. In *MICRO*.
[21] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. 2018. Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers. In *CCS*.
[22] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO*.
[23] Zhimin Chen and Yujie Zhou. 2006. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *CHES*.
[24] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. 2017. Does Coupling Affect the Security of Masked Implementations?. In *COSADE*, Sylvain Guilley (Ed.).
[25] Jean-Sébastien Coron. 2014. Higher Order Masking of Look-Up Tables. In *Eurocrypt*.
[26] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. 2014. Higher-Order Side Channel Security and Mask Refreshing. In *FSE*.
[27] Joan Daemen and Vincent Rijmen. 2020. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition.* Springer.
[28] Nicolas Debande, Maël Berthier, Yves Bocktaels, and Thanh-Ha Le. 2012. Profiled Model Based Power Simulator for Side Channel Evaluation. Cryptology ePrint Archive, Report 2012/703.
[29] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. 2012. Side-channel vulnerability factor: A metric for measuring information leakage. In *ISCA*.
[30] Jerry den Hartog, Jan Verschuren, Erik P. de Vink, Jaap de Vos, and W. Wiersma. 2003. PINPAS: A Tool for Power Analysis of Smartcards. In *SEC*.
[31] ETH Zurich. [n.d.]. Pulp Platform. https://pulp-platform.org/. (accessed: 20-Nov-2021).
[32] Muhammad Arsath K. F, Vinod Ganesan, Rahul Bodduna, and Chester Rebeiro. 2020. PARAM: A Microprocessor Hardened for Power Side-Channel Attack Resistance. In *HOST*.
[33] Omnia S. Fadl, Mohamed F. Abu-Elyazeed, Mohamed B. Abdelhalim, Hassanein H. Amer, and Ahmed H. Madian. 2016. Accurate dynamic power estimation for CMOS combinational logic circuits with real gate delay model. *Journal of Advanced Research* 7, 1 (2016).
[34] R.P. Feynman, R.B. Leighton, and M. Sands. 2011. *The Feynman Lectures on Physics, Vol. II: The New Millennium Edition: Mainly Electromagnetism and Matter.* Basic Books.
[35] Wieland Fischer and Berndt M. Gammel. 2005. Masking at Gate Level in the Presence of Glitches. In *CHES*.
[36] Georges Gagnerot. 2013. *Étude des attaques et des contre-mesures associées sur composants embarqués.* Ph.D. Dissertation. Université de Limoges.
[37] Daniel Gajski and Robert H. Kuhn. 1983. New VLSI Tools - Guest Editors' Introduction. *Computer* 16, 12 (1983).
[38] Neel Gala, Arjun Menon, Rahul Bodduna, G. S. Madhusudan, and V. Kamakoti. 2016. SHAKTI Processors: An Open-Source Hardware Initiative. In *VLSI Design*.
[39] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *CHES*.
[40] Si Gao, Elisabeth Oswald, and Dan Page. 2021. Reverse Engineering the Micro-Architectural Leakage Features of a Commercial Processor. Cryptology ePrint Archive, Report 2021/794.
[41] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptographic Engineering* 8, 1 (2018).
[42] Daniel Genkin, Adi Shamir, and Eran Tromer. 2014. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *CRYPTO*.
[43] Ashrujit Ghoshal and Thomas De Cnudde. 2017. Several Masked Implementations of the Boyar-Peralta AES S-Box. In *Indocrypt*.
[44] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. 2021. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In *USENIX Security*.
[45] G. Goodwill, J.J.B. Jun, and P.Rohatgi. 2018. A testing methodology for side channel resistance validation. *NIST non-invasive attack testing workshop* (2018).
[46] Grant agreement ID: 507270. 2004. Side Channel Analysis Resistant Design Flow. https://cordis.europa.eu/project/id/507270. [Online; accessed 20-Nov-2021].
[47] Hannes Gross, Stefan Mangard, and Thomas Korak. 2016. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS*.
[48] Hannes Groß, David Schaffenrath, and Stefan Mangard. 2017. Higher-Order Side-Channel Protected Implementations of KECCAK. In *DSD*.
[49] Miao Tony He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark Mohammad Tehranipoor. 2019. RTL-PSC: Automated Power Side-Channel Leakage Assessment at Register-Transfer Level. In *VTS*.
[50] Sorin A. Huss, Marc Stöttinger, and Michael Zohner. 2013. *AMASIVE: An Adaptable and Modular Autonomous Side-Channel Vulnerability Evaluation Framework.*
[51] Yuval Ishai, Amit Sahai, and David Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*.
[52] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, Dan Boneh (Ed.).
[53] Paul C. Kocher. 1996. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*.
[54] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *CRYPTO*.
[55] Juliane Krämer, Dmitry Nedospasov, Alexander Schlösser, and Jean-Pierre Seifert. 2013. Differential Photonic Emission Analysis. In *COSADE*.
[56] Yann Le Corre, Johann Großschädl, and Daniel Dinu. 2018. Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors. In *COSADE*.
[57] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *IEEE SP*.
[58] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. 2021. A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks and Defenses in Cryptography. CoRR 2103.14244.
[59] lowRISC contributors. [n.d.]. Open Titan. https://opentitan.org/. (accessed: 20-Nov-2021).
[60] Stefan Mangard. 2004. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In *CT-RSA*.
[61] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. 2005. Side-Channel Leakage of Masked CMOS Gates. In *CT-RSA*.
[62] Ben Marshall, Dan Page, and James Webb. 2021. MIRACLE: MIcRo-ArChitectural Leakage Evaluation. Cryptology ePrint Archive, Report 2021/261.
[63] David McCann, Elisabeth Oswald, and Carolyn Whitnall. 2017. Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages. In *USENIX Security Symposium*.
[64] Thomas S. Messerges. 2000. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms.* Ph.D. Dissertation. University of Illinois at Chicago.
[65] Thomas S. Messerges. 2000. Securing the AES Finalists Against Power Analysis Attacks. In *FSE*.
[66] Lauren De Meyer, Elke De Mulder, and Michael Tunstall. 2020. On the Effect of the (Micro)Architecture on the Development of Side-Channel Resistant Software. Cryptology ePrint Archive, Report 2020/1297.
[67] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. 2019. Glitch-Resistant Masking Revisited: or Why Proofs in the Robust Probing Model are Needed. *TCHES* 2019, 2 (2019).
[68] Adib Nahiyan, Jungmin Park, Miao Tony He, Yousef Iskander, Farimah Farahmandi, Domenic Forte, and Mark Mohammad Tehranipoor. 2020. SCRIPT: A CAD Framework for Power Side-channel Vulnerability Assessment Using Information Flow Tracking and Pattern Generation. *ACM TODAES* 25, 3 (2020).
[69] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. 2006. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security*.
[70] Colin O'Flynn and Jasper van Woudenberg. 2021. *The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks.* No Starch Press.
[71] Dan Page. 2002. Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel. Cryptology ePrint Archive, Report 2002/169.
[72] Kostas Papagiannopoulos and Nikita Veshchikov. 2017. Mind the Gap: Towards Secure 1st-Order Masking in Software. In *COSADE*.
[73] Davide Poggi, Philippe Maurine, Thomas Ordas, Alexandre Sarafianos, and Jérémy Raoult. 2020. Finding EM leakages at design stage: a simulation methodology. Cryptology ePrint Archive, Report 2020/1198.
[74] Jean-Jacques Quisquater and David Samyde. 2001. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*.
[75] Jan M. Rabaey. 2009. *Low Power Design Essentials.* Springer.
[76] Oscar Reparaz. 2016. Detecting Flawed Masking Schemes with Leakage Detection Tests. In *FSE*.
[77] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Consolidating Masking Schemes. In *CRYPTO*.
[78] Unai Rioja, Lejla Batina, and Igor Armendariz. 2020. When Similarities Among Devices are Taken for Granted: Another Look at Portability. In *Africacrypt*.

