Secure Your SoC: Building System-on-Chip Designs for Security

Shivam Bhasin[1] Trevor E. Carlson[2] Anupam Chattopadhyay[1] Vinay B.Y. Kumar[1]
Avi Mendelson[1,3] Romain Poussier[1] Yaswanth Tavva[2]
[1]Nanyang Technological University, Singapore
{sbhasin,anupam,vinay.kumar,rpoussier}@ntu.edu.sg
[2]National University of Singapore, Singapore
tcarlson@comp.nus.edu.sg, yaswanth@u.nus.edu
[3]Technion – Israel Institute of Technology, Haifa, Israel
avi.mendelson@technion.ac.il

***Abstract*—Modern System-on-Chip designs (SoCs) are becoming increasingly complex and powerful, catering to a wide range of application domains. Their use in security-critical tasks calls for a holistic approach to SoC design, including security as a first-class architecture constraint, rather than adding security only as an afterthought. The problem is compounded by the inclusion of multiple, potentially untrusted, third party components in the SoC design. To address this challenge systematically, this paper explores four distinct and important aspects of designing secure SoCs. First, starting at the component level, an evaluation framework for assessing component security against physical attacks is proposed. Second, a scalable simulation framework is developed to integrate these secure components which offers flexibility for early- and late-stage SoC development. Third, dynamic and static techniques are proposed to determine when the system is under attack, with a key focus on Hardware Trojans as threat. Finally, a design strategy for integrating untrusted components into a SoC through hardware Root-of-Trust is outlined. For each of these aspects we present early-stage evaluations, and show how these complement each other towards the design of a secure SoC.**

I. Introduction

Modern System on Chip systems (SoCs) are powerful, effective and complex. The integration of a wide variety of components in an SoC enables them to both accomplish the task at hand, as well as perform that task efficiently. However, the co-existence of heterogeneous components within an SoC raises multiple security concerns. Only by handling each of these concerns in a systematic way can one ensure the design of a trusted and reliable system.

This paper presents recent developments toward designing a secure SoC under the SOCure framework [1]. The main objective of the SOCure project is to assure hardware security-by-design in the presence of untrusted on-chip Intellectual Properties (IPs), as well as modern, and invasive, physical attacks. To this end, this paper describes four distinct aspects of designing secure SoCs.

The first part looks at physical attacks at the component level. Components in a SoC like cryptographic accelerators (e.g. AES), form the backbone of the security subsystem and must be protected against a range of attacks. We propose a generic evaluation framework to assess the security of various components at different granularity. At the cost of precision in evaluation results, the designer can evaluate security of design at different design stages from right from algorithm specification to precise evaluation considering implementation details and device properties.

Next, we propose a scalable simulation framework, Hydra. As the complexity of an SoC increases, hardware design, from early debugging to feature development becomes time consuming. The need for flexible simulation infrastructures that provide efficient, targeted simulation solutions for the current phase of development, becomes necessary for fast and efficient evaluation. To address this issue, we present a unified management and simulation environment to manage resources across a variety of back-end simulation platforms. With this system, we aim to demonstrate that automation improves productivity, and allows for ease of migration to the appropriate simulator for the work in each stage of development.

Furthermore, SoC designers deal with a variety of components (or IPs) from multiple untrusted parties allowing the possibility of hardware Trojan in those IPs. To protect such systems against security attacks is a challenging task that requires out-of-the-box thinking. As no single tool can be used to protect the entire system, we propose to take a hierarchical approach that allows hosting a set of tools simultaneously, to use a Security Management Unit (SMU) to control the complexity of such an attack. We also propose to use a new design methodology, termed Design for Security (DFS), in order to identify when the system is under attack.

Finally, we demonstrate an approach to integrate untrusted IPs into a system based on hardware root of trust. Recently, these problems are being addressed systematically through the design of secure SoCs, where a Physically Unclonable Function (PUF) along with True Random Number Generator (TRNG) are used for key management. This enables an opportunity to address the vulnerabilities of existing protocols through the RoT, albeit at the cost of increased resource usage and performance overhead. We report our experimental studies using a RISC-V based secure SoC and Xilinx KC705 FPGA board and selective protocols, such as integration of untrusted SoC components.

