Challenges and Opportunities of Security-Aware EDA

JAKOB FELDTKELLER and PASCAL SASDRICH, Ruhr University Bochum, Germany
TIM GÜNEYSU, Ruhr University Bochum, Germany and DFKI, Cyber-Physical Systems, Germany

The foundation of every digital system is based on hardware in which security, as a core service of many applications, should be deeply embedded. Unfortunately, the knowledge of system security and efficient hardware design is spread over different communities and, due to the complex and ever-evolving nature of hardware-based system security, state-of-the-art security is not always implemented in state-of-the-art hardware. However, automated security-aware hardware design seems to be a promising solution to bridge the gap between the different communities.

In this work, we systematize state-of-the-art research with respect to security-aware Electronic Design Automation (EDA) and identify a modern security-aware EDA framework. As part of this work, we consider threats in the form of information flow, timing and power side channels, and fault injection, which are the fundamental building blocks of more complex hardware-based attacks. Based on the existing research, we provide important observations and research questions to guide future research in support of modern, holistic, and security-aware hardware design infrastructures.

CCS Concepts: • **Hardware** → **Electronic design automation**; • **Security and privacy** → **Hardware attacks and countermeasures;**

Additional Key Words and Phrases: Hardware design, electronic design automation, computer aided design, Information Flow Analysis, Side-Channel Analysis, Fault Injection Analysis

ACM Reference format:
Jakob Feldtkeller, Pascal Sasdrich, and Tim Güneysu. 2023. Challenges and Opportunities of Security-Aware EDA. *ACM Trans. Embedd. Comput. Syst.* 22, 3, Article 43 (April 2023), 34 pages.
https://doi.org/10.1145/3576199

1 INTRODUCTION

Security of digital systems can only be achieved when restricting the systems to benign behavior only. Ultimately, this requires limits to the possible states and transitions of the underlying hardware, which in turn is configured by software. Hence, the structure and design of hardware play a significant role in system security.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. The work described in this paper has been supported by the German Federal Ministry of Education and Research BMBF through the project FlexKI (01IS22086I), 6GEM (16KISK038), and VE-HEP (16KIS1345). The work was also supported by the European Commission through the project CONVOLVE (101070374).
Authors' addresses: J. Feldtkeller and P. Sasdrich, Ruhr University Bochum, Horst Görtz Institute for IT Security, Universitätsstraße 140, Bochum, Nordrhein-Westfalen, 44801, Germany; emails: {jakob.feldtkeller, pascal.sasdrich}@rub.de; T. Güneysu, Ruhr University Bochum, Horst Görtz Institute for IT Security, Universitätsstraße 140, Bochum, Nordrhein-Westfalen, 44801, Germany and DFKI, Cyber-Physical Systems, Bibliothekstraße 5, Bremen, Bremen, 28359, Germany; email: tim.gueneysu@rub.de.

1539-9087/2023/04-ART43 $15.00
https://doi.org/10.1145/3576199

Unfortunately, during the last decades, research has shown that physical properties of hardware systems can leak sensitive information (*side-channel attacks*) [87, 88] or be manipulated to undermine security (*fault injection attacks*) [22, 24]. Even worse, hardware-based attacks can also be executed from software and thus remotely [20, 79, 132]. Only recently, the aspect of hardware-oriented security again received considerable attention with the discovery of *microarchitectural attacks* [52, 58, 86, 95, 122, 131]. Those attacks exploit vulnerabilities at the microarchitectural level to violate the security guarantees of otherwise secure software.

Once fabricated, vulnerabilities within hardware are hardly rectifiable while a single successful attack may suffice to overcome all remaining security measures. Hence, hardware should be considered as *Root-of-Trust* with fundamental importance in system security. Failing to do so can have severe consequences and eliminate security measures of the entire system. As a consequence, security must become a first-class design constraint for hardware, already considered during the initial stages of the design cycle.

However, system and hardware security are complex and fast-evolving areas where even security experts have hard times keeping up with the pace of new developments, attacks, and countermeasures. Hence, simple education of hardware and computer architects cannot lead to a large-scale shift towards hardware security, as it is urgently needed by the community. As a consequence, we see the integration of *security awareness* into the hardware design flow as the most promising approach to transfer the required knowledge from security experts to hardware architects. More specifically, we understand security-aware EDA as a framework, where transformations are already done, with security implications in mind, thus, security becoming a fourth design constraint, alongside speed, area, and power [111]. As such, security-aware EDA provides the opportunity to bundle security knowledge into tools that can be used to automatically test or enhance hardware designs with state-of-the-art security features. Ideally, it provides an interface between hardware designers and security experts to drive and fuel innovations within both communities. Most importantly, it may lead to widespread adoption of *security by design* methodologies.

While many EDA tools are historically grown and mostly optimize for speed, area, and power, they hardly respect security as a fundamental design constraint, which requires a holistic view of the system. Further, with each transformation, including pre- or post-processing for different EDA tools, translation to other design levels, or optimization for speed, area, or power, the risk of introducing security vulnerabilities increases. For this, current security analysis is a downstream process in the design cycle, even though an exhaustive analysis is hardly possible at this time, and rectification of vulnerabilities is most expensive due to complexity and analysis limitations.

Contribution: In this work, we aim at the integration of security as an integral part of the hardware design flow. For this, we provide an extensive overview and systematization with respect to state-of-the-art research in the field of security-aware EDA. Hence, we consider tools that perform automated security analysis or design protection and their integration into the EDA context. Building upon our systematization, we summarize important observations and conclude research challenges to indicate the direction for future research. Eventually, we identify a modern and holistic security-aware EDA framework developed around the concept of a unified Intermediate Representation (IR), i.e., a single way of representing the design internally within the entire EDA framework, and argue for its benefits from a security perspective.

Although this is not the first proposal for the integration of security-awareness into hardware design, e.g., [28, 62, 80, 85, 111, 140], none of those works provide an extensive review of existing research in security-aware EDA, nor do the authors consider a hardware design framework with unified IR based on the principle of integration. For this, we like to emphasize the high demand for a systematic and holistic approach that we aim to provide with this work.

Outline: In the following, we first provide some core concepts used throughout the paper in Section 2. Then we give an overview and discussion of security-aware EDA tools in four different threat domains: information flow in Section 3, timing side channels in Section 4, power side channels in Section 5, and fault injection in Section 6. In Section 7, we then discuss general observations and directions and outline a framework for security-aware EDA before we conclude in Section 8.

2 PRELIMINARIES

We first introduce the considered adversary model and the taxonomy used throughout the paper.

2.1 Adversary Model

In this work, we consider a design process with trusted EDA tools and fabrication environments, excluding the threat of *Hardware Trojans*, as those are maliciously introduced during design or fabrication. Further and in accordance with Kerckhoff's Principle [72], the adversary is granted full knowledge of the design specification, structure, and layout however, excluding device internals such as secret and sensitive data. By providing the adversary with the circuit structure the threat of Intellectual Property (IP) Piracy is omitted, since the adversary has trivial knowledge of the IP anyway. Nevertheless, the adversary may be able to manipulate and monitor the device physically or virtually to target secret or sensitive data after fabrication has been completed. We emphasize that Hardware Trojans and IP Piracy are real-world concerns when fabrication is not trusted or the adversary has no full knowledge of the circuitry. Note that we restrict our focus in this work to synchronous hardware devices only, as this is the most common design paradigm today.

In particular, we consider attacks based on information flow (Section 3), timing side channels (Section 4), power side channels (Section 5), or fault injections (Section 6). Those attack vectors are most common and well-researched while more complicated attacks often can be modeled as a combination of these basic attack vectors.

2.2 Taxonomy

Chinnery et al. [35] divide the history of EDA tool development into the *Age of Invention*, the *Age of Implementation*, and the *Age of Integration*, each with the following characteristics.

Age of Invention. During the first age, the basic algorithms for various EDA tasks are invented, which are feasible as long as only small circuits are processed.

Age of Implementation. In the second age, abstraction methods and sophisticated data structures are used to create advanced and efficient algorithms, to enhance scalability. Each algorithm works independently and a complete synthesis flow consists of multiple algorithms executed one after another.

Age of Integration. During the third age, common messaging protocols and agreed-upon semantic design structures are used to create highly integrated EDA environments, where different tools can interact with each other. This helps to solve complicated optimization problems, where design decisions can have severe and unpredictable consequences later in the synthesis flow. In order to fit into an integrated EDA environment, a tool must adhere to four design principles. First, the tool is required to communicate with other tools through a defined interface and operate on a shared design structure. Second, the tool should be modular, i.e., constructed out of independent parts, to reduce side effects and facilitate reuse. Third, the tool should operate incrementally, i.e., update local changes without the need to reprocess the entire design. And lastly, the tool should sparsely access data to reduce the memory footprint.

In this work, we aim to categorize existing security-aware EDA tools into those three ages, to analyze the current state of security-aware EDA, and to provide future directions for the research

Table 1. Taxonomy for Ages of Security-Aware EDA

	Taxonomy	Description	Aspect
Age of Implementation	**Coverage of Application**		**Scalability**
	Building Block	*Only considers building blocks and specific kinds of logic implementation.*	
	Entire Design	*Considers full designs as combinations of building blocks.*	
	Purpose of Application		**Scalability**
	Cryptographic	*Focuses on cryptographic applications only.*	
	CPU	*Focuses on classical software processing entities.*	
	General	*Processes arbitrary designs.*	
	Method of Abstraction		**Abstraction**
	Libraries	*Usage of libraries as simple method of abstraction.*	
	Design level	*Design level as simple indication for the level of abstraction.*	
Age of Integration	**Channel of Communication**		**Interoperability**
	Database	*A dedicated database is used.*	
	Messaging	*An interface and protocol definition exist.*	
	Files	*Additional information are written to a file.*	
	Annotation	*Additional information are written as annotations into the design file itself.*	
	Intermediate Representation		**Representation**
	Dedicated IR	*The tool uses a dedicated intermediate representation.*	
	Modular Design		**Modularity**
	Monolithic	*Tool is designed as one monolithic block.*	
	Front-/Backend	*Tool provides a separate frontend for translation into an internal representation.*	
	Modular	*Tool consist of separate modules that can be used individually.*	
	Processing		**Locality**
	Incremental	*Local and incremental processing without reprocessing of the entire design.*	
Security-Aware	**Security Features**		**Scalability**
	Custom	*Defined individually for each application in subsequent sections.*	
	Security Metric		**Prediction**
	Secure/Insecure	*Binary metric.*	
	Security Levels	*Discrete metric with defined interpretation.*	
	Quantitative	*Continues metric with space for interpretation.*	

community. By comparing security-aware EDA tools with best practices from the general EDA community, we can identify obstacles that prevent a tight integration of security into the hardware design flow. While the main drivers in traditional EDA are scalability and advances in technology, security-aware EDA tools are additionally driven by advances in security modeling. Hence, we expect security models to mature within a security domain during the *Age of Implementation* and across security domains in the *Age of Integration*.

To capture the evolution of security-aware EDA tools we propose the taxonomy outlined in Table 1. The different categories are selected in order to highlight features related to the *Age of Implementation*, measuring supported complexity, and the *Age of Integration*, by measuring the ability for cooperation. In addition, we introduce categories to capture advances in security awareness by analyzing the underlying security modeling. Traditional metrics of scale are explicitly not considered, as those need additional interpretation. Further, we report for every tool whether it is *dynamic* (using data from simulation or execution) or *static*, and whether it provides a security analysis (A) or transformation (T) pass.

Within each security domain, we group the tools by the taken approach, order each group by publication year and assign the *Age of Invention* to the primal tool. Subsequent tools within a group are considered in the *Age of Implementation* if they extend the approach in the direction of complexity or security awareness. To qualify for the *Age of Integration* a tool must follow the above-stated design principles and hence operate on a dedicated IR, have some interface for communication, and adhere to the modular and incremental design principle. The reported analysis is based on the referenced papers, as most of the tools do not provide open-source code.

3 INFORMATION FLOW ANALYSIS

The state of a digital system is defined by the information it stores and manipulates, while the flow of information transfers the system from one possible state to another. Hence, system security needs to protect access to sensitive information as well as the system itself from manipulation through information originating from potentially malicious entities. To distinguish information in need of protection from potentially malicious information usually security classes are defined based on source, meaning, and objective of the processed information. Hence, a security class groups information with equal security demands (mainly in terms of *security levels*) and defines the related security properties (e.g., a high-security level for cryptographic keys that should remain secret).

While from a functional point of view, any additional flow of information within the system is permitted as long as functional correctness is ensured, from a security perspective, information flow must be limited to a strictly necessary minimum. In particular, as information flow can transform the state, data, and behavior of the system untrusted, unauthorized, or unintended information flow might violate established security goals.

For this, permitted and prohibited information flow is specified by a security policy, including definitions of security classes and the allowed information flow between them. In this regard, *confidentiality*, and *integrity* are common security goals that are expressed in security policies. While confidentiality requires only authorized *access* to information, thus prohibiting flows to endpoints of less restrictive security classes (i.e., classes where a superset has access to), integrity requires only authorized *manipulation* of information, thus prohibiting flows to endpoints of more restrictive security classes (i.e., classes that a subset are allowed to manipulate).

Given this, Information Flow Analysis (IFA) verifies the compliance of implementations with an established security policy, through mostly manual labeling of information in accordance with security classes while tracking and checking the flow and propagation of labels through the system. The precision of IFA, given in terms of false positives [65], is mostly determined by the label propagation scheme and granularity of building blocks [4]. Through isolation of system parts, which handle information of different security classes and analysis of the interaction between system parts, *non-interference* [54] ensures absence of information flows that are not explicitly allowed by the security policy.

3.1 Domain-Specific Taxonomy

In accordance with the general taxonomy we introduce the following IFA-specific security features.

Arbitrary Security Policy is supported if the policy is not predefined or restricted by the tool.

Variable Granularity of labels allows information bundling of different sizes by a label.

