cyberagentur

ECOSYSTEM FOR TRUSTWORTHY IT

LOS 4: FORMAL SECURITY GUARANTEES FOR TRUSTWORTHY HARDWARE SUPPLY CHAINS

Dominik Sisejkovic,
Lennart M. Reimann,
Maja Malenko,
Rainer Leupers

January 2023

Herausgeberin:
Agentur für Innovation in der Cybersicherheit GmbH

Disclaimer

Impressum

Herausgeberin: Agentur für Innovation in der Cybersicherheit GmbH
Große Steinstraße 19, 06108 Halle (Saale), Germany
E-Mail: kontakt@cyberagentur.de
Internet: www.cyberagentur.de
Twitter: https://twitter.com/CybAgBund

Die Nutzungsrechte liegen bei der Herausgeberin.

Lizenz: CC BY-NC-ND 4.0: https://creativecommons.org/licenses/by-nc-nd/4.0/
Erscheinungsdatum: 12.07.2023
Redaktion: Abteilung Sichere Systeme, Referat Sichere Hardware und Lieferketten

Los 4: Formal Security Guarantees for Trustworthy Hardware Supply Chains

Dominik Sisejkovic, Lennart M. Reimann, Maja Malenko, Rainer Leupers

January 2023

Contents

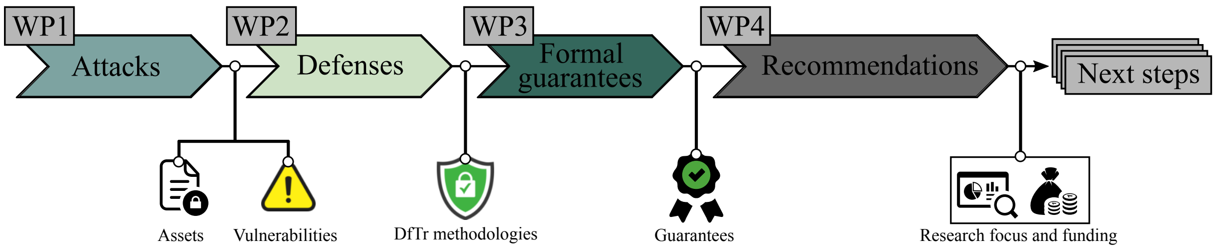


Figure 1.: Study workflow and concept.

The contemporary Integrated Circuit (IC) supply chain follows a horizontal business model in which Intellectual Property (IP) owners rely on the involvement of external design houses and untrusted, off-site foundries. This mode of operation accommodates the need for a short time to market and reduces design and production costs. However, the inclusion of third parties has led to the rise of serious security and trust concerns throughout the IC supply chain, including IP piracy, IC counterfeiting, and malicious circuit modifications known as Hardware Trojans (HTs) [1].

In the last decades, hardware Trojans have been identified as a potential source of severe security vulnerabilities [2]. These malicious circuit modifications can be utilized to exploit, manipulate, and control electronic systems through judiciously inserted hardware backdoors. HTs can potentially lead to a wide range of attacks, including information leakage, denial of service, and reliability reduction, among others. As digital hardware plays a major role in all sectors of the modern age, HTs can incur significant damage and have dangerous consequences in the context of telecommunications, automotive electronics, medical devices, financial infrastructures, and military systems—just to name a few. Even though HTs have not been reported in the wild, the sheer fact that these tiny modifications can be implanted into hardware even with basic tooling [3] offers a potent motivation to design security mechanisms.

In the last decades, many efforts have been invested in exploring, implementing, and testing various methodologies to ensure trustworthiness throughout the IC supply chain. For example, the United States Defense Advanced Research Project Agency (DARPA) has issued multiple funding programs to support trustworthy electronics research and development, including the IRIS [4], TRUST [5], and SHIELD [6] program, among others. The potential of this issue has also been recognized within Germany. The German Federal Ministry of Education and Research (BMBF) has issued a funding program for 2021–2024 to tackle the challenges of trustworthy microelectronics for Germany and Europe [7; 8]. Unfortunately, to this day, a formal process of ensuring the trustworthiness of hardware across the IC supply chain has not been defined. Therefore, within this study, we evaluate existing protection mechanisms to define the required *focus and effort of future research challenges* to guarantee secure hardware throughout the supply chain.

This study follows the overall concept presented in Figure 1, consisting of four Work Packages (WPs). The first WP concerns the identification of the asset, its different formats, and its interaction with potential vulnerabilities throughout the Hardware (HW) supply chain. Hereby, we focus on the *point of contact* between the asset on different abstraction levels and untrusted parties or their tooling to gather a set of possible vulnerabilities. Within WP1, we further analyze the specific scenario of the untrusted foundry as the common denominator for most IP owners who have to operate fabless due to cost reasons.

WP2 focuses on the analysis of state-of-the-art protection and detection mechanisms w.r.t. malicious modifications at different supply-chain stages and HW abstraction levels. These so-called Design-for-Trust (DfTr) methodologies are further evaluated for their applicability for securing the critical asset from WP1.

In the next step, within WP3, we focus on possible formal guarantees of the DfTr methodologies identified in WP2. The existing guarantees are further extended with an analysis of the requirements necessary to enable an end-to-end procedure to formally secure the identified asset throughout the supply chain.

Finally, based on the study from the previous work packages, in WP4, we compile a list of relevant research challenges that aim to enable a formally verified and trustworthy hardware supply chain. The challenges are weighted w.r.t. the work complexity, applicability, and relevance. The research recommendation is presented alongside time estimations[1].

[1] Note that the recommendations are based on possible challenges that might be addressed within the frame of 10 to 15 years.

Part I.

WP1: Attacks Enabled by Malicious HW Manipulations

Part I of this study focuses on the mechanics of the electronics supply chain and its possible attack vectors. For this purpose, the entire supply chain is described to allow for a detailed inspection of every stage. The asset that requires protection is identified throughout the supply chain. For the security analysis, we build upon the *zero-trust* model to avoid overlooking any possible vulnerabilities [9]. To document the vulnerabilities, we analyze how the asset comes into contact with third-party Intellectual Properties (IPs), tools, and entities. Thus, every *contact point* between the asset and untrusted parties is regarded as a vulnerability.

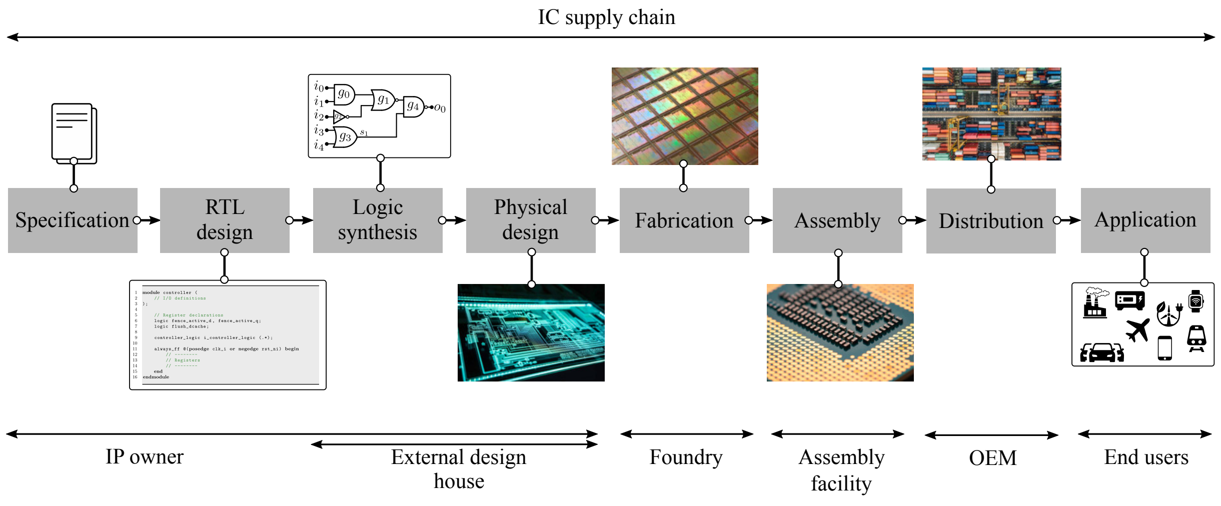
Chapter 1 illustrates the electronics supply chain. The legitimate (trusted) IP takes many forms throughout the chain. Thus, the asset is represented for every stage of the supply chain in its current format.

Chapter 2 depicts every point of contact between the different representations of the asset and a third party, or their tooling and IP. The points describe the vulnerabilities and allow the identification of the attack vectors and their respective threat level. Hereby, a special focus is given to malicious design modifications known as hardware Trojans.

In Chapter 3, a specific focus is laid on untrustworthy external design houses and foundries as the business backbone of many IP owners. This deep reliance on outsourcing sensitive assets to third parties for design and manufacturing is marked by a loss of control and assurance thereof, thus introducing a major window of vulnerability that requires more attention.

1. Classification of Critical Assets

In this study, we focus solely on the hardware design—the IP—as the asset. This chapter depicts the electronics supply chain (Fig. 1.1) and the different formats of the asset during every link in the chain [10; 11].



```
module controller (
    // I/O definitions
);

    // Register declarations
    logic fence_active_d, fence_active_q;
    logic flush_dcache;

    controller_logic i_controller_logic (.*);

    always_ff @(posedge clk_i or negedge rst_ni) begin
        // --------
        // Registers
        // --------
    end
endmodule
```

Figure 1.1.: The electronics supply chain.

The asset changes state while going through the different stages. Every state is explained in the section for the respective stage in the supply chain. The asset formats are summarized at the end of every section in a table. Additionally, figures at the end of every section illustrate how the asset changes its form, from, e.g., readable text files to a layout design to the final chip.

1.1. IP Owner

The report considers the IP owner to be the trusted party, who wants to protect the asset from untrusted party in the supply chain. Scenarios that consider a trusted client that provides a specification to the untrusted IP owner is not considered. The IP owner can have multiple descriptions of the same Hardware (HW) that represent the HW on different abstraction levels (Fig. 1.2). The design of the HW is typically driven by multiple application-dependent objectives, such as power, area, and performance. These design goals are defined in the 'specification'. As we focus on the supply chain, we do not consider security flaws present in the specification, but only vulnerabilities, introduced within the supply chain during contact with untrusted parties, their IP, or their tooling.

Once the specification is completed, the HW designer starts implementing the design in a Hardware Description Language (HDL). Common choices for HDLs operate on the Register-Transfer Level (RTL), such as Verilog and VHDL [12]. An implementation on RTL allows fine-grained design choices due to the low abstraction level. However, the low abstraction level

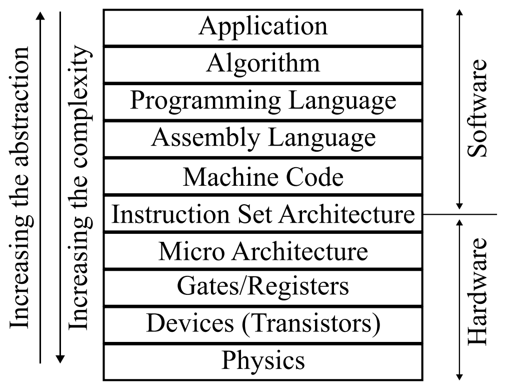


Figure 1.2.: Different hardware abstraction layers.

results in a more complex debug environment. Thus, an increasing number of design houses implement their designs in languages on higher abstraction levels. This ranges from software application languages, such as C (compiled to RTL via high-level synthesis [13]) to architecture description languages such as nML [14] that can be utilized to generate RTL automatically. Many commercial and open-source tools exist to process descriptions on higher abstraction levels to generate RTL [15]. Additional novel HW description languages include Chisel [16] and Bluespec [17], which can be used to generate Verilog.

Furthermore, modern HW designers use Virtual Prototypes (VPs) to simulate their design and enable early development of software for the intended HW. Some tools allow a generation of RTL code from the VP descriptions, such as SystemC [13]. But many designers implement the VP next to the hardware design.

Some components of the final RTL description are not designed in-house but are bought from third-party vendors. A third-party HW description on RTL is called soft IP. The additional RTL code is connected to the in-house design concluding the work on this abstraction level.

Once the RTL code is finalized, it needs to be transformed to a layout [18]. The resulting GDSII description is required by a foundry to manufacture the chip. This is done in two steps. First, logic synthesis is used to generate the gate-level netlist from the RTL description. This can be done using modern Electronic Design Automation (EDA) tools, such as Synopsys Design Compiler [19]. Second, the gate-level netlists is further processed via placement and routing to generate a spatial description of the HW—the so-called layout. Some IP owners do this process in-house, while others transmit their RTL design description to an external design house, where they conduct the logic synthesis as well as the placement and routing process. In this study, we will cover both scenarios, a supply chain with and without an external design house.

Table 1.1 and Figure 1.3 illustrate the different forms of the asset in the hands of the IP owner.

1.2. External Design House

An external design house can be the first third party to have full direct access to the IP. The designer gives the design house access to the HW description in the form of the gate-level netlist or the RTL description. The external designers are responsible for processing the description to enable the manufacturing by a foundry, resulting in a functional, in-silicon chip.

Description	Format	Abstraction level
Specification	Text file or piece of paper	Specification
Virtual prototype	Text file (SystemC, ...)	System level
Architecture description	Text file (nML, Bluespec, Chisel,...)	Architecture level
RTL design	Text file (Verilog, VHDL, SystemVerilog, ...)	Register-Transfer level
Gate-level netlist	Text file (Verilog)	Gate-level

Table 1.1.: List of assets in the hands of the IP owner.

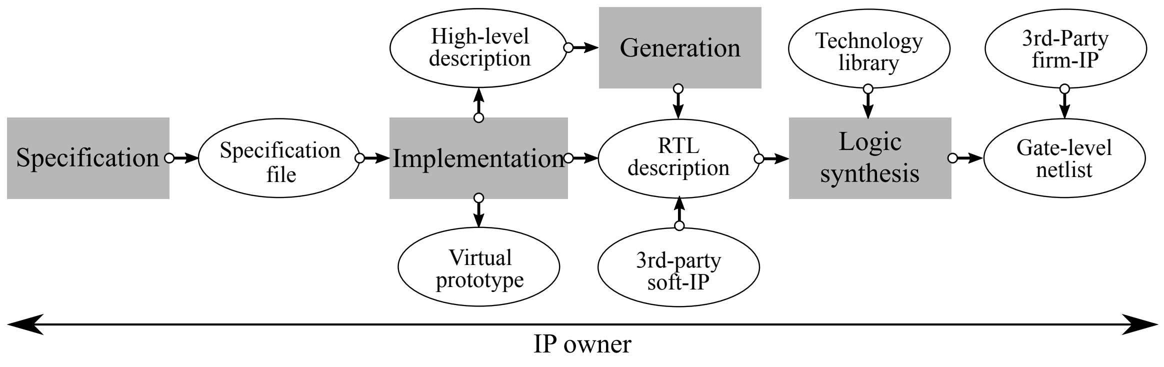


Figure 1.3.: The asset changing format in the hands of the IP owner.

In case the external designers process the RTL description, logic synthesis needs to be done, mapping the generic description onto a technology library. The resulting gate-level netlist can be used to estimate the clock speed and area of the resulting chip.

The netlist is technology-dependent but does not give any information about the physical location of every gate. Placement and routing are conducted by the design house using modern EDA tools, typically by Cadence or Synopsys. This step ensures the generation of a 3-dimensional description defining the placement and connectivity of every single transistor in the chip. Additionally, the clock tree needs to be designed with the intent that the clock signal reaches every design component in time. The final design in GDSII format (=the layout) can be forwarded to the foundry for manufacturing.

An additional service that is offered by most design houses is the implementation of the Design for Test (DfT) logic. Further wiring and registers are added to the design to allow the testing of the internal state of the design. Scan-chains offer easy access to the internal signal state of the architecture at runtime. The signals are accessible via an additional interface, often JTAG, to allow the analysis of the architecture's behavior on new software or input patterns. The interface is often encrypted or destroyed after the chip is tested to avoid the leakage of sensitive data after releasing the final chip [20].

Table 1.2 and Figure 1.4 illustrate the different formats of the asset in the hands of the external design house.

Description	Format	Abstraction level
RTL design	Text file (Verilog, VHDL, SystemVerilog, ...)	Register-transfer level
Gate level netlist	Text file (Verilog)	Gate Level
Layout	GDSII Design	Layout level

Table 1.2.: List of asset formats in the hands of the external design house.

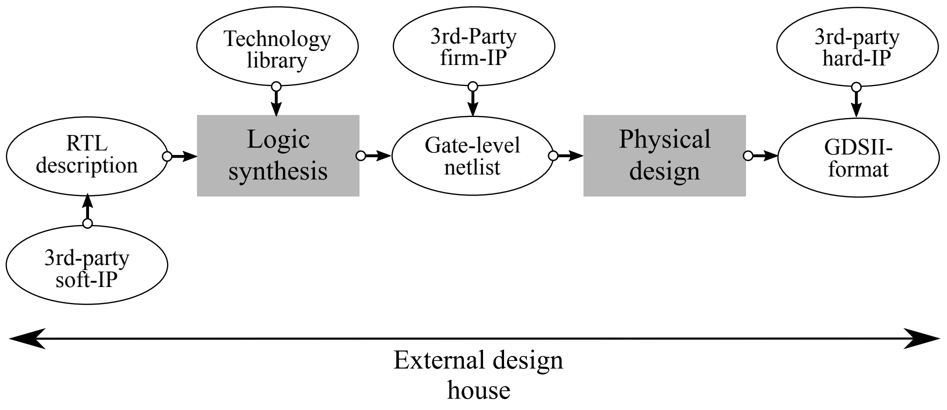


Figure 1.4.: The asset changing format within the hands of the external design house.

1.3. Foundry

Many modern designers cannot maintain their own foundry. Therefore, the manufacturing process is outsourced to a third party. The foundry receives the layout of the design and returns a batch of manufactured chips.

The technology library chosen for the logic synthesis needs to represent a technology that can be provided by the foundry, as the manufacturing equipment must match the exact specifications of the underlying technology. In many cases, the foundry also provides the packaging of the chip, allowing other parties to implement the IP on a circuit board, e.g., a Printed Circuit Board (PCB). The ports are connected to pins, and the chip is packaged for protection against physical damage.

Table 1.3 and Figure 1.5 illustrate the different formats of the asset in the hands of the foundry.

Description	Format	Abstraction level
Layout	Text file (GDSII format)	Layout
Final chip	Physical design, not combined into a device yet	Physical layer
Packaged chip	Physical design - packaged - allows integration into PCB	Physical layer

Table 1.3.: List of asset formats in the hands of the foundry.

1.4. Assembly

The assembly step in the supply chain can include multiple entities. The assembly constructs multiple components to fit a certain application. For a single chip batch, this can lead to either the same product or diversified products.

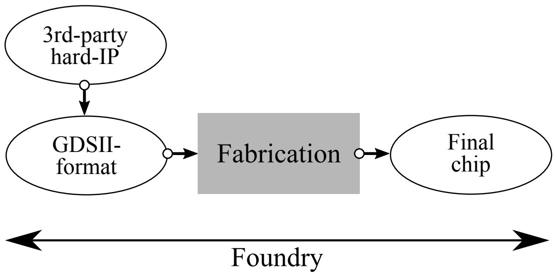


Figure 1.5.: The asset changing format within the hands of the foundry.

A common strategy for assembling the final product is the modular system. Several smaller components are assembled to form intermediate products, which are further assembled with other intermediate products to form a larger module or the final product. Multiple production entities may be involved in developing the individual modules. In this context, the client can receive a packaged chip of the owner's IP or an unpackaged version, depending on how the different components are assembled.

Table 1.4 and Figure 1.6 illustrate the different formats of the asset in the hands of the assembly facility.

Description	Format	Abstraction level
Final chip	Physical design, not combined into a device yet	Physical layer
Intermediate device	Physical design, combined with some additional components	Physical layer
Final device	Physical design, packaged	Physical layer

Table 1.4.: List of asset formats in the hands of the assembly facility.

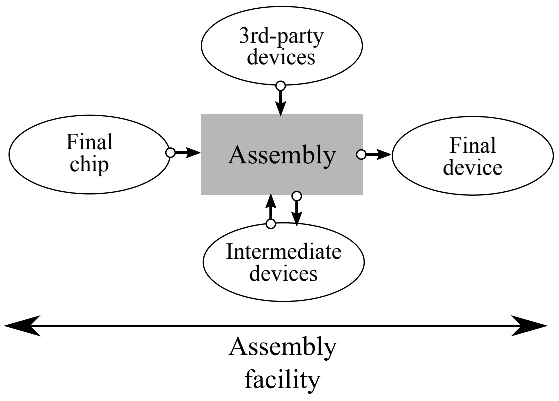


Figure 1.6.: The asset changing format within the hand of the assembly facility.

1.5. OEM

After all components are assembled to form the final product, the chips are distributed by either the IP owner or an Original Equipment Manufacturer (OEM) procurement is utilized. The product can be sold to an OEM, which distributes the product to be rebranded and used in other companies' products.

Table 1.5 and Figure 1.7 illustrate the different formats of the asset in the hands of the OEM.

Description	Format	Abstraction level
Final device	Physical design, packaged	Physical layer

Table 1.5.: List of asset formats in the hands of the OEM.

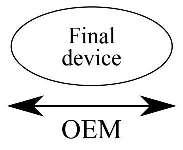


Figure 1.7.: The asset changing format within the hands of the OEM.

1.6. End Users

The final design, which includes the asset, is sold to the end user.

Table 1.6 and Figure 1.7 illustrate the different formats of the asset in the hands of the end user.

Description	Format	Abstraction level
Final device	Physical design, packaged, combined with other components	Physical layer

Table 1.6.: List of asset formats in the hands of the end user.

The final asset chain that corresponds to the supply chain in Figure 1.1 is illustrated in Figure 1.8.

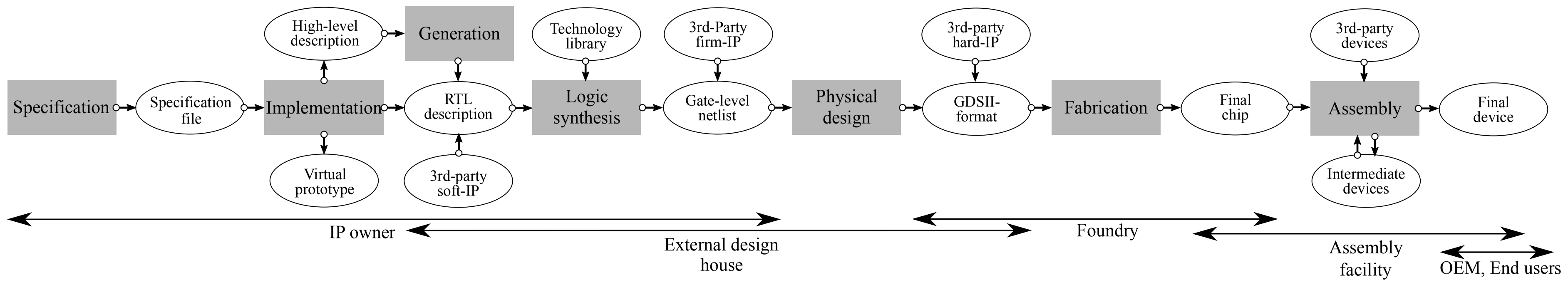


Figure 1.8.: The asset changing format within the hands of the end user.

2. Electronics Supply Chain Threats: Hardware Manipulations

The electronics supply chain operates in a horizontal fashion, optimizing for short time-to-market deadlines and cost reduction. These objectives, however, force legitimate IP owners to rely on the inclusion of a plethora of external entities, as well as closed-source, third-party EDA tools. This complexity and distributed nature of the supply chain have led to a profusion of security vulnerabilities due to one simple reason—*unverifiable assurance of trust* between the involved parties. One of the most critical security threats enabled by this absence of verifiability are malicious design modifications known as Hardware Trojans (HTs). HTs have become a driving force for security research for more than a decade [2]. Unfortunately, the issue of HTs is far from resolved.

To further analyze the background and source of HTs[1], this chapter takes a closer look at state of the art in HT design, classification, and implementation in Section 2.1. Possible attack models that lead to HT insertion are analyzed in Section 2.2. A threat evaluation is performed through the mentioned analysis, with a final conclusion in Section 2.3. Finally, open challenges are summarized in Section 2.4. *Note that we consider any form of malicious manipulation, regardless of whether logic is removed, bypassed, or added within a design, as a HT.*

2.1. Hardware Trojans

Hardware Trojans can be defined as malicious, intentional, and stealthy modifications of integrated circuits during the entire HW supply chain [21]. The malicious behavior can be embodied in the form of information leakage, power dissipation, denial of service, performance degradation, or, more generally, in a behavior that is not originally intended. The modification must be intentional; otherwise, the Trojan is equivalent to a random fault. The implementation of Trojans is assumed to be stealthy to enable the modification to pass through any form of tests and security checks, thus allowing the Trojan to remain dormant in the chip until activation.

2.1.1. Trojan Components

The research community has defined the following HT components: trigger and payload [22; 23; 24]. A visualization is given in Figure 2.1. The *trigger* mechanism activates the payload on a certain event. This event can be arbitrarily complex, including external signals (e.g., through a signal captured by an antenna), specific input values or a series of input patterns, specific circuit states, number of cycles, sensor values, and others. The *payload* is the manifestation of the malicious behavior that results in the intended attack.

[1] Within the scope of this study, we sometimes refer to hardware Trojans as "Trojans".

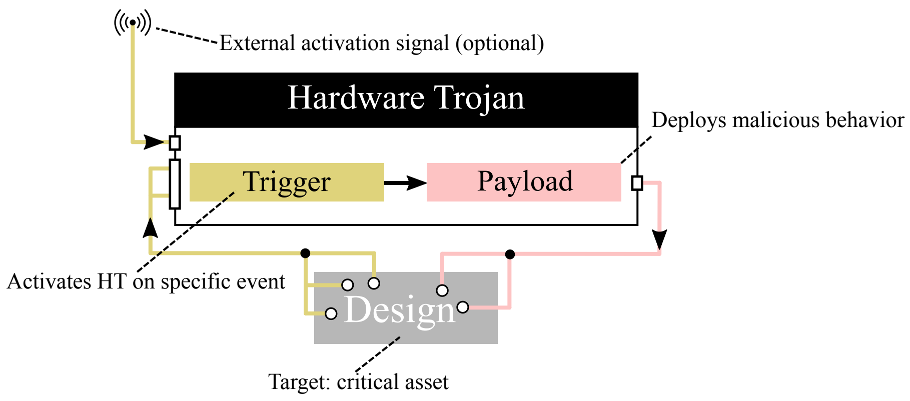

External activation signal (optional)
Hardware Trojan
Deploys malicious behavior
Trigger
Payload
Activates HT on specific event
Design
Target: critical asset

Figure 2.1.: Hardware Trojan components.

The implementation variety of the trigger and the payload has, in principle, no limits. To make things worse, HTs can be injected into the target design at any stage within the HW supply chain. Thus, HTs create a vast attack landscape that still remains an open problem.

2.1.2. Classifications

The complexity of the HT-enabled attack landscape has triggered efforts to create a classification of HT implementations, with the intention of easing the path toward potential detection or prevention methods by better understanding how Trojans could be implemented. In the following, we discuss existing classification concepts. Note that most classification systems are evolving in time, as researchers identify novel ways of injecting and designing Trojans. Hence, the following data captures the state of classifications in the referenced publications.

2.1.2.1. Physical, Activation, and Action Characteristics

An early attempt at classifying hardware Trojans was presented in [25; 26] with the objective of facilitating the proper evaluation of the effectiveness of HT detection methods. Here, HTs are decomposed based on three criteria: physical, activation, and action characteristics. This classification is shown in Figure 2.2.

