

# Chapter 2

## Security Rule Check

Adib Nahiyani, Kan Xiao, Domenic Forte, and Mark Tehranipoor

### 2.1 Introduction

With the emergence of information technology and its critical role in our daily lives, the risk of cyber attacks is larger than ever before. Many security systems or devices have critical assurance requirement. Their failure may endanger human life and environment, cause serious damage to critical infrastructure, hinder personal privacy, and undermine the viability of whole business sectors. Even the perception that a system is more vulnerable than it really is (e.g., paying with a credit card over the Internet) can significantly impede economic development. The defense against intrusion and unauthorized use of resources with software has gained significant attention in the past. Security technologies, including antivirus, firewall, virtualization, cryptographic software, and security protocols, have been developed to make systems more secure.

While the battle between software developers and hackers has raged since the 1980s, the underlying hardware was generally considered safe and secure. However, in the last decade or so, the battlefield has expanded to hardware domain, since emerging attacks on hardware are shown to be more effective and efficient than traditional software attacks in some aspects. For example, while the cryptographic algorithms have been improved and become extremely difficult (if not impossible) to break mathematically, their implementations are often not. It has been demonstrated that the security of cryptosystems, system on chips (SoCs), and microprocessor circuits can be compromised using timing analysis attacks [1], power analysis

---

A. Nahiyani (✉) • D. Forte • M. Tehranipoor  
University of Florida, Gainesville, FL, USA  
e-mail: [adib1991@ufl.edu](mailto:adib1991@ufl.edu); [dforte@ece.ufl.edu](mailto:dforte@ece.ufl.edu); [tehranipoor@ece.ufl.edu](mailto:tehranipoor@ece.ufl.edu)

K. Xiao  
Intel, Santa Clara, CA, USA  
e-mail: [kan.xiao@intel.com](mailto:kan.xiao@intel.com)

attacks [2], exploitation of design-for-test (DFT) structures [3–5], and fault injection attacks [6]. These attacks can effectively bypass the security mechanisms built in the software level and put devices or systems at risk. These hardware based attacks aim to exploit the vulnerabilities in the design which are introduced either unintentionally or intentionally during the IC design flow.

Many security vulnerabilities in ICs can be unintentionally created by design mistakes and designer's lack of understanding of security problems. Further, today's CAD tools are not equipped with understanding security vulnerabilities in integrated circuits. Therefore, a tool can introduce additional vulnerabilities in the circuit [7, 8]. These vulnerabilities can facilitate attacks such as fault injection or side-channel based attacks. Also, these vulnerabilities can cause sensitive information to be leaked through observable points which are accessible to an attacker or give unauthorized access to an attacker to control or affect a secured system.

Vulnerabilities can also be intentionally introduced in ICs in the form of malicious modifications, referred to as hardware Trojans [9]. Due to short time-to-market constraints, design houses are increasing being dependent on third party to procure IPs. Also, due to the ever increasing cost of manufacturing ICs, design houses rely on untrusted foundry and assembly for fabricating, testing, and packaging ICs. These untrusted third party IP owners or foundries can insert hardware Trojans to create backdoors in the design through which sensitive information can be leaked and other possible attacks (e.g., denial of service, reduction in reliability, etc.) can be performed.

It is of paramount importance to identify security vulnerabilities during hardware design and validation process, and address them as early as possible due to the following reasons: (1) there is little or no flexibility in changing or updating post-fabricated integrated circuits; (2) The cost of fixing a vulnerability found at later stages during the design and fabrication processes is significantly higher following the well-known rule-of-ten (the cost of detecting a faulty IC increases by an order of magnitude as we advances through each stage of design flow). Moreover, if a vulnerability is discovered after manufacturing while the IC is in-field, it may cost a company millions of dollars in lost revenues and replacement costs.

Identifying security vulnerabilities requires extensive knowledge of hardware security, which design engineers lack due to the high complexity and diversity of hardware security issues. Hence, hardware security engineers are required to analyze circuit implementations and specification, and identify potential vulnerabilities. This requires engineers with significant knowledge of different vulnerabilities stemming from diverse set of existing and emerging attacks. It is prohibitively expensive for a design house to maintain a large team of security experts with high expertise while the growing complexity of modern designs significantly increases the difficulty of manual analysis of security vulnerabilities. Poor security check could result in unresolved security vulnerabilities along with large design overhead, development time, and silicon cost [10].