Table 2. Information Flow Analysis Methods for Hardware in Literature

	Building Block	Entire Design	CPU	Generic	Libraries	Design Level	Arbitrary Policy	Variable Granularity	Label Propagation	Dependent Types	Downgrading	Secure/Insecure	Security Level	Quantitative	Database	Messaging	Files	Annotation	Dedicated IR	Monolithic	Front-/Backend	Modular	Incremental	Dynamic/Static	Pass Type	Year	Age
GLIFT [134]		●		●	◑	Gate	○	○	●	○	○	●			○	○	○	○	○	●			○	D	A	2009	Invention
RTLIFT [4]		●		●	●	RTL	○	●	●	○	○	●			○	○	○	○	○	●			○	D	A	2017	Implementation
TaintHLS [106]		●		●	●	RTL	○	●	●	○	○	●			○	○	○	○	◑	●			○	D	A	2019	Implementation
Caisson [94]		●		●	○	FSM	●	●	◑	●	◑	●			○	○	○	○	○	●			○	S	A	2011	Invention
Sapper [93]		●		●	○	FSM	●	●	●	●	○	●			○	○	○	○	○	●			○	S	T	2014	Implementation
SecVerilog [142]		●		●	○	RTL	●	●	○	●	○	●			○	○	○	○	○	●			○	S	A	2015	Implementation
ChiselFlow [47]		●		●	○	RTL	●	●	●	●	●	●			○	○	○	○	●		●		○	S	A	2018	Implementation
ASSURE [71]		●		●	○	HLS	●	●	●	●	●	●			○	○	○	○	●	●			○	S	A	2018	Implementation
SecChisel [37]		●		●	○	RTL	●	●	●	●	●	●			○	○	●	●	●			●	○	S	A	2019	Implementation
QIF-Verilog [59]		●		●	●	RTL	○	●	●	○	●			●	○	○	○	○	◑		●		○	S	A	2019	Invention
QFlow [112]		●		●	○	RTL	○	○	●	○	●			●	○	○	○	○	◑		●		○	S	A	2021	Implementation
VeriSketch [5]		●		●	○	RTL	○	●	●	●	○	●			○	○	●	●	◑			●	○	S	T	2019	Invention
Transynther [100]		●	●		○	Exec	○	○	○	○	○			●	○	○	○	○	○			●	○	D	A	2020	Invention

Symbols indicate whether the method has the property (●), has the property partially (◑), or does not have the property (○).

Label Propagation is supported if (some) labels are inferred automatically.
Dependent Types support labels depending on variables that do not affect labeled information.
Downgrading enables explicit ways to defined security policy violations that are accepted.

3.2 Current Research

Table 2 lists modern and state-of-the-art IFA methodologies for EDA, grouped based on methods that add a shadow logic to the design, those that introduce a domain-specific language, those using quantitative metrics, those that provide constructive IFA through program synthesis, and those that use black-box analysis.

3.2.1 Shadow Logic. At runtime, security labels can be propagated dynamically in parallel to functional information, by inserting additional *shadow logic* and *shadow registers* into the design. This approach was first used by Gate Level Information Flow Tracking (GLIFT) [134] based on gate-level information and a simple two-class policy with uni-directional information flow. By observing that all types of information flow have the same appearance at gate level, GLIFT tracks all flows using the same shadow logic components, which operate on both security labels and processed information to increase precision. The additional logic and registers can be created automatically using various algorithms with different complexity and precision trade-offs [14, 64, 65], but in any case significantly increases the circuit size [64]. To decrease the area footprint of the label propagation logic other methods move the shadow logic creation to the Register Transfer Level (RTL). While Register Transfer Level Information Flow Tracking (RTLIFT) [4] uses RTL information only, to distinguish between explicit and implicit information flows, TaintHLS also includes information from High-Level Synthesis (HLS) to treat data path and control logic differently. In general, HLS means transferring a non-hardware-specific into a hardware-specific design description. This includes introducing a notion of time and parallelism inherent to hardware. Both, RTLIFT and TaintHLS, use a library of label-propagation elements for efficient logic insertion and can assign a label to more complicated data structures than a single bit. Although GLIFT, RTLIFT, and TaintHLS can create shadow logic from an unmodified system description at the respective level, the specific labeling is provided at run time and is restricted to two-class policies. In general, more complex policies are possible but only with a significant increase in complexity [63]. While shadow logic, like all dynamic IFA, can only verify policy conformity for a specific execution, Star-Logic [133] provides a framework to get static assurances from those methods through exhaustive simulation. Similarly, formal theorem provers can work on formal models automatically derived from GLIFT circuits for policy verification [108].

3.2.2 Language-Based IFA. Static IFA, as design-time analysis with respect to information flow, allows verification against arbitrary security policies and enables early feedback in the design flow with minimal overhead for the final design. In general, most existing approaches introduce new security-typed Hardware Description Languages (HDLs), where variables are extended with security labels. However, this enforces the usage of a domain-specific language and might require a more complex design to comply with a security policy statically. Caisson [94] was the first security-typed HDL and explicitly models hardware as Finite State Machine (FSM). A designer has to provide a security label for all registers and states of the FSM, where dependent types are allowed for states only and sharing of registers between security classes is not supported. This approach was extended by Sapper [93], which uses the concept of shadow logic to add label propagation and error handling to the design, to statically enforce the defined policy. SecVerilog [142], ChiselFlow [46, 47], and SecChisel [37] extend language-based IFA to RTL designs in general. Such analysis at RTL is accurate, as long as the operational semantic is equal to the semantic of RTL simulation [142]. Similar to Caisson, SecVerilog does not support label propagation. ChiselFlow can propagate labels within a single module, and SecChisel can automatically infer all internal labels from the labels of the top-module inputs and outputs. Similarly, ASSURE [71] operates on a behavioral description extended with security labels to identify information flows during HLS. This allows to remove vulnerabilities more easily and with lower costs compared to IFA at RTL.

3.2.3 Quantitative IFA. QIF-Verilog [59] introduces language-based IFA using quantitative metrics of information flow of a simple two-class security policy. In contrast to non-interference, quantitative IFA [128] allows verifying that forbidden flows do not exceed a certain threshold. QFlow [112] improves QIF-Verilog by using a more tight quantitative metric based on the Posterior Bayes Vulnerability [2, 34] and Markov chains, however, at the cost of more restrictive assumptions (e.g., independent inputs).

3.2.4 Program Synthesis for Constructive IFA. VeriSketch [5] also introduces a new domain-specific language, but in contrast to other languages, uses counterexample-guided program synthesis [129] to construct information flow secure logic for a not entirely defined RTL sketch. Particularly, sketches only contain structural information provided by the designer, while VeriSketch instantiates the concrete logic satisfying the given security policy.

3.2.5 Black-Box Analysis for IFA. Transynther [100] uses fuzzing as a popular black-box analysis method to find information flows that can be exploited in a Meltdown-type manner [95]. Like all black-box methods, Transynther requires an executable design format and is, therefore, only applicable during later design stages. For fuzzing, new potential exploit programs are created by reusing building blocks form existing Meltdown-type attacks, and used as input. Then the effectiveness of the exploit is evaluated using a timing side channel (as discussed in Section 4). This timing side channel requires the existence of a vulnerable cache and is used only for evaluation. As fuzzing is based on random choices, Transynther cannot guarantee the absence of any Meltdown-type vulnerability.

3.3 Observations and Research Challenges

Comparing state-of-the-art research in security-aware EDA with respect to IFA, we discuss essential observations and outline possible directions for extensions and future work.

3.3.1 Security Model. We observe that IFA in EDA is a homogeneous field, where different tools provide similar features building upon each other. We explain this by the simple and uncontroversial underlying security model, i.e., non-interference.

3.3.2 Security Analysis. IFA is a well-matured field of research, and plenty of EDA tools exist for static and dynamic analysis of hardware designs. In general, all presented methods require the designer to specify a security policy and appropriate initial labeling of at least top-module in- and output ports. It is unlikely that this process can be fully automated, as a deep understanding of the system context is required and weak policies and wrong labeling can invalidate any security assurances. Hence, the community should work with experts from other fields to find a practical and secure way of representing security policies and initial labeling. Ideally, that information is intelligible for designers while allowing automatic transformation into machine-readable formats.

3.3.3 Secure-Design Generation. The state of the art offers two different approaches for generation of IFA-secure hardware. Either by adding extra shadow logic to enforce a defined policy dynamically [93] or via program synthesis [5]. The latter results in a more efficient logic but has to overcome the problem of scalability. Even more challenging and without current support is the automatic transformation of a design with security flaws into a secure design without an additional monitor for dynamic verification. Ideally, this allows low-level control over the resulting logic.

3.3.4 Security-Aware Optimization. Currently, no optimization passes explicitly respect the constraints defined by a given security policy, i.e., no information-flow-aware optimization methods. However, as the information flow is deeply embedded in the behavioral structure of the design, most performed optimization should be fine even without special treatment. Nevertheless, a formal analysis that could provide security guarantees is missing. This is especially important since some optimization can violate the separation of information flow, e.g., removing redundant structures.

3.3.5 Security-Oriented EDA Tool-Flow. The last changes to the logical structure of a design are done during technology mapping. Hence, security guarantees need to span from the check of a security policy to technology mapping. There is a trend towards IFA at higher levels of abstraction, which result in more efficient methods but also requires more efforts in security-aware optimization and transformation. A state-of-the-art EDA tool-flow loses any guarantees of the information flow as soon as some logical transformation is performed after IFA. Hence, formal guarantees are required but missing for the entire design flow.

3.3.6 Required Information and Modules. Tools with different complexity/precision trade-offs collect various data during label propagation. Over-approximating methods require security labels of circuit inputs to derive security labels of circuit outputs. In contrast, more precise methods also consider the operation and input values themself. Besides, the abstraction level of the analysis influences the precision. While information flow has a uniform appearance at gate level, one can distinguish *explicit* (e.g., direct assignments) and *implicit* (e.g., indirect control of assignments) flows at RTL and different functional units during HLS. In the context of static IFA, tools need to propagate security labels (either directly or symbolically) and then check for policy violations by direct label comparison or an Satisfiability Modulo Theories (SMT) solver. An SMT solver gets as input some mathematical formula with constraints to variables and will output whether there exists a valid solution. In that, SMT is a generalization of the satisfiability problem of Boolean functions. Dynamic IFA tools create shadow registers for sequential circuits and shadow logic for combinational parts. Then the concrete policy verification is done during runtime or design time using simulation.

3.3.7 Downgrading. An interesting feature of today's tools is downgrading, which allows certain policy violations if they are secure (e.g., information flow from key to the ciphertext in encryption) and is often required to be specified explicitly by the designer. This poses the question of automatic identification of downgrade opportunities. One way for automation is quantitative IFA [128], where the design of meaningful metrics is challenging. Hence, we expect future tools to explore different ways of automatic downgrading, with and without quantitative IFA.

4 TIMING SIDE-CHANNEL ANALYSIS

For correct manipulation of information, any practical system requires some notion of time, either implicit, e.g., by order of instructions, or explicit, e.g., in terms of clock signals. However, as the processed information determines the sequence of system transformations, the timing behavior might be correlated. In such cases, *data-depended timing variance* might reveal (partial) information of the system [87], opening a side channel to observe and infer sensitive information. In contrast to other side channels, timing side channels are also accessible from the software layer and even remotely.

Removing existing correlation between information and timing behavior successfully protects against timing-side-channel attacks, however, at the cost of always taking a constant and worst-case amount of time when processing confidential information. Hence, Timing Side-Channel Analysis (TSCA) looks for data-dependent timing variances by analyzing secret-dependent execution paths and their timing behavior. Naturally, TSCA can be performed through IFA, as system timing is also related to control and implicit information flow. However, adding the notion of time to IFA requires additional timing models, while direct analysis of timing behavior is possible in low-level design descriptions where the timing behavior is explicit.

4.1 Domain-Specific Taxonomy

In accordance with the general taxonomy we introduce the following TSCA-specific security features.

Flow Type Discrimination supports distinguishing timing from other information flows.

Time Blocking Detection enables detection of measures that prevent timing flow propagation.

4.2 Current Research

Table 3 lists state-of-the-art TSCA methodologies for hardware design, grouped based on methods that rely on traditional IFA, those that add a shadow logic to trace timing flows explicitly, those that use static IFA techniques to analyze timing directly, those that use unique program execution, those that use timing constraints to construct constant-time hardware, and those that use black-box analysis for TSCA.

4.2.1 Traditional IFA. As timing variances originate from information-dependent control flow, i.e., implicit information flow, IFA is generally able to detect timing side channels but cannot discriminate between functional and timing-based flows [134]. As a result, a timing model is used, assuming that any secret-dependent implicit information flow results in timing leakage. This

Table 3. Timing Side-Channel Analysis Methods for Hardware in Literature

	Building Block	Entire Design	CPU	Generic	Libraries	Design Level	Flow Type Discrimination	Time Blocking Detection	Secure/Insecure	Security Level	Quantitative	Database	Messaging	Files	Annotation	Dedicated IR	Monolithic	Front-/Backend	Modular	Incremental	Dynamic/Static	Pass Type	Year	Age
see Table 2							○	○																
Oberg et al. [103]		●		●	○	Gate	◑	○	●			○	○	○	○	○	●			○	D	A	2013	Invention
Clepsydra [3]		●		●	●	RTL	●	●	●			○	○	○	○	◑	●			○	D	A	2017	Invention
VeriSketch [5]		●		●	○	RTL	●	●	●			○	○	○	○	◑			●	○	S	T	2019	Implementation
Qin et al. [109]		●		●	●	Gate	●	●	●			○	○	○	○	○	●			○	D	A	2020	Implementation
Iodine [53]		●		●	○	RTL	○	●	●			○	○	○	○	◑		●		○	S	A	2019	Invention
PASCAL [89]		●		●	○	RTL	●	●	●			○	○	○	○	○	●			○	S	T	2019	Implementation
XENON [78]		●		●	○	RTL	○	●	●			○	○	○	●	◑		●		◑	S	A	2020	Implementation
UPEC [41]		●	●		○	RTL	●	●	●			○	○	○	○	○	●			○	S	A	2019	Invention
Fadiheh et al. [40]		●	●		○	RTL	●	●	●			○	○	○	○	○	●			○	S	A	2020	Implementation
Peter & Givargis [105]		●		●	○	RTL*	◑	◑	●			○	○	○	●	●	●			○	S	T	2016	Invention
ASSURE [71]		●		●	○	RTL*	○	●	●			○	○	○	○	●	●			○	S	T	2018	Implementation
Covert Shotgun [48]		●	●		○	Exec	●	○			●	○	○	●	○	○	●			○	D	A	2016	Invention
ABSynth [57]		●	●		○	Exec	●	○			●	○	○	●	○	○			●	○	D	A	2020	Implementation
Osiris [139]		●	●		○	Exec	●	○			●	○	○	●	○	○			●	○	D	A	2021	Implementation

Symbols indicate whether the method has the property (●), has the property partially (◑) or does not have the property (○). An entry with *, indicates that the method uses the reduction to the indicated design level.

approach allows traditional IFA methods (listed in Table 2) to ensure *timing-sensitive non-interference*. Oberg et al. use different pairs of input traces to conclude the absence of functional information flow through system simulation and report a timing flow if traditional IFA still reports an existing flow [103, 104]. Hence, the tool can identify timing flows in the absence of functional flows but cannot distinguish between them if both coexist.