The rest of the paper is organised as follows. Section 2 and 3 present the evaluation and simulation framework for a complex SoC respectively. Section 4 describes the hierarchical protection of SoC system against Hardware Trojan. Section 5 presents an approach to integrate of untrusted IPs in a system based on hardware root of trust. Final conclusions are drawn in Section 6.

II. Security evaluations against physical attacks: A time versus accuracy tradeoff

The security evaluation of cryptographic implementations with respect to physical attacks (faults, side channels, etc) is a complex and yet mandatory step. When it comes to analysing the security of the final implementation, a complete and thorough evaluation is preferred in order to show the precise security level. However, such an analysis is very time-consuming and will only be performed during the final design/implementation steps. If a security flaw is exhibited, a new design must be proposed and a new evaluation must be performed, resulting in a long time to market and eventual monetary losses. This loss can be prevented if the security flaw is found earlier in the design steps. For that purpose, this section puts together a security evaluation framework that allows a trade-off between the accuracy of the results at different design stages.

A. Framework

The overall framework is depicted in Figure 1. It takes as an input a given implementation of primitives and underlying set of countermeasures. Then, for a given threat considered, it will output up to three evaluation methodologies, ranging from fast and loose to complex and accurate.

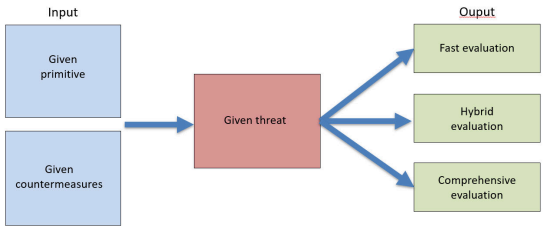

Fig. 1. Overview of the evaluation framework.

Considering several evaluation levels allow having dedicated methodologies for the different steps occurring during the design of a product. More practically, we aim at easing the development by providing tools and methods to detect, analyse and compare the security of different implementations at earlier stages and different levels of granularity. That is, we divide the evaluations into three different ranges that we now exhibit in the context of side-channel attacks:

- **Fast evaluation** will contain measurement-free tools to estimate the security level of a design choice. This will be done through the use of information theory metrics and shortcut formulas [2]. thus producing fast tools with a coarse security estimate. At pre-silicon level, tools as proposed in [3] can be helpful.
- **Hybrid evaluation** will compute a security estimate through basic measurements or simulations. Leakage detection/mapping [4], [5] and shortcut formulas augmented with real traces are different tool examples that will be used for this type of evaluation. The estimation can also be extented to standards like FIPS140-2 or common criterea by exisiting techniques such as [6].
- **Comprehensive evaluation** will contain methods to compute a precise estimate of the actual security of a given implementation. This will involve performing advanced attacks and pre-processing and will output meaningful security graphics [5]

While previous examples, focused on side-channel attacks for simplicity, the same approach can be extended to other physical attacks like fault attacks based on evaluation approaches as proposed in [7].

B. Assessing The Countermeasure Jungle

A designer has a range of choices to adopt a countermeasure. The proposed evaluation framework is capable of evaluating these countermeasures at different granularity levels and thus representing adversaries of different capabilities. While fast evaluation only gives a basic overview of a complex countermeasure like high-order masking, it can easily assess the security of simple countermeasures such as jitter or noise addition. Hybrid evaluation can carefully evaluate more complex countermeasures like hiding [8] and to an extent masking, based on a more accurate leakage model and noise setting. Finally, a comprehensive evaluation is required for complex but provable countermeasures like higher-order masking considering advanced device-level effects like coupling leading to combination of shares [9], [10].

C. Applicability

Methodologies that belong to the fast evaluation range can be used at the early design steps, in order to (e.g.) compare sets of primitives and countermeasures. This allows selecting the ones that meet the desired security margins. Hybrid evaluation strategies will be used to get a fine-grained analysis, eventually aiding the implementation step. Finally, due to its complexity and accuracy, comprehensive evaluation techniques are the methods of choice when it comes to computing the security of the final implementations.