The physical class describes different hardware implementations of Trojans, including the distribution, structure, size, and type. The distribution refers to the location of the Trojan's implementation within the chip's layout. The structure category captures the case when the layout must be regenerated due to the injected HT; thus, either a layout change exists or not. The size category describes the number of components in the chip that have been added, changed, or deleted due to the HT. Finally, the type category segregates Trojans into the functional and parametric classes. The former includes Trojans that are manifested through the physical addition or deletion of transistors or gates. The latter describes Trojans that rely on modifications to existing wires, gates, and logic.

The second decomposition accounts for the activation characteristics that trigger the HT's malicious behavior. The characteristics are separated into externally and internally activated Trojans. External activation can be manifested in the form of an antenna that receives a certain trigger signal or specific sensor values used to interact with the chip's environment. Internally activated Trojans capture two subclasses: always-on and conditional HTs. Always-on HTs are

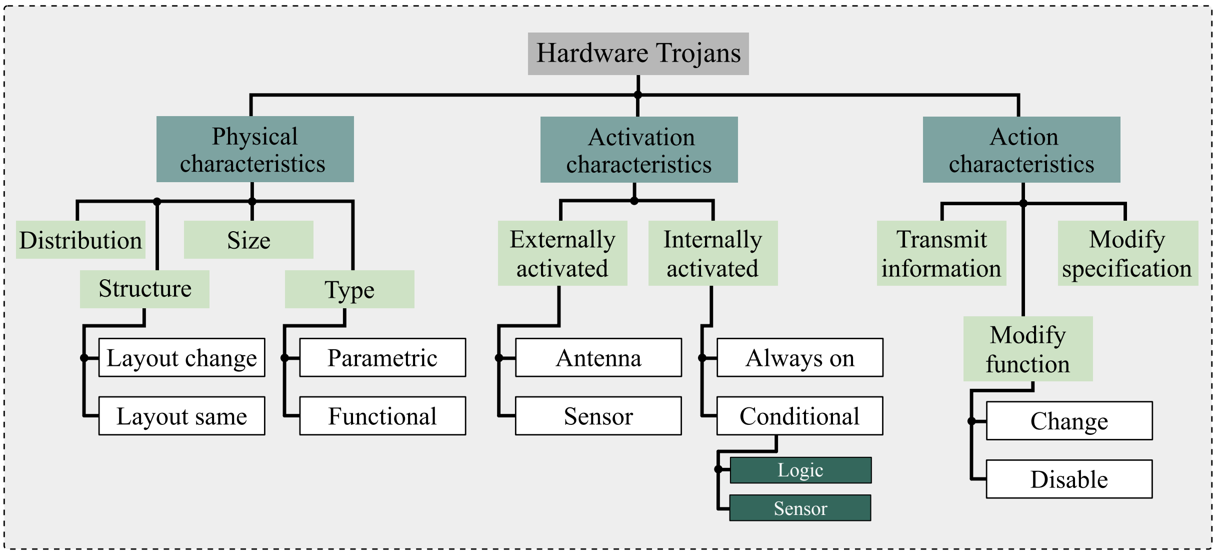


Figure 2.2.: Hardware Trojan classification based on physical, activation, and action characteristics.

continuously active and can corrupt the chip's functionality at any time. Conditional Trojans, on the other hand, activate only on the occurrence of a specific event, such as specific logic or sensor values.

The third class captures the action characteristics of Trojans. Note that "action" can be considered as the payload—as discussed in Section 2.1.1. This class targets three different behavioral traits: transmission of information, functional modification, and modification of the specification. The transmit-information class captures Trojans that enable or facilitate the transmission of secret keys or, in general, data to an attacker. The modification-of-function class describes Trojans that affect the chip's intended functionality by adding, removing, or bypassing logic. Finally, the modify-specification class includes Trojans that target the change of parametric values of the chip, such as the power or delay characteristics.

2.1.2.2. Activation Mechanism and Effect

A high-level classification of hardware Trojans was presented in [24]. The classification segregates Trojan types based on variations in activation mechanisms and the payloads' effect. The classification is presented in Figure 2.3.

The trigger classifies HTs into analog and digital Trojans. The former captures activation mechanisms through analog computing, such as device aging and temperature variations. The latter represents Trojans that are activated by a Boolean function. Trojans triggered by analog effects are considered to attack process steps that compromise the reliability of the target chips. Examples include specific values of on-chip sensors or voltage variations. Digitally triggered Trojans are further classified as combinational and sequential. Combinational HTs listen to specific in-circuit values that do not require the support of multiple states. An example are one-time rare inputs or node values[2]. Sequential Trojans rely on a certain combination or value of states, such as a certain value of a counter after a selected number of cycles.

[2] A node value is the value of the output wire of a node (gate) in the circuit.

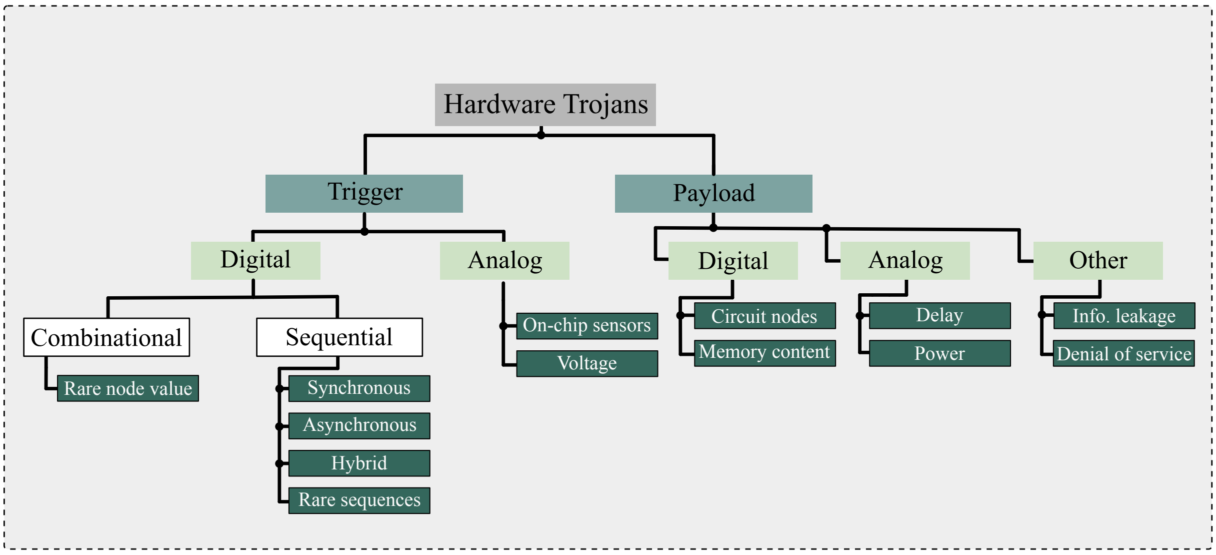


Figure 2.3.: Hardware Trojan classification based on trigger and payload mechanisms.

The payload mechanism segregates Trojans based on the type of effect the Trojans have on the infected circuit operation, including digital, analog, and other responses. Digital payloads can, for example, change the value of certain signals within the circuit or the system's memory. Analog payloads influence the power and delay characteristics of the target chip. Other payloads might result in information leakage by creating additional paths between critical (secret) keys or data and primary outputs, or block further chip operation, thus leading to a Denial of Service (DoS) attack.

2.1.2.3. A Comprehensive Classification

A comprehensive classification of hardware Trojans was proposed and refined in [27; 10]. The classification is visualized in Figure 2.4. This classification was developed to enable the comparison of HT-detection methods. Moreover, it was used as a guideline to compile the currently largest collection of online-available "trust benchmarks"[3]—a set of Trojan-infected hardware benchmarks. This classification looks at the entire spectrum of Trojan types spanning from the supply-chain phase in which the Trojan is inserted to the physical characteristics of the Trojans' implementation.

At the highest level, the classification starts with Trojans that are inserted in different supply-chain stages, thus covering the specification, design, fabrication, testing, and assembly and packaging phase. Next, the abstraction level at which the Trojan is injected is considered. This includes Trojans designed and inserted at the system level down to the physical layout. In the following step, Trojans are segregated based on the activation mechanisms. Here, similar to the classification in Section 2.1.2.1, the trigger mechanism can be always-on or conditionally triggered by a selected event. Next, the Trojan effect is used as a separation criterion. Thus, Trojans are classified based on the manifestation of their payload in the form of a functional circuit change, performance degradation, information leakage, and denial of service. The location criterion describes the exact location, such as a design module, where the HT is inserted. Finally, Trojans are distinguished based on their physical characteristics in the same manner as in Section 2.1.2.1.

[3] The benchmarks are freely available here: `https://www.trust-hub.org/`

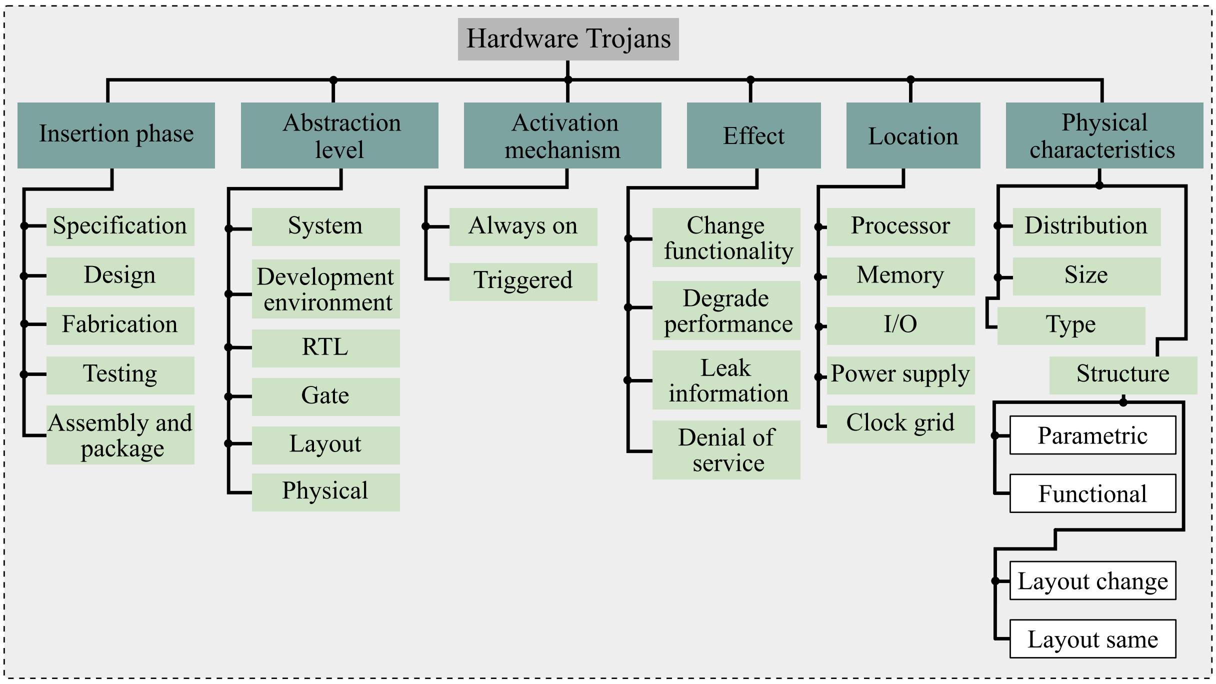


Figure 2.4.: A comprehensive hardware Trojan classification.

2.1.2.4. Reverse-Engineering-Based Classification

The previous classification systems offer detailed guidelines on how to describe, categorize, and evaluate hardware Trojans. Unfortunately, the classifications simply try to capture as many implementation features as possible, thereby speculating what traits HTs might have. Another downside of this descriptive classification approach is that it does not create any relation to defensive approaches: what does a Trojan class tell us about how we can protect against it? To answer this question and overcome the endless feature accumulation of Trojan types, a consolidated classification was presented in [28]. This classification system aims at separating all Trojans into two categories, Class-1 hardware Trojans (C1HTs) and Class-2 hardware Trojans (C2HTs), based on a single criterion—the requirement of Reverse Engineering (RE) for Trojan design and insertion. The consolidated system is shown in Figure 2.5.

C1HTs cover Trojans that are RE-dependent. Thus, the attacker must invest a certain effort to understand the target's (asset's) design specification (at an arbitrary level) to design and insert a design-specific hardware Trojan. Consequently, a C1HT allows for a *controllable* trigger leading to a *high-impact* attack in a *known* application environment. In a more simple model, C1HTs comprise all Trojans for which at least one design component—the trigger **or** the payload—is RE-dependent. An example of such a Trojan is a modification of the decoder of a RISC-V microarchitecture that filters out a specific sequence of (software) instructions and, upon triggering, leads to a complete stop of instruction execution, i.e., a DoS attack. Hereby, we assume that the attacker must identify the exact functionality of the asset (=processor), its specifications (=RISC-V instruction set), its microarchitectural design traits (=location of the decoder), and the application area (=e.g., control unit of a radar system). Evidently, this configuration can lead to disastrous attack vectors. Since a certain RE effort is required for the injection of class-1 Trojans, any form of protection mechanism that increases the RE effort might be used as a deterrent.

C2HTs includes Trojans that are RE-independent. Therefore, an attacker is able to insert this

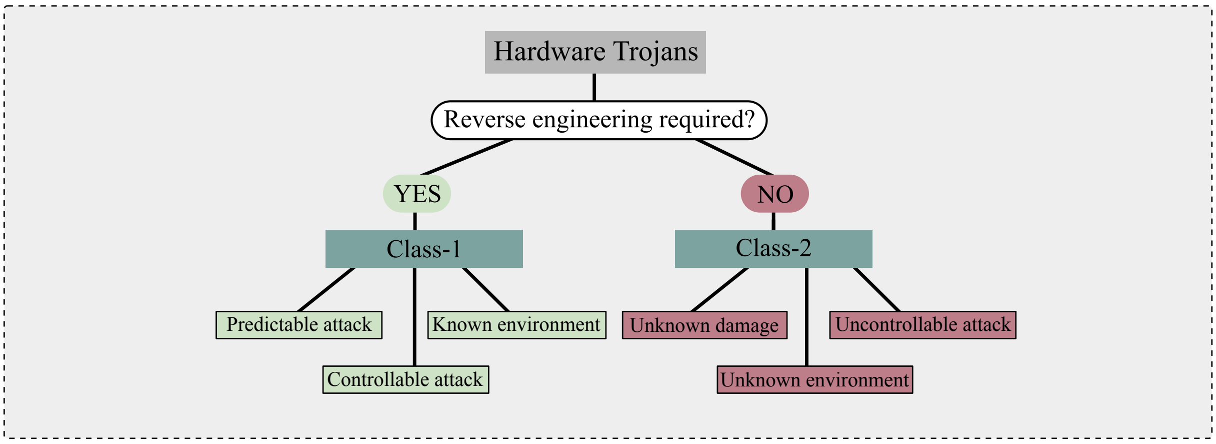


Figure 2.5.: Hardware Trojan classification based on the reverse-engineering effort.

type of hardware Trojans into a design at any stage or abstraction level without any knowledge about the underlying asset. Consequently, C2HTs are malicious modifications for which both design components—the trigger **and** the payload—are RE-independent. As no RE effort is required for the design and insertion of a C2HT, it can always be inserted regardless of potential protection mechanisms. Nevertheless, such a Trojan might be of low impact, high detectability, result in random attacks, or by chance, might never be triggered. An example of a class-2 Trojan is an XOR gate that takes two inputs: a selection input and a random wire from the asset. When the selection bit is (in an arbitrary way) set to 1, the second input is inverted, thus injecting some form of faulty behavior into the circuit. Otherwise, no change is introduced. Evidently, such a Trojan can always be inserted into the design without any knowledge about the asset or its application domain. More details can be found in [22].

2.1.3. Example Trojan Implementations

As discussed in Section 1, a wide range of entities, tools, and third-party IP is involved in the design of a single chip. Moreover, the involved parties are often spread across multiple companies operating in different countries and continents. Alongside the endless and intricate implementation variations of HTs, the distributed nature of the supply chain makes it extremely hard to catch hardware Trojans in the wild. Thus, so far, no confirmed case of a "real world" HT is known. Moreover, the question is whether a company that detects an infected chip would publicly announce it? The infection might remain a secret to protect the company's business. Nevertheless, some alleged Trojans have been documented. One prominent example is described in the case of a Syrian radar system that failed to warn of an incoming airstrike. The failure was speculated to have been caused by HTs integrated into the defense systems [29; 30].

The academic examples, on the other hand, are abundant and reach across all previously discussed classifications [24; 31; 32; 33; 34; 10; 35; 36; 37], some of which have also been put into silicon for demonstration and measurements [38; 39]. Moreover, a recent study also demonstrated the automatic insertion of hardware Trojans into circuits by means of standard EDA development tools [3]. Note that a practically unlimited number of Trojan examples can be fabricated as Trojans are, in terms of implementation, not different from any other hardware circuit—one fundamental reason why it is so challenging to detect these alterations.

Table 2.1.: Threat levels for hardware Trojan insertion.

Level	Access format	Direct access?	Access environment	Comment
LOW	Third-party IP	✗	Trusted	HT only in external files.
HIGH	Third-party IP	✗	Untrusted	HT only in external files.
ELEVATED	EDA tool	✓	Trusted	HT insertion requires a high level of automation.
HIGH	EDA tool	✓	Untrusted	HT insertion requires a high level of automation.
ELEVATED	Employee(s)	✓	Trusted	HT embedded in in-house environment.
HIGH	Employee(s)	✓	Untrusted	HT embedded in off-site environment.

2.2. Attack Models

This section takes a closer look at different attack models that are enabled by the *contact point* between the asset (the legitimate IP) and untrusted or external entities, tools, or third-party IPs. Hereby, we look at each stage of the supply chain individually. For each stage, we analyze what attack scenario is enabled by the available contact points. Every scenario is categorized by a *threat level*. The threat levels are further described in Section 2.2.1. An overview of all HT-based threats are shown in Figure 2.6.

2.2.1. Threat Levels

The threat levels used in this study are listed in Table 2.1. The objective of the threat levels is to express the probability of an attack at a certain stage of the supply chain, regardless of its effort and impact. Two criteria define the threat level: the access format and environment. These criteria are defined as follows:

- **Access format:** The access format defines how the malicious entity gets access to the asset. The format includes third-party IP, EDA tools, and rouge employees.
  - Third-party IP: The HT is embedded in a third-party IP. Thus, the HT can only influence the asset through externally included files.
  - EDA tool: The EDA tool has complete access to the asset and inserts the HT.
  - Employee: A malicious employee has complete access to the asset and inserts the HT.
- **Access environment:** The access environment includes two scenarios: trusted and untrusted. The former indicates that the HT is (willingly or unwillingly) inserted into the asset in an environment that is supposed to be trusted (e.g., in the case of the IP owner). The latter captures HT insertions in external, off-site environments that are considered untrusted.

Using the criteria, we can distinguish three threat levels that are mainly directed by the **access environment**. A visualization of the decision tree that determines the threat level is shown in Figure 2.7. If the attack takes place in an untrusted environment, every threat, regardless of its access format, is labeled with a HIGH threat level. The reason is that an untrusted environment has full and direct access to the asset.

If the attack takes place in a trusted environment, the threat level depends on the **access format**. If the HT is embedded in a third-party IP (=no direct access), it is marked with a LOW threat

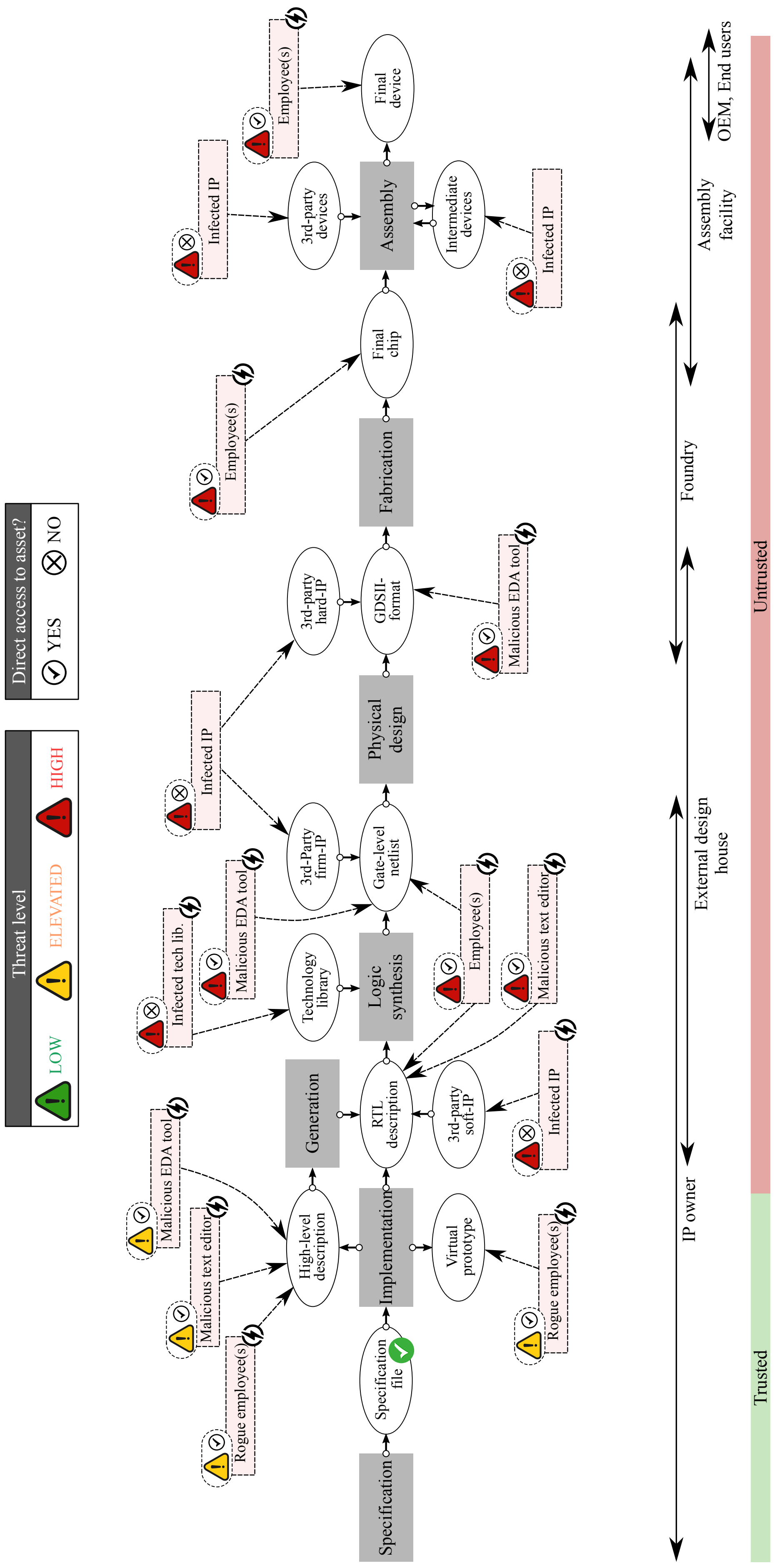


Figure 2.6.: Hardware Trojans in the electronics supply chain.

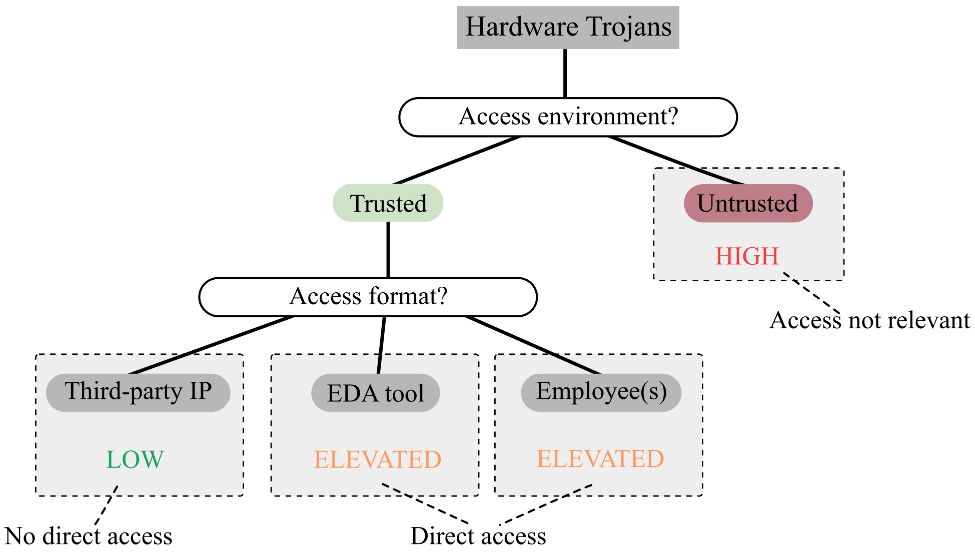


Figure 2.7.: Threat-levels decision tree.

level. If the access is direct, e.g., through a malicious EDA tool or rogue employees, the threat level is set to ELEVATED.

2.2.2. Hardware Trojans: Insertion Effort and Impact

Each scenario in the following sections is described with two additional parameters: effort (E) and impact (I). E captures the effort required to insert a design-specific hardware Trojan (=class-1 HT). I describes the potential impact of the hardware Trojan (=impact of the attack caused by a hardware Trojan). We estimate the effort and impact based on three variables:

- Environment type (τ): trusted or untrusted (see Section 2.2.1).
- Reverse engineering effort (κ): the effort required to fully understand the target design in order to insert a design-specific (class-1) Trojan.
- Direct access (δ): the type of access to the target design (see Section 2.2.1).

The variable values are presented in Table 2.2. The effort depends on the type of environment

Table 2.2.: Effort and impact variables.

(a) Environment (τ).

Environment type	Value
Trusted	$\tau = 1$
Untrusted	$\tau = 2$

(b) RE effort (κ).

RE effort	Value
RTL or higher	$\kappa = 1$
Gate level	$\kappa = 2$
Layout level	$\kappa = 3$
Chip level	$\kappa = 4$
Device level	$\kappa = 5$

(c) Direct access (δ).

Direct access	Value
No	$\delta = 1$
Yes	$\delta = 2$

and reverse-engineering effort:

$$E = \frac{\kappa}{\tau}. \tag{2.1}$$

An untrusted environment facilitates the insertion of a hardware Trojan (=lower effort) as the insertion takes place in an isolated, external location. A trusted environment increases the effort, as a malicious behavior of an attacker might be more detectable, thus constraining the actions of the adversary. Moreover, the effort decreases with higher abstraction levels. For example, it stands to reason that the effort to insert a design-specific Trojan is lower on RTL than on layout level, as less reverse-engineering must be performed to understand the target design.

The impact of the Trojan depends on the environment and the access to the target design:

$$I = \tau \cdot \delta. \tag{2.2}$$

The impact is higher in untrusted environments, because the malicious party can move more freely. Furthermore, having direct access to the target design allows for a more design-specific implementation of the Trojan, thus resulting in a (potentially) higher impact.

Using the definitions above, we can look at corner cases to better understand the variables:

- Highest effort: trusted environment and layout level (no known example).
- Lowest effort: untrusted environment and RTL (or higher) level (external design house, RTL).
- Highest impact: untrusted environment and direct access (external design house/foundry, employees).
- Lowest impact: trusted environment and no direct access (IP owner, third-party IP).

2.2.3. IP Owner

The IP owner is considered a trusted environment. The possible attack vectors in this stage are listed in Table 2.3. The specification is assumed to be trusted and not affected by malicious modifications. All attack vectors with direct access to the asset are labeled with an `ELEVATED` threat level. The reason is that even though direct access is provided, the attack takes place in a trusted environment in which (almost) all employees are supposed to be trusted. Thus, even malicious employee(s) or EDA tools might be less effective in this environment. Third-party IPs are labeled as `LOW` as no direct access to the asset is provided.