4.2.2 Shadow Logic. The idea of shadow logic, i.e., a parallel logic dedicated to information flow tracking, can also be applied to track timing flows explicitly. Clepsydra [3] identifies timing variances through register updates that are controlled by sensitive signals and traces their flow through the system by introducing a dedicated shadow logic. The tool stops the propagation of timing flows when a register fully controlled by non-sensitive signals is reached. VeriSketch [5] extends this approach to create functional logic providing timing-sensitive non-interference out of not entirely specified RTL sketches, using program synthesis. Qin et al. provide another extension of the idea by directly forwarding the number of cycles required to compute a value through the shadow logic [109]. Thereby, registers increase the value propagating through the shadow logic while combinatorial logic forwards those values and selects the most significant. Naturally, addition of shadow logic adds some overhead in terms of area and power consumption. Hence, this method is of limited use for constrained devices.

4.2.3 Static TSCA. Like shadow logic for dynamic timing analysis, techniques inspired by static IFA can also be used to do static analysis of the system timing behavior. The first approach in this direction was Iodine [53], which marks wires and registers as *live* whenever their value is determined by the current cycle. Then, widely adopted formal verification tools for assertion checking are used to verify that all possible executions, under specified assumptions, run in constant time. Similarly, PASCAL [89] enumerates all possible paths from a sensitive variable to an output and uses formal methods to check that all paths are balanced. If paths are unbalanced, PASCAL proposes to add a cycle counter for affected registers. XENON [78] uses the analysis technique of Iodine and extends it in two ways. First, it enhances scalability by analyzing the timing behavior for each submodule individually and concludes the overall timing behavior based on the results. Second, it improves usability by providing counterexamples and suggestions for variables that could be labeled non-sensitive when XENON fails to prove constant-time security. In general, the use of formal methods is limited in the design size they can handle, making those approaches hard to scale.

4.2.4 Unique Program Execution. For some systems, it is sufficient to ensure that executed software always observes constant-time behavior. Unique Program Execution Checking (UPEC) [41] formally verifies that the content of a secret memory location is not able to alter the observation at the architectural level (defined by the Instruction Set Architecture (ISA)) of the processor in a given time window or provides a counterexample. For efficiency, the tool also reports timing differences at the microarchitectural level (not observable by software), as those are always a precondition of observable leakage. The separation of microarchitectural and architectural state variables in the input RTL design has to be specified by the user. Internally, interval property checking, which is a type of bounded model checking, is used in combination with appropriate constraints. Fadiheh et al. [40] extend this method for more complex scenarios by transforming UPEC to a trace property and adding some more constraints, without restricting generality. Specifically, this allows the analysis of out-of-order execution. The analysis based on unique program execution is specific to the analyzed software and does not provide any general guarantees.

4.2.5 Timing Constraints. During HLS a scheduler is used to assign operations to specific clock cycles. Peter and Givargis add additional constraints to this scheduler to balance execution time [105], resulting in additional idle states for short basic blocks of the Control-Flow Graph (CFG). ASSURE [71] extends this approach by creating two different controller FSMs, i.e., state machines that control the behavior of the circuit, distinguishing outputs observable by authorized and unauthorized entities. While outputs observable by unauthorized entities are forced to be constant in time, other outputs may have a variable-time behavior, and responsibilities are delegated to subsequent processes handling this data.

4.2.6 Black-Box Analysis for TSCA. In later design stages an executable model of the design is usually available, making black-box analysis methods viable options. A simple analysis for contention-based timing side channels of multi-scalar or hyper-threading processors is Covert Shotgun [48], which uses a brute-force-like method to compare the execution time of different instruction pairs. This approach was refined by ABSynth [57], which first creates a leakage map of different instruction pairs at the target hardware and then uses an evolutionary search algorithm to find the best attack code for a given victim software. As for Covert Shotgun, contention-based timing side channels can be detected only. A similar approach is implemented by the fuzzing-based tool Osiris [139], which is able to find timing side channels, which are not based on resource contention. For this, a machine-readable ISA definition is utilized to create an attack sequence consisting of a reset, a trigger, and a measurement instruction. Afterwards, the attack sequence is executed with and without the trigger instruction, while the execution time is measured. If a timing difference is detected, the side channels are validated and clustered based on heuristics. However, as no design internals are used, these methods can not report any root causes for existing timing vulnerabilities.

4.3 Observations and Research Challenges

Comparing state-of-the-art research in security-aware EDA with respect to TSCA, we discuss essential observations and outline possible directions for extensions and future work.

4.3.1 Security Model. As in the domain of IFA, there is a simple and clear security model for TSCA, namely *constant-time execution*. This results in tools with similar features even as different approaches are explored. Due to the similarity with general IFA, most methods seem to be inspired by methods from this domain.

4.3.2 Security Analysis. Similar to IFA, the analysis of timing side channels is a well-matured field with approaches doing IFA and approaches specifically tailored to TSCA. For the identification of timing side-channels, it is sufficient to analyze the behavior of registers. While only unbalanced register updates, i.e., a register update that is optional and depends on a (sensitive) control signal, can cause timing leakage, registers that are fully controlled by (non-sensitive) signals block the propagation of timing variances [3]. Hence, RTL is the natural abstraction for analysis passes. However, current analysis methods are limited to designs with a single clock domain. Hence, a natural and interesting extension are methods that consider multiple clock signals and their interaction.

4.3.3 Secure-Design Generation. The construction of TSCA-secure logic follows three different approaches: (i) the construction of additional logic to balance the timing behavior of different paths [89], (ii) the construction of secure logic using program synthesis [5], and (iii) the construction of secure logic during HLS [71, 105]. Especially the last approach is simple and efficient, as timing is introduced during HLS via a scheduler by construction and analysis of a CFG. Again, an interesting extension of existing methods is the generation of designs with multiple clock domains.

4.3.4 Security-Aware Optimization. For functional correctness, *retiming* optimization, i.e., moving logical gates across register boundaries within modules, is usually viable as long as timing behavior at module boundaries remains intact [125]. Depending on the adversarial observation or manipulation capabilities, such optimization might introduce subtle but exploitable timing variances. Otherwise, optimization techniques trivially maintain established timing properties at cycle granularity and require no additional consideration.

4.3.5 Security-Oriented EDA Tool-Flow. As transformation and optimization are viable in the context of TSCA, with the only exception being retiming, a complete tool flow can be constructed from existing EDA tools with disabled retiming.

4.3.6 Required Information and Modules. Most presented methods discriminate between timing variances caused by sensitive or non-sensitive information and, thus, require some prior IFA with variable labeling. Afterwards, we divide TSCA into detecting sources of timing variances and the propagation of existing flows with integrated detection of timing flow blocking. In an additional step, formal verification methods guarantee the absence of timing side channels at arbitrary execution.

5 POWER-SIDE-CHANNEL ANALYSIS

Other physical characteristics than timing can also correlate with the processed information and hence form a side channel. In general, Side-Channel Analysis (SCA) exploits such secret-dependent leakage of physical systems by observing a physical source that correlates with sensitive internal information [49, 67, 87, 88]. In this section, we focus on Power-Side-Channel Analysis (PSCA) as the most prominent and well-studied example of SCA. A similar survey of tools for side-channel evaluation is given by Buhan et al. [25]. However, while they focus on tools for leakage assessment of mostly software-executing systems, we survey tools for analysis and transformation within an EDA tool-flow.

Over time, many PSCA countermeasures have been presented, where *Boolean masking* (based on *secret sharing*) is the most promising approach due to its formal and sound security foundation [33]. Unfortunately, due to design flaws, inaccurate models, or violated assumptions, many proposals were insecure and had to be withdrawn. As a consequence, research started to focus on the development of more accurate formal models for adversaries, security notions, and physical execution environments to facilitate the formal verification of countermeasures, designs, and implementations [11–13, 31, 38, 42, 69].

5.1 Domain-Specific Taxonomy

In accordance with the general taxonomy we introduce the following PSCA-specific security features.

Uniform Sharing ensures that valid sharings for each unshared value occur equiprobable.

Physical Defaults considers and models the presence of unintentional physical effects such as *Glitches*, *Transitions*, and *Cross-Talk*.

Composability analyzes or uses the ability to compose masked gadgets securely through *Non-Interference (NI)* [11], *Strong Non-Interference (SNI)* [12], or *Probe-Isolating Non-Interference (PINI)* [31].

5.2 Current Research

Table 4 lists modern and state-of-the-art PSCA research on security-aware EDA, grouped based on the analysis and construction methods that target rather specific countermeasures, those that do probing-security analysis, those that provide composition-based probing security, those based on leakage analysis, and those based on power simulation.

5.2.1 *Specific Countermeasures.*

Among the candidates in Table 4, VerMI [6] is a limited case as the tool only focuses on verification of Boolean masking in terms of Threshold Implementations (TIs) and their fundamental properties *non-completeness* and *uniformity*. Further, uniformity is checked through simulation, hence, limited by circuit complexity and number of primary inputs.

5.2.2 *Probing Security.*

The candidates for probing security perform analysis of masked circuits against the commonly used, simple, and abstract d-probing model. In this model, an adversary gets access to a circuit that can be executed multiple times, while prior to each invocation, the adversary selects up to d wires of which the carried values are leaked on execution. As a seminal work, REBECCA [23] verifies masked circuits based on the estimation of Fourier coefficients, which allows characterization of statistical dependencies between gates and inputs and can predict potential leakage. In contrast to this, `maskVerif` [10] uses a more efficient language-based probabilistic IFA but can result in false negatives due to overly conservative transformations. Besides improved performance, the latest version of `maskVerif` supports a wider range of security notions, including verification of NI and SNI properties for masked gadgets. As an alternative solution, SILVER [83] is based on Reduced Ordered Binary Decision Diagrams (ROBDDs) to verify d-probing security, NI, SNI, PINI, and uniformity of a masked gadget checking statistical independence of binary random variables. VRAPS [16] adapted the random probing model as d-probing model extension to capture leakages from multiple successive manipulations of the same sensitive information. Most recently, `IronMask` [19] used arithmetic characterization and Gaussian elimination to significantly speed

Table 4. Power-Side-Channel Analysis Methods for Hardware in Literature

	Building Block	Entire Design	Cryptographic	Generic	Libraries	Design Level	Uniform Sharing	Glitches	Transitions	Cross-Talk	Non-Interference	Strong NI	Probe Isolating NI	Secure/Insecure	Security Level	Quantitative	Database	Messaging	Files	Annotation	Dedicated IR	Monolithic	Front-/Backend	Modular	Incremental	Dynamic/Static	Pass Type	Year	Age
VerMI [6]		●	●		○	Gate	◑	○	○	○	○	○	○	●			○	○	○	○	○	●			○	S	A	2018	Invention
`maskVerif` (v1) [11]	●		●		○	Gate	○	○	●	○	○	○	○	●			○	○	○	○	○	●			○	S	A	2015	Invention
`maskVerif` (v2) [12]	●		●		○	Gate	○	○	●	○	●	●	○	●			○	○	○	○	○	●			○	S	A	2016	Implementation
REBECCA [23]	●		●		○	Gate	○	●	○	○	○	○	○	●			○	○	○	○	○	●			○	S	A	2018	Implementation
`maskVerif` (v3) [10]	●		●		○	Gate	○	●	●	○	●	●	○	●			○	○	○	○	●		●		○	S	A	2019	Implementation
SILVER [83]	●		●		○	Gate	●	●	○	○	●	●	●	●			○	○	○	○	●		●		○	S	A	2020	Implementation
VRAPS [17]	●		●		○	Gate	○	○	○	○	○	○	○			●	○	○	○	○	○	●			○	S	A	2020	Implementation
IronMask [15]	●		●		○	Gate	○	●	●	○	●	●	●	●			○	○	○	○	●	●			○	S	A	2021	Implementation
PROLEAD [101]		●		●	○	Gate	○	●	●	○	○	○	○			●	○	○	○	○	○	●			○	D	A	2022	Implementation
`tightPROVE` [18]		●	●		○	RTL	○	○	○	○	●	●	○	●			○	○	○	○	○	●			○	S	A	2018	Invention*
`tightPROVE`$^+$ [19]		●	●		○	RTL	○	○	○	○	●	●	○	●			○	○	○	○	○	●			○	S	A	2020	Implementation*
TORNADO [19]		●	●		○	RTL	○	○	○	○	●	●	○	●			○	○	○	○	○			●	○	S	T	2020	Implementation*
`fullVerif` [29]		●	●		●	Gate	○	●	○	○	○	○	●	●			○	○	○	○	○			●	○	D	A	2020	Implementation
AGEMA [82]		●		●	●	Gate	○	●	○	○	○	○	●		●		○	○	○	○	◑			●	○	S	T	2022	Implementation
SAIREDA [44]		●		●	○	Gate	○	●	○	○	●	●	●		●		○	○	○	●	●			●	○	S	T	2022	Implementation
AMASIVE [143]		●		●	○	RTL	○	◑	○	○	○	○	○	●			○	○	○	○	●			●	○	S	A	2012	Invention
AMASIVE+ [66]		●		●	○	RTL	○	◑	○	○	○	○	○	●			○	○	○	○	●			●	○	S	T	2017	Implementation
RTL-PSC [60]		●	●		○	RTL	○	○	○	○	○	○	○			●	○	○	○	○	○	●			○	D	A	2019	Implementation
RTL-PAT [107]		●		●	○	RTL	○	○	○	○	○	○	○			●	○	○	●	○	○			●	○	D	A	2022	Implementation
FORTIFY [90]		●		●	○	Gate	○	○	○	○	○	○	○			●	○	○	○	○	○	●			◑	S	A	2022	Implementation
Karna [127]		●		●	●	Gate	○	●	●	○	○	○	○			●	○	○	●	●	◑			●	○	D	T	2019	Invention
PACA [141]		●		●	○	Gate	○	●	●	○	○	○	○			●	○	○	○	○	○	●			○	D	A	2021	Implementation
ACA [77]		●		●	○	Gate	○	●	●	○	○	○	○			●	○	○	●	○	○			●	○	D	A	2022	Implementation
PathFinder [96]		●		●	●	Gate	○	●	●	○	○	○	○			●	○	○	●	●	○			●	○	D	T	2022	Implementation

Symbols indicate whether the method has the property (●), has the property partially (◑), or does not have the property (○).

up the verification of arithmetic gadgets in the *d*-probing and random probing model. Another approach to speed up checks for probing security is taken by PROLEAD [101], which does not perform a rigorous formal verification but instead uses a statistical independence test (G-Test [98]) to check for the independence of secrets and probed values. Here the confidence level, i.e., the probability of false negatives, can be adjusted to the needs of the user and controls the number of performed simulations. In general, a higher confidence level requires more time and provides a higher assurance, nevertheless without providing proof of security. The reason is that it cannot be guaranteed that all possible test vectors are indeed tested.