[79] Riscure. 2002. Inspector SCA. https://www.riscure.com/security-tools/inspector-sca. [Online; accessed 20-Nov-2021].

[80] Matthieu Rivain and Emmanuel Prouff. 2010. Provably Secure Higher-Order Masking of AES. In *CHES*.

[81] Theodore Roth and Klaus Rudolph. 2001. SimulAVR. https://www.nongnu.org/simulavr/. [Online; accessed 20-Nov-2021].

[82] Satoh Lab. 2017. Lookup table based AES Verilog design. http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html. (accessed: 20-Nov-2021).

[83] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. 2021. An Inside Job: Remote Power Analysis Attacks on FPGAs. *IEEE Des. Test* 38, 3 (2021).

[84] Kai Schramm and Christof Paar. 2006. Higher Order Masking of the AES. In *CT-RSA*.

[85] Nader Sehatbakhsh, Baki Berkay Yilmaz, Alenka G. Zajic, and Milos Prvulovic. 2020. EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals. In *HPCA*.

[86] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. 2021. Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. In *NDSS*.

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

[88] D. Singh, J.M. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, and T.J. Mozdzen. 1995. Power conscious CAD tools and methodologies: a perspective. *Proc. IEEE* 83, 4 (1995).

[89] Patanjali SLPSK, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. 2019. KARNA: A Gate-Sizing based Security Aware EDA Flow for Improved Power Side-Channel Attack Protection. In *ICCAD*.

[90] Céline Thuillet, Philippe Andouard, and Olivier Ly. 2009. A Smart Card Power Analysis Simulator. In *CSE (2)*.