2.2.4. External Design House

The external design house is considered an untrusted environment. The attack vectors are listed in Table 2.4. All contact points of the asset with third-party IP, tools, and employees are labeled as `HIGH` threats since the untrusted party has full and uncontrolled access to the asset.

2.2.5. Foundry

The foundry is considered an untrusted environment. The attack vectors are listed in Table 2.5. All contact points of the asset with third-party IP, tools, and employees are labeled as `HIGH` threats

Table 2.3.: Attack model: IP owner.

Asset	Untrusted entity	Example scenario(s)	Threat level	Direct access	*E*	*I*
Virtual prototype	Rogue employee(s)	Hiding faulty behavior of trigger software.	LOW	✗	1	1
Architectural desc.	Rogue employee(s)	Manual inclusion of HT.	ELEVATED	✓	1	2
Architectural desc.	Malicious third-party text editor.	Automatic insertion of HTs based on static code analysis.	ELEVATED	✓	1	2
Architectural desc.	Malicious EDA tool: code generation	Automatic insertion of HTs based on static code analysis.	ELEVATED	✓	1	2
RTL design	Rogue employee	Manual inclusion of HT.	ELEVATED	✓	1	2
RTL design	Malicious third-party text editor.	Automatic insertion of HTs based on static code analysis.	ELEVATED	✓	1	2
RTL design	Infected third-party IP.	Infected soft IP.	LOW	✗	1	1
Gate-level netlist	Rogue employee(s)	Manual inclusion of HT.	ELEVATED	✓	2	2
Gate-level netlist	Malicious EDA tool: logic synthesis	Automatic insertion of HTs based on static code analysis.	ELEVATED	✓	2	2
Gate-level netlist	Infected technology library	HT hides in specific cells.	LOW	✗	2	1
Gate-level netlist	Infected third-party IP	Infected firm IP.	LOW	✗	2	1

Table 2.4.: Attack model: External design house.

Asset	Untrusted entity	Example scenario(s)	Threat level	Direct access	*E*	*I*
RTL design	Employee(s)	Manual inclusion of HT.	HIGH	✓	0.5	4
RTL design	Malicious third-party text editor.	Automatic insertion of HTs based on static code analysis.	HIGH	✓	0.5	4
RTL design	Infected third-party IP.	Infected soft IP.	HIGH	✗	0.5	2
Gate-level netlist	Employee(s)	Manual inclusion of HT.	HIGH	✓	1	4
Gate-level netlist	Malicious EDA tool: logic synthesis	Automatic insertion of HTs based on static code analysis.	HIGH	✓	1	4
Gate-level netlist	Infected technology library	HT hides in specific cells.	HIGH	✗	1	2
Gate-level netlist	Infected third-party IP	Infected firm IP	HIGH	✗	1	2
Layout	Infected third-party IP	Infected hard IP.	HIGH	✗	1.5	2
Layout	Malicious EDA tool: physical design	Automatic insertion of HTs based on netlist analysis.	HIGH	✓	1.5	4

Table 2.5.: Attack model: foundry.

Asset	Untrusted entity	Example scenario(s)	Threat level	Direct access	*E*	*I*
Layout	Infected third-party IP	Infected hard IP.	HIGH	✓	1.5	4
Layout	Employee(s)	Manipulation of layout.	HIGH	✓	1.5	4
Chip	Employee(s)	Manipulation of dopant polarity of existing transistors.	HIGH	✓	2	4

Table 2.6.: Attack model: Assembly.

Asset	Untrusted entity	Example scenario(s)	Threat level	Direct access	*E*	*I*
Chip	Infected third-party devices	Infected third-party devices.	HIGH	✗	2	2
Chip	Employee(s)	Manipulation of the package by, e.g., appending antennas (triggers).	HIGH	✗	2	2

Table 2.7.: Attack model: OEM.

Asset	Untrusted entity	Example scenario(s)	Threat level	Direct access	*E*	*I*
Chip	Employee(s)	Exchanging devices for infected ones.	HIGH	✗	2.5	2

since the untrusted party has full and uncontrolled access to the asset. Note that the foundry sometimes includes an external mask house for mask generation. However, we assume that the mask generation is part of the fabrication process, thus being an untrusted environment.

2.2.6. Assembly

The assembly is considered an untrusted supply-chain stage. At this point in the chain, the hardware is already manufactured and placed in silicon. Thus, malicious changes on the design level are not possible. Nevertheless, the assembly facility can still manipulate and adjust the chip's packaging by, for example, adding malicious HW blocks or manipulating the packaging and pins. The attack vectors for this stage are listed in Table 2.6.

2.2.7. OEM

The OEM is considered an untrusted entity. In this stage, the design is already packaged. Thus, complex design changes are not possible. Nevertheless, the OEM can still exchange legitimate devices with infected ones. The attack vectors for this stage are listed in Table 2.7.

2.2.8. End Users

End users are considered untrusted. However, malicious manipulations are typically not an attack vector deployed at this stage of the supply chain. Nevertheless, end users can reverse engineer the chip, steal the IP, and extract secret keys.

2.2.9. The Cooperative Scenario

The attack models considered so far have been analyzed as individual stages with the assumption that no cooperation between multiple malicious entities exists—as typically assumed in the scientific community. However, it stands to reason that the inclusion of hardware Trojans is a complex and tedious process, especially if the HT should exert a high-impact and targeted attack. Therefore, a cooperative attack might be even more likely than just having a single, isolated malicious entity. One example of Trojan insertion could look like follows:

1. IP owner: An in-house malicious employee shares knowledge about what chip is being produced (e.g., processor) and for what purpose, i.e., the exact intended application domain (=control unit of an airplane).
2. External design house: The external design house inserts a design-dependent Trojan with the payload blocking the execution of instruction (=denial of service attack). The trigger is configured to listen to an external antenna.
3. Assembly: an external antenna is mounted within the package.
4. End user: a malicious employee (e.g., of an airplane construction company) inserts the infected processor into the target system (e.g., a specific airplane).

This attack scenario involves many untrusted parties across the supply chain. Thus, its implementation requires a high amount of planning, investment, and cooperation. However, its outcome might become more likely if untrusted parties are placed alongside the supply chain instead of having to design and insert a Trojan in a single step.

2.3. Threat Evaluation

The presented analysis in this chapter results in a clear message: the contemporary microelectronics supply chain is riddled with vulnerabilities. To provide more focus on high-priority vulnerabilities, we have introduced the effort and impact measures. Based on these measures, we can gather the following observations:

- A trusted environment can include low effort HT insertion, but typically results in low impact Trojans.
- An untrusted environment results in high-impact Trojans, specifically if direct access to the target design is provided. Thereby, it does not matter if the design and insertion of the Trojan is manual or automatic.
- All HT insertion scenarios in an untrusted foundry result in high-impact Trojans with a relatively low insertion and design effort.
- The lowest-effort and highest-impact HTs can be inserted by an external, untrusted design house operating on the RTL level.

The analysis confirms that external design houses and foundries pose the greatest threat within the microelectronics supply chain, and therefore, the primary focus of protection and detection research should be directed towards these supply-chain stages.

2.4. Open Challenges

- What effort, tools, and knowledge are required to insert a design-specific (class-1) hardware Trojan at different stages and abstraction levels in the supply chain?
- How to secure the hardware design while in the hands of external parties?
- How likely is a cooperative attack scenario, and how to protect against it?

3. The Untrusted Design House and Foundry

One of the most prevalent attack scenarios within the HW supply chain is the *malicious external design house and foundry* scenario. The reason that this scenario is frequently discussed within the scientific and industrial community is that most IP owners have to outsource parts of the design and fabrication services to third parties to uphold their business model. This reliance is driven by multiple factors, including tight time-to-market deadlines, lack of necessary skills, and exceptionally high construction and operational costs of semiconductor foundries [40; 41; 1]. As these third parties are often placed in off-site locations spread around the globe without any form of *assurance of trust* in the design and production process, a malicious modification, i.e., an HT, is a viable attack vector. Note that compared to other external entities, such as the assembly facility or the end user, both the external design house and the foundry receive the asset—the design files—in a form that is still "easily"[1] modifiable before it is hard coded into silicon.

In this chapter, we take a closer look at the conditions that enable an HT attack while the critical asset is in the hands of external parties. Section 3.1 summarizes the capabilities of the external parties. Section 3.2 introduced the concept of reverse engineering. Finally, Section 3.3 outlines the open challenges.

3.1. The Attack Model: Assumptions

The attack scenarios of an untrusted design house and foundry have been discussed in Section 2.2. The following is true for both malicious entities:

- The entity is considered untrusted.
- The entity receives full access to the design. The external design house receives either the RTL or gate-level design. The foundry receives the final layout.
- The entity operates without any restrictions or control by the legitimate IP owner.
- To insert a design-specific hardware Trojan (=class-1 Trojan as per Section 2.1.2.4), a certain amount of RE effort is required.

As we can see from the assumptions above, an important factor that determines the nature of the inserted Trojan is reverse engineering. Therefore, to understand the challenges and opportunities in Trojan detection and protection, we need to take a deeper look at the entire RE process.

[1] Compared to modifying the chip after the fabrication is done.

3.2. Reverse Engineering

Hardware RE is defined as the extraction of a set of specifications for a hardware design by someone other than the original design (IP) owner [42]. Thus, RE has historically been seen as a malicious act of significant relevance to governments, the military, and the industry. However, RE can also be utilized as a tool to check if the chip adheres to the intended specification, i.e., to exclude an infection with hardware Trojans or, simply, bugs. Evidently, in the context of malicious external entities, RE is used as a vehicle to decompose and understand the chip's functionality to insert a stealthy, design-specific hardware Trojan.

3.2.1. Flow

The primary objective of RE is to obtain an abstraction level of the asset that can be further analyzed and manipulated. In order to bring the asset to the most favorable abstraction level (ideally to its original specification), the attacker must invest a certain amount of Reverse Engineering Effort (REE). As the abstraction level of an asset decreases, a greater amount of REE is required to fully understand the design and insert C1HTs. The REE in each stage of the supply chain depends on various factors, including design complexity, level of obfuscation, availability of tools, human resources, financial resources, time, and others. Table 3.1 lists some of the available academic and industrial tools that only partially automate the RE process, which still requires manual pre- and post-processing steps. A fully automated, non-destructive, and zero-fault RE is not yet available. Furthermore, due to the sequential nature of the RE workflow, as illustrated in Figure 3.1, the correctness of reverse engineering an asset in one abstraction level depends on the accuracy of the RE process in the previous (lower) abstraction levels.

Asset	Untrusted entity	REE	Technology knowledge needed	Tools
Device	End User OEM	5	Yes	🟥 Delayering and imaging tools [43; 44] TAEUS, ScanCAD
Chip	Assembly	4	Yes	🟥 Delayering and imaging tools [43; 44] ICWorks [45], ScanCAD, ChipJuice [46], TAEUS 🟩 Degate [47]
Layout	Foundry External design house	3	Yes	🟥 NETEX 🟩 ReGDS [48]
Netlist	External design house	2	No	🟩 HAL [49], DANA [50]
RTL	External design house	1	No	🟩 VeriIntel2C, V2C

Table 3.1.: RE tools: 🟩 academic or 🟥 industrial.

In the following sections, we will discuss different RE techniques performed by the untrusted entities in the supply chain, on different abstraction levels of the asset, starting from PCB/board level down to the netlist level. As shown in Figure 3.1, in the process of RE, the asset will change its format, however, this time in the opposite direction. A systematic study of the challenges for fully automated RE on different abstraction levels is presented in [51; 52; 53].

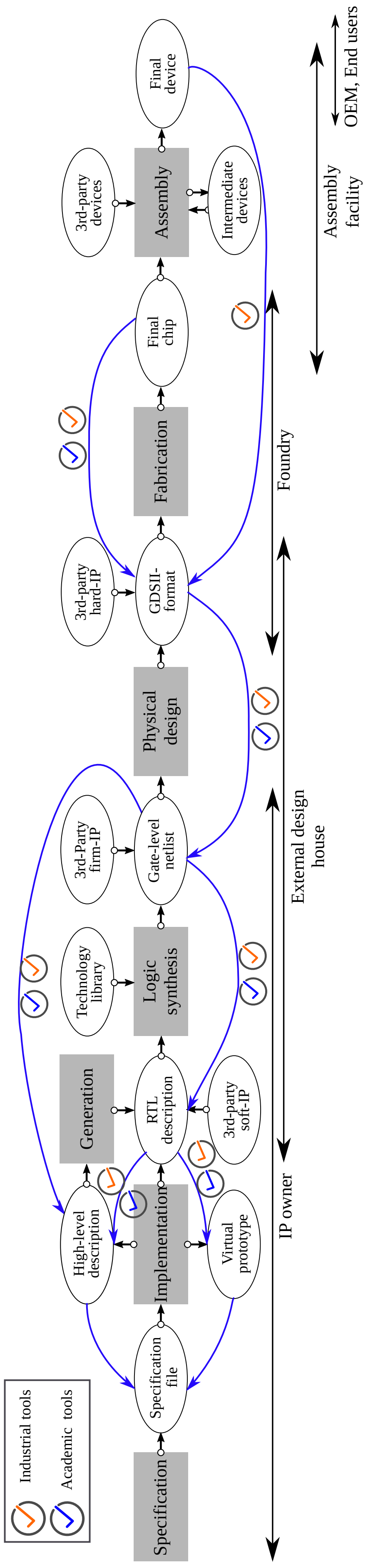


Figure 3.1.: Reverse engineering performed on different forms of the asset.

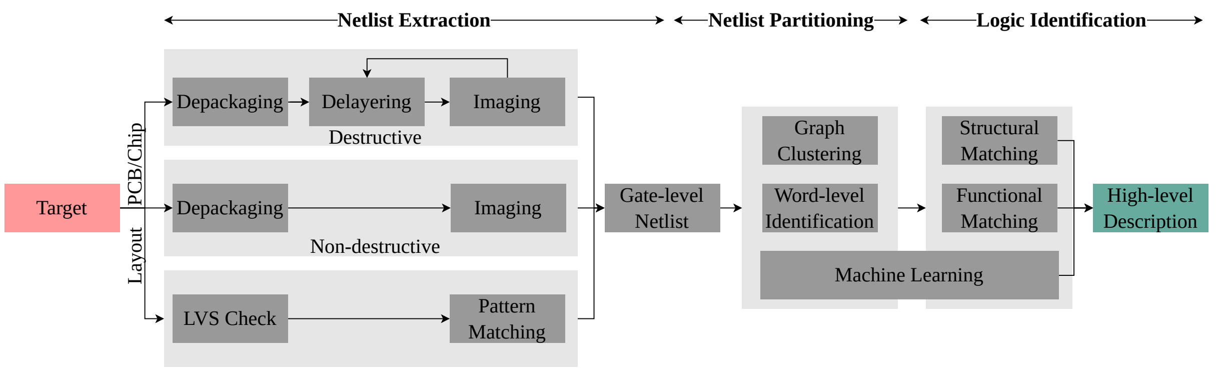


Figure 3.2.: An overview of the RE process on different abstraction layers of the asset.

3.2.1.1. PCB-level RE

The goal of PCB-level RE is to identify the components on the board (e.g., CPUs, memories, and communication ports) and their interconnections to determine the system's functionality and ultimately reconstruct the schematic of the board. This information can be used to clone the device or identify areas where potentially malicious features may be added. Integrated Circuit (IC) markings, manufacturer's logos, or die markings can easily identify some standard off-the-shelf components mounted on the PCB. In contrast, semi or fully customized circuits are typically not marked, and therefore more difficult to identify. Once the IC marking of a component is identified, the component's data sheet, if publicly available, can uncover detailed information about its functionality. After identifying the PCB components, it is necessary to determine the PCB type (e.g., single-sided, double-sided, or multi-layered).

PCB analysis can be destructive or non-destructive. A destructive RE process begins with PCB imaging to find the location and orientation of the components identified on the outer layers in order to remove them. The process continues with solder mask removal (i.e., desoldering) to expose the copper traces on the top and bottom layers. In a multi-layered PCB, the board is then delayered to access the inner copper layers. Images of each layer are taken to determine its composition (vias, connectors, and traces) and thickness [54]. Destructive PCB RE challenges and limitations are similar to those presented in Section 3.2.1.2, including accumulated errors and damage during desoldering and delayering (e.g., broken traces and disconnected vias), low-quality images, and others. Non-destructive PCB-level RE is performed using X-ray computed tomography (X-ray CT) which eliminates the delayering process and speeds up the PCB imaging time [55]. By using X-rays, a radiograph of the object is produced, which shows its composition, thickness, and any defects that may be present. An object's internal structure is visualized without the interference of overlayer and underlayer structures. The images captured during destructive delayering or non-destructive X-ray imaging are analyzed, and a PCB layout netlist is constructed. The layout could ultimately be converted into board schematics using commercially available software. Although impressive, the results depend heavily on the construction features of the PCB. In addition to high machine costs, the non-destructive method is further limited by the field of view of the X-ray system, which can make it difficult to obtain high-quality X-ray images of the entire PCB area [54; 56; 57]. Tear Apart Everything Under the Sun (TAEUS) is a service provider that offers PCB and chip-level RE to discover infringements within a product. ScanCAD is another service provider that uses destructive and non-destructive methods for PCB RE to generate schematics of the board.

During the manufacturing process, PCBs can be subjected to malicious modifications. A malicious modification is usually detected by comparing the reconstructed PCB schematic with a golden PCB schematic, which is an image taken from a PCB manufactured by a trusted and authorized source. However, since this report focuses on malicious modifications at the IC level, we will not examine further attacks and countermeasures on the PCB level [58; 59; 60].

3.2.1.2. Chip-level RE

The goal of chip-level RE is to identify the internals of the chip and ultimately retrieve its functionality at a higher level of abstraction. Chip-level RE consists of two phases, physical RE and functional RE, as illustrated in Figure 3.2. Physical RE extracts the gate-level netlist from the physical chip, while the subsequent functional RE understands its functionality and specification. This section takes a closer look at the techniques for physical RE. Functional RE is described in the following sections.

As with PCB-level RE, chip-level RE can also be destructive and non-destructive. A typical chip consists of a die, a lead frame, wire bonding or solder bumps, and a molding encapsulant. Chips are packaged with ceramic or plastic materials, using wire bond or flip chip packaging techniques. Destructive RE of a packaged chip involves several steps [45; 57]. The chip is first depackaged (or decapsulated) to recover the die using chemical, mechanical, laser ablation, or Focused Ion Beam (FIB) methods. The die is then cleaned to prevent dust-related artifacts from affecting any subsequent stages. Methods which achieve high material removal accuracy and affect a specific controlled area (e.g., plasma FIB, laser ablation) are expensive and require a high level of operational skills. To acquire images of all chip layers, delayering and imaging are performed alternately. The number of layers, their material and thickness, vias, and connectors are identified by cross-sectional imaging using Scanning Electron Microscopy (SEM), Transmission Electron Microscopy (TEM), FIB, or Helium Ion Microscopy (HIM). Due to its wide availability and reasonable cost, SEM is the most commonly used tool for IC imaging. The delayering process is then performed in multiple stages, usually by combining wet/plasma etching, grinding, and polishing [61; 62]. Due to variation in chip thickness caused by manufacturing process variations, it is best to have one die for each delayering stage. Mechanical equipment used for chip delayering includes: semi-automated polishing and milling machine, CNC milling machine, ion beam milling machine, laser, and others. To recreate the chip, a number of high-resolution images of each layer are taken, stitched together, and analyzed to recover the circuit [63; 61]. The circuit analysis includes automated and manual steps. An automated image recognition software is used to identify the cells from the standard cell library and obtain the transistor-level netlist. Extraction of gates by defining the functionality of the identified standard cells is a non-automated process, usually as a result of the unavailability of standard cell libraries. To detect stitching errors and imperfections caused by the image recognition process, design rule checks are performed [45]. After the entire area is processed, the gate-level netlist is obtained. These steps are followed by a functional interpretation of the results, as described in Section 3.2.1.4.

Although destructive techniques for chip-level RE are still the most prevalent, non-destructive approaches also exist. Non-destructive chip-level RE is performed using X-ray computed tomography (X-ray CT) which eliminates the delayering process and speeds up the PCB imaging time for the upper metal layers of an IC. X-ray CT can resolve features in ICs up to 14.6 nm in size [64]. However, it is associated with significant overhead in the image acquisition time frame. This method is not yet perfect, and it does not provide as much information about the chip's internals as invasive techniques. Future developments may make it a significant non-destructive RE tool. Photon emission microscopy can also help in RE by probing and decoding IC functionality in near-infrared spectra, but cannot be exploited for full-scale RE. Scan chain RE, on the other hand, is a non-destructive RE method that exploits the scan chains inserted into the device for production test to allow easy access to the circuit internals. This technique gives a netlist approximation that is logically equivalent to the netlist [65].

The process of destructive chip-level RE is complex, expensive, time-consuming, and only partially automated. Each step in the process requires a high level of precision and operational skills. Images might be imperfect, cell image recognition could be erroneous, and consequently stitching for nanoscale node technologies becomes challenging. Therefore, a certain error rate is expected in the final netlist. In practice, the error rate should be less than 1%. Generally, technology libraries are proprietary, which could be a problem for RE performed by OEMs and end users.

There is limited support from industry and academia for chip-level RE. Degate [47] is an open-source framework for reverse engineering ICs. The tool generates a gate-level netlist after receiving layer images and a standard cell library, but it does not offer support for further netlist analyses. There are several commercial service providers that offer tools for chip-level RE. ChipWorks (now merged with Techinsights) developed ICWorks [45], which provides destructive techniques for netlist extraction and subsequent analysis. Texplained developed ChipJuice [46], an automatic RE tool which can process layer images and, if given the standard cell library, reconstructs the netlist.

3.2.1.3. Layout RE

External design houses and foundries with access to the GDSII layout files of the design can perform a layout RE without the need to go through the tedious and expensive RE steps described in the sections above.

ReGDS [48] is a layout RE framework that reconstructs the transistor-level netlist based on the GDSII layout files and the technology library, and after identifying the logic gates, recovers the original gate-level netlist. The framework first uses Layout vs Schematic comparison (LVS) tools to extract the transistor-level information, and subsequently uses graph matching algorithms to facilitate logic gate identification.

3.2.1.4. Netlist RE

As mentioned in Section 3.2.1.2, chip-level RE consists of two phases, physical RE and functional RE. While physical RE extracts the netlist from a given chip, the goal of functional (netlist) RE is to obtain a high-level description of the chip's functionality for further interpretation and validation. The approaches described in this section assume that the gate-level netlist is previously extracted using the mechanisms described in Sections 3.2.1.2 and 3.2.1.3. A systematic description of the methods and algorithms for netlist RE is given in [53].

The netlist RE itself consists of two phases, netlist partitioning and logic identification (Figure 3.2). Netlist partitioning is performed on the flattened netlist, which does not contain any information regarding the hierarchy, boundaries, and functionality of modules. The netlist is broken down into smaller structures (i.e., submodules), which are then analyzed separately. Several approaches for netlist partitioning exist, including word-level structure identification [66; 67; 68] and graph clustering [69; 70; 71; 72].

Logic identification is achieved through syntactic or semantic analysis. Syntactic analysis is also referred to as structural or topological matching, while semantic analysis is referred to as functional or behavioral matching. The success of both approaches depends on the completeness and correctness of a pool of known design components, i.e., a component library or golden library. Structural matching compares the structure of each submodule with the library modules using the principle of subgraph isomorphism. Alternatively, functional matching performs formal equivalence checks between the partitioned submodules and the library modules. Logic identification might use precise methods, which require a perfect match, or heuristic methods, which require an approximate match. Moreover, logic identification can be performed on data-path components as well as on Finite State Machines (FSMs). In practice, combining different methods can increase the chances of extracting the complete netlist functionality.

Analysis of gate-level netlists to obtain a higher level description was first presented in [73; 74; 75]. Authors in [75] describe manually driven techniques for identifying combinational structures and present a semi-automated library-based module recognition algorithm. This study, however, focuses on reverse-engineering circuits (i.e., ISCAS-85) that are custom built and considerably smaller and simpler than today's complex chip designs. Automated functional logic identification techniques are presented in [76; 77; 66; 78; 79; 80]. Structural logic identification, on the other hand, is presented in [81; 82; 77; 83; 71; 69; 84; 72]. Furthermore, identification of datapath components is presented in [75; 66; 68; 81; 78; 71; 84; 50], while identification of FSMs is presented in [82; 77; 83; 71; 85]. The authors in [68] use a combination of structural and functional matching algorithms. They first published the feasibility of reverse engineering a realistically sized gate-level netlist. The identified submodules, however, are small and belong to the combinational datapath (e.g., register files, adders, counters), excluding the random sequential logic.

Machine Learning (ML) has the potential to increase the RE efficiency and enhance its automation, not only on netlist abstraction level, but also during image analysis in PCB and chip-level RE [86; 87; 88]. However, only recently has there been an increase in the number of ML-based RE studies. The limited size of the training circuit dataset may account for insufficient research in this area. The first ML-based RE attempt utilizing Convolutional Neural Networks (CNN) was described in [89; 90]. In [89], multipliers and dividers in larger circuits that contained additional arithmetic operations were detected with a relatively high degree of precision for single-class problems. Recently, authors in [91; 92] introduced GNN-RE, an ML-based platform that leverages Graph Neural Networks (GNNs) to analyze flattened netlists, identify module boundaries, and classify submodules based on their structural and functional features.

Table 3.2 summarizes the main characteristics of the netlist RE approaches mentioned above. It is evident that most of the techniques are tested with small circuits and their accuracy depends not only on the correctness and completeness of the component library, but also on the assumption of perfect netlist extraction and subsequent partitioning. In [69], a high number of incorrect matches are reported due to the incorrect partitioning of the netlist. Hiding of module boundaries and hierarchy information after synthesis makes the process of netlist reverse engineering challenging. Additionally, synthesis optimizations typically result in gate-level structures that are interconnected and shared, and thus difficult to be easily translated to their high-level counterparts. Perfect matching approaches [76; 68; 79] highly depend on the existence of a library of known components, where a submodule with identical design exists, preferably built with identical synthesis options and cell library. This assumption, however, is unrealistic. Approximate algorithms [69; 84; 72], on the other hand, are computationally more efficient and less constrained by the size, technology, and accuracy of the netlist.

Netlist RE is not fully automated, and the human element plays an important role [52]. HAL [49] is an interactive open-source framework for graph-based netlist analysis. Using HAL, time-consuming and complex RE processes can be automated. HAL partially recovers module boundaries and hierarchy, but it cannot recover a full complex netlist automatically. The recovered modules can be further manually inspected in details. DANA [50] is a plug-in to HAL for identifying high-level register structures in flattened gate-level netlist. It provides guidance for human analysts by structuring and condensing the otherwise incomprehensible sea of gates.

Papers	**Method**	**Circuit type**	**Benchmark size**	**Accuracy**[†]
Doom [73]	Functional matching	Standard cells	<100	100%
Hansen [75]	Mostly manual	Datapath modules	3.5K	N/A
Shi [82]	Structrural matching	FSMs	N/A	N/A
Li [76; 66]	Functional matching	Communication circuit library	<1000	N/A
Subramanyan [68]	Structural and functional matching	Datapath modules	15K	75%
Meade [71]	Structural matching	Control registers	12K	80-100%
Soeken [79]	Functional matching	Datapath modules	3.5K	N/A
Werner [72]	Structural fuzzy matching	Crypto modules	100K	85-95%
Albartus [50]	Dataflow analysis	Register structures	100K	90%
Dai [89]	ML-CNN	Datapath modules	N/A	45-90%
Alrahis [91]	ML-GNN	Datapath modules	3.5K	99%

Table 3.2.: Comparison of different netlist RE methods. †The numbers in this column should not be compared directly, since each work addresses different RE method, circuit type, and approximation.