5.2.3 Composition-Based Probing Security. The tools `tightPROVE` [17] and `tightPROVE`$^+$ [18] analyze a circuit, assuming securely implemented `AND`, `XOR`, `NOT`, and special share-refreshing `REF` gadgets (for masking, refreshing random masks is required to prevent leakage from dependencies between intermediate values). Given this, verification of the composition of such gates eventually either results in a security proof for the *d*-probing model or discovers a security flaw, which directly relates to a successful set of probes. However, as share-refreshing at carefully selected positions in the circuit can fix the security flaws, TORNADO [18] can produce secure masked bit-slice implementations in order to achieve the desired level of d-probing security. Similarly, the `fullVerif` verification framework [29] analyzes correct implementation of masked implementations based on the composition of glitch-robust and trivially composable gadgets (under the PINI security notion). Generating a hierarchy-preserving, pre-synthesized netlist composed of the masked gadgets, the tool extracts the data-flow graph through simulation and verifies it for secure composition (based on internal annotation propagation and composition rules). Based on Mealy machines [99], AGEMA [82] generates masked circuits out of arbitrary unprotected netlists. Leveraging IFA only the required part of the circuit is masked using a variety of PINI gadgets [29, 81, 84]. One step further goes SAIREDA [44], which optimizes the amount of randomness used for mask-refreshing after achieving side-channel security via gadget insertion. Specifically, the tool groups gadgets into different *clusters* such that gadgets within a cluster can share randomness while maintaining probing security. However, this process does not include the evaluation and verification of the masked gadgets, which still has to rely on additional checks using one of the *direct probing security* verification tools in Table 4.

5.2.4 Leakage Analysis. Due to missing physical and layout information, verification of security on HDL representations follows the approach of leakage simulation and assessment. For this, designs are simulated under various inputs and leakage is modeled through a leakage function and operation noise. For this, the AMASIVE [66, 143] framework uses graph-based circuit modeling in combination with Hamming-weight and Hamming-distance power models for leakage prediction while security analysis is performed using a correlation-based distinguisher. In contrast to this, RTL-PSC [60] uses simulation-based generation of the switching activity information to estimate the power leakage distribution. Eventually, using the Kullback-Leibler divergence and success rate based on the estimated power-leakage distribution, vulnerable design blocks are identified and reported. Later, RTL-PAT [107] optimized this approach using state-of-the-art RTL simulation resulting in Value Change Dump (VCD) files, making the evaluation faster and the tool more modular. Recently, FORTIFY [90] proposed an analytic way of leakage estimation based on IFA and conditional probabilities between intermediate values and secrets. However, for efficient analysis, FORTIFY assumes that all input signals to a logical gate are independent of each other, which is usually not true in reality. Hence, the resulting leakage assessment must be seen as an approximation.

5.2.5 Power Simulation. At lower levels, power simulation can be performed to locate the source of power side-channel leakage [75]. In this regard, Karna [141] divides a circuit with placement information into regions and evaluates the security of each region based on a Test Vector Leakage Assessment (TVLA) [55] with simulated power traces. For regions where the leakage surpasses a specified threshold the leakage is reduced by reconfiguring the affected gates with respect to V_{dd}, V_t, and *size*. Similarly, PACA [141] determines occurring leakage by power simulation in combination with a specific leakage model (e.g., Hamming-weight). Then the correlation between the toggle count of a single gate and the occurring leakage is computed to identify the gates that contribute most to the leakage. ACA [77] extends PACA by using non-specific TVLA instead of a specific leakage model and increases efficiency by averaging the simulation values within specified time windows, as proposed by Kiaei et al. [76]. Building upon the vulnerability ranking of individual gates, PathFinder [96] identifies leaking paths by clustering vulnerable gates from the same computation path using static graph analysis. Afterwards, the vulnerable parts are protected by a combination of masking and random pre-charging. Similar to the presented techniques for power simulation, there also exists preliminary work for pre-silicon evaluation in the context of EM side channels [137], however, not yet in a fully automatic way. The effectivity of methods in this category is normally analyzed empirically, without formal models.

5.3 Observations and Research Questions

Comparing state-of-the-art research in security-aware EDA with respect to PSCA, we discuss essential observations and outline possible directions for extensions and future work.

5.3.1 Security Model. Endeavoring a comprehensive analysis of cryptographic implementations most methods employ a variant of the *probing* adversary model to prove the absence of side-channel leakage. Despite being of simple and tangible nature, this model limits the analysis to atomic building blocks due to an exponential increase in complexity with each additional probe. This motivates auxiliary properties, used to verify entire designs through a composition of secure building blocks. Other methods model side-channel leakage as correlated to some function of intermediate values (e.g., Hamming weight or distance). While often fast and easy to evaluate, this approach is rather attack specific and captures only parts of existing leakage.

5.3.2 Security Analysis. The analysis and verification of security for PSCA are challenging for scalability, both in design complexity and security order. A significant speed-up is achieved by reducing the analysis to gadgets of a specific form or verifying the secure composition of abstract gadgets. However, this lacks the ability of general analysis and, thus, imposes some overhead only for the sake of verification. In addition, identification and detection of (power) side channels are challenging tasks at design time due to their inherent physical character and uncertain manifestation. For this, accurate modeling of leakage behavior requires detailed information on the physical implementation and execution environment, which only becomes available at lower levels such as RTL and gate-level. As long as the information is not available with sufficient accuracy, hypothetical power and leakage models are considered. For this, all presented approaches are located at RTL or gate-level and consider dedicated adversary, leakage, or power models. Please note there is extensive research on evaluation of side-channel security after fabrication (e.g., [8, 50, 97]). While valuable, they are not part of the EDA landscape, and we leave their discussion here.

5.3.3 Secure-Design Generation. The construction of provable secure circuits in the abstract probing model is well established via the composition of secure gadgets [18, 82]. While simple, this approach suffers from a local, unnecessary overhead to achieve global security in the context of PSCA. This directly poses the question of other constructive methods based on masking that are more efficient than the composition of gadgets. constructive methods based on hiding are much more efficient [127] however validated experimentally only. An important direction for future research is the generation of circuits that account for leakage from transitions (switching activity between clock cycles) into account. Preliminary work in this direction was done by Cassiers and Standaert [32], however, without tool support.

5.3.4 Security-Aware Optimization. There exist some work in the reduction of required gadgets (e.g., [17, 18, 29]) that has influenced the existing methods for secure-design generation. Other works optimize the randomness consumption of masked software implementations [15, 30, 36, 43, 56, 68, 138]. Nevertheless, we do not cover those works here as the transformation to hardware is not always trivial, and they do not offer automatic optimization passes (except for SAIREDA [44]). In general, the glitch-extended probing model [42] has to be considered for hardware. In this model, the adversary gains insights into the values at the last synchronization point, i.e., the last dependent register stage. Therefore, the implemented function does not impact the leakage and optimization that respects register boundaries (hence, without retiming) preserve established security properties.

5.3.5 Security-Oriented EDA Tool-Flow. While logic transformations between register stages do not affect the glitch-robust probing security, other low-level design decisions have an impact.

For example, the routing impacts the occurring cross-talk, and the power network links different sources of leakage together. However, state-of-the-art PSCA tools do not consider such low-level effects. Hence, this points to a promising and imperative line of research for a complete PSCA-aware design flow. Research in this direction will likely uncover a mismatch between theory and practice as not all sources of leakage, e.g., power networks and static leakage, are covered by current leakage models.

5.3.6 Required Information and Modules. Formal verification of probing security and secure composition of gadgets requires a pre-processing step, identifying the information flow of secret data within the circuit. Further, verification of secure composition first requires the verification of gadgets, ideally supporting compartmentalization of tasks and responsibilities. Experience has shown that security does not stop at design or implementation boundaries but requires a holistic view of the system and its environment. Although security should be constructed and evaluated in a bottom-up approach, ensuring solid and sound foundations, extension to higher levels of abstraction is indispensable.

6 FAULT INJECTION ANALYSIS

Correct operation of a system is only guaranteed under well-defined conditions, while any execution outside the specified conditions might result in undefined behavior and a corrupted system state. In this regard, security mechanisms are no exception, and any event outside specified boundaries is considered a fault and might affect system security properties. For this, malicious fault injection can lead to security violations, e.g., breach of integrity, leakage of cryptographic keys [22, 24], or the disabling of security mechanisms.

The different nature of faults requires dynamic handling at runtime, either by detecting or correcting any occurring fault. While fault correction is self-sufficient, countermeasures based on detection need to handle faults appropriately, either by aborting or rendering the system behavior useless for an adversary (*infection*). This runtime requirement enforces some overhead in the actual implementation, mostly due to redundancy based on information, area, or time. However, even as error handling can not be done statically, a given system can be analized for its susceptibility against fault injection statically. Hence, one major objective of Fault Injection Analysis (FIA) is to distinguish faults that might result in an exploitable behavior and those that *just* lead to a system error.

6.1 Domain-Specific Taxonomy

In accordance with the general taxonomy we introduce the following FIA-specific security features.

Fault Model allows the attacker to fix (*stuck-at*) or change (*bit flip*) the value of a bit.
Location distinguishes between *gate-input* and *gate-output* fault locations.
Distribution describes faults that occur *uniformly* (randomly) or have a *bias*.
Quantity allows *univariate* (single) or *multivariate* (multiple) fault injections.
Attack Specific methods have narrow applicability.
Countermeasures can be analyzed for sufficiency under some considered fault attacks.
Mechanism distinguishes *error detection*, *error correction*, and *infection* countermeasures.
Redundancy uses *information* (codes), *area* (parallelism), or *timing* (repetition) to detect or correct faults.

6.2 Current Research

Table 5 (generic taxonomy) and Table 6 (domain-specific taxonomy) list state-of-the-art FIA methodologies for hardware design, grouped based on the methods handling rather specific fault

Table 5. Fault Injection Analysis Methods for Hardware in Literature (General Taxonomy)

	Building Block	Entire Design	Cryptographic	Generic	Libraries	Design Level	Secure/Insecure	Security Level	Quantitative	Database	Messaging	Files	Annotation	Dedicated IR	Monolithic	Front-/Backend	Modular	Incremental	Dynamic/Static	Pass Type	Year	Age
AVFSM [102]	●			●	○	Gate*			●	○	○	○	○	◑	●			○	D	A	2016	Invention
AutoFault [26]		●	●		○	RTL/Gate		●		○	○	○	○	○	●			○	D	A	2017	Invention
Gay et al. [51]		●	●		○	RTL/Gate		●		○	○	○	○	○	●			○	D	A	2019	Implementation
XFC [74]		●	●		○	Algo			●	○	○	○	○	○	●			○	S	A	2017	Invention
DFARPA [73]		●	●		○	Algo+Gate			●	○	○	○	○	○			●	○	S	T	2018	Implementation
SAFARI [117]		●	●		●	Algo			●	○	●	○	○	○			●	○	S	T	2019	Implementation
SOLOMON [130]	●		●		○	Algo+RTL			●	○	○	○	○	○	●			○	S	A	2020	Implementation
ExpFault [119]		●	●		○	Algo+Exec			●	○	○	○	○	○	●			○	D	A	2018	Invention
VerFI [7]		●		●	○	Gate			●	○	○	◑	○	○		●		○	D	A	2019	Implementation
SoFI [137]		●		●	○	Gate			●	○	○	○	○	○	●			○	D	A	2021	Implementation
FIVER [116]		●		●	○	Gate			●	○	○	○	●	●		●		○	S	A	2021	Invention
Eldib et al. [39]	●			●	○	Gate	●			○	○	○	○	◑	●			◑	S	T	2016	Invention
Avatar [118]		●		●	●	Gate			●	○	○	○	○	○	●			○	S	T	2022	Invention

Symbols indicate whether the method has the property (●), has the property partially (◑), or does not have the property (○). An entry with * indicates that the method uses the reduction to the indicated level.