D. Example

As an example, we consider the case where a designer aims at implementing a secure block cipher on a micro-controller. For that purpose, (s)he has the choice between several ciphers (e.g. AES, PRESENT) and several software platforms (e.g. ATmega series, Cortex-M series). We further assume that (s)he wants to implement a masked version of the cipher to get some side-channel resistance. We now show how our framework can be applied to help in solving these design choices.

1) Phase 1: As a first step to discriminate between the numerous combinations, we suggest to first apply a fast evaluation to isolate some candidates. Based on some prior knowledge of the signal to noise ratios from different devices, an overview of the different masking algorithm and block ciphers, we use shortcut formulas to estimate the leakage information, the success rate and other metrics related to a side-channel attacks [2], [11].

2) Phase 2: Once few implementation candidates remain, we suggest going for a hybrid evaluation strategy as a second shortlisting procedure. Using actual traces from early stage implementations of the remaining candidates or carefully simulated traces, the designer can compute information theory metrics and more accurate formulas, along with statistical tests to verify the soundness of the masking algorithm [2], [4].

3) Phase 3: For the final stage, we assume that the designer is left with only one or very few implementation candidates for which (s)he desires to know the actual security. To that purpose, (s)he would apply an intensive comprehensive evaluation that would require performing multiple repetitions of actual profiled horizontal DPAs, mixing leakage mapping, dimensionality reduction techniques to output accurate security graphics [5].

We emphasize that while a comprehensive final evaluation should be done on an actual implementation, the ability to detect threats at earlier design stages can be crucial in term of overall design time. Simulated environments are important tools to enhance the performance of fast and hybrid evaluations, whose goal is to detect issues early and to compare different methods in a high-level view. To that purpose, the next section introduces our simulation system **Hydra**.

III. Accelerating Secure Systems Research with Automated Distributed Simulation

Systems research can be both complex and extremely time consuming. From idea to implementation, designs need to be evaluated and verified from multiple levels: starting from models to architecture simulation and finally RTL designs. In this section, we describe the current state of hardware simulation methodologies, and describe how both a holistic and automated approach is needed to address the varied concerns seen during the development process.

TABLE I
FUNDAMENTAL SIMULATION TRADE-OFFS. EVEN THOUGH TECHNIQUES LIKE FPGA-BASED SIMULATION RUN DESIGNS FASTER, THEIR PHYSICAL SIMULATION CAPACITY IS LIMITED, AND THEIR SETUP COSTS ARE HIGH, PREVENTING FAST EVALUATION DURING LATENCY-CRITICAL EARLY-PHASE DEVELOPMENT.

Methodology	Impl. time	Level of detail	Startup time	Sim. time	Sim. capacity
Modeling	high	low	fast	very fast	high
SW simulation	medium	high	fast	slow	high
HW sim. (FPGA)	high	high	slow	fast	low

Traditionally, some of the most important characteristics for simulation have been the performance and level of detail of the simulation solutions used. But, given the availability of new classes of simulation (FPGAs[1], cloud-computing-based, etc.), and the development stage of the design, there is now a larger space of simulation parameters that can be taken into account to optimize the user's time-to-result.

In fact, given the broad applicability of these tools, understanding the key characteristics of each is critical. A few key examples of these are the (1) implementation time, (2) simulation detail required, (3) simulation startup costs, (4) simulation time and (5) simulation capacity (See Table I for an overview of these trade-offs for common methodologies). There tends to be a tension between the level of accuracy and the simulation performance seen; thus, with new methodologies like FPGA-based solutions, faster does not always mean a shorter time-to-solution.

For example, given a small design, very short test simulations and early stage evaluations, software-based RTL simulators can be more practical and provide an overall lower latency. This is because of the startup cost for some solutions (number 3, above, for FPGA-based simulation), can be very large (hours) compared to the time it might take to simulate a short debugging run in software (minutes). On the other hand, the use of hardware accelerated simulation is often preferred for extensive evaluations and longer runs (including full-system, kernel and other experiments that require long time frames).

There is, therefore, no all-in-one solution that addresses the needs of each simulation situation. Instead, there is a need to move towards a progressive approach with a common framework that supports multiple simulation options. In addition to simulation itself, the management of a large number of simulations is needed to allow for checking, tracking, collection and archiving the results. At the same time, the platform should be flexible enough to migrate to new simulators and scale across machines spanning a variety of architectures and capabilities (which is the result of consistent system refreshes to maintain efficiency and performance). To handle each of these tasks, we propose **Hydra**, a unified, flexible management platform for distributed multi-system simulations.