With the emergence of open source hardware, when full or partial knowledge of the design is made public, RE becomes easier [93]. It has been noted previously that netlist partitioning can lead to faulty or incorrect submodules, which can make subsequent logic identification significantly more challenging, as errors in the submodules make formal-based analysis ineffective. If, however, the entire design is based on an open source design, which is easy to integrate into the golden library, the partitioning can generally be resolved with little to no error. With partially open source designs, separating the open source parts from unknown or proprietary hardware is simpler, greatly reducing the effort of partitioning the remaining part of the design. On the other hand, open source designs make it more challenging to protect commercial IP solutions if they are built on top of open source, easily identifiable hardware.

3.3. Open Challenges

- How to quantify complexity, cost, and effort for RE?
- How to quantify the success criteria and the amount of retrieved information of the RE process?
- How to implement automatic, non-destructive, and zero-fault RE on every abstraction level of the asset?

Part II.

WP2: State of the Art in Protection Mechanisms and Detection Methods

Part I depicted the different formats the asset can take. For every format and step in the supply chain, the vulnerabilities are listed and classified. For those vulnerabilities, common state-of-the-art attacks are elaborated on.

In this part, the list of possible attacks on the asset is evaluated using known detection and protection mechanisms. In this evaluation, gaps in the defense of the supply chain are identified. First, Chapter 4 lists the passive defense mechanisms, the *detection* procedures. The detection methods are used to find vulnerabilities, to support the designer in the removal of malicious modifications from the asset. Second, the active defense mechanisms, the *protection* procedures, are listed in Chapter 5. These methods implement changes to the asset to prevent manipulations by adversaries. Finally, in Chapter 6, the lists of state-of-the-art detection and protection mechanisms are evaluated on their applicability and security assurance. Additionally, the defense mechanisms are laid out along the supply chain to identify gaps.

4. Detection of Malicious HW Manipulations

In this chapter, we summarize the known detection mechanisms that can be used to identify malicious modifications in the asset. We assign the mechanisms into one of two classes: pre-silicon and post-silicon. Some mechanisms are designed for analysis before the asset is manufactured. These focus on the asset at the design stage. The post-silicon mechanisms focus on the manufactured design. As no formal description of every manufactured asset is given, fewer algorithms are provided for the post-silicon phase. The taxonomy and classification for the detection schemes are illustrated in Figure 4.1.

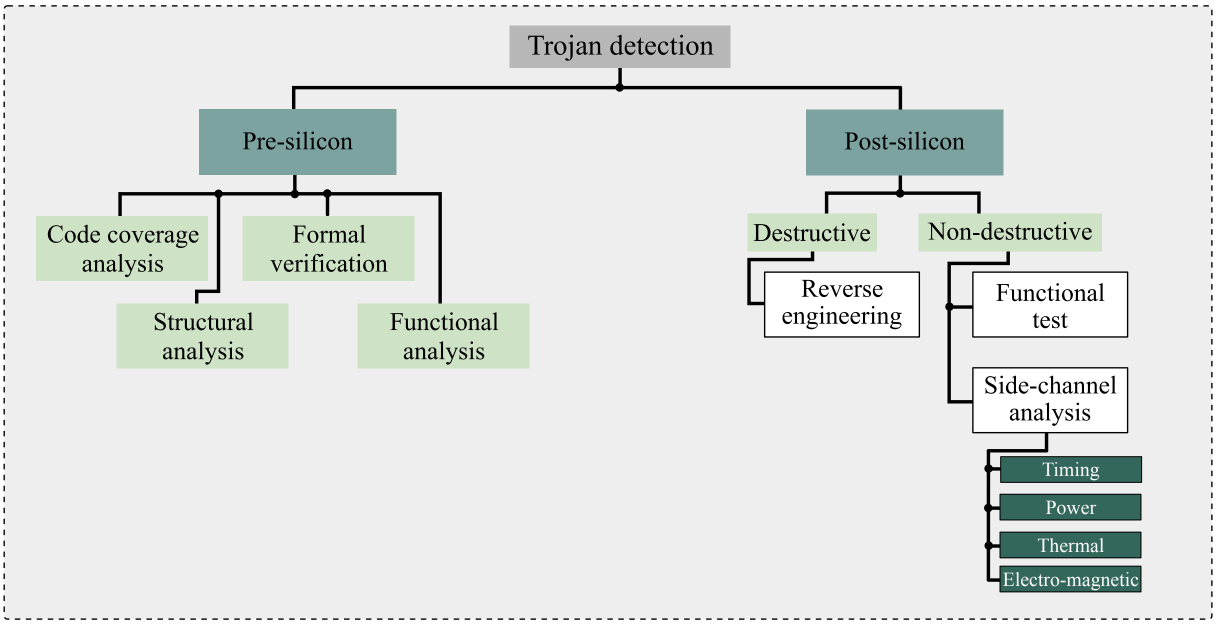


Figure 4.1.: Taxonomy of hardware Trojan detection mechanisms [1].

4.1. Pre-silicon Trojan Detection

4.1.1. Code Coverage Analysis

Code coverage describes the percentage of code executed during functional verification [1]. For common functional verification, this is done to develop new test suites that cover the non-executed parts of the code.

In hardware, this analysis can be used to find non-activated components of the hardware during simulation using functional tests. Functional tests might find a malicious hardware modification but cannot give complete assurance if not all possible input-output sequences are tested. Unfortunately, the complexity of modern hardware designs limits the feasibility of functional testing for Hardware Trojan (HT) detection due to a very high number of required tests.

Common test suites do not consider security, but test parts of the chip's functionality. The tests can be applied to the design at different stages of the supply chain. Simulation tools or virtual prototypes allow the execution of the test suites on the individual abstraction layers. Virtual Prototypes, Register-Transfer Level (RTL), and netlist simulators [15] enable the application of different input patterns, alongside the comparison to expected outputs. Thus, the designed applications can be executed on the designed hardware at multiple steps of the supply chain.

Many developed detection techniques focus on identifying HT triggers (see Section 2.1.1). A common property of Trojan triggers is that common testing should not activate them. This means the Trojans have uncommon (rare) trigger conditions. This property can be exploited to design a more focused HT detection process. For example, functional tests can identify areas in the Intellectual Property (IP) that are not activated by the tests. These areas are marked as potential trigger sources. Once identified, input combinations that activate the parts are determined [94; 95]. These input combinations are used for additional functional tests. The observation of dramatic changes in the input-output behavior is used to identify HTs. A formal description of the hardware is required, thus reverse-engineering is a requirement for most HT identification techniques. The tests can be conducted on all possible abstraction levels and the manufactured chip. These methodologies have been shown to have detection rates above 90% for a set of benchmarks [95].

4.1.2. Formal Verification

Traditionally, formal methods have been applied to software for finding security bugs and increasing the test coverage [1]. Many techniques can be reused for hardware to prove the trustworthiness of the system, and thus, the absence of hardware Trojans.

4.1.2.1. Reusing Existing Verification Techniques for Security

As hardware security gains more focus, researchers elaborate on whether existing verification techniques can be reused to prove security properties or identify malicious modifications and vulnerabilities. For example, assertions are commonly used to verify the functional correctness of a hardware description. The assertions are integrated into the description and can be verified using simulations or model checkers. Simulations prove the properties of the test set, while model checkers can prove them for all possible input patterns. This verification technique can also be reused to detect hardware Trojans [96]. Reusing existing techniques does not result in additional work for security verification.

Other techniques, such as [97], reuse hardware rewriting techniques to identify information leaking HTs. The resulting dependency graphs are utilized to identify undesired information flows in the hardware descriptions.

4.1.2.2. Equivalence Checking

Equivalence checking is a technique that proves 'equivalence' between two descriptions of the same asset. In this context, it means that two designs exhibit the same behavior. The checks can be conducted using descriptions on multiple abstraction layers. A common practice is to show that the RTL description is equivalent to the post-synthesis gate-level netlist. This elaboration demonstrates that the logic synthesis was successful, and no illegal modifications were introduced.

Therefore, the evaluation shows that the two descriptions are functionally equivalent, and no malicious modification can be found in one, but not the other. For the elaboration, commercial [98] and open-source [99] tools can be used. Commercial tools are offered by Synopsys [98], Siemens EDA [100] and Cadence [101]. Smaller companies such as Lubis EDA [102], also offer their services to assist a designer in verifying their asset.

It is important to note that the formal equivalence check can overlook (malicious) modifications, even if the equivalence check terminates successfully. For example, undefined behavior in RTL results in "don't care" states which can be used for malicious implementations in the gate-level netlists. The undefined states in the one description lead to a misclassification of the equivalence [103]. Thus, to prevent this vulnerability, all possible internal states must be covered in the hardware description.

Most equivalence checkers only support analysis for descriptions in VHDL, Verilog, SystemVerilog, and GDSII [104]. Therefore, hardware descriptions on different abstraction levels, e.g., specification, are not yet supported. Additionally, a formal definition is required for the analysis. Therefore, physical chips need to be reverse-engineered (see Section 3.2) to a description that can be used in the equivalence check process.

4.1.2.3. Security Properties Enforcement

This section summarizes developed techniques that are not specifically designed to detect hardware Trojans [111]. The methodologies are developed to enforce properties in hardware designs, focusing on the three cornerstones of (hardware) security: confidentiality, integrity, and availability. Once the security properties are proven for a given hardware design, they can be proven again at a later stage of the hardware supply chain. The second proof shows the property is still valid, and no malicious modification was implemented that resulted in breaking the property. Some of these techniques are summarized in **Los 2**. Therefore, this report only discusses a few methodologies for each cornerstone.

Information Flow Analysis (IFA) [112; 113; 114] is a technique to prove that certain sensitive data may not be leaked to adversaries via untrusted components or output ports. If the analysis is done at every step in the asset chain, it can be proven that no hardware Trojan has been implemented that leaks the information *covered* by the IFA. In this scenario, the analysis focuses on proving the confidentiality property. IFA requires a formal description of the hardware. This means the analysis cannot be conducted on a manufactured chip, if it is not reverse-engineered. Additionally, no IFA techniques have been developed for abstraction layers below the gate level. IFA requires a hardware description. The integrity property can also be proven using IFA. The analysis can show that sensitive data cannot be modified by untrusted components, i.e., no untrusted data can flow to the sensitive target.

The biggest threat to the "availability" property are Denial of Service (DoS) attacks. As these attacks can be specifically designed for every hardware system, it is fairly complicated to protect against unknown hardware Trojan implementations. As an example, for GPS systems, the transmitter could be turned off, the calculations can be manipulated, and data can be replaced or removed. Some known vulnerabilities, such as uncontrolled access to boot code, can be detected by analyzing the access rights. Other vulnerabilities might be overlooked, as the designer cannot be certain of what he is looking for. Incorrect behavior can only be avoided by enforcing the desired behavior, which needs to be specified for every small state of the hardware system.

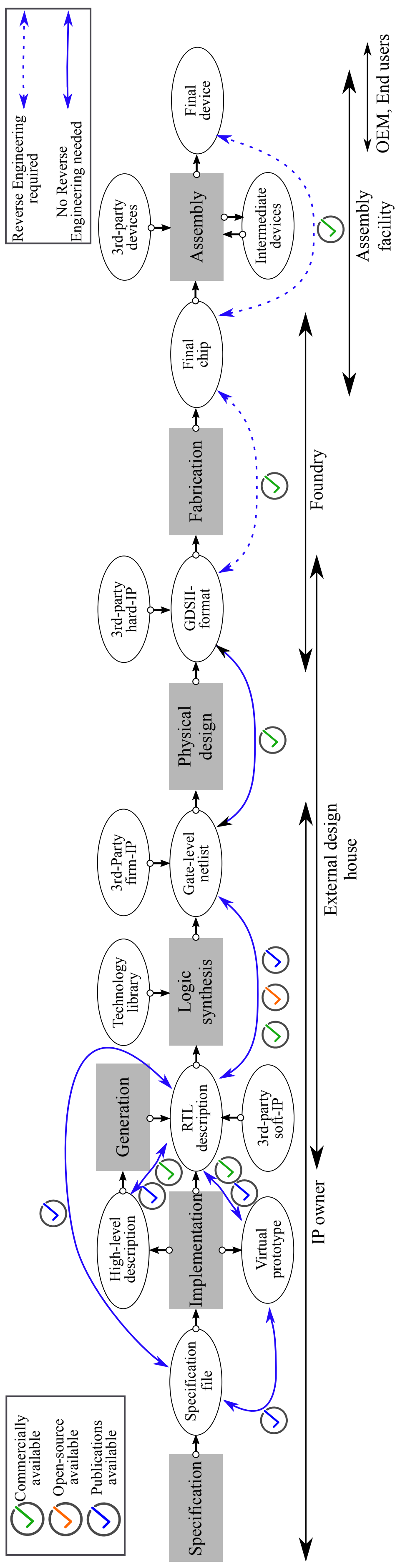


Figure 4.2.: In what steps of the asset chains can equivalence checking be applied?

Table 4.1.: Available equivalence checking techniques for different formats of the asset. Two different descriptions of the same asset are illustrated as description type A and B in the table.

Description type A	Description type B	Are commercial tools available?	Are open-source tools available?	Are publications available?
Specification to System Level				
UML	SystemC TML	No	No	Yes [105]
Specification to Register Transfer Level				
UML	RTL	No	No	Yes [106]
System Level to Architecture Level				
SystemC	nML + PDG	No	No	No
System Level to Register Transfer Level				
SystemC or C/C++	Verilog or VHDL	Yes [108] [98]	No	Yes [107] [109] [110]
Architecture Level to Register Transfer Level				
nML + PDG	Verilog	No	No	No
nML + PDG	SystemVerilog	No	No	No
nML + PDG	VHDL	No	No	No
Register Transfer Level to Gate Level				
Verilog	Verilog	Yes [98][100]	Yes [99]	Yes, many
SystemVerilog	Verilog	Yes [98][100]	Yes [99]	Yes, many
VHDL	Verilog	Yes [98][100]	Yes [99]	Yes, many
Gate Level to Layout Level				
Verilog	GDSII	Yes [98]	No	No
Layout Level to Fabricated Chip				
GDSII	Fabricated chip	Reverse engineering to a formal description required.		
Fabricated Chip to to Final Device				
Fabricated chip	Final device	Reverse engineering to a formal description required.		

Many security engineers utilize a known list of vulnerabilities: Common Weakness Enumeration (CWE) [115]. CWE lists known vulnerabilities commonly found in software and hardware designs. Researchers constantly work on the automatic processing of the list to identify the "common weaknesses" in the asset. The vulnerabilities all endanger at least one of the three cornerstones of security. Enforcing the properties at each step of the supply chain would allow the identification of malicious manipulations that implement any of the listed weaknesses.

4.1.3. Structural Analysis

Code coverage analysis exploits the unlikelihood of HTs triggered by identifying inactive hardware components during functional testing. Structural analysis conducts this search for Trojans in a static fashion. The hardware design description is analyzed statically to identify possible HT triggers.

Some methodologies use machine learning to identify input patterns that might trigger a potential HT [116]. The methodology processes a set of hardware descriptions as a training set.

Reinforcement learning is used to generate a subset of the desired input patterns. A variety of detection methodologies utilize that many HTs share certain architectural structures. Thus, pattern recognition techniques can be used to identify hardware Trojans in a hardware description. A set of known Trojan benchmarks is used to extract patterns to form a score-based pattern recognizer that can classify unknown netlists as malicious or benign [117]. Similarly, some frameworks develop graph neural networks and gradient-boosted tree classifiers to identify malicious modifications in hardware descriptions [118; 119]. These classifiers depend on the completeness of the benchmark set to detect unknown Trojans in new hardware designs.

Additional research focuses on the identification of hardware Trojans in 3rd Party Intellectual Property (3PIP). Some published works reuse software virus detection techniques to detect HTs. Standard features of Trojan triggers are extracted from a dataset of features [120]. The feature analyses result in a high detection rate and are operated on the gate level. Similar to the Trojan trigger identification techniques for in-house IP, additional techniques are developed for 3PIP. The Trojan triggers are identified by locating hardware regions with a low likelihood of being activated [121]. An Automatic Test Pattern Generation (ATPG) tool is used to identify signals that are "hard-to-excite" and/or propagate.

Gaikwad et al. developed a machine-learning framework that identifies suspicious nets in a hardware description. The advantage of the framework VIPR [122], is that it doesn't require a set of trusted benchmarks for the training set. This allows an assisted identification of hardware Trojans in 3PIP without having a trusted copy.

4.1.4. Functional Analysis

The main difference between logic testing and functional analysis is that functional analysis utilizes random input patterns during simulation. The output is observed to identify any unusual behavior.

These techniques do not utilize the property of unlikely Trojan triggers, but aim to use heuristics to reduce the required test space. Some approaches utilize randomized methodologies, while others apply probabilistic approaches. The reduced and specified test set is used again to observe dramatic changes in the expected input-output behavior of the asset [123; 124; 125].

In addition to the random approaches, software testing techniques such as fuzzing are reused for hardware [126]. The hardware description is converted to a software description to allow the application of software fuzzing techniques. Semi-valid inputs are generated to result in unwanted behavior of the hardware, such as a DoS attack. The approach has been shown to identify unwanted behavior of the hardware, thus also covering the identification of DoS HTs.

4.2. Post-silicon Trojan Detection

The mechanisms used to detect hardware Trojans in a manufactured chip are discussed below.

4.2.1. Reverse Engineering (Destructive)

Destructive techniques require a manufactured chip to be depackaged and analyzed layer by layer. Each layer is scanned, before being etched off to gather images of the following layers. The resulting digital copy of the depackaged chip can be compared to the design description from the pre-silicon stages. A detailed description of the entire Reverse Engineering (RE) process is available in Section 3.2.

The RE process may take up to several months, depending on the complexity of the chip [1]. Additionally, the analysis is costly and gives only information about the single depackaged chip. Thus, no security guarantee can be given across all fabricated chips—unless all are thoroughly reverse engineered. However, as RE is, in most cases, a destructive procedure, the analyzed chips cannot be used after the check [1]. A random sample survey could be conducted to allow a high detection coverage.

As discussed in Section 3.2.1.1 and Section 3.2.1.2, RE can also be conducted on PCB- and chip-level.

4.2.2. Functional and Fault-Model Derived Tests

In this subsection, we differentiate between functional tests that allow the detection of errors implemented by a faulty design and fault tests that enable the identification of manufacturing faults that alter the intended functionality. Both types of tests have been shown to facilitate the identification of hardware Trojans.

4.2.2.1. Fault-Model Derived Tests

Fault tests can be automatically generated for a design using ATPG. Different fault models are used to identify fabrication-induced faults, such as the stuck-at fault model, stuck-open fault model, bridging faults, delay fault model, and IDDQ model. Fault tests are a common practice [127]. ATPG tools generate the required test pattern set to identify faults in the manufactured chip. Most IP owners use commercial tools to generate the required test patterns, such as Synopsys Testmax [128]. As ATPG-guided tests are meant to detect faulty chips, fault-model derived testing is only applicable for the post-production asset.

Moreover, the standard stuck-at-fault test allows the identification of gates that are either stuck at a logic 0 or 1. This model requires the generation of a test pattern that allows forwarding the actual value at a gate to observe the value at the primary (observable) output of the chip. Therefore, this technique can be used to identify possible leakages that might allow the forwarding of sensitive data to untrusted outputs for the generated input patterns.

4.2.2.2. Functional Tests

In the following, we focus on standard testing procedures used to verify the chip's correct operation after fabrication. The tests include the functional tests discussed in Section 4.1.

Functional and fault tests might find a malicious hardware modification but cannot give complete assurance if not all possible input-output sequences are tested. Due to the complexity of modern hardware designs, such a high number of tests is not feasible.

Additionally, an adversary might design the hardware Trojans to avoid being identified with the standard fault and functional tests. In some cases, a Trojan might not change the functionality of the original circuit, but transmit information via nonfunctional means, such as an antenna [1]. Thus, specific techniques are required to identify such threats.

Less information about the design is known for 3PIP. Thus, special functional tests are designed to detect possible malicious modifications. For manufactured 3PIP, additional approaches have been suggested that try to detect the modifications despite the lack of information. In [129], the authors illustrate a method for the identification of malicious modifications in manufactured 3PIP. The designer buys two units of the same chip and compares the input-output patterns. The technique shows promise in the identification of abnormalities.

4.2.3. Side-Channel Signal Analysis

These post-silicon detection techniques identify hardware Trojans by measuring physical parameters, such as delay, power, and radiation [1]. Most side-channel detection schemes require a "golden Integrated Circuit (IC)", a manufactured unit of the same IC that is not infected with Trojans. As any additional circuitry results in side effects such as a higher power consumption, modifications can be identified by comparing the measured parameters with the golden IC. For example, authors in [130] use the signature of a non-infected circuit's power consumption and electromagnetic dissipation to identify abnormalities in possibly infected ICs.

Researchers have shown that the identification of hardware Trojans using side-channel analysis works to some degree [1], but the high number of possible side-channels results in a low detection rate. Signatures or fingerprints need to be collected for every substructure in the IC to facilitate the identification of small abnormalities. At the same time, process variations result in a high degree of noise, which deems the detection of small abnormalities infeasible. Additionally, in most scenarios, a golden IC is not available.

4.3. Open Challenges

- How much security can equivalence checking offer?
- Is there a detection scheme that can find any possible hardware Trojan?

5. Protection Against Malicious HW Manipulations

After we discussed the detection mechanisms, we elaborate on the other type of defense algorithms, the protection schemes. Protection methodologies can prevent malicious modifications, such as hardware Trojans, in the first place. The implementation of class-1 Trojans is prevented using different strategies that all rely on the same principle, reducing the amount of information an attacker can get about the functionality of a chip. First, we discuss the most popular approach, logic locking. Afterwards, methodologies such as split manufacturing and layout camouflaging are presented.

5.1. Logic Locking

Logic Locking (LL) is a Design-for-Trust (DfTr) technique that aims to protect the **integrity** of hardware designs at different supply-chain stages and design levels. LL modifies the hardware design by introducing logic changes that bind the correct chip operation to a secret activation key. This change has two main implications. First, the Hardware (HW) design's functional behavior depends on the correctness of the key. If a correct key is applied, the design performs as intended. Otherwise, erroneous outputs are generated. Second, the added key-dependent logic induces structural changes in the design. Thus, LL "obfuscates" the hardware both functionally and structurally. Note that logic locking is sometimes referred to as logic obfuscation or logic encryption. The multitude of names is a result of the different implications of the methodology on hardware. However, the term "logic locking" has been widely accepted as the standard naming by the research community in recent years.

In this section, we take a closer look at the fundamentals of logic locking, from its security implications within the supply chain to the vast evolutionary landscape of attacks and schemes. More details on logic locking are available in [22; 131; 132; 133; 134; 135; 136].

5.1.1. Logic Locking in the Electronics Supply Chain

The inclusion of LL in the HW supply chain is visualized in Figure 5.1[1]. Here, we assume that logic locking is deployed at the gate level; however, as discussed in later sections, the methodology applies to other design levels as well. The presented flow assumes the inclusion of an external design house for the layout generation. The IP owner represents the trusted entity that intends to design and produce a legitimate chip. In this stage, RTL description of the HW design is logically synthesized to a gate-level netlist. The synthesis can be either technology-dependent or technology-independent depending on the locking software[2]. Once a

[1] Some details of the flow have been omitted for simplicity.
[2] In both cases, another synthesis round is typically deployed to further integrate any changes induced by the locking mechanism.

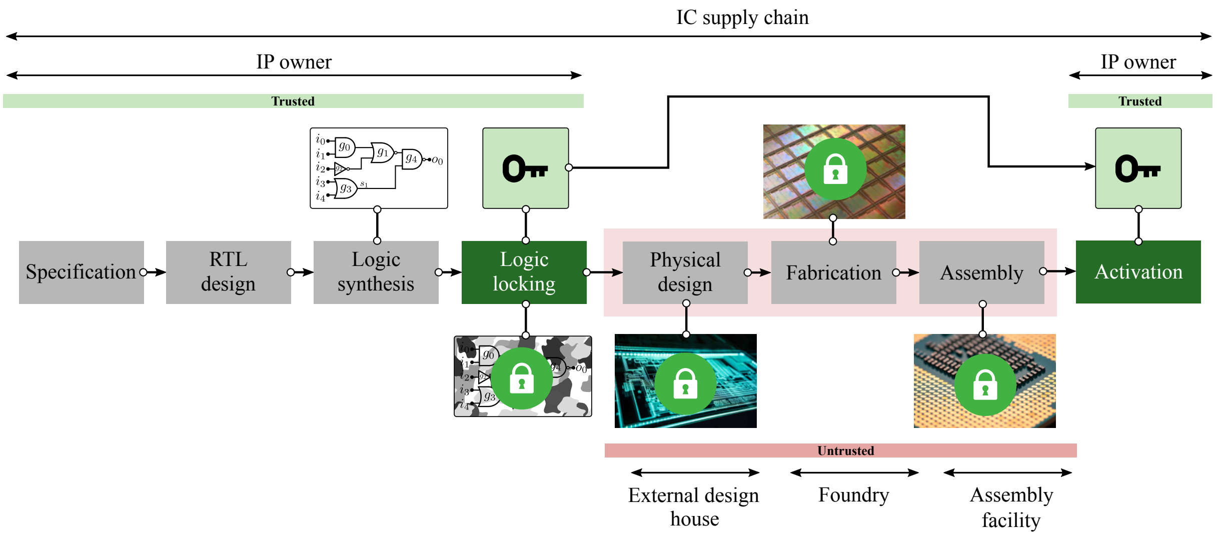


Figure 5.1.: Logic locking in the electronics supply chain.

gate-level netlist is available, LL is deployed, thereby producing a locked netlist and a secret key. The secret key remains with the legitimate IP owner. Note that the key is not required for the layout generation or any production and testing steps. Thus, the LL has no negative impact on the standard design and fabrication flow. The locked netlist is transferred to the external, untrusted parties for layout generation, fabrications, and assembly. Once the final chip is prepared, it is sent back to the IP owner for activation. The secret key is embedded into the chip through a non-volatile memory, such as flash, e-fuse, or EEPROM [137]. This flow has successfully been implemented and showcased by HENSOLDT Cyber GmbH through the production of the Made in Germany RISC-V (MiG-V)—the first fully logic-locked commercial processor [138; 139; 140].

Security implications: The main objective of LL is to *conceal* the design's original functionality from an untrusted entity. This property is known as functional secrecy [141; 142]. A side-effect of this process is the locking mechanism; the chip cannot be used without the correct key. However, this property can be achieved without comprehensive locking procedures [142]. LL protects the integrity of designs by creating a structural and functional dependency on a secret (activation) key. Therefore, the security assumption is that the secret key must be identified to successfully reverse-engineer and understand the HW design, and ultimately insert a high-impact, design-dependent hardware Trojans (=class-1 HT as per Section 2.1.2.4). Thus, finding the correct activation key is a proxy for the functional secrecy of the design. Moreover, it stands to reason that the reverse-engineering effort of an untrusted entity *increases* due to LL introducing additional logic into the design. Hence, logic locking and reverse engineering are two opposing streams.

5.1.2. Attack Model

Many key-recovery attacks on logic locking have been introduced. These attacks operate based on the following assumptions [143]:

- The attacker has access to the locked design (at some level).
- The location of the key-input pins is known.

- The deployed locking scheme is known.
- The attacker has access to an activated chip to use as *oracle* for retrieving golden input/output pairs.

This model adheres to Kerckhoffs's principle, as the only unknown is the secret key. The last assumption categorizes all attacks into Oracle-Guided (OG) and Oracle-Less (OL) attacks. The OG attack model assumes an oracle (=activated instance of the chip) is available. This is typically the case in a high-volume production with multiple production rounds where an attacker can simply acquire an activated copy of the chip on the open semiconductor market. Hereby, it is important to note that the security assumption within the OG model includes a secure, tamper-proof memory for key storage.