Table 6. Fault Injection Analysis Methods for Hardware in Literature (Domain-specific Taxonomy)

	Stuck-At	Bit Flip	Gate-Output Faults	Gate-Input Faults	Uniform Faults	Biased Faults	Univariate Faults	Multivariate Faults	Attack Specific	Countermeasures	Error Detection	Error Correction	Infection	Redundant Information	Redundant Area	Redundant Timing
AVFSM [102]	●	○	○	○	○	○	●	○	●	●	●	○	○	○	○	○
AutoFault [26]	○	●	●	○	○	●	●	○	●	◑						
Gay et al. [51]	○	●	○	○	○	●	●	●	●	●						
XFC [74]	○	●	◑	○	○	●	●	○	●	○						
DFARPA [73]	●	●	○	○	○	●	●	●	●	○	○	○	○	○	○	○
SAFARI [117]	○	●	●	○	○	●	●	○	●	○	●	○	●	●	●	○
SOLOMON [130]	○	●	●	○	○	●	●	○	●	○						
ExpFault [119]	○	●	○	○	○	●	●	○	●	●						
VerFI [7]	●	●	●	◑	●	●	●	●	○	●						
SoFI [137]	●	●	●	○	●	●	●	●	○	●						
FIVER [116]	●	●	●	●	●	●	●	●	○	●						
Eldib et al. [39]	○	○	○	○	○	○	○	○	●	●	○	○	●	○	○	○
Avatar [118]	◑	◑	○	○	○	●	○	●	○	○	●	○	○	○	●	○

Symbols indicate whether the method has the property (●), has the property partially (◑), or does not have the property (○).

models, those that generate algebraic equations for analysis, those that analyze a high-level cipher specification, those based on fault simulation, those based on symbolic fault injection, and those that use property-driven synthesis.

6.2.1 Specific Fault Models. An Attacker can use fault injection to manipulate an FSM to provoke the transition to a sensitive state. This scenario can be analyzed by AVFSM [102]. During synthesis, and in particular design optimization, some *don't care* states and transitions might be added to an FSM. An attacker can inject a fault that transfers the FSM to such a don't care state, which in turn has a transition to a sensitive state. Since don't care states do not exist at the RTL, AVFSM takes a gate level description and generates the associated FSM with don't care states, to analyze it further.

6.2.2 Algebraic Equation Generation. A more general type of fault attack is Algebraic Fault Attack (AFA), where algebraic cryptanalysis is enhanced by additional equations obtained through faults. AutoFault [26] was the first to automatically generate the necessary algebraic equations from a hardware description based on Tseitin transformations [135]. Gay et al. extended this approach to a multivariate fault setting [51].

6.2.3 High-Level Cipher Specification. Most existing fault injection attacks target specific cryptographic systems and exploit mathematical properties. As this information is already available at a high-level cipher description, an analysis at this level can guide the selection and integration of countermeasures. Such analysis is done in XFC [74], which uses IFA techniques to trace the propagation of specified faults through a block cipher, find potential key distinguishers for Differential Fault Analysis (DFA), i.e., differentials between correct and faulty intermediate states, and ultimately provide the attack complexity for a given fault location. A similar analysis is done in DFARPA [73] where the outcome is used to inform the placement of registers, to make coarse-grain faults less exploitable, and insert watchdog ring oscillators, to detect more focused fault injections. SAFARI [117] leverages XFC to generate a DFA-secure hardware description automatically. Thereby, appropriate countermeasures are selected based on the vulnerable locations and the corresponding attack complexity as reported by XFC. Then the cipher description is altered accordingly. Afterwards, the countermeasure-enhanced specification is synthesized to RTL. SOLOMON [130] also extends XFC by mapping the reported vulnerable locations from the high-level cipher description to a specific hardware implementation at RTL or gate level.

6.2.4 Simulation-Based FIA. By simulation of both the correct and faulty behavior of a system, valuable insights about possible attacks and the effectiveness of countermeasures can be drawn. Such simulation is used by ExpFault [119] to find potential key distinguishers for DFA, using techniques from data mining. A mathematical description of the implemented cipher is then used to estimate the complexity of the related attack. A more general approach takes VerFI [7] by providing the means for simulation based on a wide variety of fault models. The tool can evaluate the effectiveness of countermeasures by reporting detected, undetected, and ineffective faults, however human expertise is required for setup and evaluation. SoFI [137] uses executable security properties to directly identify exploitable faults, making the evaluation more comprehensive, however, at the cost of a more complex setup phase due to the necessity to generate appropriate security properties.

6.2.5 Symbolic FIA. Formal guarantees with respect to fault injection can be given by creating a system model that incorporates information from both correct and faulty execution for all possible inputs. FIVER [116] generates such a model based on ROBDDs as an efficient means for symbolic simulation. In particular, a ROBDD is created, such that a satisfying assignment of the

ROBDD yields a successful fault injection by generating the XOR ROBDD of the correct and faulty ROBDD. Thereby, the faulty ROBDD is generated by replacing the Boolean functions of internal gates with other Boolean functions according to a fault model [115]. In the end, a simple evaluation provides the number of effective, ineffective, and detected faults, where the designer has to mark the detection flag of detection-based mechanisms.

6.2.6 Property-Driven Synthesis. Some fault attacks, i.e., Fault Sensitivity Analysis (FSA), use timing differences at gate granularity, e.g., by providing a higher clock frequency, to obtain sensitive information statistically. Eldib et al. [39] propose inductive synthesis for finding a netlist such that the gate level timing behavior is independent of any sensitive signal. However, since inductive synthesis scales poorly with large designs, the authors use a divide-and-conquer approach to divide the system into smaller entities that are handled separately. In a different matter, redundancy in area suffers from the fact that the same fault in all redundant parts is very likely when all of them are confronted with the same stress level during the fault injection, impeding the capability for fault detection/correction. Avatar [118] strengthens two-times area redundancy by choosing different gate variants (different in terms of size, supply voltage, and threshold voltage) for the two redundant parts, building upon the observation that fault susceptibility is correlated to the delay properties of the chosen gate implementation. In this, the approach has some similarity with Karna [127] (cf., Section 5.2). By choosing one fast and one slow implementation, for the duplication, Avatar increases the difference in fault behavior in the redundant parts without changing the actual logic. However, a fast implementation increases the power consumption, while a slow variant decreases the operational frequency. For that, Avatar tries to fulfill user-specified power and delay constraints, providing the option to trade security and performance. Currently, it is an open question how to extend this approach to more than two-times redundancy.

6.3 Observations and Research Challenges

Comparing state-of-the-art research in security-aware EDA with respect to FIA, we discuss essential observations and outline possible directions for extensions and future work.

6.3.1 Security Model. In contrast to other domains, FIA has not yet a precise security model and finding one is complicated through the vast amount of possible faults. Therefore, we see many different approaches and features in FIA tools for security-aware EDA and a manual specification of specific security properties is required.

6.3.2 Security Analysis. There exist different approaches for analysis in the context of FIA, both general and specific to attack types. In absence of a clear criterion that distinguishes secure from insecure designs (c.f., simulation as proof technique in the context of PSCA), the general approaches analyze the occurrence and propagation of *effective faults*, i.e., values that are different from a fault-free execution with the same input. On the downside, this results in an exponential increase of the analysis complexity for the number of injected faults, prohibiting the analysis of large designs. Hence, clear criteria and more efficient methods are required. For specific attacks the security criteria are better understood. Thus, fault analysis is much more direct and efficient.

6.3.3 Secure-Design Generation. Existing tools can harden circuits against specific fault attacks, i.e., DFA [73, 117] or clock glitching [39]. General protection is trivially achieved by repetition with error detection/correction at the end, optionally hardened with different fault susceptibility [118]. Again, due to the missing criteria for security, more efficient countermeasures against general fault attacks are more complex to apply automatically. Besides, Table 5 clearly shows that none of the existing methods provide an automatic implementation of error correction capabilities. However, recent research in error-correcting countermeasures shows a path towards integration into an automatic hardware design flow, e.g., [123]. Also, no methods supports the implementation of temporal redundancy. However, as a system transformation for temporal redundancy requires significant changes to the controller FSM and changes to the timing behavior of the system, this is rather expected. Nevertheless, we assume that an automatic implementation is possible starting at a system-level description (using HLS), providing a reasonable alternative for area-restricted applications.

6.3.4 Security-Aware Optimization. Today, no tools consider logic optimization in the context of FIA. However, in this context, it is especially vital as protection against fault injection usually uses some redundancy, which optimization techniques try to overcome. Luckily, the introduced redundancy is not always detectable by simple optimization processes. Nevertheless, a thorough analysis is required to provide strong guarantees.

6.3.5 Security-Oriented EDA Tool-Flow. For FIA, there are open questions in all areas of the EDA tool-flow. Especially the automatic implementation of generic protection mechanisms and subsequent security-aware transformations are currently missing. However, the entire flow has to be considered, as the logical implementation, the chosen technology and configuration, and the placement of logic impact the susceptibility against faults.

6.3.6 Required Information and Modules. Nearly all presented methods require the designer to specify the concrete fault model, i.e., the location and fault type under consideration, due to the enormous large set of possibilities. Hence, considerable human expertise is required to receive meaningful results, even when, in reality, only a few faults are exploitable. An efficient method for discarding harmless faults could enable tools more feasible for inexperienced users. Further, the presented methods operate either at the algorithmic level to find cipher-specific attacks or at a low-level hardware description to analyze implementation-specific behavior. Therefore, effective FIA requires information from various abstraction levels to assess the concrete fault behavior and exploitability. After definition of interesting fault locations, the tool analyzes the propagation of faults within the system to assess the faulty behavior. This step is rather simple and done via standard IFA or simulation techniques. Finally, to determine the impact of fault injection, the exploitability and the complexity of potential attacks are analyzed. The specific approach for this step considers the fault type and adversary model, and various methods exist in the literature.

6.3.7 General Applications. One common denominator is that all existing methods target cryptographic implementations, where an attacker tries to learn some information about a secret key. As for real-world implementations, it is often much easier to circumvent the execution of encryption or security mechanisms altogether or to alter the result of final security checks. The analysis of such attacks is vital, e.g., considering general-purpose processors that might operate on sensitive data, but will differ from those for cryptographic systems.

7 DISCUSSION AND DIRECTIONS

All preceding sections show that security-aware EDA is an active field of research as new approaches are introduced constantly while existing approaches are refined and improved. At the same time, none of the presented tools are particularly designed for integration, despite some progress in the right direction, which we mostly attribute to best practices in software engineering rather than conscious decisions. As a consequence we conclude that security-aware EDA is currently still in the *Age of Implementation*, based on the observation that *state-of-the-art security-aware EDA is isolated with respect to security domains and abstraction levels.*

7.1 Isolation in Security Domains

We observe that most tools are focused on one particular security domain, except for IFA and TSCA, which inherently share some common properties, as all timing flows result from implicit information flows. Nevertheless, even though more advanced methods for TSCA have evolved from general IFA, these approaches often focus solely on timing flows, hence disregarding the aspect of integration.

Normally, practical applications require protection against multiple attack vectors, and a security-aware design paradigm should take a holistic view to tackle issues across security domains. Some research into cross-domain countermeasures already exists, e.g., [114, 120], and could be adopted by the security-aware EDA community. We argue that an integrated and security-aware EDA framework is especially suited for the task of cross-domain security, as different modules can remain specialized to their security domain and still negotiate a common solution. Of course, this requires that a cross-domain-secure solution exists and the tool has the means to find and apply it efficiently. With VERICA [113] exists some early work in this direction that combines the capabilities of SILVER [83] and FIVER [116] and, hence, verifies for both SCA and FIA in a combined attacker setting.

Therefore, an integrated and security-aware EDA framework can ensure security across security-domain boundaries.

7.2 Isolation at Abstraction Levels

We further observe that most tools operate isolated at one particular abstraction level or during a specific reduction. This isolation can have severe security consequences when subsequent transformations are not considered, which potentially degrade security. Again there exist a few exceptions but only in the domain of FIA, where some methods use both an algorithmic and an implementation-specific description for analysis. Those examples highlight the potential of cross-abstraction analysis and transformation passes.

In general, passes on a higher level of abstraction benefit of lower expenses in terms of complexity and resources and can generate timely feedback about design issues, particularly at design levels that are more applicable to human designers, eventually resulting in faster recovery from errors. In addition, the design is more mutable, as most of the implementation details are not determined yet, however, this also means that it is harder to predict the actual impact of design decisions. Diametrically opposed, passes at lower levels, especially at the lowest level can guaranty meaningful results with respect to the actual hardware device and estimate the impact of changes more precisely, but at the cost of higher expenses. In consequence, security awareness can be integrated into EDA following three major approaches (or combinations of them), i.e., iteratively updating security-related information synchronous to design transformations along with constant analysis for violations, execution of proven security-preserving design transformations only (as already considered for software compilers [9, 21, 126, 136]), or using the framework of composable security [27].

Unfortunately, not all information is available at all design levels equally. For example, the full combinational logic is only specified at the gate level, while higher levels might rely on behavioral abstraction. In contrast, the hierarchy of modules is often already resolved at the gate level, available at a structural level, and not yet entirely specified at even higher levels. Hence, an efficient design flow should anticipate later design stages, already gathering the required information, and preparing design directives. Such cross-abstraction-layer thinking of (security-aware) EDA tools has the potential to reduce the burden of the user as information, normally provided manually, can potentially be inferred at a higher level automatically.

Therefore, an integrated and security-aware EDA framework can provide information across abstraction levels for efficient security analysis and transformation.

7.3 Holistic, Integrated, and Security-Aware EDA

Based on this, we now argue that a holistic, integrated, and security-aware design paradigm is required to tackle the complexity of today's security landscape. A holistic view is required to overcome the current isolation, both in security-domains and abstraction levels, and to set up a strong line of defenses. The resulting complexity calls for an integrated approach. Finally, efficient and effective security measures require consideration of security implications during all design transformation. Hence, security should be established as a fundamental design constraint, alongside speed, area, and power. In this context, artificial intelligence and machine learning will certainly play a role. However, this part is extensively discussed by Koblah et al. [85], and we refer the interested reader to their work.