The key reasons for building this type of platform are enumerated as follows:

1) The need for a unified interface for progressive hardware systems simulation.
2) The ability to easily migrate between different simulation methodologies, using the same job format, constrains and designs, depending on the current stage of the design (early discovery vs. system refinement, etc.).
3) The need for unified resource management platform.
4) The need for an automated, distributed, multi-system simulation.

[1]Prior work [12] uses FPGA for accelerated simulation and energy estimation while Simmani [13] uses Amazon Web Services (AWS) F1 (FPGA) instances to accelerate power modeling. Both works do not measure the power of the FPGA itself, but instead use the FPGA as a way to simulate new ASIC and SoC designs, achieving results that allow for a direct comparison to software-based simulations.

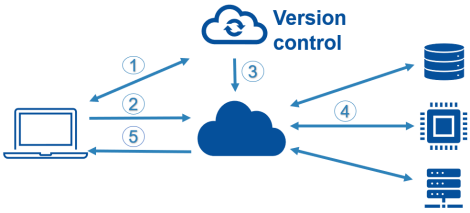

Fig. 2. The SOCure simulation framework, Hydra. The steps of the methodology include (1) saving the design to version control, (2) configuring and queuing jobs, (3) design file consolidation, (4) design simulation, and (5) viewing and processing results.

The Hydra Framework (Figure 2) achieves cycle-level distributed RTL simulation using two main approaches – (1) FPGA-accelerated simulation, and (2) software-based simulation. The main components of infrastructure, apart from the users themselves, include (1) the job management system, and (2) the simulation backends that provide both the FPGA-accelerated and software-based methodologies.

The job system acts as the backbone to submit new tasks, consolidate the hardware designs, run, and finally fetch the results. It has three main components – (1) the user, (2) the server and (3) the worker nodes. Our infrastructure supports a constraint-based resource management system, with technology library requirements and tool requirements (like Synopsys, Cadence or Xilinx tools). With a given *task definition*, where the user specifies the designs (using version control ID numbers) to be instantiated, the system compiles, simulates and exchanges data between runs for reproducible, consistent results. The key feature of Hydra is the ability to choose the system most appropriate for the task at hand, and the phase of the design that the designer is in[2]. Given the overhead of managing large-scale compilation, simulation and evaluation frameworks, building a framework that is flexible from the ground-up allows hardware designers to select the tool that is appropriate for the stage of the design that is currently being evaluated.

IV. Hierarchical protection of SoC system against Hardware Trojan Horses

This part focuses on IoT devices which are comprised out of multi IP modules. Such devices are typically being assembled to form a System-on-a-Chip (SoC) based processor. Thus, each of these modules, may be developed by a different team and probably use different design tools. Under such an distributed development environment, there are many potential ways, one can insert Trojan Horses to the system [14]. This may be done as part of almost any design stage; e.g., at the IP level, at integration time, at the tool chain or even at packaging time.

Protecting an existing system against Trojan horses is a huge task, thus this research advocate the notion DFS; i.e., to consider security protection aspects, as part of the developing flow and not to add it at the end of the developing process.

In order to achieve this goal we suggest to have the protection at different granularity levels:

- at the system level, we suggest to define a ”security monitor module” based on the use of machine learning algorithms in order to identify abnormal behaviors
- at the IP integration level, we suggest to use encrypted ports and channels
- we suggest to force inserting “markers” that will serve as assertions to guarantee correct behavior of the system
- we suggest to extend the use of debugging interfaces, such as JTAG [15] that were designed to assist the post silicon phase, and use them in order to exam the internal state of the machine, at any given time so that we could assist the Security Control Unit to detect when the system is under attack.
- New standards such as iJTAG [16], allow the testing unit to access the internal scan logic [17]. We are planning to extend these interfaces in order to exam if an IP contains an Hardware Trojan horse.