Such limitations suggest to automate the process of security vulnerability analysis during design and validation phases. In this chapter, we present a framework, called Design Security Rule Check (DSeRC) [11], to be integrated in the

conventional design flow to analyze vulnerabilities of a design and assess its security at various stages of design process, namely register transfer level (RTL), gate-level netlist, design-for-test (DFT) insertion, physical design, etc. DSeRC framework is intended to be used as security subject matter expert for design engineers. To achieve this, one needs a comprehensive list of vulnerabilities for ICs. The vulnerabilities are then tied with rules and metrics, so that each vulnerability can be quantitatively measured. The DSeRC framework will allow the semiconductor industry to systematically identify vulnerabilities and security issues before tape-out in order to include proper countermeasures or refine the design to address them.

The rest of the chapter is organized as follows: In Sect. 2.2, we present what are the security assets and attack models. Here, we also talk about who are potential adversaries and how they can get unauthorized access to assets. In Sect. 2.3, we present the proposed DSeRC framework, vulnerabilities, and the associated metrics rules. We present in detail how the vulnerabilities are introduced at different stages of design process. We also present a brief overview of the rules and metrics which are required to quantitatively analyze vulnerabilities. In Sect. 2.4, we discuss the required tasks for the development DSeRC framework. Finally, Sect. 2.5 concludes the chapter.

## 2.2 Security Assets and Attack Models

To build a secure integrated circuit, a designer must decide what assets to protect, and which of the possible attacks to investigate for. Further, IC designers must also understand who the players (attackers and defenders) are in the IC design supply chain and have the means for quickly evaluating the security vulnerabilities and the quality of the countermeasures against a set of well-defined rules and metrics. Three fundamental security factors (security assets, potential adversaries, and potential attacks) are associated with security checks in integrated circuits, which are discussed in the following.

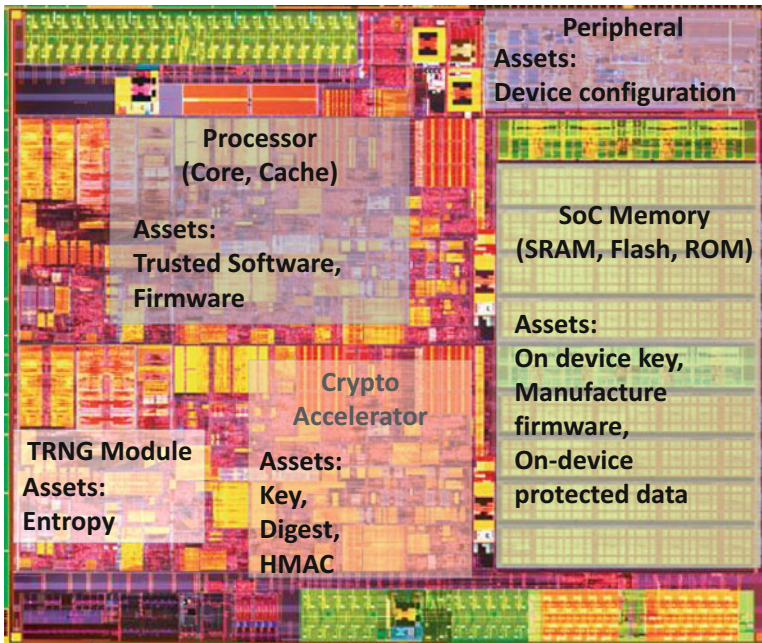
### 2.2.1 Asset

As defined in [10], asset is a resource of value which is worth protecting with respect to the adversary. An asset may be a tangible object, such as a signal in a circuit design, or may be an intangible asset, such as controllability of a signal. Sample assets that must be protected in an SoC are listed below [12]:

- **On-device key:** Secret key, e.g., private key of an encryption algorithm. These assets are stored on chip in some form of non-volatile memory. If these are breached, then the confidentiality requirement of the device will be compromised.

- **Manufacture firmware:** Low level program instructions, proprietary firmware. These assets have intellectual property values to the original manufactures and compromising these assets would allow an attacker to counterfeit the device.
- **On-device protected data:** Sensitive data such as user's personal information and meter reading. An attacker can invade someone's privacy by stealing these assets or can benefit himself/ herself by tampering with these assets (meter reading).
- **Device configuration:** Configuration data determining which resources are available to users. These assets determine which particular services or resources are available to a particular user and an attacker may want to tamper with these assets to gain illegal access to these resources.
- **Entropy:** Random numbers generated for cryptographic primitives, e.g., initializing vector or cryptographic key generation. Successful attacks on these assets would weaken cryptographic strength of a device.