[91] Kris Tiri, David D. Hwang, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. 2005. Prototype IC with WDDL and Differential Routing - DPA Resistance Assessment. In *CHES*.

[92] Kris Tiri and Ingrid Verbauwhede. 2006. A Digital Design Flow for Secure Integrated Circuits. *TCAD* 25, 7 (2006).

[93] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. 1996. Instruction Level Power Analysis and Optimization of Software. In *VLSI Design*.

[94] Elena Trichina, Tymur Korkishko, and Kyung Hee Lee. 2005. Small Size, Low Power, Side Channel-Immune AES Coprocessor: Design and Synthesis Results. In *Advanced Encryption Standard – AES*.

[95] Ali Jahanian Vahid Samadi Bokharaie. 2021. Power side-channel leakage assessment and locating the exact sources of leakage at the early stages of ASIC design process. *J. Supercomput* (2021).

[96] Vamshi PN. 2019. Hardware-implementation of AES Verilog. https://github.com/pnvamshi/Hardware-Implementation-of-AES-Verilog. (accessed: 20-Nov-2021).

[97] Nikita Veshchikov. 2014. SILK: high level of abstraction leakage simulator for side channel analysis. In *PPREW@ACSAC*.

[98] Nikita Veshchikov and Sylvain Guilley. 2017. Use of Simulators for Side-Channel Analysis. In *EuroS&P Workshops*.

[99] Matthew Walker, Sascha Bischoff, Stephan Diestelhorst, Geoff Merrett, and Bashir Al-Hashimi. 2018. Hardware-Validated CPU Performance and Energy Modelling. In *ISPASS*.

[100] Dan Walters, Andrew Hagen, and Eric Kedaigle. 2014. *SLEAK: A Side-Channel Leakage Evaluator and Analysis Kit.* Technical Report. The MITRE Corporation. [Online; accessed 20-Nov-2021].

[101] Felix Wegener and Amir Moradi. 2018. A First-Order SCA Resistant AES Without Fresh Randomness. In *COSADE*.

[102] Yuan Yao, Tarun Kathuria, Baris Ege, and Patrick Schaumont. 2020. Architecture Correlation Analysis (ACA): Identifying the Source of Side-channel Leakage at Gate-level. In *HOST*.

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

[104] Mark Zhao and G. Edward Suh. 2018. FPGA-Based Remote Power Side-Channel Attacks. In *IEEE-SP*.

A METRICS FOR COMPARING SIMULATED AND REAL TRACES

Here we give some examples:

- *Dynamic Time Warping.* To evaluate SILK [97], the authors compute the distance between a set of measured traces and the simulator's output. Dynamic Time Warping (DTW) computes the distance between two temporal series, even with different elements. Several leakage models are used to generate simulated traces. The authors instantiate DTW with two distance metrics, Euclidean and correlation-based distance. For a fair comparison, they add noise to the simulated traces. Notably, the two used distance metrics give different results regarding what constitutes a realistic trace set.

- *Power correlation.* Although the term is defined in [102], this is a well-known measure used when analyzing side-channel leakage. The approach assumes the knowledge of the key and requires the explicit choice of a target variable and leakage model. The authors of ELMO [63] compare (visually) the power correlation trace produced by ELMO with the power correlation trace obtained from measured traces. If the correlation traces follow a similar trend, the model and the measured traces capture similar side-channel leakage.
- *Leakage detection comparison.* The de facto technique for leakage detection is TVLA [45]. It comes then as no surprise that the practice of comparing a t-test trace produced by a leakage emulator with the t-test trace produced by the reference traces, using either a fixed vs. random t-test [63, 86] or a fixed vs. fixed t-test [76] is widespread. To compute the t-test trace, the datasets 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 emulator, Debande et al. [28] 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 that of the measured traces (and that of the traces produced by the profiled leakage models), the conclusion is that profiled models that closely follow the behavior of 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 intermediate target value. NCSIM [33] uses the same metric, and the similarity of a DPA attack performed on the internal MOV operation demonstrates the advantage of using an emulator for the design of a secure chip. The authors of PARAM [32] also use DPA to compare the reference architecture's resistance before and after applying hardening countermeasures.

B CASE STUDIES

In this section, we explore the answers to the question: *How effective are the existing tools at verifying and eliminating SCA vulnerabilities*? 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 significantly between the different contributions. While some verification tools showcase a toy example [30], others explore a wide variety of scenarios. 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 for a leakage verification tool. In the following, we present a representative selection of use cases.