In the OL attack model, on the other hand, the attacker is not able to get an activated copy of the chip. This is the case in a low-volume, high-security environment where the chip is intended for high-critical applications.

5.1.3. Logic Locking vs. Reverse Engineering

A certain effort is required to insert class-1 hardware Trojans. As LL aims at protecting the integrity of HW, i.e., against malicious modifications, it effectively increases the RE effort. Unfortunately, it remains an open challenge to estimate the RE effort required to insert HTs. Therefore, the security of LL has thus far been evaluated through the retrievability of the key. Unfortunately, this property may or may not be in relation to the actual RE effort. Consequently, the effectiveness of LL in preventing Trojan insertion is far from determined.

5.1.3.1. The Metric Problem

How do we measure the security of logic locking? As previously discussed, LL aims to achieve functional secrecy. Hereby, its effectiveness is approximated through the retrievability of the key. Therefore, the majority of existing metric proposals try to capture certain structural or behavioral features of locked circuits that might suggest how effective the underlying locking policy is against certain key-retrieving attacks [144; 145; 146; 147]. This "key-centric" measurement approach stems from the difficulty of evaluating the *additional* RE effort caused by logic locking. Consequently, it becomes clear that we are far from a concrete security measure.

5.1.3.2. The Interplay of Attack Models

The evolutionary landscape of logic locking is riddled with many attack vectors, both in the OG and OL model. The OL model does not required a golden, activated, in-silicon design instance. Therefore, this attack model is always applicable and realistic. Many powerful and game-changing attacks have, however, been presented in the OG model, requiring an activated chip as reference with a secure key storage in place. Unfortunately, a secure key storage does not exist as the key can always be retrieved through some form of physical attack. This yields an interesting question: is the OG attack model, despite its inclusion of powerful attacks, relevant? The simple answer is—yes. Assuming that a secure key storage and activated chip are not available, an attacker can still resort to OG attacks by reconstructing an activated chip manually. This scenario can be embedded within the OL model. For example, the following scenario showcases a valid utilization of OG attacks:

1. The attacker receives the locked target pre-silicon design (e.g., gate-level format).
2. The attacker can compare the target design to an existing database to find the most similar design.
3. The most similar design is used as oracle (or a similar one if certain information sources suggest a different implementation).

Therefore, even if a secure key storage is not available, OG attacks can still play an important role. The core question, however, remains—how to disable a comparison to other designs, i.e., how to prevent any form of oracle-less attacks?

5.1.4. The Evolution of Logic Locking

Logic locking can be broadly classified into combinational and sequential LL. The former performs key-dependent manipulations of the combinational logic within a design, thereby (usually) having no explicit effect on the internal states of the circuit. The latter obfuscates the state space of the circuit. The vast majority of available work focuses on combinational LL, partially due to the simplicity of deploying combinational manipulations that inject considerable functional and structural changes in a design. The evolution of LL is, with some details omitted, represented in Figure 5.2.

5.1.4.1. Gate-Level LL

LL at gate level has been a prominent research stream, resulting in many proposed locking policies. In the first years of logic locking (starting from 2008), the security objectives of the proposed schemes were not entirely clear. Therefore, different schemes focused on different objectives, such as increasing functional output corruption for incorrect key values, creating strong functional interference among the inserted key-dependent gates, minimizing low-controllability signals in the netlist, and others [148; 149; 150; 151; 152; 153].

In 2015, a potent attack was introduced; the Boolean Satisfiability Problem (SAT) attack [154]. This attack type utilizes powerful SAT solvers to find a correct activation key. It turned out, all previous schemes are vulnerable to this type of attack. Therefore, a new research stream started with the introduction of SAT-based attacks with a clear security objective—SAT resilience. Example LL mechanisms that aim at SAT resilience are point-function-based locking, the insertion of SAT-unresolvable structures, and routing-based locking, among others [155; 156; 157; 158; 159; 160].

An interesting locking approach is manifested in parametric locking. This LL category aims to protect the parametric features of the design, such as power, delay, and reliability characteristics. Compared to traditional locking, parametric locking typically does not induce incorrect output behavior for incorrect keys, but rather incorrect parametric traits, e.g., incorrect power consumption and performance [161].

One drawback of traditional SAT-resilient schemes is the low corruptibility for incorrect keys. This specific trait is a key ingredient in thwarting SAT-based attacks. Unfortunately, this behavior is

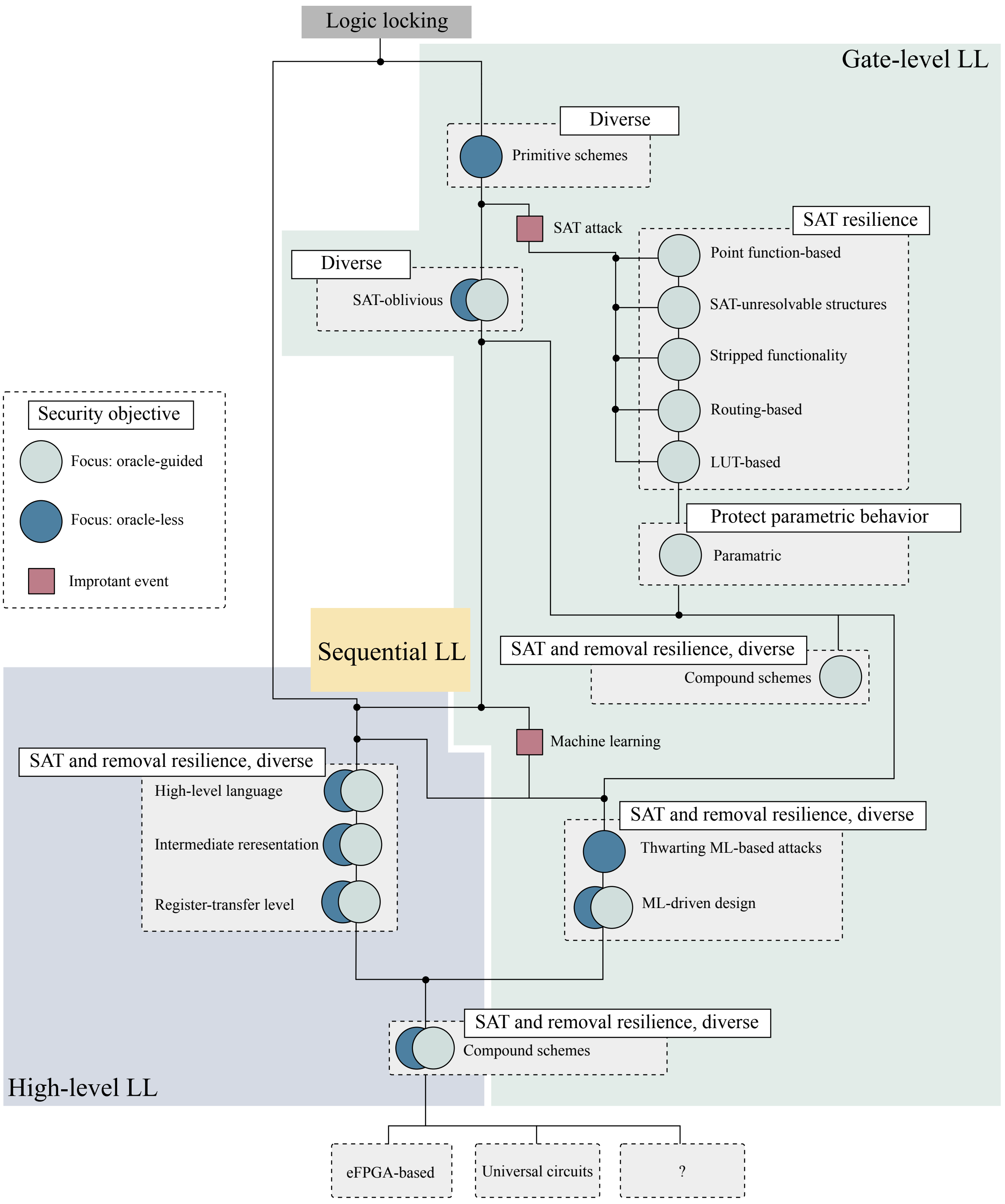


Figure 5.2.: Evolution of logic locking.

going against security expectations, as high output corruption is typically expected. Moreover, the first iteration of SAT-resilient schemes suffers from removable and identifiable locking structures, thus enabling removal attacks. To mitigate these two pitfalls, SAT-resilient schemes can be combined with traditional locking, implementing the so-called compound schemes.

Starting around 2018, Machine Learning (ML) concepts have been introduced into the domain of logic locking. This initiated a new research stream focusing on two main topics [162; 163; 164]: thwarting ML-based attacks on LL and ML-driven locking policies. The existing research clearly shows the potential of machine learning in driving the design of logic-locking schemes, as well as challenging different security aspects of locking policies [165; 166; 167; 168; 169; 170]. The main reason "ML and logic locking" is an interesting and promising research direction is that ML can uncover new or confirm previously unconfirmed LL-related leakages [164].

5.1.4.2. High-Level LL

Another research direction is represented by logic locking at higher abstraction levels. Typically, this includes three categories [135]. The first categories concerns the application of LL at high-level language representations before high-level synthesis is deployed [3]. The second category includes the deployment of LL on the intermediate representations generated in the high-level-synthesis process. The final category comprises LL schemes that operate directly at RTL. The main driving motivation to operate LL at higher abstraction levels is the direct availability of semantics, such as operators, control flow, and constants. These semantics are broken down during logic synthesis, making it extremely challenging to retrieve the original semantics on gate level. Therefore, recent studies started to explore the security of LL at higher levels [171; 172; 173; 174; 175; 176].

5.1.4.3. Attacks on Logic Locking

Throughout the evolutionary landscape of LL, a plethora of deobfuscation attacks have been introduced. We can categorize all attacks as follows [177]:

- Functional attacks: These attacks exploit the effects that LL has on the functional (output) behavior of locked designs. For example, an attacker could try to determine a correct activation key by measuring the output corruptibility for many incorrect keys, thereby mutating the key until the corruptibility reaches zero.

- Side-channel attacks: These attacks exploit key-related information that hides in power, timing and delay characteristics. For example, if incorrectly deployed, logic-locking structures can lead to higher power, delay, or area during the logic synthesis process. These hints can be used to deduce the correct key and remove the additional structures.

- Structural attacks: This attack class exploits the structural (topological) characteristics of the logic-locking-induced residue. For example, vulnerable schemes might include LL structures that can easily be identified as redundant logic for incorrect key values.

- Physical attacks: This attack class includes attacks that have a direct physical impact on locked circuits, such as optical probing, fault injection, tampering and others. As an example, a physical attack can try to extract the correct activation key by optically probing certain registers that buffer the correct key from the key storage to the locked design (assuming an oracle-guided attack).

A detailed overview of attacks is available in [22; 131; 135; 178].

[3] High-level synthesis is the process of translating high-level code (e.g., written in C/C++) into synthesizable register-transfer-level constructs.

5.1.4.4. Fundamental Problems

Although substantial research efforts have been invested in the domain of logic locking, we are still far away from provably secure solutions. The main unresolved challenges can be summarized as follows:

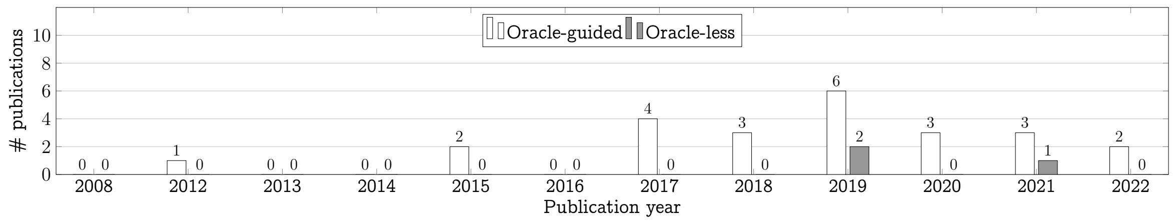
Secure key storage: One fundamental problem in the OG model is the security of the key. In this model, an oracle chip can be used as a golden reference to steer a plethora of attacks. Hereby, the assumption is that the key cannot be read from the chip. Unfortunately, this is not (yet) true. Multiple findings have shown that the key can be acquired through physical methods without having to attack the logic-locking scheme:

- Multiple sources have demonstrated the extraction of keys through probing and fault-injection attacks (=physical attacks) on locked ICs in the presence of a tamper-proof memory [179; 180; 181].
- As the attack model assumes the key-input pins are known and identifiable, an adversary can design a simple hardware Trojan that leaks the key value after the chip is activated [182]. This can be done with class-2 Trojans, i.e., without design-specific knowledge.

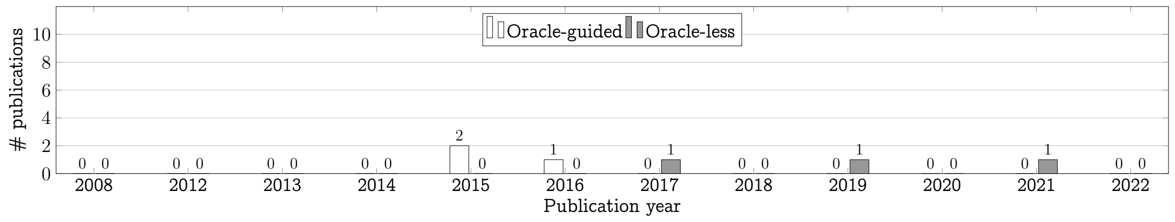
At first glance, the mentioned issues discredit the validity of the OG model, since the key can be found regardless of the underlying locking policy. However, the OG model still plays an important role in the reverse engineering process, as discussed in Section 5.1.3.2.

Security metrics and variability: Despite many efforts, a concrete security metric for logic locking does not exist. Existing proposals mostly focus on functional implications (such as corruptibility) or the success in thwarting certain attacks, such as SAT-resilience. However, these metrics only paint part of the picture. The real, expected impact of logic locking—the increase of reverse-engineering complexity—has, so far, not been measured. This problem exists simply because of the complexity of reverse engineering and the inclusion of diverse manual, semi-manual, and automated steps. Thus, measuring the reverse engineering effort would offer a direct, tangible measure of the security of logic locking—a challenge that still persists. Moreover, even if such an empirical metric would exist, a formal procedure to test the effectiveness of LL against hardware Trojans still remains an open question.

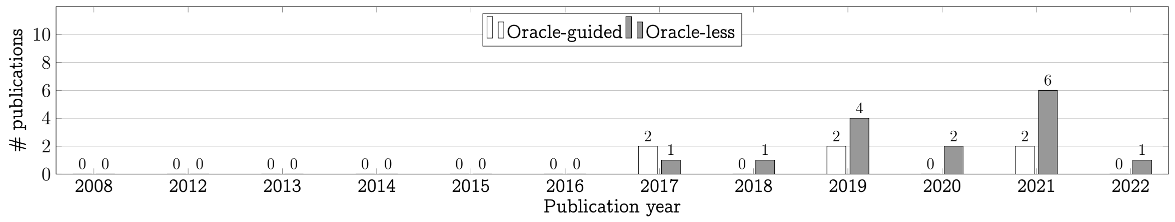
Universal circuits: An interesting solution to most problems in logic locking lies within the concepts of universal circuits [183; 142]. Based on the well-known and many decades old cryptographic primitive introduced by Valiant [184], universal circuits can be programmed to represent any circuit up to a given size. In terms of security, a universal circuit could represent almost any form of hardware functionality, thereby always preserving the same structure. Thus, universal circuits implement the highest form of obfuscation. Unfortunately, the cost of implementation is far beyond acceptable levels. A middle-ground solution has been explored in the form of Embedded Field-Programmable Gate Arrays (eFPGAs). In this configuration, selected design modules are replaced with fully reconfigurable soft eFPGA or ready-made eFPGA hard macros [135; 185]. These macros are later programmed to implement the desired functionality after production. However, more investigations are required to ensure that FPGA-based obfuscation is secure and cost-effective [186; 187; 135].



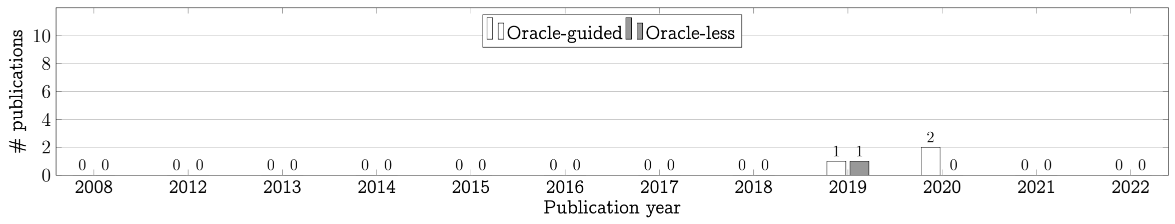
(a) **Functional** attacks.



(b) **Side-channel** attacks.



(c) **Structural** attacks.



(d) **Physical** attacks.

Figure 5.3.: Evolution of publications: attacks on logic locking.

5.1.4.5. Current Trends

Despite the problems mentioned in Section 5.1.4.4, the scientific community continues to design new schemes and attacks to push logic locking towards more secure solutions. To better understand the current trends, we can look at how the publications in the domain of attack and scheme design have progressed. In terms of attacks, as visualized in Figure 5.3, we can identify the following trends:

- Functional attacks remain an active research area.
- Oracle-less attacks are becoming more popular. This trend is mostly related to the problem of secure key storage, as mentioned in Section 5.1.4.4. Moreover, machine learning has boosted the development of oracle-less attacks, as ML-driven approaches work well in this attack model.
- Structural attacks are becoming ever more present. The reason is twofold. First, oracle-less attacks are currently more realistic. Second, ML-driven attacks are successful in learning, processing, and attacking structural representations of hardware, such as netlists (graphs).

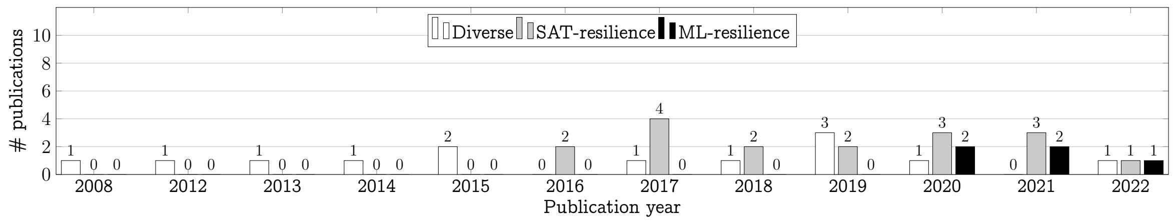


Figure 5.4.: Evolution of logic-locking schemes.

- Physical attacks are slowly being introduced. As in the previous observation, these are closely related to the key-storage issue.

In terms of scheme design (Figure 5.4), the following observations can be made:

- Despite the fundamental problems of oracle-guided attacks, SAT-resilient scheme design remains an active research field.
- Since 2020, ML-resilience has become a strong research stream.

These observations are also visible in more detail in Table 5.1 and 5.2.

Table 5.1.: Logic-locking schemes: security statistics (part 1). Specific security objectives are marked with color: SAT-resilience and ML-resilience.

Scheme	Year	Vulnerable in OG	Vulnerable in OL
Random Logic Locking (RLL) [148; 149]	2008	✓	✓
Strong (Secure) Logic Locking (SLL) [188; 150]	2012	✓	✓
Fault Analysis-Based Logic Locking (FLL) [151]	2013	✓	✓
AND-OR [189]	2014	✓	?
Test-Aware Locking (TAL) [152]	2015	✓*	✓*
Logic Cone Analysis Logic Locking (LCALL) [153]	2015	✓*	✓*
SARLock [155]	2016	✓	✓
Anti-SAT [190; 191]	2016	✓	✓
Tenacious and Traceless Logic Locking (TTLock) [158]	2017	✓	✓
Cyclic locking [157]	2017	✓	✓
Weighted logic locking [192]	2017	?	?
AND-Tree Insertion (ATI) scheme [193]	2017	?	?
Stripped-Functionality Logic Locking (SFLL)-HD/flex [194]	2017	✓	✓
SFLL-fault [195]	2018	?	✓*
SRCLock [196]	2018	✓	?
Logic Cone Size-Based Logic Locking (LCSBLL) [197]	2018	✓*	✓*

* Based on a theoretical evaluation, as no empirical data was found.
✓ At least one attack in this attack model can break the scheme.
? The vulnerability has not been evaluated yet in full extent.

Another interesting point lies in the statistics of vulnerable and unbroken schemes, as presented

Table 5.2.: Logic-locking schemes: security statistics (part 2). Specific security objectives are marked with color: SAT-resilience and ML-resilience.

Scheme	Year	Vulnerable in OG	Vulnerable in OL
Redundancy Attack Resistant Logic Locking (RARLL) [198]	2019	✓*	?
Inter-Lock [199; 200; 139]	2019	?	?
TGA-Resistant Logic Locking (TGARLL) [201]	2019	✓*	?
(M)-CAS-Lock [202]	2019	✓	✓
LOOPLock [203]	2019	?	✓
SFLL-rem [204]	2020	?	?
Bilateral locking [159]	2020	✓	?
Strong Anti-SAT (SAS) [205]	2020	?	?
SAS [205]	2020	✓	?
Truly Random Logic Locking (TRLL) [206]	2020	✓*	?
Scalable Attack-Resistant Obfuscation (SARO) [162]	2020	?	?
UNSAIL [163]	2020	✓*	?
LOOPLock 2.0 [207]	2021	?	✓*
(G-)Anti-SAT [156]	2021	?	?
Robust SAS (RSAS) [208]	2021	?	?
Symmetric Multiplexer (MUX) [209]	2021	✓*	✓
Deceptive Multiplexer Logic Locking (D-MUX) [164]	2021	✓*	✓
LeGO [210]	2022	?	?
SATConda [211]	2022	?	?
IsoLock [212]	2022	?	✗

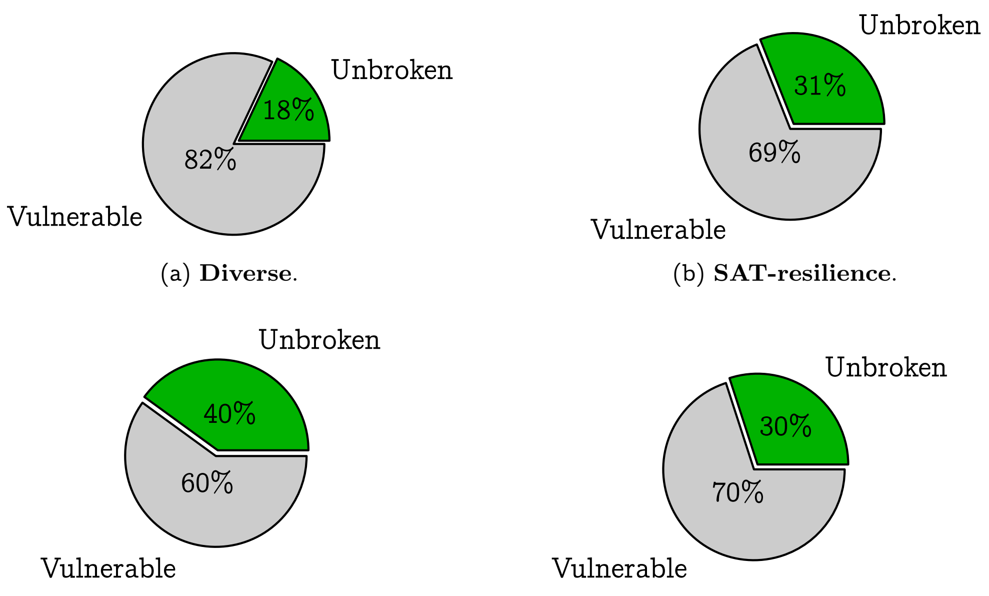
* Based on a theoretical evaluation, as no empirical data was found.
✓ At least one attack in this attack model can break the scheme.
? The vulnerability has not been evaluated yet in full extent.
✗ The current evaluations indicate that the scheme is not vulnerable.

in Figure 5.5. Across different security objectives (diverse, SAT-resilience, ML-resilience), the percentage of broken (vulnerable) schemes ranges from 69% to 82% (with 75% on average). The percentage of unbroken schemes would suggest secure schemes exist[4]. However, the reality is different; some proposed schemes have simply not been evaluated yet. Therefore, only a handful of secure schemes exist, *thereby being secure only in specific attack models and against specific attacks.*

5.2. Functional Filler Cells

Filler cells are often used to fill unused spaces in the layout, in order to improve the debugging, yield, and power characteristics of the chip. An attacker might replace some of these nonfunctional cells with Trojan-related cells. One potential way to mitigating this issue is to use functional

[4] Secure in terms of empirical metrics.



(a) **Diverse**. (b) **SAT-resilience**. (c) **ML-resilience**. (d) **All** objectives.

Figure 5.5.: Percentage of unbroken and vulnerable LL schemes for different security objectives.

filler cells. An example implementation is known as Built-In Self-Authentication (BISA) [213; 214; 215]. This technique fills the unused layout spaces with functional filler cells connected into a combinational and testable circuit. A failure during testing indicates that the protected area might have been manipulated.

5.3. Split Manufacturing

Split manufacturing divides the design into the Front End of Line (FEOL) and Back End of Line (BEOL). FEOL includes the transistors and lower metal layers[5]. BEOL includes the remaining higher metal layers. The low-cost BEOL layers can be fabricated in a trusted in-house foundry. The expensive FEOL layers are fabricated in an untrusted, high-end foundry. In this setting, only the transistors and a limited number of connections are exposed to the untrusted foundry [216; 217; 218]. Split manufacturing offers a potential way to reduce the cost of in-house production and lower the security risk associated with outsourcing the IC fabrication. Despite these promising features, the security of split manufacturing still remains under debate [219].

5.4. Layout Camouflaging

Layout camouflaging conceals the layout of selected types of logic gates by making them appear identical. A standard gate can be camouflaged by using real and dummy contacts, which can enable different functionalities. Automated image processing techniques can easily identify the functionality of standard gates if regular layouts are used. However, automated RE is more difficult when camouflaging is used [220; 221; 220; 222; 223; 224].

[5] The exact number of layers is not defined.

5.5. Comparison of Design-for-Trust Techniques

Among the presented DfTr solutions, logic locking has been identified as the only active protection mechanism that can potentially protect against untrusted entities throughout the microelectronics supply chain [225; 22; 134; 226]. Therefore, logic locking has been the focal point of research in this domain for more than a decade, resulting in a plethora of locking policies and deobfuscation attacks.

5.6. Open Challenges

- How to protect the key from physical attacks?
- How to measure the impact of logic locking on the reverse-engineering effort?
- How to design generic, indistinguishable circuits?

6. Applicability of Detection and Protection Methods to the Asset

After the detection and protection mechanisms were presented in the previous chapters, their applicability to every part of the asset chain is discussed below. First, it is elaborated on the applicability of the different forms of detection schemes in the identification of hardware Trojans in the asset in the supply chain. Second, the same elaboration is conducted w.r.t. the protection against the implementation of hardware Trojans.

6.1. Detection Techniques

Table 6.1 illustrates which detection mechanisms can be used to identify malicious modifications in the asset at a certain link of the supply chain. The cells that contain "Yes" indicate that commercial, open-source, or scientifically published solutions exist for the respective form of the asset and the detection mechanism.

The pre-silicon detection mechanisms can only be applied to the asset in the pre-silicon form, which means that the chip is not manufactured yet. For the manufactured chip, non-destructive reverse engineering is required to yield a formal description of the chip, e.g., in GDSII format.