Our analysis shows that some of the proposed methods are only necessary because of the lack of security-awareness in EDA. For instance, SOLOMON [130] could be replaced by a hardware design flow that analyzes the system-level description for vulnerable locations and then keeps track of those locations while transformations and reductions take place. Similarly, AVFSM [102] could be directly integrated into a security-aware design flow by ensuring that FSM optimization does not lead to exploitable don't care states. Hence, a security-unaware design flow requires the development and use of additional tools that compensate for the lack of security consideration. This is more complex and leads to less optimal solutions than integrating security into the design flow as a first-class design constraint.

Other methods could be significantly improved by an integrated design approach. For example, Eldib et al. assume equal latency for all gate types [39], a requirement that can be dropped when leveraging state-of-the-art timing analysis tools. A vast majority of other tools focus primarily on the identification of vulnerabilities. While a fundamental task, tool-assisted transformation into secure systems and automated repair of flawed designs is the desirable objective, but supreme discipline of security-aware EDA. In this sense, we would expect a boost in research of constructive security methods through integration, as they become easier to construct and evaluate.

7.3.1 Global Adversary Model. In an integrated, holistic, and security-aware EDA framework, security properties and goals should be derived from a globally defined *adversary model* to ensure a simple and usable design flow. Similar to hardware description, the user would describe the adversary at a high level of abstraction which then is compiled alongside the hardware design and adjusted to the different abstraction layers [61]. However, as this is an entirely new branch of research and clearly out of the scope for this paper, we leave the discussion of this complex field of research here.

7.3.2 Unified IR. An important aspect of integration is an agreed-upon semantic design structure. Modern software compiler frameworks, e.g., LLVM [91] or MLIR [92], are developed around a unified IR to provide a single representation for optimization and analysis spanning across all compilation stages. With FIRRTL [70] and LLHD [121], for the first time, this approach was transferred to the hardware context, enabling modernization and innovation in HDLs.

Similarly, an IR tailored to the context of security-aware EDA could lead to innovations in this field. The example of IFA shows the potential of such an approach. There, a specific type system was introduced by language-based methods, representing different security classes. Other security domains could ease analysis and transformation by introducing particular types and operations as well. For instance, `tightPROVE` [17] assumes securely implemented gates, which can be modeled by own operations making an automatic distinction between secure and insecure gates simple. To

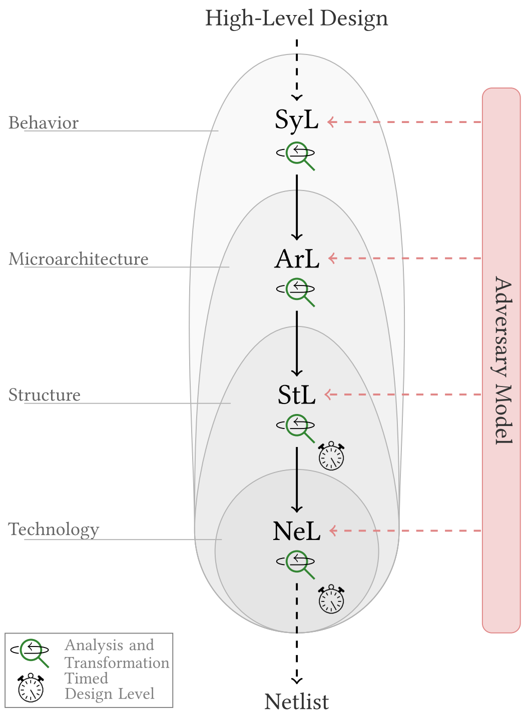

Fig. 1. Overview of the proposed design flow.

deal with the added complexity of such a security-aware IR, one could define an IR dialect [92] for each security domain. Then each EDA pass only needs to handle the dialects required for its task. We leave the specifics of such a security-aware IR for future work.

The design flow of hardware systems can be roughly divided into four layers of abstraction:

System Level (SyL) provides a behavioral or algorithmic description of the system, mostly without hardware-specific details.

Architectural Level (ArL) provides a coarse-grain structure, by defining components and their relation to each other, specifying the micro-architecture.

Structural Level (StL) defines a fine-grain structure of modules, their hierarchy, and I/O relations, based on a Register-Transfer description with explicit clock-based timing behavior. This is the abstraction of traditional HDLs and the reduction from SyL to StL is considered as HLS, with the ArL as a potential intermediate step proposed by Sharifian et al. [124].

Netlist Level (NeL) provides a technology-specific description that solely operates on signals and logic gates. The reduction from StL to NeL is considered as classical *logic synthesis* while *physical synthesis* denotes the process of *place & route* transformations at NeL.

Using the concept of multi-level IR, where every level is a subset of higher levels, we propose an EDA design flow consisting of those four abstraction layers and inspired by existing work [70, 121, 124]. Starting with a design written in a high-level language and ending with a technology-specific netlist, our design flow is depicted in Figure 1. Each intermediate level provides opportunities for automatic *analysis* and *transformation*, before the design is transferred to the next level through automatic *reduction*, i.e., removing and replacing no longer supported language constructs.

Trade-offs between automation and low-level control are possible through input languages targeting other abstraction levels. However, the concept of multi-level IR naturally allows the description in lower abstraction levels, as they are part of SyL anyway.

In the following, we discuss the exemplary design flow for a hardware accelerator. The input to the tool flow is a software-like, algorithmic description of the design at SyL that contains complex data structures (like vectors or data classes), operations on those data structures, and complex control structures (like loops or functions). However, since the lower IR levels are a subset of SyL, the designer can also define hardware-specific module borders or use registers and operations on bits to specify exactly how the final design should operate. This allows easy descriptions of the functionality and fine-grain control of the outcome where necessary. In addition, the designer provides some specifications of the adversary model that defines the security requirements. From this point onward, all transformations are automated but guided by the designer through parameter selection. First and still on SyL, an IFA is performed to ensure that the design has no trivial leakage of secrets and to propagate security classes and security requirements via type systems and annotations through the design. Hence, each operation is annotated afterwards with the protection it should provide. An information-flow secure final design is guaranteed by performing each downstream transformation such that no new information flow is created that violates the established security. When reducing to ArL, all complex control structures are removed and instead hardware modules are introduced that operate in parallel or consecutive and are controlled by a dedicated control unit. For that, each operation is assigned to some hardware module by deciding which operations can share some hardware resource (same security level with non-overlapping execution time such that no new violating information flow is created), which operations can run in parallel, and which require the completion of other modules. This coarse-grain *scheduling & binding* already considers which operations should run in constant time to prepare for efficient operation scheduling. In addition, the security requirements of the dedicated control unit are assessed and annotated accordingly. Reduction to StL includes the replacement of complex data structures to simpler structures, like vectors of numerical values of specified size. This allows the introduction of hardware registers and hence fine-grained scheduling and binding within modules. While doing this, constant-time execution is enforced wherever necessary. Of course, this can require cross-module analysis to ensure that the interaction of different modules is still constant-time. The last reduction is from StL to NeL and translates everything to operations taken from a specified technology library operating on single bits. During this process, masked gadgets are used to protect against PSCA, introducing specific gadget operations as an intermediary step. Also, FSA is considered during this transformation by ensuring equal path delay for sensitive data within each clock cycle. Lastly, *place & route* ensures that different shares of the same intermediate are processed and routed far enough apart to prevent leakage from cross-talk.

7.3.3 Incremental Design. Again, our analysis in this paper clearly shows that none of the presented tools are designed following an incremental paradigm, i.e., all tools operate on the entire design at once while small and local changes always require a reprocessing of the entire design. This leads to limitations with respect to the size that can be efficiently handled, and the number of executions that are practically feasible. Using incremental tools would allow analysis passes to be executed frequently, to assess the security impact of transformations and transformation passes to reverse insecure design changes and explore different alternatives. An incremental design approach is possible whenever compositional properties [27] hold, i.e., when the security analysis can be divided into the analysis of sub-components and the interaction of those sub-components with each other. Then, individual components can be changed without affecting the overall security as long as the compositional property is maintained. Finding compositional properties is easy for some security domains, e.g., IFA and TSCA, while it is more complex for others, e.g., PSCA and FIA. Specifically, for IFA, *non-interference* allows trivial composition, while for TSCA, changes are generally allowed as long as no new registers are introduces. In contrast, PSCA requires a thorough analysis of the possible leakage after composition, or composition needs to be considered during construction. However, this is extensively researched in the literature [12, 16, 29, 31]. Similarly, FIA usually requires some sort of *checkpoints* [1, 110, 123], where faults are detected and handled appropriately, or some isolation of instances [45] to achieve composition. As a result, more complex notions impose additional costs that ideally get removed later in the design flow by some optimization. The existing compositional notions for individual security domains are an important step. However, ultimately, we require rules for composition under multiple security domains at once [45, 113]. Such an incremental design flow is most efficient in combination with a *secure by construction* methodology, in which composable structures are used for construction and then marked. Otherwise, composable properties need to be detected from scratch, which is currently an open problem.

In the following, we list major advantages of the proposed framework compared to the state-of-the-art methodology.

- Reduced **infrastructure implementation and verification** effort due to reuse of analysis and transformation passes even across abstraction levels.
- Reduced **design verification** effort as a unified IR reduces the necessary translations to a minimum and provides a simple and unambiguous design representation.
- Enhanced **usability** as a modular design allows to enable required passes only and simplifies maintenance. In addition, automatic information gathering across abstraction levels reduces the required user expertise.
- Enhanced **efficiency** through information accumulation and sharing across abstraction levels and modules as well as an incremental design paradigm.
- Enhanced **security** through a holistic view that overcomes the isolation in security domains and abstraction levels, providing guarantees for the final design.
- Support of **innovation** through easy integration of new input languages, simple adaption to new security concepts and threat models, and reduced implementation effort. Experts can focus on their domain of expertise and provide tools for a broader community.

8 CONCLUSION

In this work, we provided an extensive overview of existing research for security-aware EDA, considering information flow, timing and power side channels, and fault injection threats. In essence, this overview clearly shows the potential of security-aware EDA in closing the gap between security and hardware design experts. Nevertheless, our systematization of existing knowledge shows that security-aware EDA currently is in the *Age of Implementation*, demonstrated by isolation with respect to security domains and abstraction levels. Hence, a huge effort is necessary to bring together different methods and approaches to take a holistic view of security.

For this, we propose a modern, security-aware design flow developed around a unified IR and tool integration to overcome those limitations and fuel innovation, both in hardware design and security. Our design framework provides a path towards a holistic view on hardware-based system security at design time by enabling interaction between security passes across design levels and security goals. To this, we see the framework of Universal Composable Security [27] in combination with a unified IR [91] as the fundamental and most promising building blocks to tackle future challenges of security-aware EDA, such as the integration of scalable and efficient constructive security passes, and the analysis of complex hardware architectures to identify sophisticated microarchitectural attacks.

REFERENCES

[1] A. Aghaie, A. Moradi, S. Rasoolzadeh, A. R. Shahmirzadi, F. Schellenberg, and T. Schneider. 2020. Impeccable circuits. *IEEE Trans. Computers* 69, 3 (2020), 361–376. https://doi.org/10.1109/TC.2019.2948617

[2] M. S. Alvim, K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith. 2020. *The Science of Quantitative Information Flow*. Springer. https://doi.org/10.1007/978-3-319-96131-6

[3] A. Ardeshiricham, W. Hu, and R. Kastner. 2017. Clepsydra: Modeling timing flows in hardware designs. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 147–154. https://doi.org/10.1109/ICCAD.2017.8203772

[4] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner. 2017. Register transfer level information flow tracking for provably secure hardware design. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 1691–1696.

[5] A. Ardeshiricham, Y. Takashima, S. Gao, and R. Kastner. 2019. VeriSketch: Synthesizing secure hardware designs with timing-sensitive information flow properties. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*. Association for Computing Machinery, New York, NY, USA, 1623–1638. https://doi.org/10.1145/3319535.3354246

[6] V. Arribas, S. Nikova, and V. Rijmen. 2018. VerMI: Verification tool for masked implementations. In *25th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2018, Bordeaux, France, December 9–12, 2018*. IEEE, 381–384. https://doi.org/10.1109/ICECS.2018.8617841

[7] V. Arribas, F. Wegener, A. Moradi, and S. Nikova. 2020. Cryptographic fault diagnosis using VerFI. *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7–11,* (2020), 229–240. https://doi.org/10.1109/HOST45689.2020.9300264

[8] A. Barenghi, L. Breveglieri, N. Izzo, and G. Pelosi. 2021. Exploring cortex-m microarchitectural side channel information leakage. *IEEE Access* 9 (2021), 156507–156527. https://doi.org/10.1109/ACCESS.2021.3124761

[9] G. Barthe, A. Basu, and T. Rezk. 2004. Security types preserving compilation. In *Verification, Model Checking, and Abstract Interpretation*. Springer Berlin, 2–15.

[10] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, and F.-X. Standaert. 2019. maskVerif: Automated verification of higher-order masking in presence of physical defaults. In *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11735)*. Springer, 300–318. https://doi.org/10.1007/978-3-030-29959-0_15

[11] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub. 2015. Verified proofs of higher-order masking. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26–30, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9056)*. Springer, 457–485. https://doi.org/10.1007/978-3-662-46800-5_18

[12] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini. 2016. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24–28, 2016*. ACM, 116–129. https://doi.org/10.1145/2976749.2978427

[13] G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F.-X. Standaert, and P.-Y. Strub. 2017. Parallel implementations of masking schemes and the bounded moment leakage model. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10210)*. 535–566. https://doi.org/10.1007/978-3-319-56620-7_19

[14] A. Becker, W. Hu, Y. Tai, P. Brisk, R. Kastner, and P. Ienne. 2017. Arbitrary precision and complexity tradeoffs for gate-level information flow tracking. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[15] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud. 2016. Randomness complexity of private circuits for multiplication. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 616–648. https://doi.org/10.1007/978-3-662-49896-5_22

[16] S. Belaïd, J.-S. Coron, E. Prouff, M. Rivain, and A. R. Taleb. 2020. Random probing security: Verification, composition, expansion and new constructions. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12170)*. Springer, 339–368. https://doi.org/10.1007/978-3-030-56784-2_12

[17] S. Belaïd, D. Goudarzi, and M. Rivain. 2018. Tight private circuits: Achieving probing security with the least refreshing. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11273)*. Springer, 343–372. https://doi.org/10.1007/978-3-030-03329-3_12

[18] S. Belaïd, P.-É.Dagand, D. Mercadier, M. Rivain, and R. Wintersdorff. 2020. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12107).* Springer, 311–341. https://doi.org/10.1007/978-3-030-45727-3_11

[19] S. Belaïd, D. Mercadier, M. Rivain, and A. R. Taleb. 2021. IronMask: Versatile verification of masking security. *IACR Cryptol. ePrint Arch.* 2021 (2021). http://eprint.iacr.org/2021/1671.