The security assets are known to the hardware designers based on the target specifications of a design. For example, a designer knows that the private encryption key used by the crypto-module is an asset and also knows where the key is located in the SoC. Different types of assets and their locations in an SoC are shown in Fig. 2.1.



**Fig. 2.1** Assets in SoC. *Source: Intel*

### 2.2.2 *Potential Access to Assets*

The aim of an attack is usually to obtain access to assets. There are three types of attacks to gain access to assets depending on attackers' abilities: remote attacks, non-invasive physical attacks, and invasive physical attacks.

**Remote Attacks** In this case an attacker has no physical access to the device; i.e., the attacker cannot touch the device. The attacker can still perform timing [13] and electromagnetic [14] side-channel attacks to remotely extract private key from devices such as smartcards. It has also been demonstrated that an attacker can remotely access the JTAG port and compromise the secret key stored in smartcard of a set-top box [15].

It is also possible to remotely access the scan structure of a chip. For example, in automotive applications, the SoC controlling critical functions such as breaks, power-train, air bags go into "test-mode" every time the car is turned off or on. This key-off/on tests ensure that these critical systems are tested and working correctly before every drive. However, modern cars can be remotely turned on or off, either by trusted parties such as roadside assistance operators or by malicious parties as shown in recent news. Remotely turning the car on or off, allows access to the SoC's test mode which can be used to obtain information from the on-chip memory or force unwanted functions.

Remote attacks also include those that utilize the weakness of hardware, such as buffer overflow, integer overflow, heap corruption, format string, and globbing [16].

**Non-Invasive Physical Attacks** Such attacks are usually of low-budget and do not cause the destruction of the device under attack. Basic attacks consist of using the primary inputs and outputs to take advantage of security weaknesses in the design to obtain sensitive information. Additionally, more advanced attacks use JTAG debug, boundary scan I/O, and DFT structures to monitor and/or control system intermediate states, or snoop bus lines and system signals [4]. Other attacks consist of injecting faults to cause an error during computation of cipher algorithms and exploit the faulty results to extract the asset (e.g., private key).

**Semi-Invasive Physical Attacks** Semi-invasive attacks fall between non-invasive and invasive physical attacks. These attacks present a greater threat because they are more effective than non-invasive attacks but can be performed at a much lower cost than an invasive physical attack. Semi-invasive physical attacks require partial depackaging the chip to get access to its surface; but unlike invasive attacks, these attacks do not require complete removal of the internal layers of the chip. Such attacks include injecting faults to modify SRAM cells content or change the state of a CMOS transistor and gain control of a chip's operation or bypass its protection mechanisms [17].

**Invasive Physical Attacks** These fall under the most sophisticated and expensive attacks and require advanced technical skills and equipment. In these attacks, chemical processes or precision equipment can be used to physically remove

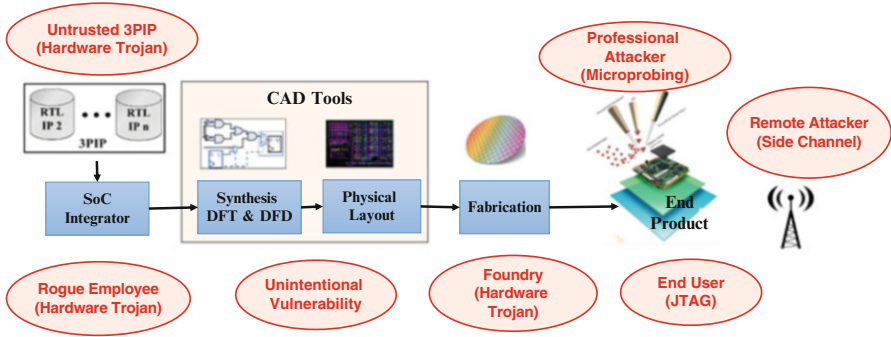


Fig. 2.2 Potential adversaries in different stages of SoC design process

micrometer thin layers of the device. Micro-probes can then be used to read values on data buses or inject faults into internal nets in the device to activate specific parts and extract information. Such attacks are usually invasive, require total access to the device, and result in destruction of the device.

### 2.2.3 Potential Adversary for Intentional Attacks

It is important to understand the potential adversaries who would utilize the security vulnerabilities to perform attacks. This may help designers in comprehending adversaries' capabilities and choosing right countermeasures depending on the target adversary. Adversary might be an individual or a party who intend to acquire, damage, or disrupt an asset for which he/she does not have permission to access. Considering an integrated circuit design process and entities involved in it, adversaries can be categorized into *insiders* and *outsiders*. Figure 2.2 shows the potential adversaries in different stages of SoC design process.