Post-silicon detection mechanisms are not applicable to non-manufactured chips. The asset could be manufactured to yield a chip, so that post-silicon schemes could be used. However, the manufacturing of a chip solely to use the less successful post-silicon schemes seems illogical.

As indicated in Table 6.1, verifying the integrity of the specification is not supported, as it represents the highest abstraction layer. Most research work assumes the specification is correct or that a manual inspection is sufficient to identify errors. As mentioned before, we assume the specification is also Trojan-free.

Detection Schemes / Design Steps	Pre-silicon Analysis				Post-silicon Analysis		
	Code Coverage Analysis	Formal Verification	Structural Analysis	Functional Analysis	Destr. Reverse Engineering	Functional & Fault Testing	Side-Channel Analysis
Specification file	No	No	No	No	No	No	No
Virtual prototype	Yes	Yes	Yes	Yes	No	No	No
High-Level description	Yes	Yes	Yes	Yes	No	No	No
RTL design	Yes	Yes	Yes	Yes	No	No	No
Gate-Level netlist	Yes	Yes	Yes	Yes	No	No	No
GDSII format	No	Yes	No	Yes	No	No	No
Final chip	RE required	RE required	RE required	RE required	Yes	Yes	Yes
Final device	RE required	RE required	RE required	RE required	Yes	Yes	Yes

Table 6.1.: An illustration of what detection schemes can identify modifications of the asset in the current description format (left column). Therefore, it identifies whether modifications have been conducted in the last design step that lead to the current asset format.

Protection scheme / Format	Logic locking	Functional filler cells	Split manufacturing	Layout camouflaging
Specification file	No	No	No	No
Virtual prototype	No	No	No	No
High-level description	Yes	No	No	No
RTL design	Yes	No	No	No
Gate-level netlist	Yes	No	No	No
GDSII format	Yes	Yes	Yes	No
Final chip	Yes	Yes	No	Yes
Final device	Yes	Yes	No	Yes

Table 6.2.: The applicability of protection schemes on various design formats in the microelectronics supply chain.

6.2. Protection Techniques

The applicability and effectiveness scope of the protection mechanisms discussed in Chapter 5 on every design format throughout the supply chain is presented in Table 6.2. Hereby, a "yes" entry indicates that a certain mechanism can in principle increase the security of a design of the given format. However, *it does not indicate a conclusive and complete security assurance*. Thus, the exact security implication must be evaluated individually in detail for each case.

As discussed in the previous chapter, logic locking provides security properties across different formats, starting from a high-level description. As the effect of logic locking reaches up until the correct key is uploaded to the final chip, the earlier the protection mechanism is deployed, the greater its effectiveness. Functional filler cells are tightly bound to the layout. Thus, the security implications of these cells only start at the GDSII level. Split manufacturing only affects the fabrication process. After the fabrication is done, the effect of split manufacturing has no impact on reverse engineering, as the original (target) design has not been changed in any way through the process—only the fabrication itself is split. Finally, layout camouflaging requires the active inclusion of a trusted foundry. Therefore, its protection only spans the design formats after fabrication.

6.3. Open Challenges

- Is post-manufacturing equivalence checking feasible?
- Are the abstraction layers incompatible with the concept of security when using equivalence checking?
- How to actively protect design formats starting from the specification?

Part III.

WP3: Formal Security Guarantees, Auditability and Trust Models

This part of the report discusses possible formal security guarantees for the asset in the hardware supply chain. Most presented detection and protection mechanisms use empirical analysis to prove their effectiveness.

Formal methods offer mathematical proof and certainty for their success. Therefore, we focus on these methods in this part to develop a complete security guarantee for the hardware supply chain.

The formal methods have some requirements that need to be satisfied. The requirements are discussed in Chapter 7. Afterward, the detection and protection schemes from Part II are discussed in Chapter 8 to elaborate on whether they can offer a formal assurance for the hardware supply chain.

7. Requirements for Formal Guarantees

This chapter deals with the requirements for achieving formal security guarantees. First, the requirements are listed. Second, each point in the list is discussed in detail in the following sections.

The requirements are as follows:

1. A complete formal description of the asset.
2. A formal description of the protection or detection scheme.
3. A formal proof of the success of the defense against a malicious modification for the protection or detection schemes.

7.1. A Complete Formal Description of the Asset

In the given context, a formal description is a written description of the asset that can be processed by an algorithm. Common hardware description languages offer this formal description. The different Hardware Description Language (HDL) can operate on different abstraction levels, such as Verilog on Register-Transfer Level (RTL) or SystemC on system level. Specifications need to be formatted in a manner that can be compared to other descriptions or processed by algorithms checking for vulnerabilities.

In addition to having a formal description, the description must be complete. As discussed in section 4.1.2.2, the higher abstraction levels offer a more dense and simpler environment for describing the asset, which might allow internal states and functional behavior to remain undefined. These undefined states are ignored in detection mechanisms, such as equivalence checking, thus malicious behavior can be hidden within these blind spots. Therefore, all internal states and possible functional behavior need to be defined at a higher level.

Achieving this is fairly complex, as it goes against the nature of the design methodology itself. Higher abstraction levels are used to neglect details and ease the design process. An RTL description is independent of power, timing, and area definitions, whereby GDSII descriptions incorporate the information in its format.

Additionally, manufactured chips must be reverse-engineered to yield a formal description, which can be processed. Reverse engineering needs to be conducted for every manufactured chip, if the integrity of each device shall be proven.

As a summary, it may be stated that the **two major challenges** for this requirement are:

1. Higher abstraction layers use fewer details to describe the asset. Hardware Trojans can be hidden in this abstraction.

2. A formal description of a manufactured chip is only available if all final devices are reverse-engineered.

7.2. A Formal Description of the Protection or Detection Scheme

Most methods depend on algorithms that require a formatted description to run as a computer program. If such a computer program exists, it can be processed to offer mathematical proof. Some approaches utilize the conversion of a program or algorithm into a graph structure, that can be further analyzed. For hardware descriptions, petri nets, control-data-flow-graphs or abstract syntax trees are common representations that can be analyzed using mathematical algorithms. Computer programs can be represented similarly. Known mathematical structures, such as graphs, can be used to employ known mathematical lemmas. Algorithms could be represented similarly. However, the goal of the protection and detection scheme needs to be formulated to fulfill the third requirement.

7.3. A Formal Proof of the Success of the Defense Against Hardware Trojans

The final goal is to have complete proof that no malicious modifications have been implemented in the asset during any step of the supply chain. Traditionally, formal methods consist of two components: assumptions and properties. The properties can be proven under the given assumptions. Assumptions can be made to reduce complexity and make the computation feasible. Assumptions can consider the behavior for undefined states or, e.g., disallow certain input patterns. Once the properties are proven, they are solely proven under the given assumptions.

7.3.1. Detection mechanisms

For a detection mechanism to offer a formal proof of the success of the identification of a hardware Trojan, three challenges need to be overcome.

1. The detection mechanism needs to detect any kind of hardware Trojan. It should not matter whether they endanger the confidentiality, integrity, or availability property of the asset. A combination of multiple detection mechanisms that cover all properties together is also acceptable.
2. The detection mechanism should be applicable at every link of the supply chain.
3. The detection scheme should offer a formal proof for its success.

If the listed challenges are overcome, the third property is fulfilled. The third challenge is currently only covered by formal methods.

7.3.2. Protection Mechanisms

Formally proving the success of active protection mechanisms is an extremely difficult task—if not impossible. This is supported by the lack of research results in this domain. Nevertheless, the community must continue to research and develop active protection mechanisms, as these are, at the moment, the only line of defense against malicious modifications after production. Nevertheless, even though it might not be possible to formally prove the effectiveness of a protection scheme, it might be possible to formally prove that certain modifications cannot be made due to a provable absence of knowledge. For example, in terms of universal circuits, any change in the structure (thus, functionality) of the design cannot be made based on design-specific knowledge, as the structure of the circuit provides no information about the intended functionality. This assumes that the full circuit is replaced with a configurable block. In this scenario, it can be proven that design-dependent modifications (class-1 hardware Trojans) cannot be intentionally inserted, as no information leakage exists. However, design-independent Trojans are still feasible, as well as design and manufacturing faults.

8. Possible Formal Security Guarantees for the Hardware Supply Chain

Some state-of-the-art technologies try to work on the challenges explained in the chapter above. As most security frameworks focus on their functionality, the completeness of the underlying hardware description is sometimes disregarded. This chapter discusses some of the frameworks that try to offer a formal security guarantee. We discuss their functionality and their relation to the listed requirements, such as the influence of an incomplete hardware description or the different abstraction layers on their effort to give a formal security guarantee for the hardware supply chain. Additionally, possible gaps in the guarantee are pointed out to establish a possible roadmap for future research.

8.1. Detection Mechanisms

8.1.1. Formal Methods

In the field of hardware Trojan detection schemes, the obvious approach for a formal security guarantee is the usage of a formal method such as equivalence checking and the enforcement of security properties. The two approaches offer a formal solution to identify malicious modifications by utilizing mathematical models that can prove a property such as confidentiality or equivalence.

However, although formal methods have been shown to be capable of identifying malicious modifications, further analysis might be required to discuss how hardware Trojans might be overlooked when applying the formal methods to protect the supply chain.

As discussed in Section 4.1.2.2, the abstraction of the description of the asset influences the level of detail. Although formal methods offer formal proof of a security property, the first requirement for formal assurance needs to be covered as well. The lack of detail in higher abstraction layers could be abused to hide additional modifications that cannot be identified when comparing the description with a version of a higher abstraction layer. The ***influence of the level of abstraction*** on the possibility of hiding hardware Trojans needs to be elaborated further.

Additionally, manufactured chips do not fulfill the first requirement. A formal description is only available if ***Reverse Engineering (RE)*** is conducted. Furthermore, if all assets must be protected, all devices must be reverse-engineered. As destructive approaches would deem every asset useless after the analysis, non-destructive mechanisms must be used to generate a formal description of the asset. Furthermore, fast approaches are required to allow formal proof for every manufactured chip.

As not all formal methods are supported for every abstraction layer for the asset, the ***gaps need to be filled***. Most model checkers and theorem provers are embedded in commercial tools by Synopsys and Cadence. If a lack of trust in such third-party providers is existent, as discussed in

Section 2.2, additional *open-source* implementations need to be considered and implemented.

8.1.2. Other methods

Other detection mechanisms, such as functional, structural, and code coverage analysis, are shown empirically to be successful in the identification of hardware Trojans. Therefore, the third requirement is not fulfilled.

Therefore, mathematical methods describing a metric of the success of the detection methods in identifying each type of hardware Trojan are required. The success of all of the listed detection methods is shown empirically using benchmarks in all publications. Thus, the task of developing mathematical proof can be labeled as a complex problem.

Additionally, the problems about requirement one, listed in the previous subsection are valid for the other methods as well. Abstraction layers and RE influence the success in the identification of malicious modifications.

8.2. Protection Mechanisms

The protection schemes discussed in Chapter 5 are all active protection schemes relying on two security pillars: increasing the RE or hardware Trojan insertion effort. For example, logic locking aims to increase the RE effort. Functional filler cells target the increase of insertion difficulty. Unfortunately, *none of the discussed protection mechanisms includes a security guarantee*. The main reason a guarantee cannot be given lies in the nonexistence of tangible and measurable security metrics based on the RE effort and insertion difficulty. How much RE is required to insert a class-1 hardware Trojan? How much is the effort impacted by the different protection mechanisms? These questions remain unanswered, despite a handful of smaller studies along these topics. Without the answers to these questions, it will be difficult to formalize the security implications of existing protection mechanisms. Therefore, at the time of writing this report, a protection mechanism with formal security guarantees does not exist.

8.3. Open Challenges

- How to deploy end-to-end equivalence checking to secure the complete hardware supply chain?
- What is the influence of the abstraction layer on the success of the formal methods?
- How to enable formally-secure active protection approaches?

Part IV.

WP4: Research Gaps and Recommendations

This part discusses the research gaps and recommendations for future research projects in the field of secure hardware supply chains. Chapter 9 lists research needs to secure the hardware supply chain from malicious modifications. Chapter 10 presents an approximation of the required project costs and time effort for the recommended research goals.

9. List of Research Needs

The open challenges listed throughout this study are summarized into Research Goals (RGs). Each RG contains several Work Streams (WSs). To enable a secure microelectronics supply chain, it is important to look at two challenges. (1) How to formally secure the supply chain and reach mathematically proven security guarantees throughout the hardware design and fabrication flow. Evidently, this goal fulfills the highest level of security possible, potentially leading to high-risk and long-term projects. (2) Therefore, it is crucial to also support research activities on lower-risk, short-term projects that enable best-effort security. These two macro-goals are described in the following.

9.1. Goal: Formal Security Guarantees

To enable a formally secure hardware design and fabrication flow, the following must be achieved:

- **RG1**: End-to-end, automatic, zero-fault, and non-destructive Reverse Engineering (RE) from the physical device to high abstraction levels. Having a complete RE flow from the final chip will be a key-enabler for end-to-end Equivalence Checking (EQ), any form of Hardware Trojan (HT) detection, as well as the effectiveness estimation of active protection methodologies. This research gap includes the following research streams:
  - **WS1.1**: The complexity of the RE process makes it extremely challenging to estimate the complexity, cost, and time effort of reverse engineering. It is fundamentally important to understand these properties to enable designing and evaluating potential protection methods.
  - **WS1.2**: The RE process involves many complex non-standardized manual, semi-manual, and automatic steps. Therefore, reverse-engineering every produced device for security checks remains infeasible. Thus, the community must provide automatic tools that facilitate an end-to-end RE process with supportive complexity, effort, and time metrics.
- **RG2**: End-to-end and complete EQ from high abstraction levels to the final physical device. This EQ chain will ensure the final fabricated, packaged, and embedded device remains fully equivalent to its original design description. The completeness of the flow must ensure that any form of manipulation is detected by the EQ process. This research gap includes the following research streams:
  - **WS2.1**: The evaluation of the influence of the abstraction layer on the effectiveness of EQ. The common design methodology relies on the principle of abstraction and Electronic Design Automation (EDA) tools to generate the final detailed description. This lack of detail in the higher abstraction layers can be utilized to hide Trojans,

when comparing the low level design with an abstract definition. The influence of the abstraction on EQ shall be analyzed.

  – **WS2.2**: Enabling EQ at high abstraction layers is an important step in closing the end-to-end EQ process. Currently, certain high-level descriptions of hardware cannot be used as input to modern EQ checkers.

  – **WS2.3**: Enabling post-fabrication EQ is the final step towards closing the EQ chain. One potential way to close this end of the chain is to reverse-engineer each produced device back to a format that can be checked with EQ. However, the current RE flows remain incomplete and faulty.

  – **WS2.4**: Trust must remain a cornerstone of the equivalence-check process. Therefore, it is important that the community offers open-source EQ tools that allow processing complex designs described in modern hardware description languages.

9.2. Goal: Best-Effort Security

As the research gaps in Section 9.1 are estimated to be of long-term and high-risk, it is important to support lower-risk research gaps that might not result in formal and complete security guarantees, but offer best-effort security. These research goals include the following:

- **RG3**: Active protection mechanisms might not provide formal security guarantees, but remain an important and, at the moment, only line of defense against malicious modifications during external design and fabrication steps. However, the effectiveness of active protection methods is bound to many unanswered questions, including:

  – **WS3.1**: Protecting hardware from physical attacks plays an important role in extending the security benefits of active protection mechanisms, such as logic locking, beyond a single fabrication batch. For example, tamper- and read-proof memories are supposed to ensure the security of secret keys, such as logic-locking activation keys. Unfortunately, various physical inspection methods often circumvent such memories, resulting in successful key retrieval. Note that this issue persists even if a formally secure logic locking scheme exists. Hence, the importance of physically-protected chips is a prerequisite for a long-term security beyond the fabrication process.

  – **WS3.2**: A potentially fully secure protection lies within generic, indistinguishable, and reconfigurable circuits. These circuits could embed any form of functionality, thereby always preserving the same circuit structure. This configuration would allow only design-independent, low-impact HTs, as no information is retrievable from the circuit about its intended functionality. Unfortunately, the design of such circuits is often limited by the implementation cost. Supporting the development of cost-efficient universal circuits or approximations thereof in the form of reconfigurable circuits offers the potential for the highest security levels. Assuming such circuits would exist, formal security guarantees could potentially be compiled based on the structural regularity of these circuits.

  – **WS3.3**: The exact impact of active protection mechanisms on the RE effort remains unknown. However, having an estimation of this impact remains a fundamental tool to perform security estimations.

- **WS3.4**: Active protection mechanisms have not been formalized thus far, and formal security assurances do not exist. It remains unclear if a formally secure active protection is realizable.

- **RG4**: The design and detection of class-1 HTs remains in the focus of security research. To facilitate this goal, the following must be considered:
  - **WS4.1**: A tangible estimation of the effort, tools, and skills required to design and insert class-1 hardware Trojans is an important vehicle to understand what active protection mechanisms must provide and how to protect against Trojan insertion. Currently, this process is only supported by assumptions and small-scale studies.
  - **WS4.2**: EQ remains a golden standard to detect design manipulations. However, it is unclear to which extent HTs can be designed to circumvent detection by EQ.
  - **WS4.3**: Post-silicon EQ still remains a long-term goal, which heavily depends on fault-free RE. Therefore, it is relevant to also support best-effort research on post-silicon, non-EQ-based HT detection methods.

The discussed research goals and their work streams are summarized in Table 9.1.

Table 9.1.: Research goals and work streams.

Goals	**Title**
RG1	End-to-end, automatic, zero-fault, and non-destructive RE
WS1.1	Quantification of RE complexity, cost, and time effort
WS1.2	Automatic and zero-fault RE from physical devices to high-level design descriptions
RG2	End-to-end and complete EQ
WS2.1	Influence of the abstraction layer on the effectiveness of EQ
WS2.2	Enabling EQ at high abstraction layers
WS2.3	Enabling post-fabrication EQ
WS2.4	Open-source EQ tools
RG3	Active protection mechanisms
WS3.1	Protecting against physical attacks
WS3.2	Designing cost-effective, generic, and indistinguishable circuits
WS3.3	Impact of active protection mechanisms (e.g., logic locking) on RE
WS3.4	Applicability of formal security evaluations for active protection mechanisms
RG4	Class-1 HTs: design and detection
WS4.1	Effort, tools, and skills requirements for the insertion of class-1 HTs
WS4.2	Effectiveness and limits of EQ for HT detection
WS4.3	Post-silicon, non-EQ-based HT detection methods

10. Recommendation for Action

The recommendation for action at the level of main research goals is presented in Figure 10.1. Hereby, as previously discussed, we can categorize the four RGs based on the final outcome into (1) formal security guarantees and (2) best-effort security. **The RGs in (1) are of higher priority.** Moreover, the specific work streams for each RG are shown in Figures 10.2-10.5.

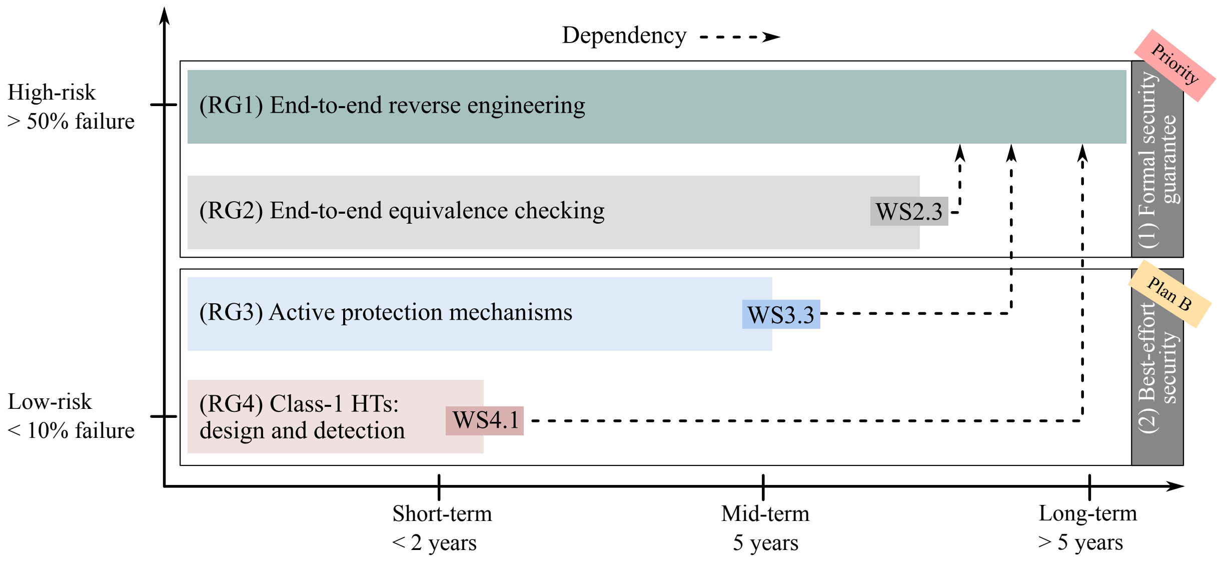
10.1. Cost Estimation

The cost and time estimation of the mentioned RGs is based on our experience in BMBF and industrial projects. The main cost unit is a Person Years (PYs). The majority of PYs are typically based on academic research costs. However, some work streams are likely to involve a certain number of industry-based PYs. Moreover, certain work streams require expensive equipment. These cases are listed in the following:

- WS1.2 will likely include industry partners, as high-level design abstractions are often available in third-party software solutions.
- WS1.3 will likely include industry partners, as certain modifications of the fabrication process might be an important component in post-fabrication EQ.
- WS2.2 will require expensive imaging equipment to automatically delayer fabricated chips.

The cost estimation per RG in PYs is as follows:

- (RG1) End-to-end reverse engineering: 75 PY.
- (RG2) End-to-end equivalence checking: 40 PY.
- (RG3) Active protection mechanisms: 25 PY.
- (RG4) Class-1 HTs: design and detection: 15 PY.



Dependency
High-risk > 50% failure
Low-risk < 10% failure
(RG1) End-to-end reverse engineering
(RG2) End-to-end equivalence checking
WS2.3
(RG3) Active protection mechanisms
WS3.3
(RG4) Class-1 HTs: design and detection
WS4.1
(1) Formal security guarantee
Priority
(2) Best-effort security
Plan B
Short-term < 2 years
Mid-term 5 years
Long-term > 5 years

Figure 10.1.: Recommendation for action at the level of research goals.

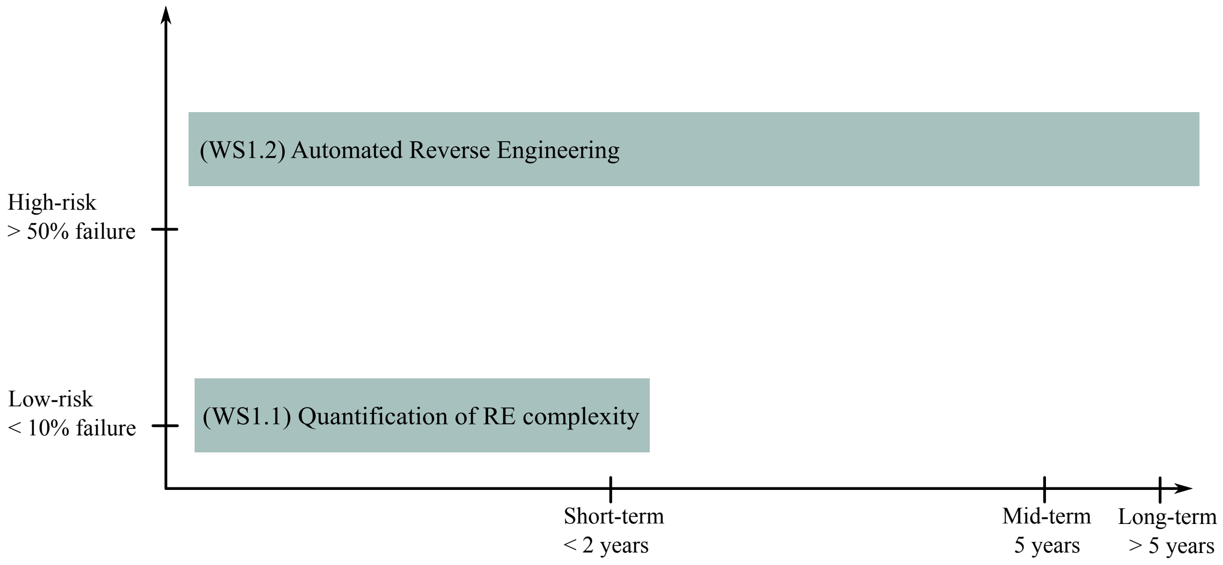


Figure 10.2.: RG1: Recommendation for action at the level of work streams.

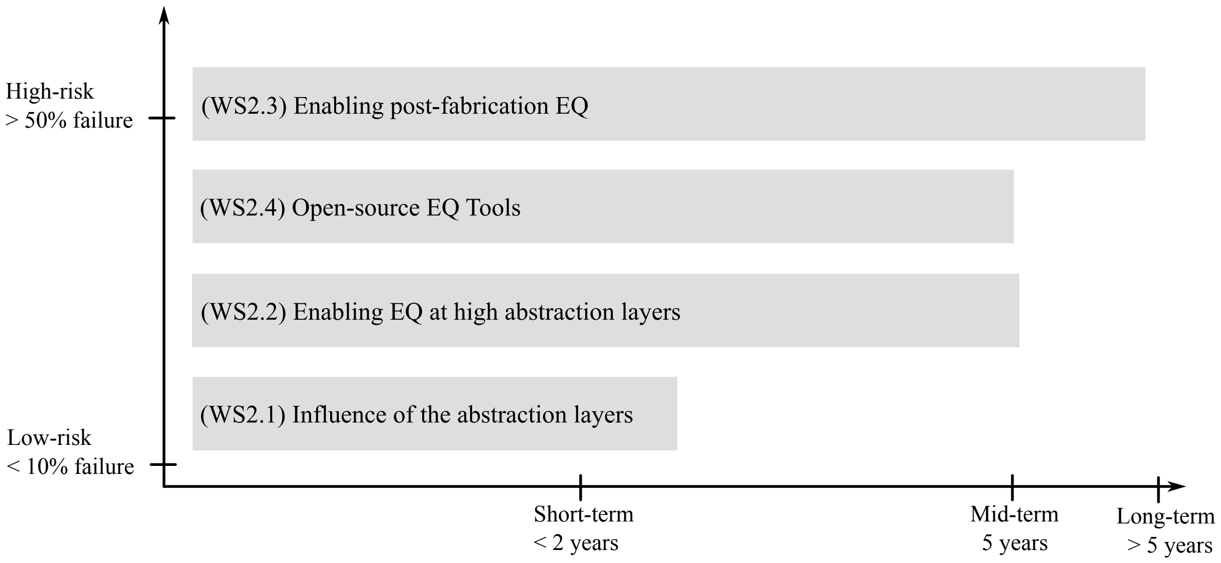


Figure 10.3.: RG2: Recommendation for action at the level of work streams.