[20] D. J. Bernstein. 2005. Cache-timing attacks on AES. (2005).

[21] F. Besson, A. Dang, and T. Jensen. 2019. Information-flow preservation in compiler optimisations. In *CSF 2019 - 32nd IEEE Computer Security Foundations Symposium.* IEEE, Hoboken, NJ, United States, 1–13. https://hal.inria.fr/hal-02180303.

[22] E. Biham and A. Shamir. 1997. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology — CRYPTO'97.* Springer Berlin, Berlin, 513–525.

[23] R. Bloem, H. Groß, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter. 2018. Formal verification of masked hardware implementations in the presence of glitches. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10821).* Springer, 321–353. https://doi.org/10.1007/978-3-319-78375-8_11

[24] D. Boneh, R. A. DeMillo, and R. J. Lipton. 1997. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology — EUROCRYPT'97.* Springer Berlin, Berlin, 37–51.

[25] I. Buhan, L. Batina, Y. Yarom, and P. Schaumont. 2022. SoK: Design tools for side-channel-aware implementations. In *ASIA CCS'22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022.* 756–770. https://doi.org/10.1145/3488932.3517415. arXiv:https://eprint.iacr.org/2021/497.

[26] J. Burchard, M. Gay, A. M. Ekossono, J. Horáček, B. Becker, T. Schubert, M. Kreuzer, and I. Polian. 2017. AutoFault: Towards automatic construction of algebraic fault attacks. In *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC).* 65–72.

[27] R. Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS'01).* IEEE Computer Society, USA, 136.

[28] R. Canetti, M. van Dijk, H. Maleki, U. Rührmair, and P. Schaumont. 2020. Using universal composition to design and analyze secure complex hardware systems. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE).* 520–525.

[29] G. Cassiers, B. Grégoire, I. Levi, and F.-X. Standaert. 2021. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers* 70, 10 (2021), 1677–1690. https://doi.org/10.1109/TC.2020.3022979

[30] G. Cassiers and F.-X. Standaert. 2019. Towards globally optimized masking: From low randomness to low noise rate: Or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 2 (Feb. 2019), 162–198. https://doi.org/10.13154/tches.v2019.i2.162-198

[31] G. Cassiers and F.-X. Standaert. 2020. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 2542–2555. https://doi.org/10.1109/TIFS.2020.2971153

[32] G. Cassiers and F.-X. Standaert. 2021. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 2 (2021), 136–158. https://doi.org/10.46586/tches.v2021.i2.136-158

[33] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. 1999. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO'99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1666).* Springer, 398–412. https://doi.org/10.1007/3-540-48405-1_26

[34] K. Chatzikokolakis, C. Palamidessi, and P. Panangaden. 2008. On the Bayes risk in information-hiding protocols. *J. Comput. Secur.* 16, 5 (2008), 531–571. https://doi.org/10.3233/JCS-2008-0333

[35] D. Chinnery, L. Stok, D. Hathaway, and K. Keutzer. [n. d.]. Design FLow. In *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology*, Luciano Lavagno, Igor L. Markov, Grant Martin, and Louis K. Scheffer (Eds.). CRC Press.

[36] J.-S. Coron, A. Greuet, and R. Zeitoun. 2020. Side-channel masking with pseudo-random generator. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III.* 342–375. https://doi.org/10.1007/978-3-030-45727-3_12. arXiv:https://eprint.iacr.org/2019/1106.pdf.

[37] S. Deng, D. Gümüundefinedoundefinedlu, W. Xiong, S. Sari, Y. S. Gener, C. Lu, O. Demir, and J. Szefer. 2019. SecChisel framework for security verification of secure processor architectures. In *Proceedings of the 8th International Workshop*

on Hardware and Architectural Support for Security and Privacy (HASP'19). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3337167.3337174

[38] A. Duc, S. Dziembowski, and S. Faust. 2019. Unifying leakage models: From probing attacks to noisy leakage. *J. Cryptology* 32, 1 (2019), 151–177. https://doi.org/10.1007/s00145-018-9284-1

[39] H. Eldib, M. Wu, and C. Wang. 2016. Synthesis of fault-attack countermeasures for cryptographic circuits. In *Computer Aided Verification*. Springer International Publishing, Cham, 343–363.

[40] M. R. Fadiheh, J. Müller, R. Brinkmann, S. Mitra, D. Stoffel, and W. Kunz. 2020. A formal approach for detecting vulnerabilities to transient execution attacks in out-of-order processors. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218572

[41] M. Rahmani Fadiheh, D. Stoffel, C. W. Barrett, S. Mitra, and W. Kunz. 2019. Processor hardware security vulnerabilities and their detection by unique program execution checking. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25–29, 2019*. 994–999. https://doi.org/10.23919/DATE.2019.8715004

[42] S. Faust, V. Grosso, S. M. Del Pozo, C. Paglialonga, and F.-X. Standaert. 2018. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120. https://doi.org/10.13154/tches.v2018.i3.89-120

[43] S. Faust, C. Paglialonga, and T. Schneider. 2017. Amortizing randomness complexity in private circuits. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3–7, 2017, Proceedings, Part I*. 781–810. https://doi.org/10.1007/978-3-319-70694-8_27. arXiv:https://eprint.iacr.org/2017/869.pdf.

[44] J. Feldtkeller, D. Knichel, P. Sasdrich, A. Moradi, and T. Güneysu. 2022. Randomness optimization for gadget compositions in higher-order masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022), 188–227. https://doi.org/10.46586/tches.v2022.i4.188-227

[45] J. Feldtkeller, J. Richter-Brockmann, P. Sasdrich, and T. Güneysu. 2022. CINI MINIS: Domain isolation for fault and combined security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7–11, 2022*. 1023–1036. https://doi.org/10.1145/3548606.3560614

[46] A. Ferraiuolo. 2017. Security results for SIRRTL, a hardware description language for information flow security. (2017). https://ecommons.cornell.edu/handle/1813/57673.

[47] A. Ferraiuolo, M. Zhao, A. C. Myers, and G. E. Suh. 2018. HyperFlow: A processor architecture for nonmalleable, timing-safe information flow security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. Association for Computing Machinery, New York, NY, USA, 1583–1600. https://doi.org/10.1145/3243734.3243743

[48] A. Fogh. 2016. Covert Shotgun: Automatically Finding SMT Covert Channels. https://cyber.wtf/2016/09/27/covert-shotgun/.

[49] K. Gandolfi, C. Mourtel, and F. Olivier. 2001. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2162)*. Springer, 251–261. https://doi.org/10.1007/3-540-44709-1_21

[50] S. Gao, E. Oswald, and D. Page. 2021. Reverse engineering the micro-architectural leakage features of a commercial processor. *IACR Cryptol. ePrint Arch.* (2021), 794. https://eprint.iacr.org/2021/794.

[51] M. Gay, T. Paxian, D. Upadhyaya, B. Becker, and I. Polian. 2019. Hardware-oriented algebraic fault attack framework with multiple fault injection support. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 25–32.

[52] Q. Ge, Y. Yarom, D. Cock, and G. Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptogr. Eng.* 8, 1 (2018), 1–27. https://doi.org/10.1007/s13389-016-0141-6

[53] K. v. Gleissenthall, R. G. Kıcı, D. Stefan, and R. Jhala. 2019. IODINE: Verifying constant-time execution of hardware. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1411–1428. https://www.usenix.org/conference/usenixsecurity19/presentation/von-gleissenthall.

[54] J. A. Goguen and J. Meseguer. 1982. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy*. 11.

[55] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. 2011. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop*, Vol. 7. 115–136.

[56] V. Goyal, Y. Ishai, and Y. Song. 2022. Private circuits with quasilinear randomness. *IACR Cryptol. ePrint Arch.* 2022 (2022). http://eprint.iacr.org/2022/250.

[57] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi. 2020. ABSynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23–26, 2020*. https://www.ndss-symposium.org/ndss-paper/absynthe-automatic-blackbox-side-channel-synthesis-on-commodity-microarchitectures/.

[58] D. Gruss. 2017. *Software-based Microarchitectural Attacks*. Ph. D. Dissertation. Graz University of Technology. arXiv:1706.05973 http://arxiv.org/abs/1706.05973.

[59] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin. 2019. QIF-Verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 91–100. https://doi.org/10.1109/HST.2019.8740840

[60] M. T. He, J. Park, A. Nahiyan, A. Vassilev, Y. Jin, and M. M. Tehranipoor. 2019. RTL-PSC: Automated power side-channel leakage assessment at register-transfer level. In *37th IEEE VLSI Test Symposium, VTS 2019, Monterey, CA, USA, April 23-25, 2019*. IEEE, 1–6. https://doi.org/10.1109/VTS.2019.8758600

[61] W. Hu, A. Althoff, A. Ardeshiricham, and R. Kastner. 2016. Towards property driven hardware security. In *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*. 51–56. https://doi.org/10.1109/MTV.2016.12

[62] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li. 2021. An overview of hardware security and trust: Threats, countermeasures, and design tools. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 40, 6 (2021), 1010–1038. https://doi.org/10.1109/TCAD.2020.3047976

[63] W. Hu, D. Mu, J. Oberg, B. Mao, M. Tiwari, T. Sherwood, and R. Kastner. 2014. Gate-level information flow tracking for security lattices. *ACM Trans. Des. Autom. Electron. Syst.* 20, 1 (2014). https://doi.org/10.1145/2676548

[64] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. 2011. Theoretical fundamentals of gate level information flow tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 8 (2011), 1128–1140.

[65] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. 2012. On the complexity of generating gate level information flow tracking logic. *IEEE Trans. Information Forensics and Security* 7, 3 (2012), 1067–1080. https://doi.org/10.1109/TIFS.2012.2189105

[66] S. A. Huss and O. Stein. 2017. A novel design flow for a security-driven synthesis of side-channel hardened cryptographic modules. *Journal of Low Power Electronics and Applications* 7, 1 (2017), 4.

[67] M. Hutter and J.-M. Schmidt. 2013. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8419)*. Springer, 219–235. https://doi.org/10.1007/978-3-319-08302-5_15

[68] Y. Ishai, E. Kushilevitz, X. Li, R. Ostrovsky, M. Prabhakaran, A. Sahai, and D. Zuckerman. 2013. Robust pseudorandom generators. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8–12, 2013, Proceedings, Part I*. 576–588. https://doi.org/10.1007/978-3-642-39206-1_49

[69] Y. Ishai, A. Sahai, and D. A. Wagner. 2003. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2729)*. Springer, 463–481. https://doi.org/10.1007/978-3-540-45146-4_27

[70] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 209–216. https://doi.org/10.1109/ICCAD.2017.8203780

[71] Z. Jiang, S. Dai, G. E. Suh, and Z. Zhang. 2018. High-level synthesis with timing-sensitive information flow enforcement. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'18)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3240765.3243415

[72] A. Kerckhoffs. 1883. La cryptographie militaire. In *Journal Des Sciences Militaires*, Vol. 9.

[73] M. Khairallah, R. Sadhukhan, R. Samanta, J. Breier, S. Bhasin, R. S. Chakraborty, A. Chattopadhyay, and D. Mukhopadhyay. 2018. DFARPA: Differential fault attack resistant physical design automation. In *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19–23, 2018*. 1171–1174. https://doi.org/10.23919/DATE.2018.8342190

[74] P. Khanna, C. Rebeiro, and A. Hazra. 2017. XFC: A framework for exploitable fault characterization in block ciphers. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[75] P. Kiaei, Z. Liu, R. K. Eren, Y. Yao, and P. Schaumont. 2021. Saidoyoki: Evaluating side-channel leakage in pre- and post-silicon setting. *IACR Cryptol. ePrint Arch.* (2021), 1235. https://eprint.iacr.org/2021/1235.

[76] P. Kiaei, Z. Liu, and P. Schaumont. 2022. Leverage the average: Averaged sampling in pre-silicon side-channel leakage assessment. In *GLSVLSI'22: Great Lakes Symposium on VLSI 2022, Irvine CA USA, June 6–8, 2022*. 3–8. https://doi.org/10.1145/3526241.3530337

[77] P. Kiaei, Y. Yao, Z. Liu, N. Fern, C.-B. Breunesse, J. Van Woudenberg, K. Gillis, A. Dich, P. Grossmann, and P. Schaumont. 2022. Gate-level side-channel leakage assessment with architecture correlation analysis. *CoRR* abs/2204.11972 (2022). arXiv:2204.11972. http://arxiv.org/abs/2204.11972.

[78] R. Kici. 2020. *Verifying Constant-Time Execution of Hardware*. Ph. D. Dissertation. UC San Diego. https://escholarship.org/uc/item/6mx0g513.

[79] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[80] J. Knechtel, E. B. Kavun, F. Regazzoni, A. Heuser, A. Chattopadhyay, D. Mukhopadhyay, S. Dey, Y. Fei, Y. Belenky, I. Levi, T. Güneysu, P. Schaumont, and I. Polian. 2020. Towards secure composition of integrated circuits and electronic systems: On the role of EDA. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*. 508–513. https://doi.org/10.23919/DATE48585.2020.9116483

[81] D. Knichel and A. Moradi. 2022. Low-latency hardware private circuits. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7–11, 2022*. 1799–1812. https://doi.org/10.1145/3548606.3559362

[82] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich. 2022. Automated generation of masked hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 589–629. https://doi.org/10.46586/tches.v2022.i1.589-629

[83] D. Knichel, P. Sasdrich, and A. Moradi. 2020. SILVER - statistical independence and leakage verification. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Virtual, December 7–11, Proceedings*.