A special attention will be given during our research to new techniques in reverse engineering. Figure 3, which is taken from [18], provides an insight view of the way reverse engineering is usually done today. The process is done in two phases, the first aims to extract the netlist, or similar representation of the internal structure of the chip, and the second phase aims at “extracting” the possible functionalities of the given processor. By comparing the expected functionalities to the actual behavior, we hope to be able to identify Trojan horses, back doors, special circuits that were inserted to ease side-channel attacks [19], etc.

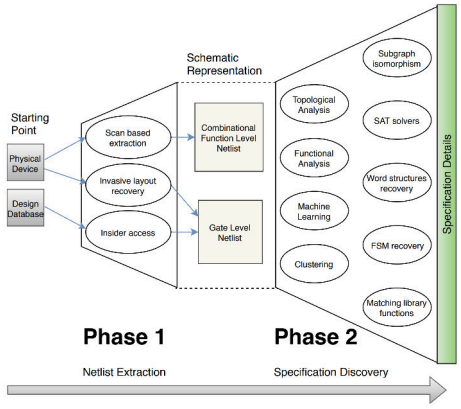

Fig. 3. Phases in Reverse Engineering.

Our research already indicated [17] that the use of the internal scan channel, can be used to extract a graph that will be close enough to the actual netlist. Based on that graph, we are applying algorithms, which are based on graph theory [20] or machine learning [21] to perform the extraction of the functionality of a given chip.

[2]For example, software RTL simulation might be preferred for simulating 1 million cycles or fewer, while FPGA-accelerated simulation can be recommended for much longer runs.

One of the major problems of using these techniques for "real-life" cases, is the fact that modern processors are using encryption and other countermeasures to protect the access to the internal scan. Thus, this issue doesn't exist when applying these techniques at the integration time, or at the testing time, since here, we can use the given interfaces to the testing environment in order to extract other information regarding the chip that the IP provider may try to hide.

V. Secure Integration of Untrusted SoC Components through Hardware Root of Trust

Trusted computing base (TCB) in a secure system is defined to include components trusted by design, hardware and software. Anything outside the TCB is potentially untrusted, for instance: software*, external memory*, on-chip components (incl. hardware Trojans). In securely integrating such components, while cryptographic-primitives* have an indispensable role, parallels from fault-tolerance parlance could be drawn to work with 'security faults' due to these components—to one or more of prevent, detect or correct such 'faults', using techniques such as fault-masking and fault-isolation*. The words with an asterisk mark the scope for this section.

A trusted execution environment (TEE) on a secure platform provides some isolation guarantees between programs co-executing over untrusted shared resources: e.g., memory hierarchy. A TEE, with remote attestation, offers a lightweight solution to trusted computing compared to expensive alternatives such as fully-homomorphic encryption, secure multi-party computation, or authenticated memory encryption with integrity trees. The security of a TEE, and indeed any higher level security solution, presumes a secure hardware root-of-trust (RoT) beneath it. Most commercial SoC vendors today include a TEE framework. The isolation guarantees, however, are often weak without other side-channel attack protections.

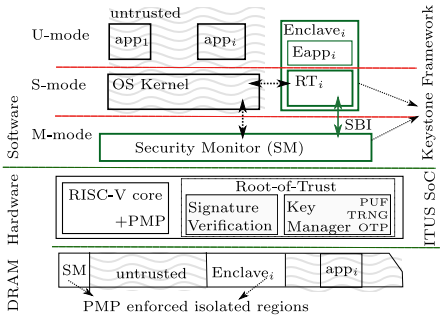

Fig. 4. Keystone ([22], Fig. 1) + ITUS [23]

This section presents an outline of integration of the open Keystone Enclaves TEE framework [22] into ITUS [23], a RISC-V based secure SoC FPGA prototype (Fig. 4).

A. Keystone

Keystone is implemented entirely as software, and only presumes that the RISC-V cores support the Physical Memory Protection (PMP) specification. It enables process isolation—the entire process stack—on external memory. As illustrated in Fig. 4 (based on Fig. 1 in [22]), the Enclave$_i$, together with its private runtime (RT) and the U-mode application, all reside in an isolated enclave address space. The core of Keystone is implemented in the M-mode security monitor (SM), which manages the PMP configuration registers. The SM itself lives in a dedicated PMP isolated region in memory and helps to dynamically allocate regions for enclave process stack. While the SM, a part of the TCB, also implements the necessary cryptographic support—such as for secure boot, enclave measurements, and root-of-trust management—this work moves much of these functions from SM to hardware.