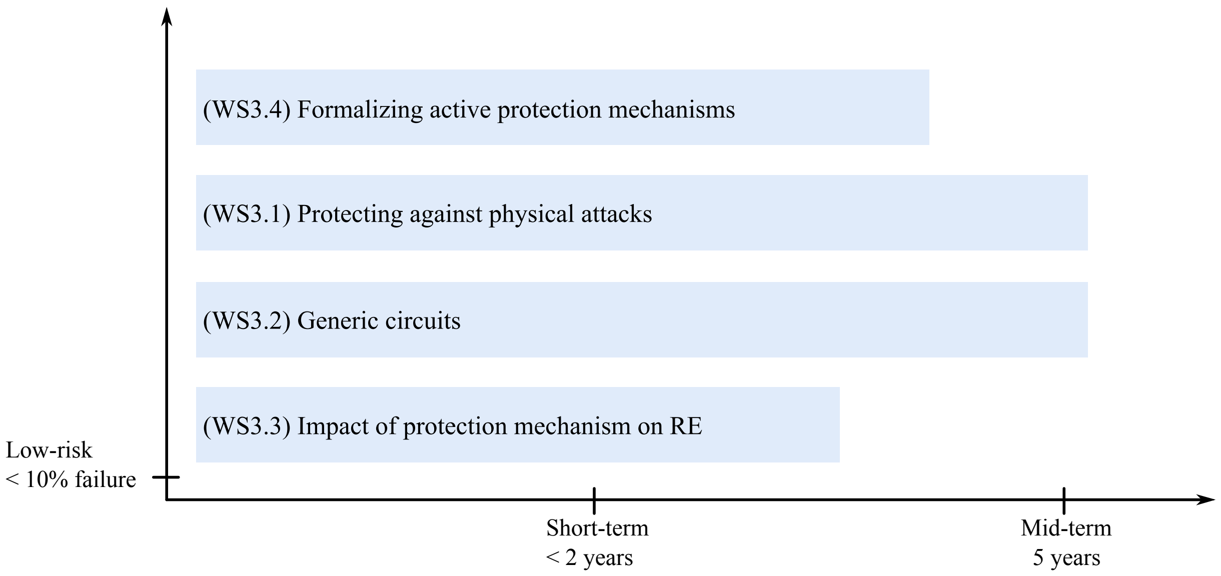


Figure 10.4.: RG3: Recommendation for action at the level of work streams.

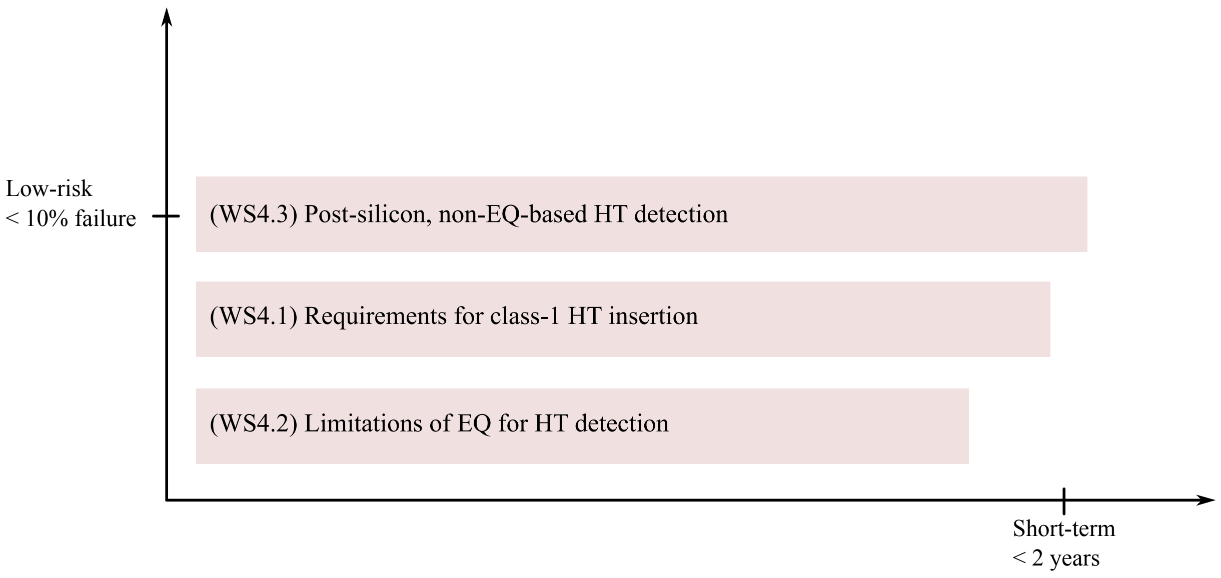

Low-risk
< 10% failure
(WS4.3) Post-silicon, non-EQ-based HT detection
(WS4.1) Requirements for class-1 HT insertion
(WS4.2) Limitations of EQ for HT detection
Short-term
< 2 years

Figure 10.5.: RG4: Recommendation for action at the level of work streams.

Acronyms

3PIP	3rd Party Intellectual Property
ATI	AND-Tree Insertion
ATPG	Automatic Test Pattern Generation
BEOL	Back End of Line
BISA	Built-In Self-Authentication
BMBF	German Federal Ministry of Education and Research
C1HT	Class-1 hardware Trojan
C2HT	Class-2 hardware Trojan
CNN	Convolutional Neural Networks
CWE	Common Weakness Enumeration
D-MUX	Deceptive Multiplexer Logic Locking
DfT	Design for Test
DfTr	Design-for-Trust
DoS	Denial of Service
EDA	Electronic Design Automation
eFPGA	Embedded Field-Programmable Gate Array
EQ	Equivalence Checking
FEOL	Front End of Line
FIB	Focused Ion Beam
FLL	Fault Analysis-Based Logic Locking
FSM	Finite State Machine
GNN	Graph Neural Network
HDL	Hardware Description Language
HIM	Helium Ion Microscopy

HT	Hardware Trojan
HW	Hardware
IC	Integrated Circuit
IFA	Information Flow Analysis
IP	Intellectual Property
LCALL	Logic Cone Analysis Logic Locking
LCSBLL	Logic Cone Size-Based Logic Locking
LL	Logic Locking
LVS	Layout vs Schematic comparison
ML	Machine Learning
MUX	Multiplexer
OEM	Original Equipment Manufacturer
OG	Oracle-Guided
OL	Oracle-Less
PCB	Printed Circuit Board
PY	Person Year
RARLL	Redundancy Attack Resistant Logic Locking
RE	Reverse Engineering
REE	Reverse Engineering Effort
RG	Research Gap
RLL	Random Logic Locking
RSAS	Robust SAS
RTL	Register-Transfer Level
SARO	Scalable Attack-Resistant Obfuscation
SAS	Strong Anti-SAT
SAT	Boolean Satisfiability Problem

SEM	Scanning Electron Microscopy
SFLL	Stripped-Functionality Logic Locking
SLL	Strong (Secure) Logic Locking
TAL	Test-Aware Locking
TEM	Transmission Electron Microscopy
TGARLL	TGA-Resistant Logic Locking
TRLL	Truly Random Logic Locking
TTLock	Tenacious and Traceless Logic Locking
VP	Virtual Prototype
WP	Work Package
WS	Work Stream

Bibliography

[1] Swarup Bhunia and Mark Tehranipoor. *Hardware Security: A Hands-on Learning Approach*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2018.

[2] Swarup Bhunia and M Tehranipoor. “The Hardware Trojan War: Attacks, Myths, and Defenses”. In: *Springer International Publishing* (2018). DOI: 10.1007/978-3-319-68511-3.

[3] Tiago Perez and Samuel Pagliarini. “Hardware Trojan Insertion in Finalized Layouts: a Silicon Demonstration”. In: arXiv, 2021. DOI: 10.48550/ARXIV.2112.02972.

[4] Defense Advanced Research Project Agency (DARP). *Integrity and Reliability of Integrated Circuits (IRIS)*. https://www.darpa.mil/program/integrity-and-reliability-of-integrated-circuits. accessed: July 2021.

[5] Defense Advanced Research Project Agency (DARP). *Trusted Integrated Circuits (TRUST)*. https://www.darpa.mil/program/trusted-integrated-circuits. accessed: July 2021.

[6] Defense Advanced Research Project Agency (DARP). *Supply Chain Hardware Integrity for Electronics Defense (SHIELD)*. https://www.darpa.mil/program/supply-chain-hardware-integrity-for-electronics-defense. accessed: July 2021.

[7] Bundesministerium für Bildung und Forschung (BMBF). *Mikroelektronik. Vertrauenswürdig und nachhaltig. Für Deutschland und Europa. Rahmenprogramm der Bundesregierung für Forschung und Innovation 2021-2024*. https://www.elektronikforschung.de/rahmenprogramm. accessed: July 2021.

[8] Bundesministerium für Bildung und Forschung (BMBF). *Vertrauenswürdige Elektronik. Forschung und Innovation für technologische Souveränität.* https://www.elektronikforschung.de/service/publikationen/vertrauenswuerdige-elektronik. accessed: July 2021.

[9] Microsoft Inc. *Zero-Trust Model*. https://www.microsoft.com/de-de/security/business/zero-trust. Oct. 2022.

[10] Bicky Shakya et al. “Benchmarking of hardware trojans and maliciously affected circuits”. In: *Journal of Hardware and Systems Security* 1.1 (2017), pp. 85–102. DOI: 10.1007/s41635-017-0001-6.

[11] Imran Abbasi et al. “TrojanZero: Switching Activity-Aware Design of Undetectable Hardware Trojans with Zero Power and Area Footprint”. In: Mar. 2019, pp. 914–919. DOI: 10.23919/DATE.2019.8714829.

[12] Niladri Maity and Reshmi Maity. “VHDL and Verilog: Unbiased Compared and Contrasted”. In: *International Journal HIT Transaction on ECCN* 2 (Apr. 2007), pp. 356–363.

[13] Philippe Coussy and Adam Morawiec. *High-Level Synthesis: From Algorithm to Digital Circuit*. 1st. Springer Publishing Company, Incorporated, 2008.

[14] Synopsys Inc. *ASIP Designer*. https://www.synopsys.com/dw/ipdir.php?ds=asip-designer. Oct. 2022.

[15] M. Birnbaum. *Essential Electronic Design Automation (EDA)*. Prentice Hall Modern Semiconductor Design Series'sub Series: Ph Signal Integrity Library. Prentice Hall PTR/Pearson Education, 2004.

[16] Jonathan Bachrach et al. "Chisel: Constructing hardware in a Scala embedded language". In: *DAC Design Automation Conference 2012*. 2012, pp. 1212–1221. DOI: 10.1145/2228360.2228584.

[17] Thomas Bourgeat et al. "The Essence of Bluespec: A Core Language for Rule-Based Hardware Design". In: *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2020. London, UK: Association for Computing Machinery, 2020, pp. 243–257. DOI: 10.1145/3385412.3385965.

[18] Vaibbhav Taraate. "Physical Design". In: *ASIC Design and Synthesis : RTL Design Using Verilog*. Singapore: Springer Singapore, 2021, pp. 245–258. DOI: 10.1007/978-981-33-4642-0_16.

[19] Synopsys Inc. *Design Compiler*. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html. Oct. 2020.

[20] A.L. Crouch. *Design-for-test for Digital IC's and Embedded Core Systems*. Prentice Hall Modern Semiconductor Design Series' Sub Series. Prentice Hall PTR, 1999.

[21] K. Xiao et al. "Hardware Trojans: Lessons Learned after One Decade of Research". In: *ACM Trans. Des. Autom. Electron. Syst.* 22.1 (May 2016). DOI: 10.1145/2906147.

[22] Dominik Sisejkovic and Rainer Leupers. *Logic Locking: A Practical Approach to Secure Hardware*. Springer, 2023. DOI: https://doi.org/10.1007/978-3-031-19123-7.

[23] R. S. Chakraborty, S. Narasimhan, and S. Bhunia. "Hardware Trojan: Threats and emerging solutions". In: *2009 IEEE International High Level Design Validation and Test Workshop*. 2009, pp. 166–171. DOI: 10.1109/HLDVT.2009.5340158.

[24] S. Bhunia et al. "Hardware Trojan Attacks: Threat Analysis and Countermeasures". In: *Proceedings of the IEEE* 102.8 (2014), pp. 1229–1247. DOI: 10.1109/JPROC.2014.2334493.

[25] Xiaoxiao Wang, Mohammad Tehranipoor, and Jim Plusquellic. "Detecting malicious inclusions in secure hardware: Challenges and solutions". In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2008, pp. 15–19. DOI: 10.1109/HST.2008.4559039.

[26] M. Tehranipoor and F. Koushanfar. "A Survey of Hardware Trojan Taxonomy and Detection". In: *IEEE Design Test of Computers* 27.1 (2010), pp. 10–25. DOI: 10.1109/MDT.2010.7.

[27] R. Karri et al. "Trustworthy Hardware: Identifying and Classifying Hardware Trojans". In: *Computer* 43.10 (2010), pp. 39–46. DOI: 10.1109/MC.2010.299.

[28] Dominik Sisejkovic and Rainer Leupers. "Hardware Trojans". In: *Logic Locking: A Practical Approach to Secure Hardware*. Cham: Springer International Publishing, 2023, pp. 13–24. DOI: 10.1007/978-3-031-19123-7_3.

[29] Subhasish Mitra, H.-S. Philip Wong, and Simon Wong. "The Trojan-proof chip". In: *IEEE Spectrum* 52.2 (2015), pp. 46–51. DOI: 10.1109/MSPEC.2015.7024511.

[30] Sally Adee. "The Hunt For The Kill Switch". In: *IEEE Spectrum* 45.5 (2008), pp. 34–39. DOI: 10.1109/MSPEC.2008.4505310.

[31] Y. Shiyanovskii et al. "Process reliability based trojans through NBTI and HCI effects". In: *2010 NASA/ESA Conference on Adaptive Hardware and Systems*. 2010, pp. 215–222. DOI: 10.1109/AHS.2010.5546257.

[32] Lang Lin, Wayne Burleson, and Christof 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, pp. 117–122.

[33] Dominik Sisejkovic et al. “Control-Lock: Securing Processor Cores Against Software-Controlled Hardware Trojans”. In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. GLSVLSI ’19. Tysons Corner, VA, USA: Association for Computing Machinery, 2019, pp. 27–32. DOI: 10.1145/3299874.3317983.

[34] Georg T. Becker et al. “Stealthy Dopant-Level Hardware Trojans”. In: *Cryptographic Hardware and Embedded Systems - CHES 2013*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 197–214.

[35] Yier Jin, Nathan Kupp, and Yiorgos Makris. “Experiences in Hardware Trojan design and implementation”. In: *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2009, pp. 50–57. DOI: 10.1109/HST.2009.5224971.

[36] Samaneh Ghandali et al. “Side-Channel Hardware Trojan for Provably-Secure SCA-Protected Implementations”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28.6 (2020), pp. 1435–1448. DOI: 10.1109/TVLSI.2020.2982473.

[37] Md Mahbub Alam et al. “Soft-HaT: Software-Based Silicon Reprogramming for Hardware Trojan Implementation”. In: *ACM Trans. Des. Autom. Electron. Syst.* 25.4 (June 2020). DOI: 10.1145/3396521.

[38] Yu Liu, Yier Jin, and Yiorgos Makris. “Hardware Trojans in wireless cryptographic ICs: Silicon demonstration & detection method evaluation”. In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013, pp. 399–404. DOI: 10.1109/ICCAD.2013.6691149.

[39] Alexander Hepp and Georg Sigl. “Tapeout of a RISC-V Crypto Chip with Hardware Trojans: A Case-Study on Trojan Design and Pre-Silicon Detectability”. In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*. CF ’21. Virtual: Catania, Italy: Association for Computing Machinery, May 2021. DOI: 10.1145/3457388.3458869.

[40] Stefan Heck, Sri Kaza, and Dickon Pinner. *Creating value in the semiconductor industry*. 2011.

[41] Jan-Peter Kleinhans and Nurzat Baisakova. *The global semiconductor value chain*. https://www.stiftung-nv.de/sites/default/files/the_global_semiconductor_value_chain.pdf. Oct. 2020.

[42] M. G. Rekoff. “On reverse engineering”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15.2 (1985), pp. 244–252. DOI: 10.1109/TSMC.1985.6313354.

[43] David C. Zhang et al. “Fast, Full Chip Image Stitching of Nanoscale Integrated Circuits”. In: 2019.

[44] Bernhard Lippmann et al. “Integrated Flow for Reverse Engineering of Nanoscale Technologies”. In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ASPDAC ’19. Tokyo, Japan: Association for Computing Machinery, 2019, pp. 82–89. DOI: 10.1145/3287624.3288738.

[45] Randy Torrance and Dick James. “The State-of-the-Art in Semiconductor Reverse Engineering”. In: *Proceedings of the 48th Design Automation Conference*. DAC ’11. San Diego, California: Association for Computing Machinery, 2011, pp. 333–338. DOI: 10.1145/2024724.2024805.

[46] “On the Impact of Automating the IC Analysis Process”. In: (2015).

[47] Martin Schobert. “Interactive Functions of the Degate Software Package”. In: (2012).

[48] Rachel Selina Rajarathnam et al. “ReGDS: A Reverse Engineering Framework from GDSII to Gate-level Netlist”. In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2020), pp. 154–163.

[49] Marc Fyrbiak et al. “HAL - The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion”. In: *IEEE Trans. Dependable Secur. Comput.* 16.3 (2019), pp. 498–510. DOI: 10.1109/TDSC.2018.2812183.

[50] Nils Albartus et al. “DANA Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.4 (Aug. 2020), pp. 309–336. DOI: 10.13154/tches.v2020.i4.309-336.

[51] Ulbert J. Botero et al. *Hardware Trust and Assurance through Reverse Engineering: A Survey and Outlook from Image Analysis and Machine Learning Perspectives*. 2020. DOI: 10.48550/ARXIV.2002.04210.

[52] Marc Fyrbiak et al. “Hardware reverse engineering: Overview and open challenges”. In: July 2017, pp. 88–94. DOI: 10.1109/IVSW.2017.8031550.

[53] Leonid Azriel and Avi Mendelson. “SoK: An Overview of Algorithmic Methods in IC Reverse Engineering”. In: Nov. 2019, pp. 65–74. DOI: 10.1145/3338508.3359575.

[54] Joe Grand. “Printed Circuit Board Deconstruction Techniques”. In: *Proceedings of the 8th USENIX Conference on Offensive Technologies*. WOOT’14. San Diego, CA: USENIX Association, 2014, p. 11.

[55] Navid Asadizanjani, Mark Tehranipoor, and Domenic Forte. “PCB Reverse Engineering Using Nondestructive X-ray Tomography and Advanced Image Processing”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 7.2 (2017), pp. 292–299. DOI: 10.1109/TCPMT.2016.2642824.

[56] Navid Asadizanjani et al. “Non-Destructive PCB Reverse Engineering Using X-Ray Micro Computed Tomography”. In: *International Symposium for Testing and Failure Analysis* (2015).

[57] Shahed E. Quadir et al. “A Survey on Chip to System Reverse Engineering”. In: *J. Emerg. Technol. Comput. Syst.* 13.1 (Apr. 2016). DOI: 10.1145/2755563.

[58] Matthew McGuire, Umit Ogras, and Sule Ozev. “PCB Hardware Trojans: Attack Modes and Detection Strategies”. In: *2019 IEEE 37th VLSI Test Symposium (VTS)*. 2019, pp. 1–6. DOI: 10.1109/VTS.2019.8758643.

[59] Jacob Harrison, Navid Asadizanjani, and Mark Tehranipoor. “On malicious implants in PCBs throughout the supply chain”. In: *Integration* 79 (2021), pp. 12–22. DOI: https://doi.org/10.1016/j.vlsi.2021.03.002.

[60] G. Piliposyan, S. Khursheed, and D. Rossi. “Hardware Trojan Detection on a PCB Through Differential Power Monitoring”. In: *IEEE Transactions on Emerging Topics in Computing* 10.02 (Apr. 2022), pp. 740–751. DOI: 10.1109/TETC.2020.3035521.

[61] Franck Courbon et al. “SEMBA: A SEM based acquisition technique for fast invasive Hardware Trojan detection.” In: *ECCTD*. IEEE, 2015, pp. 1–4.

[62] Adam G. Kimura et al. “A Decomposition Workflow for Integrated Circuit Verification and Validation”. In: *Journal of Hardware and Systems Security* 4 (2020), pp. 34–43.

[63] Nidish Vashistha et al. “Detecting Hardware Trojans Inserted by Untrusted Foundry Using Physical Inspection and Advanced Image Processing”. In: *J. Hardw. Syst. Secur.* 2.4 (2018), pp. 333–344. DOI: 10.1007/s41635-018-0055-0.

[64] Mirko Holler et al. “High-resolution non-destructive three-dimensional imaging of integrated circuits”. In: *Nature* 543.7645 (2017), pp. 402–406.

[65] Leonid Azriel, Ran Ginosar, and Avi Mendelson. “Revealing On-Chip Proprietary Security Functions with Scan Side Channel Based Reverse Engineering”. In: *Proceedings of the*

on Great Lakes Symposium on VLSI 2017. GLSVLSI '17. Banff, Alberta, Canada: Association for Computing Machinery, 2017, pp. 233–238. DOI: 10.1145/3060403.3060464.

[66] Wenchao Li et al. "WordRev: Finding word-level structures in a sea of bit-level gates". English (US). In: *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*. Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013. 2013 6th IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013 ; Conference date: 02-06-2013 Through 03-06-2013. 2013, pp. 67–74. DOI: 10.1109/HST.2013.6581568.

[67] Pramod Subramanyan et al. "Reverse engineering digital circuits using functional analysis". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2013, pp. 1277–1280. DOI: 10.7873/DATE.2013.264.

[68] Pramod Subramanyan et al. "Reverse Engineering Digital Circuits Using Structural and Functional Analyses". In: *IEEE Trans. Emerg. Top. Comput.* 2.1 (2014), pp. 63–80. DOI: 10.1109/TETC.2013.2294918.

[69] Jacob Couch et al. "Functional block identification in circuit design recovery". In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2016, pp. 75–78. DOI: 10.1109/HST.2016.7495560.

[70] Leonid Azriel et al. "Using Scan Side Channel for Detecting IP Theft". In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016. Seoul, Republic of Korea: Association for Computing Machinery, 2016. DOI: 10.1145/2948618.2948619.

[71] Travis Meade et al. "Gate-Level Netlist Reverse Engineering for Hardware Security: Control Logic Register Identification". In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. Montréal, QC, Canada: IEEE Press, 2016, pp. 1334–1337. DOI: 10.1109/ISCAS.2016.7527495.

[72] M. Werner et al. "Reverse Engineering of Cryptographic Cores by Structural Interpretation Through Graph Analysis". In: *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. 2018, pp. 13–18. DOI: 10.1109/IVSW.2018.8494896.

[73] T. Doom et al. "Identifying high-level components in combinational circuits". In: *Proceedings of the 8th Great Lakes Symposium on VLSI (Cat. No.98TB100222)*. 1998, pp. 313–318. DOI: 10.1109/GLSV.1998.665284.

[74] Gregory H. Chisholm et al. "Understanding Integrated Circuits". In: *IEEE Des. Test* 16.2 (Apr. 1999), pp. 26–37. DOI: 10.1109/54.765201.

[75] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering". In: *IEEE Des. Test* 16.3 (July 1999), pp. 72–80. DOI: 10.1109/54.785838.

[76] Wenchao Li, Zach Wasson, and Sanjit A. Seshia. "Reverse engineering circuits using behavioral pattern mining". In: *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. 2012, pp. 83–88. DOI: 10.1109/HST.2012.6224325.

[77] Yiqiong Shi et al. "Extracting functional modules from flattened gate-level netlist". In: *2012 International Symposium on Communications and Information Technologies (ISCIT)*. 2012, pp. 538–543. DOI: 10.1109/ISCIT.2012.6380958.

[78] Adrià Gascón et al. "Template-based circuit understanding". In: *2014 Formal Methods in Computer-Aided Design (FMCAD)*. 2014, pp. 83–90. DOI: 10.1109/FMCAD.2014.6987599.

[79] Mathias Soeken et al. "Simulation graphs for reverse engineering". In: *2015 Formal Methods in Computer-Aided Design (FMCAD)*. 2015, pp. 152–159. DOI: 10.1109/FMCAD.2015.7542265.

[80] Cunxi Yu and Maciej Ciesielski. “Automatic word-level abstraction of datapath”. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 1718–1721. DOI: 10.1109/ISCAS.2016.7538899.

[81] N. Rubanov. “A High-Performance Subcircuit Recognition Method Based on the Nonlinear Graph Optimization”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.11 (2006), pp. 2353–2363. DOI: 10.1109/TCAD.2006.881335.

[82] Yiqiong Shi et al. “A highly efficient method for extracting FSMs from flattened gate-level netlist”. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010, pp. 2610–2613. DOI: 10.1109/ISCAS.2010.5537093.

[83] Travis Meade, Shaojie Zhang, and Yier Jin. “Netlist reverse engineering for high-level functionality reconstruction”. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016, pp. 655–660. DOI: 10.1109/ASPDAC.2016.7428086.

[84] Travis Meade et al. “The Old Frontier of Reverse Engineering: Netlist Partitioning”. In: *Journal of Hardware and Systems Security* 2 (2018), pp. 201–213.

[85] Marc Fyrbiak et al. “On the Difficulty of FSM-based Hardware Obfuscation”. In: vol. 2018. Aug. 2019. DOI: 10.13154/tches.v2018.i3.293-330.

[86] Xuenong Hong et al. “Deep Learning for Automatic IC Image Analysis”. In: *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. 2018, pp. 1–5. DOI: 10.1109/ICDSP.2018.8631555.

[87] Kai Qiao et al. “Wire segmentation for printed circuit board using deep convolutional neural network and graph cut model”. In: *IET Image Process.* 12.5 (2018), pp. 793–800. DOI: 10.1049/iet-ipr.2017.1208.

[88] Tim Bucher et al. *AppGNN: Approximation-Aware Functional Reverse Engineering using Graph Neural Networks*. 2022. DOI: 10.48550/ARXIV.2208.10868.

[89] Yu-Yun Dai and Robert K. Braytont. “Circuit recognition with deep learning”. In: *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2017, pp. 162–162. DOI: 10.1109/HST.2017.7951826.

[90] Arash Fayyazi et al. “Deep Learning-Based Circuit Recognition Using Sparse Mapping and Level-Dependent Decaying Sum Circuit Representations”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2019), pp. 638–641.

[91] Lilas Alrahis et al. *Embracing Graph Neural Networks for Hardware Security (Invited Paper)*. 2022. DOI: 10.48550/ARXIV.2208.08554.

[92] Lilas Alrahis, Johann Knechtel, and Ozgur Sinanoglu. *Graph Neural Networks: A Powerful and Versatile Tool for Advancing Design, Reliability, and Security of ICs*. 2022. DOI: 10.48550/ARXIV.2211.16495.

[93] Johanna Baehr et al. “Open Source Hardware Design and Hardware Reverse Engineering: A Security Analysis”. en. In: *Euromicro Conference on Digital System Design DSD*. Vol. 25. Maspalomas, Gran Canarias, Spain, Sept. 2022.

[94] Sophie Dupuis et al. “Identification of Hardware Trojans triggering signals”. In: 2013.

[95] Burcin Cakır and Sharad Malik. “Hardware Trojan detection for gate-level ICs using signal correlation based clustering”. In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2015, pp. 471–476.

[96] Mohammad Eslami, Tara Ghasempouri, and Samuel Pagliarini. “Reusing Verification Assertions as Security Checkers for Hardware Trojan Detection”. In: *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. 2022, pp. 1–6. DOI: 10.1109/ISQED54688.2022.9806292.

[97] Irina Măriuca Asăvoae et al. “Hardware Trojan detection via rewriting logic”. In: *Journal of Logical and Algebraic Methods in Programming* 127 (2022), p. 100762. DOI: https://doi.org/10.1016/j.jlamp.2022.100762.

[98] Synopsys Inc. *Formality*. https://www.synopsys.com/implementation-and-signoff/signoff/formality-equivalence-checking.html. Oct. 2022.

[99] Claire Wolf. *Yosys*. http://bygone.clairexen.net/yosys/documentation.html. Oct. 2022.

[100] Siemens. *FormalPro*. https://eda.sw.siemens.com/en-US/ic/formalpro-equivalence-checking/. Oct. 2022.

[101] Cadence. *Conformal*. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/logic-equivalence-checking/conformal-equivalence-checker.html. Oct. 2022.