B.1 Software implementation of cryptographic algorithms

Reparaz [76] tests the security of six high-order implementations. The first is a "smoke test" where the aim is to reproduce the flaw found by [80] for the first-order masking scheme proposed by [4]. The emulator performs six fixed vs. fixed TVLA tests and reports that five of the six tests show leakage. The authors apply the same test to the first-order secure table recomputation scheme proposed by [25] which, as expected, "on the strength of the found evidence" is reported to be secure. Next, the authors use the tool to reproduce the second-order flaw spotted by [26] in the masking scheme proposed by [80]. The authors report that it takes the tool five seconds to find the flaw, including the time to identify the cause. The tool further reproduces the third-order vulnerability spotted by [25] in the technique used for refreshing the mask in the scheme proposed by [80]. The tools finds this flaw in less than a second. A more complex case for the tool (200 million traces and eight hours of simulation) is to reproduce the observations from [77] on higher-order implementations [16]. The authors also report a new second-order flaw found in [84], which, once found, is easily proved.

B.2 Hardware implementation of cryptographic algorithms

KARNA [89] is evaluated with three open-source cores. The first is a bit-serial implementation of the Simon block cipher [2][3]. 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[4], a minimal design which after one iteration achieves side-channel resilience. The third is an optimized AES core [12][5], on which a DPA attack is performed after the place-and-route stage, and the design is shown not to leak information with 100 000 traces. Compared with the unoptimized AES synthesized design, where the result is shown to reveal the correct key byte at approximately 2 000 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 [49] is evaluated on two AES designs based on Galois Field (GF) [1] and Look-up Table (LUT) [82]. The tool is used to identify the leaky modules in the design, using a combination of Kullback-Leibler divergence and success rate. The tool is validated

[3] reference implementation, https://opencores.org/projects/simon_core
[4] reference implementation https://opencores.org/projects/present
[5] reference implementation https://opencores.org/projects/tiny_aes

with both a gate-level netlist simulation and an FPGA simulation. The authors compare the Pearson correlation of the RTL simulations (produced by the tool) with the gate-level netlist's KL-divergence trace and conclude that the tool is successful.

AMASIVE [50] is used to analyze an unprotected hardware implementation of the PRESENT cipher [18] for the first and the last round. The tool identifies hypothesis functions for the HW and HD leakage models. The authors mount a CPA attack to confirm the tool's attack vector and report recovering the key within 10 000 traces.

B.3 Hardening of non-cryptographic hardware

PARAM [32] is used to produce a hardened implementation of a Shakti-C [38] 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 of the number of traces vs. correlation scores show the hardened microprocessor's resilience.

B.4 Combination of software implementations running on a physical target

The case study for MAPS focuses on showcasing the design flow with the tool. The specific example is a naive implementation of SIMON [9] protected with Trichina AND gate [94], which aims to minimize the number of execution cycles. The authors simulate the implementation of this cipher with and without pipeline leakage. Using a fixed vs. random t-test, the authors show that both instances leak information. In the next iteration, MAPS identifies the leakage due to register reuse. The remaining leakage comes from the two pipeline registers. After fixing the two pipeline registers' leakage, the t-test obtained from the simulated traces show no leaky points. As a final step, the authors perform a t-test on a set of reference traces measured from a physical implementation, which shows a few remaining leakage points.

SAVRASCA [98] is used to find a flaw in the AES implementation used for Version 4 of the DPA contest[6]. Using the tool, the authors note that the simulated traces size depends on the value that the microcontroller manipulates, even though the implementation is running in a constant number of cycles. Analyzing the implementation, the authors find that the number of register accesses depended on the manipulated value. In response to this finding, the new version of the DPA contest v4.2 fixed the implementation and released a new set of traces.

ASCOLD [72] is used to develop a hardened first-order, ISW-based [51] S-box with a bit-sliced RECTANGLE implementation [103]. The authors evaluate the performance of the hardened implementation 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 a *conservative* implementation, which adds to the efficient implementation of dummy instructions' insertion through register/memory clearing. Non-specific TVLA is used to evaluate the strengths of hardening countermeasure.

[6] http://www.dpacontest.org/v4/rsm_doc.php

COCO uses the open-source IBEX core[7], part of the PULP platform [31] and the OpenTitan [59] project. The main application of COCO is the verification of masked software implementations running on hardware specified at gate-level netlist. A considerable selection of masked circuits, which cover domain-oriented masking (DOM) AND gate [47], Ishai-Sahai-Wagner (ISW) AND [77], Threshold implementation (TI) AND [69] and larger implementations DOM Keccak S-box [13, 48], DOM AES S-box [43] and the Trichina AND gate [94] are presented to demonstrate the effectiveness of the tool. The scenarios cover two case studies with intentionally injected vulnerabilites [47, 48]. The implementations also cover second-order security [47, 48] and third-order security [47] (for a complete overview see Table 3 in [44]). To demonstrate COCO's ability to deliver a secure implementation, the authors map a sample of the verified netlist of IBEX cores and the DOM Keccak S-box [48] onto a Xilinx Spartan-6 FPGA. The design is then evaluated using TVLA and shown not to leak information at 100 000 traces.

[7] reference implementation https://github.com/lowRISC/ibex