B. ITUS hardware RoT

A PUF-based key manager together with the secure boot framework, in ITUS, constitutes the hardware RoT supporting Keystone (Fig. 4). The secure boot framework supports two public-key signature scheme options—the classical ECDSA and the post-quantum XMSS [24]—implemented fully as hardware, in a signature verification unit (SVU). The key manager in hardware is built around ColPUF [25], a PUF appropriate for FPGAs and accepts 16 bit challenges with 128 bit responses, error-corrected using a lightweight BCH(15,7,2) instance. It is a weak PUF, with extensible response lengths. It also includes a ring-oscillator based TRNG. Asymmetric key derivation under NIST B-233 elliptic curve is also implemented as hardware. The interface to PUF is only available to software signed by an entity authorized to access it—enabled by the secure boot framework; the PUF responses and the derived keys never leave the hardware boundary in plaintext.

1) Secure Boot: While many variations are possible, the following is our secure boot protocol. It begins at power-on, by verifying the ZSBL (zero-stage bootloader, bootROM), ensuring first-instruction-integrity. The FSBL contains a Challenge-Syndrome pair (CSP_{dev}) corresponding to the PUF, collected earlier using a custom 'enrollment bootloader', signed by an authorized party and verified by secure boot. After FSBL is verified, also in hardware, the CSP_{dev} is used to regenerate the device root key pair (SK_{dev}, PK_{dev}) from the PUF. Verification of subsequent boot stages can be requested through software calls to the SVU. To limit the exposure of SK_{dev}, other auxiliary root keys—such as for remote attestation and sealed storage—may be derived from it through the key manager.

2) Sealed storage for Keystone enclaves: On top of the isolation guarantees of Keystone, assuming additional fortification through side-channel attack protections (e.g., PMP + cache-partitioning [22]), sealed storage techniques can be used to cover additional physical threats such as on external memory (e.g., snooping, replay). This can be seen as a lightweight alternative to encryption with expensive integrity trees, used in the absence of a TEE. A sealing key requested for an enclave application (Eapp$_i$) may be derived from (one or more of) the measurement of Eapp$_i$ and the platform state (measurements taken in hardware) and the device root sealing key. This binds the encryption to Eapp$_i$, the platform configuration, and the device itself. In other words, data sealed with this key can

TABLE II
RESOURCE USAGE FOR ROT COMPONENTS IN ITUS

	Total LUTs	FFs	DSP48s
Key manager	7913	3454	-
PUF+BCH(15,7,2)	6924	2051	-
TRNG	637	272	-
SHA256	1402	1546	-
Elliptic Curve Keygen†	21473	249	-
SVU+XMSS Verify [24]	12162	16348	-
SVU+ECDSA Verify†	27170	6722	-
RocketTile (reference)	29081	14061	34
core+FPU	4074 + 14585	1582 + 4163	4 + 20

† these modules would share the curve point-multiplier

only be unsealed (decrypted) under those exact circumstances enabling regeneration of the sealing key.

C. Evaluation and discussion

The overheads due to the decision to implement most of the RoT components as dedicated hardware, to reduce the software footprint in the TCB, can be seen in Table II. From a security perspective, the key manager necessarily must remain in hardware. The hardware for key generation, signing and verification can be implemented with significant hardware reuse for both XMSS and ECDSA schemes. While this is an early FPGA prototype, later ASIC iterations of the SoC would replace some of the cryptographic IPs with countermeasures against physical attacks and leakage.

VI. CONCLUSIONS

Guaranteeing security of a SoC is becoming a challenging problem due to growing body of threats, ranging from physical side-channel attacks, to untrusted third party components piggybacking hardware Trojan horses. In this paper, we outline four key aspects of a secure SoC design, namely, low level attack evaluation, high level simulation, static and dynamic mitigation of attacks and integration of untrusted components through hardware root-of-trust. We present early evaluation results and show that the complementary nature of these approaches can enforce security. Designing a secure SoC beyond FPGA prototypes with in-depth scrutiny is envisioned in the next phase.