[84] D. Knichel, P. Sasdrich, and A. Moradi. 2022. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 323–344. https://doi.org/10.46586/tches.v2022.i1.323-344

[85] D. S. Koblah, R. Y. Acharya, D. Capecci, O. P. Dizon-Paradis, S. Tajik, F. Ganji, D. L. Woodard, and D. Forte. 2022. A survey and perspective on artificial intelligence for security-aware electronic design automation. *CoRR* abs/2204.09579 (2022). https://doi.org/10.48550/arXiv.2204.09579. arXiv:2204.09579.

[86] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19–23, 2019*. 1–19.

[87] P. C. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology — CRYPTO'96*. Springer Berlin, 104–113.

[88] P. C. Kocher, J. Jaffe, and B. Jun. 1999. Differential power analysis. In *Advances in Cryptology - CRYPTO'99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15–19, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1666)*. Springer, 388–397. https://doi.org/10.1007/3-540-48405-1_25

[89] X. Lai, M. Jenihhin, J. Raik, and K. Paul. 2019. PASCAL: Timing SCA resistant design and verification flow. In *25th IEEE International Symposium on On-Line Testing and Robust System Design, IOLTS 2019, Rhodes, Greece, July 1–3, 2019*. 239–242. https://doi.org/10.1109/IOLTS.2019.8854458

[90] A. V. Lakshmy, C. Rebeiro, and S. Bhunia. 2022. FORTIFY: Analytical pre-silicon side-channel characterization of digital designs. In *27th Asia and South Pacific Design Automation Conference, ASP-DAC 2022, Taipei, Taiwan, January 17–20, 2022*. 660–665. https://doi.org/10.1109/ASP-DAC52403.2022.9712551

[91] C. Lattner and V. Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*. Palo Alto, California.

[92] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. A. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. 2021. MLIR: Scaling compiler infrastructure for domain specific computation. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2021, Seoul, South Korea, February 27–March 3, 2021*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[93] X. Li, V. Kashyap, J. K. Oberg, M. Tiwari, V. R. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong. 2014. Sapper: A language for hardware-level security policy enforcement. In *Architectural Support for Programming Languages and Operating Systems, (ASPLOS'14), Salt Lake City, UT, USA, March 1–5, 2014*. ACM, 97–112. https://doi.org/10.1145/2541940.2541947

[94] X. Li, M. Tiwari, J. K. Oberg, V. Kashyap, F. T. Chong, T. Sherwood, and B. Hardekopf. 2011. Caisson: A hardware description language for secure information flow. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'11)*. ACM, New York, NY, USA, 109–120. https://doi.org/10.1145/1993498.1993512

[95] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18)*. USENIX Association, Berkeley, CA, USA, 973–990. http://dl.acm.org/citation.cfm?id=3277203.3277276.

[96] H. Ma, Q. Zhang, Y. Gao, J. He, Y. Zhao, and Y. Jin. 2022. PathFinder: Side channel protection through automatic leaky paths identification and obfuscation. In *DAC'22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10–14, 2022*. 79–84. https://doi.org/10.1145/3489517.3530413

[97] B. Marshall, D. Page, and J. Webb. 2022. MIRACLE: MIcRo-architectural leakage evaluation - a study of micro-architectural power leakage across many devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 1 (2022), 175–220. https://doi.org/10.46586/tches.v2022.i1.175-220

[98] J. H. McDonald. 2014. *Handbook of Biological Statistics*. Vol. 2. Sparky House Publishing, Baltimore.

[99] G. H. Mealy. 1955. A method for synthesizing sequential circuits. *The Bell System Technical Journal* 34, 5 (1955), 1045–1079.

[100] D. Moghimi, M. Lipp, B. Sunar, and M. Schwarz. 2020. Medusa: Microarchitectural data leakage via automated attack synthesis. In *29th USENIX Security Symposium, USENIX Security 2020, August 12–14, 2020*. 1427–1444. https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-medusa.

[101] N. Müller and A. Moradi. 2022. PROLEAD: A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022), 311–348. https://doi.org/10.46586/tches.v2022.i4.311-348

[102] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor. 2016. AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[103] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner. 2013. A practical testing framework for isolating hardware timing channels. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1281–1284.

[104] J. Oberg, S. Meiklejohn, T. Sherwood, and R. Kastner. 2014. Leveraging gate-level properties to identify hardware timing channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 9 (2014), 1288–1301.

[105] S. Peter and T. Givargis. 2016. Towards a timing attack aware high-level synthesis of integrated circuits. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. 452–455. https://doi.org/10.1109/ICCD.2016.7753326

[106] C. Pilato, K. Wu, S. Garg, R. Karri, and F. Regazzoni. 2019. TaintHLS: High-level synthesis for dynamic information flow tracking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 5 (2019), 798–808. https://doi.org/10.1109/TCAD.2018.2834421

[107] N. Pundir, J. Park, F. Farahmandi, and M. M. Tehranipoor. 2022. Power side-channel leakage assessment framework at register-transfer level. *IEEE Trans. Very Large Scale Integr. Syst.* 30, 9 (2022), 1207–1218. https://doi.org/10.1109/TVLSI.2022.3175067

[108] M. Qin, W. Hu, X. Wang, D. Mu, and B. Mao. 2019. Theorem proof based gate level information flow tracking for hardware security verification. *Computers & Security* 85 (2019), 225–239. https://doi.org/10.1016/j.cose.2019.05.005

[109] M. Qin, X. Wang, B. Mao, D. Mu, and W. Hu. 2020. A formal model for proving hardware timing properties and identifying timing channels. *Integration* 72 (2020), 123–133. https://doi.org/10.1016/j.vlsi.2020.02.001

[110] S. Rasoolzadeh, A. R. Shahmirzadi, and A. Moradi. 2021. Impeccable circuits III. In *IEEE International Test Conference, ITC 2021, Anaheim, CA, USA, October 10–15, 2021*. 163–169. https://doi.org/10.1109/ITC50571.2021.00024

[111] P. Ravi, Z. Najm, S. Bhasin, M. Khairallah, S. S. Gupta, and A. Chattopadhyay. 2019. Security is an architectural design constraint. *Microprocess. Microsystems* 68 (2019), 17–27.

[112] L. M. Reimann, L. Hanel, D. Sisejkovic, F. Merchant, and R. Leupers. 2021. QFlow: Quantitative information flow for security-aware hardware design in verilog. In *39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24–27, 2021*. 603–607. https://doi.org/10.1109/ICCD53106.2021.00097. arXiv:2109.02379.

[113] J. Richter-Brockmann, J. Feldtkeller, P. Sasdrich, and T. Güneysu. 2022. VERICA - verification of combined attacks automated formal verification of security against simultaneous information leakage and tampering. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022, 4 (2022), 255–284. https://doi.org/10.46586/tches.v2022.i4.255-284. arXiv:http://eprint.iacr.org/2022/484.

[114] J. Richter-Brockmann and T. Güneysu. 2020. Improved side-channel resistance by dynamic fault-injection countermeasures. In *31st IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2020, Manchester, United Kingdom, July 6–8, 2020*. 117–124. https://doi.org/10.1109/ASAP49362.2020.00029

[115] J. Richter-Brockmann, P. Sasdrich, and T. Güneysu. 2021. Revisiting fault adversary models - hardware faults in theory and practice. *IACR Cryptol. ePrint Arch.* 2021 (2021). https://eprint.iacr.org/2021/296.

[116] J. Richter-Brockmann, A. R. Shahmirzadi, P. Sasdrich, A. Moradi, and T. Güneysu. 2021. FIVER – robust verification of countermeasures against fault injections. *IACR Cryptol. ePrint Arch.* 2021 (2021). http://eprint.iacr.org/2021/936.

[117] I. Roy, C. Rebeiro, A. Hazra, and S. Bhunia. 2019. SAFARI: Automatic synthesis of fault-attack resistant block cipher implementations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019), 1. https://doi.org/10.1109/TCAD.2019.2897629

[118] P. B. Roy, P. SLPSK, and C. Rebeiro. 2022. Avatar: Reinforcing fault attack countermeasures in EDA with fault transformations. In *27th Asia and South Pacific Design Automation Conference, ASP-DAC 2022, Taipei, Taiwan, January 17–20, 2022*. 417–422. https://doi.org/10.1109/ASP-DAC52403.2022.9712539

[119] S. Saha, D. Mukhopadhyay, and P. Dasgupta. 2018. ExpFault: An automated framework for exploitable fault characterization in block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 2 (2018), 242–276. https://doi.org/10.13154/tches.v2018.i2.242-276

[120] T. Schneider, A. Moradi, and T. Güneysu. 2016. ParTI - Towards combined hardware countermeasures against side-channel and fault-injection attacks. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14–18, 2016, Proceedings, Part II.* 302–332. https://doi.org/10.1007/978-3-662-53008-5_11

[121] F. Schuiki, A. Kurth, T. Grosser, and L. Benini. 2020. LLHD: A multi-level intermediate representation for hardware description languages. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15–20, 2020.* 258–271.

[122] M. Schwarz and D. Gruss. 2020. How trusted execution environments fuel research on microarchitectural attacks. *IEEE Secur. Priv.* 18, 5 (2020), 18–27. https://doi.org/10.1109/MSEC.2020.2993896

[123] A. R. Shahmirzadi, S. Rasoolzadeh, and A. Moradi. 2020. Impeccable circuits II. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20–24, 2020.* 1–6. https://doi.org/10.1109/DAC18072.2020.9218615

[124] A. Sharifian, R. Hojabr, N. Rahimi, S. Liu, A. Guha, T. Nowatzki, and A. Shriraman. 2019. μIR - an intermediate representation for transforming and optimizing the microarchitecture of application accelerators. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'52).* Association for Computing Machinery, New York, NY, USA, 940–953. https://doi.org/10.1145/3352460.3358292

[125] N. Shenoy. 1997. Retiming: Theory and practice. *Integration* 22, 1 (1997), 1–21. https://doi.org/10.1016/S0167-9260(97)00002-3

[126] R. Sison and T. Murray. 2019. Verifying that a compiler preserves concurrent value-dependent information-flow security. In *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9–12, 2019, Portland, OR, USA.* 27:1–27:19. https://doi.org/10.4230/LIPIcs.ITP.2019.27

[127] P. SLPSK, P. K. Vairam, C. Rebeiro, and V. Kamakoti. 2019. Karna: A gate-sizing based security aware EDA flow for improved power side-channel attack protection. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4–7, 2019.* 1–8. https://doi.org/10.1109/ICCAD45719.2019.8942173

[128] G. Smith. 2009. On the foundations of quantitative information flow. In *Foundations of Software Science and Computational Structures.* Springer Berlin , Berlin, 288–302.

[129] A. Solar-Lezama. 2013. Program sketching. *Int. J. Softw. Tools Technol. Transf.* 15, 5-6 (2013), 475–495. https://doi.org/10.1007/s10009-012-0249-7

[130] M. Srivastava, P. SLPSK, I. Roy, C. Rebeiro, A. Hazra, and S. Bhunia. 2020. SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE).* 310–313.

[131] J. Szefer. 2019. Survey of microarchitectural side and covert channels, attacks, and defenses. *J. Hardw. Syst. Secur.* 3, 3 (2019), 219–234. https://doi.org/10.1007/s41635-018-0046-1

[132] A. Tang, S. Sethumadhavan, and S. J. Stolfo. 2017. CLKSCREW: Exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16–18, 2017.* 1057–1074. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.

[133] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood. 2011. Crafting a usable microkernel, processor, and I/O system with strict and provable information flow security. In *2011 38th Annual International Symposium on Computer Architecture (ISCA).* 189–199.

[134] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood. 2009. Complete information flow tracking from the gates up. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV).* Association for Computing Machinery, New York, NY, USA, 109–120. https://doi.org/10.1145/1508244.1508258

[135] G. S. Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970.* Springer Berlin, 466–483. https://doi.org/10.1007/978-3-642-81955-1_28

[136] S. T. Vu, K. Heydemann, A. de Grandmaison, and A. Cohen. 2020. Secure delivery of program properties through optimizing compilation. In *Proceedings of the 29th International Conference on Compiler Construction (CC 2020).* Association for Computing Machinery, New York, NY, USA, 14–26. https://doi.org/10.1145/3377555.3377897

[137] H. Wang, H. Li, F. Rahman, M. M. Tehranipoor, and F. Farahmandi. 2021. SoFI: Security property-driven vulnerability assessments of ICs against fault-injection attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/TCAD.2021.3063998

[138] W. Wang, C. Guo, F.-X. Standaert, Y. Yu, and G. Cassiers. 2020. Packed multiplication: How to amortize the cost of side-channel masking?. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I.* 851–880. https://doi.org/10.1007/978-3-030-64837-4_28. arXiv:https://eprint.iacr.org/2020/1103.pdf.

[139] D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, and C. Rossow. 2021. Osiris: Automated discovery of microarchitectural side channels. *CoRR* abs/2106.03470 (2021). arXiv:2106.03470. http://arxiv.org/abs/2106.03470.

[140] K. Xiao, A. Nahiyan, and M. Tehranipoor. 2016. Security rule checking in IC design. *Computer* 49, 8 (2016), 54–61.

[141] Y. Yao, T. Tufan, T. Kathuria, B. Ege, U. Guler, and P. Schaumont. 2021. Pre-silicon architecture correlation analysis (PACA): Identifying and mitigating the source of side-channel leakage at gate-level. *IACR Cryptol. ePrint Arch.* 2021 (2021). https://eprint.iacr.org/2021/530. https://eprint.iacr.org/2021/530.

[142] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers. 2015. A hardware design language for timing-sensitive information-flow security. *ACM SIGPLAN Notices* 50, 4 (2015), 503–516. https://doi.org/10.1145/2775054.2694372

[143] M. Zohner, M. Stöttinger, S. A. Huss, and O. Stein. 2012. An adaptable, modular, and autonomous side-channel vulnerability evaluator. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*. IEEE Computer Society, 43–48. https://doi.org/10.1109/HST.2012.6224317

Received 27 June 2022; revised 18 November 2022; accepted 30 November 2022