[102] Lubis EDA. *Lubis EDA*. https://lubis-eda.com/. Oct. 2022.

[103] Wei Hu et al. “Why you should care about don’t cares: Exploiting internal don’t care conditions for hardware Trojans”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 707–713. DOI: 10.1109/ICCAD.2017.8203846.

[104] Cadence. *Conformal Equivalence Checker*. https://www.cadence.com/ko_KR/home/tools/digital-design-and-signoff/logic-equivalence-checking/conformal-equivalence-checker.html. Oct. 2022.

[105] Mehran Goli, Jannis Stoppe, and Rolf Drechsler. “Automatic equivalence checking for SystemC-TLM 2.0 models against their formal specifications”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 630–633. DOI: 10.23919/DATE.2017.7927064.

[106] Michal Grobelny, Iwona Grobelna, and Marian Adamski. “Hardware Behavioural Modelling, Verification and Synthesis with UML 2.x Activity Diagrams”. In: *IFAC Proceedings Volumes* 45.7 (2012). 11th IFAC,IEEE International Conference on Programmable Devices and Embedded Systems, pp. 134–139. DOI: https://doi.org/10.3182/20120523-3-CZ-3015.00028.

[107] Carlos Ivan Castro Marquez, Marius Strum, and Wang Jiang Chau. “Formal equivalence checking between high-level and RTL hardware designs”. In: *2013 14th Latin American Test Workshop - LATW*. 2013, pp. 1–6. DOI: 10.1109/LATW.2013.6562666.

[108] Calypto. *SLEC*. http://calypto.agranderdesign.com/slecsystem.php. Oct. 2022.

[109] Mohammed Abderehman et al. “BLAST: Belling the Black-Hat High-level Synthesis Tool”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022), pp. 1–1. DOI: 10.1109/TCAD.2022.3200513.

[110] Mohammed Abderehman, Theegala Rakesh Reddy, and Chandan Karfa. “DEEQ: Data-driven End-to-End EQuivalence Checking of High-level Synthesis”. In: *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. 2022, pp. 64–70. DOI: 10.1109/ISQED54688.2022.9806218.

[111] Jan Richter-Brockmann et al. *VERICA - Verification of Combined Attacks: Automated formal verification of security against simultaneous information leakage and tampering*. Cryptology ePrint Archive, Paper 2022/484. https://eprint.iacr.org/2022/484. 2022.

[112] Wei Hu et al. “Property Specific Information Flow Analysis for Hardware Security Verification”. In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2018, pp. 1–8. DOI: 10.1145/3240765.3240839.

[113] Lennart M. Reimann et al. “QFlow: Quantitative Information Flow for Security-Aware Hardware Design in Verilog”. In: *2021 IEEE 39th International Conference on Computer Design (ICCD)*. 2021, pp. 603–607. DOI: 10.1109/ICCD53106.2021.00097.

[114] Lennart M. Reimann et al. *Quantitative Information Flow for Hardware: Advancing the Attack Landscape*. 2022. DOI: 10.48550/ARXIV.2211.16891.

[115] MITRE. *Common Weakness Enumeration*. https://cwe.mitre.org/. Nov. 2022.

[116] Huili Chen et al. *AdaTest:Reinforcement Learning and Adaptive Sampling for On-chip Hardware Trojan Detection*. 2022. DOI: 10.48550/ARXIV.2204.06117.

[117] Masaru Oya et al. “A score-based classification method for identifying Hardware-Trojans at gate-level netlists”. In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2015, pp. 465–470. DOI: 10.7873/DATE.2015.0352.

[118] Alexander Hepp, Johanna Baehr, and Georg Sigl. “Golden Model-Free Hardware Trojan Detection by Classification of Netlist Module Graphs”. en. In: *Design, Automation and Test in Europe Conference*. Ed. by IEEE. Antwerp, Belgium: IEEE, Mar. 2022, pp. 1317–1322. DOI: 10.23919/DATE54114.2022.9774760.

[119] Rozhin Yasaei, Sina Faezi, and Mohammad Abdullah Al Faruque. *Golden Reference-Free Hardware Trojan Localization using Graph Convolutional Network*. 2022. DOI: 10.48550/ARXIV.2207.06664.

[120] Xiaoming Chen et al. “Hardware Trojan Detection in Third-Party Digital Intellectual Property Cores by Multilevel Feature Analysis”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.7 (2018), pp. 1370–1383. DOI: 10.1109/TCAD.2017.2748021.

[121] Mainak Banga and Michael S. Hsiao. “Trusted RTL: Trojan detection methodology in pre-silicon designs”. In: *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2010, pp. 56–59. DOI: 10.1109/HST.2010.5513114.

[122] Pravin Gaikwad et al. *Third-Party Hardware IP Assurance against Trojans through Supervised Learning and Post-processing*. 2021. DOI: 10.48550/ARXIV.2111.14956.

[123] Susmit Jha and Sumit Kumar Jha. “Randomization Based Probabilistic Approach to Detect Trojan Circuits”. In: *2008 11th IEEE High Assurance Systems Engineering Symposium*. 2008, pp. 117–124. DOI: 10.1109/HASE.2008.37.

[124] Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. “A Novel Technique for Improving Hardware Trojan Detection and Reducing Trojan Activation Time”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20.1 (2012), pp. 112–125. DOI: 10.1109/TVLSI.2010.2093547.

[125] Nicole Lesperance, Shrikant Kulkarni, and Kwang-Ting Cheng. “Hardware Trojan detection using exhaustive testing of k-bit subspaces”. In: *The 20th Asia and South Pacific Design Automation Conference*. 2015, pp. 755–760. DOI: 10.1109/ASPDAC.2015.7059101.

[126] Timothy Trippel et al. “Fuzzing Hardware Like Software”. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3237–3254.

[127] E. Larsson. *Introduction to Advanced System-on-Chip Test Design and Optimization*. Frontiers in Electronic Testing. Springer, 2005.

[128] Synopsys Inc. *TestMAX*. https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-atpg.html. Oct. 2022.

[129] Trey Reece and William H. Robinson. “Detection of Hardware Trojans in Third-Party Intellectual Property Using Untrusted Modules”. In: *IEEE Transactions on Computer-*

Aided Design of Integrated Circuits and Systems 35.3 (2016), pp. 357–366. DOI: 10.1109/TCAD.2015.2459038.

[130] Dakshi Agrawal et al. “Trojan Detection using IC Fingerprinting”. In: *2007 IEEE Symposium on Security and Privacy (SP ’07)*. 2007, pp. 296–310. DOI: 10.1109/SP.2007.36.

[131] Muhammad Yasin, Jeyavijayan (JV). Rajendran, and Ozgur Sinanoglu. *Trustworthy Hardware Design: Combinational Logic Locking Techniques*. Springer International Publishing, 2020. DOI: 10.1007/978-3-030-15334-2.

[132] Dominik Sisejkovic et al. “Logic Locking at the Frontiers of Machine Learning: A Survey on Developments and Opportunities”. In: *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*. 2021, pp. 1–6. DOI: 10.1109/VLSI-SoC53125.2021.9606979.

[133] Sophie Dupuis and Marie-Lise Flottes. “Logic Locking: A Survey of Proposed Methods and Evaluation Metrics”. In: *Journal of Electronic Testing* 35.3 (June 2019), pp. 273–291. DOI: 10.1007/s10836-019-05800-4.

[134] M. Yasin and O. Sinanoglu. “Evolution of logic locking”. In: *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2017, pp. 1–6. DOI: 10.1109/VLSI-SoC.2017.8203496.

[135] Hadi Mardani Kamali et al. “Advances in Logic Locking: Past, Present, and Prospects”. In: *Cryptology ePrint Archive* (2022).

[136] Dominik Sisejkovic. “Designing trustworthy hardware with logic locking”. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, Rheinisch-Westfälische Technische Hochschule Aachen, 2022. Dissertation. Aachen: Rheinisch-Westfälische Technische Hochschule Aachen, 2022, 1 Online–Ressource : Illustrationen. DOI: 10.18154/RWTH-2022-02625.

[137] M. Tanjidur Rahman et al. “Defense-in-Depth: A Recipe for Logic Locking to Prevail”. In: *Integr. VLSI J.* 72.C (May 2020), pp. 39–57. DOI: 10.1016/j.vlsi.2019.12.007.

[138] HENSOLDT Cyber GmbH. *Press Release: HENSOLDT Cyber presents MiG-V, the first RISC-V Processor “Made in Germany” for Security Applications*. https://hensoldt-cyber.com/wp-content/uploads/2020/05/20200515-HENSOLDT-Cyber-PM-MiG-V-is-ready-1.pdf. May 2020.

[139] Dominik Sisejkovic et al. “A Secure Hardware-Software Solution Based on RISC-V, Logic Locking and Microkernel”. In: *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*. SCOPES ’20. St. Goar, Germany: Association for Computing Machinery, 2020, pp. 62–65. DOI: 10.1145/3378678.3391886.

[140] Dominik Sisejkovic and Rainer Leupers. “Processor Integrity Protection”. In: *Logic Locking: A Practical Approach to Secure Hardware*. Cham: Springer International Publishing, 2023, pp. 87–113. DOI: 10.1007/978-3-031-19123-7_8.

[141] Hai Zhou, Amin Rezaei, and Yuanqi Shen. “Resolving the Trilemma in Logic Encryption”. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, pp. 1–8. DOI: 10.1109/ICCAD45719.2019.8942076.

[142] Kaveh Shamsi, David Z. Pan, and Yier Jin. “On the Impossibility of Approximation-Resilient Circuit Locking”. In: *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2019, pp. 161–170. DOI: 10.1109/HST.2019.8741035.

[143] Dominik Sisejkovic and Rainer Leupers. “Working Principle and Attack Scenarios”. In: *Logic Locking: A Practical Approach to Secure Hardware*. Cham: Springer International Publishing, 2023, pp. 27–34. DOI: 10.1007/978-3-031-19123-7_4.

[144] Yinghua Hu et al. “Security-Driven Metrics and Models for Efficient Evaluation of Logic Encryption Schemes”. In: *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*. MEMOCODE ’19. La Jolla, California: Association for Computing Machinery, 2019. DOI: 10.1145/3359986.3361207.

[145] Hai Zhou. “A Humble Theory and Application for Logic Encryption.” In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 696.

[146] Dominik Šišejković et al. “A Unifying Logic Encryption Security Metric”. In: *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. SAMOS ’18. Pythagorion, Greece: Association for Computing Machinery, 2018, pp. 179–186. DOI: 10.1145/3229631.3229636.

[147] Dominik Sisejkovic and Rainer Leupers. “Security Metrics: One Problem, Many Dimensions”. In: *Logic Locking: A Practical Approach to Secure Hardware*. Cham: Springer International Publishing, 2023, pp. 53–70. DOI: 10.1007/978-3-031-19123-7_6.

[148] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. “EPIC: Ending Piracy of Integrated Circuits”. In: *2008 Design, Automation and Test in Europe*. 2008, pp. 1069–1074. DOI: 10.1109/DATE.2008.4484823.

[149] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. “Ending Piracy of Integrated Circuits”. In: *Computer* 43.10 (2010), pp. 30–38. DOI: 10.1109/MC.2010.284.

[150] M. Yasin et al. “On Improving the Security of Logic Locking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.9 (2016), pp. 1411–1424. DOI: 10.1109/TCAD.2015.2511144.

[151] J. Rajendran et al. “Fault Analysis-Based Logic Encryption”. In: *IEEE Transactions on Computers* 64.2 (2015), pp. 410–424. DOI: 10.1109/TC.2013.193.

[152] Stephen M. Plaza and Igor L. Markov. “Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.6 (2015), pp. 961–971. DOI: 10.1109/TCAD.2015.2404876.

[153] Yu-Wei Lee and Nur A. Touba. “Improving logic obfuscation via logic cone analysis”. In: *2015 16th Latin-American Test Symposium (LATS)*. 2015, pp. 1–6. DOI: 10.1109/LATW.2015.7102410.

[154] P. Subramanyan, S. Ray, and S. Malik. “Evaluating the security of logic encryption algorithms”. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2015, pp. 137–143. DOI: 10.1109/HST.2015.7140252.

[155] M. Yasin et al. “SARLock: SAT attack resistant logic locking”. In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2016, pp. 236–241. DOI: 10.1109/HST.2016.7495588.

[156] Jingbo Zhou and Xinmiao Zhang. “Generalized SAT-Attack-Resistant Logic Locking”. In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 2581–2592. DOI: 10.1109/TIFS.2021.3059271.

[157] Kaveh Shamsi et al. “Cyclic Obfuscation for Creating SAT-Unresolvable Circuits”. In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. Banff, Alberta, Canada: Association for Computing Machinery, 2017, pp. 173–178. DOI: 10.1145/3060403.3060458.

[158] Muhammad Yasin et al. “What to Lock? Functional and Parametric Locking”. In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI ’17. Banff, Alberta, Canada: Association for Computing Machinery, 2017, pp. 351–356. DOI: 10.1145/3060403.3060492.

[159] Amin Rezaei, Yuanqi Shen, and Hai Zhou. "Rescuing Logic Encryption in Post-SAT Era by Locking Obfuscation". In: *2020 Design, Automation & Test in Europe Conference Exhibition (DATE)*. 2020, pp. 13–18. DOI: 10.23919/DATE48585.2020.9116500.

[160] Hadi Mardani Kamali et al. "InterLock: An Intercorrelated Logic and Routing Locking". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. ICCAD '20. Virtual Event, USA: Association for Computing Machinery, 2020. DOI: 10.1145/3400302.3415667.

[161] Yang Xie and Ankur Srivastava. "Delay Locking: Security Enhancement of Logic Locking against IC Counterfeiting and Overproduction". In: DAC '17. Austin, TX, USA: Association for Computing Machinery, 2017. DOI: 10.1145/3061639.3062226.

[162] Abdulrahman Alaql and Swarup Bhunia. *Scalable Attack-Resistant Obfuscation of Logic Circuits*. 2020.

[163] Lilas Alrahis et al. "UNSAIL: Thwarting Oracle-Less Machine Learning Attacks on Logic Locking". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 2508–2523. DOI: 10.1109/TIFS.2021.3057576.

[164] Dominik Sisejkovic et al. "Deceptive Logic Locking for Hardware Integrity Protection against Machine Learning Attacks". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2021). DOI: 10.1109/TCAD.2021.3100275.

[165] Dominik Sisejkovic et al. "Challenging the Security of Logic Locking Schemes in the Era of Deep Learning: A Neuroevolutionary Approach". In: *J. Emerg. Technol. Comput. Syst.* 17.3 (May 2021). DOI: 10.1145/3431389.

[166] Lilas Alrahis et al. "OMLA: An Oracle-Less Machine Learning-Based Attack on Logic Locking". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.3 (2022), pp. 1602–1606. DOI: 10.1109/TCSII.2021.3113035.

[167] Likhitha Mankali et al. "Titan: Security Analysis of Large-Scale Hardware Obfuscation Using Graph Neural Networks". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 304–318. DOI: 10.1109/TIFS.2022.3218429.

[168] Lilas Alrahis et al. "GNNUnlock+: A Systematic Methodology for Designing Graph Neural Networks-Based Oracle-Less Unlocking Schemes for Provably Secure Logic Locking". In: *IEEE Transactions on Emerging Topics in Computing* 10.3 (2022), pp. 1575–1592. DOI: 10.1109/TETC.2021.3108487.

[169] Lilas Alrahis et al. "MuxLink: Circumventing Learning-Resilient MUX-Locking Using Graph Neural Network-based Link Prediction". In: (2021). DOI: 10.48550/ARXIV.2112.07178.

[170] Lilas Alrahis et al. "GNNUnlock: Graph Neural Networks-based Oracle-less Unlocking Scheme for Provably Secure Logic Locking". In: *Accepted at DATE 2021* (2020).

[171] Christian Pilato et al. "TAO: Techniques for Algorithm-Level Obfuscation during High-Level Synthesis". In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465830.

[172] Sheikh Ariful Islam, Love Kumar Sah, and Srinivas Katkoori. "DLockout: A Design Lockout Technique for Key Obfuscated RTL IP Designs". In: *2019 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*. 2019, pp. 17–20. DOI: 10.1109/iSES47678.2019.00017.

[173] Christian Pilato et al. "ASSURE: RTL Locking Against an Untrusted Foundry". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2021), pp. 1–13. DOI: 10.1109/TVLSI.2021.3074004.

[174] Chandan Karfa et al. “Is Register Transfer Level Locking Secure?” In: *2020 Design, Automation & Test in Europe Conference Exhibition (DATE)*. Mar. 1, 2020, pp. 550–555. DOI: 10.23919/DATE48585.2020.9116261. published.

[175] Sheikh Ariful Islam, Love Kumar Sah, and Srinivas Katkoori. “High-Level Synthesis of Key-Obfuscated RTL IP with Design Lockout and Camouflaging”. In: *ACM Trans. Des. Autom. Electron. Syst.* 26.1 (Oct. 2020). DOI: 10.1145/3410337.

[176] Christian Pilato et al. “Optimizing the Use of Behavioral Locking for High-Level Synthesis”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022), pp. 1–1. DOI: 10.1109/TCAD.2022.3179651.

[177] Dominik Sisejkovic and Rainer Leupers. “Attacks and Schemes”. In: *Logic Locking: A Practical Approach to Secure Hardware*. Cham: Springer International Publishing, 2023, pp. 35–51. DOI: 10.1007/978-3-031-19123-7_5.

[178] M. Yasin et al. “Removal Attacks on Logic Locking and Camouflaging Techniques”. In: *IEEE Transactions on Emerging Topics in Computing* 8.2 (2020), pp. 517–532. DOI: 10.1109/TETC.2017.2740364.

[179] Susanne Engels, Max Hoffmann, and Christof Paar. *The End of Logic Locking? A Critical View on the Security of Logic Locking*. Cryptology ePrint Archive, Report 2019/796. https://eprint.iacr.org/2019/796. 2019.

[180] M. T. Rahman et al. “The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes”. In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2020, pp. 262–272. DOI: 10.1109/HOST45689.2020.9300258.

[181] A. Jain, M. T. Rahman, and U. Guin. “ATPG-Guided Fault Injection Attacks on Logic Locking”. In: *2020 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. 2020, pp. 1–6. DOI: 10.1109/PAINE49178.2020.9337734.

[182] Ayush Jain, Ziqi Zhou, and Ujjwal Guin. “TAAL: Tampering Attack on Any Key-Based Logic Locked Circuits”. In: 26.4 (Mar. 2021). DOI: 10.1145/3442379.

[183] Elisaweta Masserova et al. “Logic Locking-Connecting Theory and Practice”. In: *Cryptology ePrint Archive* (2022).

[184] Leslie G. Valiant. “Universal Circuits (Preliminary Report)”. In: *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*. STOC ’76. Hershey, Pennsylvania, USA: Association for Computing Machinery, 1976, pp. 196–203. DOI: 10.1145/800113.803649.

[185] Jitendra Bhandari et al. *Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction*. 2021. DOI: 10.48550/ARXIV.2111.04222.

[186] Gaurav Kolhe et al. “Breaking the Design and Security Trade-off of Look-up-Table–Based Obfuscation”. In: *ACM Trans. Des. Autom. Electron. Syst.* 27.6 (June 2022). DOI: 10.1145/3510421.

[187] Gaurav Kolhe et al. “Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality”. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2019, pp. 1–8. DOI: 10.1109/ICCAD45719.2019.8942100.

[188] J. Rajendran et al. “Security analysis of logic obfuscation”. In: *DAC Design Automation Conference 2012*. 2012, pp. 83–89. DOI: 10.1145/2228360.2228377.

[189] S. Dupuis et al. “A novel hardware logic encryption technique for thwarting illegal overproduction and Hardware Trojans”. In: *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. 2014, pp. 49–54. DOI: 10.1109/IOLTS.2014.6873671.

[190] Yang Xie and Ankur Srivastava. “Mitigating SAT Attack on Logic Locking”. In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. Ed. by Benedikt Gierlichs

and Axel Y. Poschmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 127–146.

[191] Y. Xie and A. Srivastava. “Anti-SAT: Mitigating SAT Attack on Logic Locking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.2 (2019), pp. 199–207. DOI: 10.1109/TCAD.2018.2801220.

[192] Nikolaos Karousos et al. “Weighted logic locking: A new approach for IC piracy protection”. In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 221–226. DOI: 10.1109/IOLTS.2017.8046226.

[193] Meng Li et al. “Provably Secure Camouflaging Strategy for IC Protection”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.8 (2019), pp. 1399–1412. DOI: 10.1109/TCAD.2017.2750088.

[194] Muhammad Yasin et al. “Provably-Secure Logic Locking: From Theory To Practice”. In: CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1601–1618. DOI: 10.1145/3133956.3133985.

[195] Abhrajit Sengupta et al. “ATPG-based cost-effective, secure logic locking”. In: *2018 IEEE 36th VLSI Test Symposium (VTS)*. 2018, pp. 1–6. DOI: 10.1109/VTS.2018.8368625.

[196] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. “SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware”. In: *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. GLSVLSI ’18. Chicago, IL, USA: Association for Computing Machinery, 2018, pp. 153–158. DOI: 10.1145/3194554.3194596.

[197] Sarah Amir et al. “Development and Evaluation of Hardware Obfuscation Benchmarks”. In: *Journal of Hardware and Systems Security* 2.2 (June 2018), pp. 142–161. DOI: 10.1007/s41635-018-0036-3.

[198] Leon Li and Alex Orailoglu. “Shielding Logic Locking from Redundancy Attacks”. In: *2019 IEEE 37th VLSI Test Symposium (VTS)*. 2019, pp. 1–6. DOI: 10.1109/VTS.2019.8758671.

[199] D. Šišejković et al. “Inter-Lock: Logic Encryption for Processor Cores Beyond Module Boundaries”. In: *2019 IEEE European Test Symposium (ETS)*. 2019, pp. 1–6. DOI: 10.1109/ETS.2019.8791528.

[200] D. Šišejković, F. Merchant, and R. Leupers. “Protecting the Integrity of Processor Cores with Logic Encryption”. In: *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. 2019, pp. 424–425. DOI: 10.1109/SOCC46988.2019.1570564157.

[201] Yuqiao Zhang et al. “TGA: An Oracle-Less and Topology-Guided Attack on Logic Locking”. In: *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*. ASHES’19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 75–83. DOI: 10.1145/3338508.3359576.

[202] Bicky Shakya et al. “CAS-Lock: A Security-Corruptibility Trade-off Resilient Logic Locking Scheme”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.1 (Nov. 2019), pp. 175–202. DOI: 10.13154/tches.v2020.i1.175-202.

[203] Hsiao-Yu Chiang et al. “LOOPLock: Logic Optimization-Based Cyclic Logic Locking”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2178–2191. DOI: 10.1109/TCAD.2019.2960351.

[204] Abhrajit Sengupta et al. “Truly Stripping Functionality for Logic Locking: A Fault-Based Perspective”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (2020), pp. 4439–4452. DOI: 10.1109/TCAD.2020.2968898.

[205] Yuntao Liu et al. “Strong Anti-SAT: Secure and Effective Logic Locking”. In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. 2020, pp. 199–205. DOI: 10.1109/ISQED48828.2020.9136983.

[206] Nimisha Limaye et al. "Thwarting All Logic Locking Attacks: Dishonest Oracle with Truly Random Logic Locking". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020), pp. 1–1. DOI: 10.1109/TCAD.2020.3029133.

[207] Xiang-Min Yang et al. "LOOPLock 2.0: An Enhanced Cyclic Logic Locking Approach". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.1 (2022), pp. 29–34. DOI: 10.1109/TCAD.2021.3053912.

[208] Yuntao Liu et al. "Robust and Attack Resilient Logic Locking with a High Application-Level Impact". In: *J. Emerg. Technol. Comput. Syst.* 17.3 (May 2021). DOI: 10.1145/3446215.

[209] Abdulrahman Alaql, Md Moshiur Rahman, and Swarup Bhunia. "SCOPE: Synthesis-Based Constant Propagation Attack on Logic Locking". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.8 (2021), pp. 1529–1542. DOI: 10.1109/TVLSI.2021.3089555.

[210] Abdulrahman Alaql et al. "LeGO: A Learning-Guided Obfuscation Framework for Hardware IP Protection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.4 (2022), pp. 854–867. DOI: 10.1109/TCAD.2021.3075939.

[211] Rakibul Hassan et al. "A Neural Network-Based Cognitive Obfuscation Toward Enhanced Logic Locking". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4587–4599. DOI: 10.1109/TCAD.2021.3138686.

[212] Shaza Elsharief et al. *IsoLock: Thwarting Link-Prediction Attacks on Routing Obfuscation by Graph Isomorphism*. Cryptology ePrint Archive, Paper 2022/1752. https://eprint.iacr.org/2022/1752. 2022.

[213] K. Xiao and M. Tehranipoor. "BISA: Built-in self-authentication for preventing hardware Trojan insertion". In: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2013, pp. 45–50. DOI: 10.1109/HST.2013.6581564.

[214] K. Xiao, D. Forte, and M. Tehranipoor. "A Novel Built-In Self-Authentication Technique to Prevent Inserting Hardware Trojans". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (2014), pp. 1778–1791. DOI: 10.1109/TCAD.2014.2356453.

[215] P. Ba et al. "Hardware Trust through Layout Filling: A Hardware Trojan Prevention Technique". In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2016, pp. 254–259. DOI: 10.1109/ISVLSI.2016.22.

[216] Richard Jarvis and Michael McIntyre. *Split manufacturing method for advanced semiconductor circuits*. U.S. Patent no 20040102019A1, May 2004.

[217] Yajun Yang et al. "How Secure Is Split Manufacturing in Preventing Hardware Trojan?" In: 25.2 (Mar. 2020). DOI: 10.1145/3378163.

[218] J. Rajendran, O. Sinanoglu, and R. Karri. "Is split manufacturing secure?" In: *2013 Design, Automation & Test in Europe Conference Exhibition (DATE)*. 2013, pp. 1259–1264. DOI: 10.7873/DATE.2013.261.

[219] Tiago D. Perez and Samuel Pagliarini. "A Survey on Split Manufacturing: Attacks, Defenses, and Challenges". In: *IEEE Access* 8 (2020), pp. 184013–184035. DOI: 10.1109/ACCESS.2020.3029339.

[220] Jeyavijayan Rajendran et al. "Security Analysis of Integrated Circuit Camouflaging". In: *Proceedings of the 2013 ACM SIGSAC Conference on Compute and Communications Security*. CCS '13. Berlin, Germany: Association for Computing Machinery, 2013, pp. 709–720. DOI: 10.1145/2508859.2516656.

[221] Satwik Patnaik et al. "Obfuscating the Interconnects: Low-Cost and Resilient Full-Chip Layout Camouflaging". In: *Proceedings of the 36th International Conference*

on Computer-Aided Design. ICCAD '17. Irvine, California: IEEE Press, 2017, pp. 41–48.

[222] B. Erbagci et al. "A secure camouflaged threshold voltage defined logic family". In: *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2016, pp. 229–235. DOI: 10.1109/HST.2016.7495587.

[223] Lap Wai Chow et al. *Camouflaging a standard cell based integrated circuit*. U.S. Patent no 8151235B25, April 2012.

[224] Bicky Shakya et al. "Covert Gates: Protecting Integrated Circuits with Undetectable Camouflaging". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.3 (May 2019), pp. 86–118. DOI: 10.13154/tches.v2019.i3.86-118.

[225] Sarah Amir et al. "Comparative Analysis of Hardware Obfuscation for IP Protection". In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. GLSVLSI '17. Banff, Alberta, Canada: Association for Computing Machinery, 2017, pp. 363–368. DOI: 10.1145/3060403.3060495.

[226] A. Chakraborty et al. "Keynote: A Disquisition on Logic Locking". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 1952–1972. DOI: 10.1109/TCAD.2019.2944586.

cyberagentur