ACKNOWLEDGMENT

The authors acknowledge the support from the Singapore National Research Foundation ("SOCure" grant NRF2018NCR-NCR002-0001 – www.green-ic.org/socure).

REFERENCES

[1] "Socure: Research programme in assuring hardware security by design in systems on chip," http://www.green-ic.org/socure.

[2] A. Duc, S. Faust, and F.-X. Standaert, "Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version," *Journal of Cryptology*, vol. 32, no. 4, pp. 1263–1297, 2019.

[3] R. Sadhukhan, P. Mathew, D. B. Roy, and D. Mukhopadhyay, "Count your toggles: a new leakage model for pre-silicon power analysis of crypto designs," *Journal of Electronic Testing*, vol. 35, no. 5, 2019.

[4] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, pp. 115–136.

[5] F. Durvaux and F.-X. Standaert, "From improved leakage detection to the detection of points of interests in leakage traces," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 240–262.

[6] D. B. Roy, S. Bhasin, S. Guilley, A. Heuser, S. Patranabis, and D. Mukhopadhyay, "Cc meets fips: A hybrid test methodology for first order side channel analysis," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 347–361, 2018.

[7] S. Saha, S. N. Kumar, S. Patranabis, D. Mukhopadhyay, and P. Dasgupta, "Alafa: Automatic leakage assessment for fault attack countermeasures," in *Proceedings of the 56th Annual Design Automation Conference*, 2019.

[8] M. Nassar, S. Bhasin, J.-L. Danger, G. Duc, and S. Guilley, "Bcdl: a high speed balanced dpl for fpga with global precharge and no early evaluation," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 849–854.

[9] T. De Cnudde, B. Bilgin, B. Gierlichs, V. Nikov, S. Nikova, and V. Rijmen, "Does coupling affect the security of masked implementations?" in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2017, pp. 1–18.

[10] A. Jati, N. Gupta, A. Chattopadhyay, S. K. Sanadhya, and D. Chang, "Threshold implementations of GIFT : A trade-off analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, 2020.

[11] R. Poussier, V. Grosso, and F.-X. Standaert, "Comparing approaches to rank estimation for side-channel security evaluations," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2015, pp. 125–142.

[12] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanovicc, "Strober: fast and accurate sample-based energy simulation for arbitrary rtl," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 128–139.

[13] D. Kim, J. Zhao, J. Bachrach, and K. Asanović, "Simmani: Runtime power modeling for arbitrary rtl with automatic signal selection," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 1050–1062.

[14] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration*, vol. 55, pp. 426 – 437, 2016.

[15] "IEEE standard test access port and boundary scan architecture," *IEEE Std 1149.1-2001*, pp. 1–212, 2001.

[16] "IEEE standard for access and control of instrumentation embedded within a semiconductor device," *IEEE Std 1687-2014*, pp. 1–283, 2014.

[17] L. Azriel, R. Ginosar, S. Gueron, and A. Mendelson, "Using scan side channel to detect ip theft," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3268–3280, 2017.

[18] L. Azriel, R. Ginosar, and A. Mendelson, "Sok: An overview of algorithmic methods in ic reverse engineering," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, ser. ASHES'19. ACM, 2019.

[19] L. Lin, W. Burleson, and C. Paar, "Moles: Malicious off-chip leakage enabled by side-channels," in *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, 2009.

[20] M. Fyrbiak, S. Wallat, S. Reinhard, N. Bissantz, and C. Paar, "Graph similarity and its applications to hardware security," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 505–519, 2020.

[21] R. Elnaggar, K. Chakrabarty, and M. B. Tahoori, "Hardware trojan detection using changepoint-based anomaly detection techniques," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2706–2719, 2019.

[22] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[23] V. B. Kumar, A. Chattopadhyay, J. Haj-Yahya, and A. Mendelson, "Itus: A secure risc-v system-on-chip," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 418–423.

[24] V. B. Kumar, N. Gupta, A. Chattopadhyay, M. Kasper, C. Krauß, and R. Niederhagen, "Post-quantum secure boot," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1582–1585.

[25] B. Srinivasu, P. Vikramkumar, A. Chattopadhyay, and K.-Y. Lam, "Colpuf: a novel configurable lfsr-based puf," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 358–361.