**Insiders** The design and manufacturing of integrated circuits have become more sophisticated and globally distributed with a higher possibility of being attacked by insiders who understand details of the design. An insider could be a rogue employee who work for design house, system integrator or could be a untrusted 3PIP, or a foundry. An insider:

- Has direct access to the SoC design either as an RTL or gate-level netlist, or as a GDSII layout file.
- Has high technical knowledge from the assumption that he/she is employed by a company in the IC design and supply chain.
- Has the capability to make modifications to the design, e.g., inserting hardware Trojans [9, 18]. These hardware Trojans can cause denial of service or create backdoors in the design through which sensitive information can be leaked. Other possible insider attacks are reduction in circuit reliability by manipulating circuit parameters, asset leakage, etc.

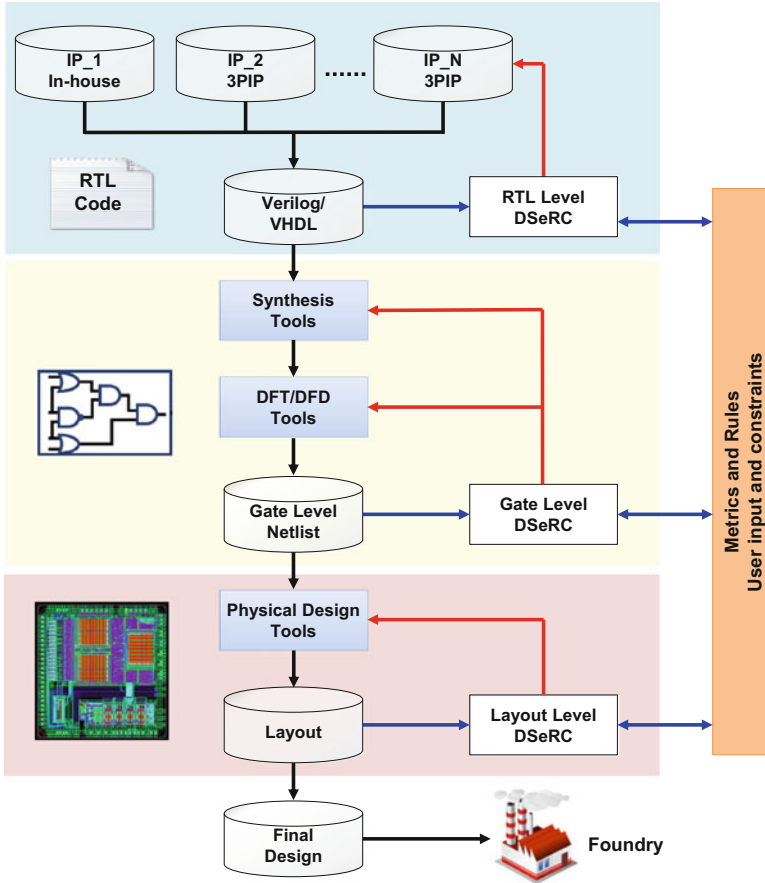
**Outsiders** This class of attackers are assumed to have access to end products in the market (e.g., a packaged IC). Outsider attackers can be divided into three groups based on their capabilities:

- **Remote hacker:** These attackers have no physical access to the device and must employ remote attacks described in Sect. 2.2.2, although they may have physical access to a similar device to develop their attack strategy. These attackers generally rely on exploiting software/hardware vulnerabilities, user errors, and design bugs to gain access to assets.
- **End user:** This group of attackers typically aim to gain free access to contents and services. In this case, the curious end users may rely on techniques already developed by professional attackers to carry their attack. For example, some hobbyists may find a way to jailbreak iPhone or Xbox gaming consoles and post the procedure on social media allowing end users with less expertise to duplicate the process. Jailbreaking allows users to install jailbreak programs and make Apple or Microsoft lose profits [19].
- **Professional attacker:** The most technically capable attackers are security experts or state-sponsored attackers whose motives are driven by financial or political reasons. These groups are capable of executing all types of attacks, including the more expensive invasive attacks described in Sect. 2.2.2, which requires removing the chip package and probing internal signals.

An *Insider* can more easily introduce or exploit the vulnerabilities in a design compared to an *outsider*. The main challenge for an *outsider* to perform an attack is that the internal functionality of the design is not known to the attacker. An *outsider* can reverse engineer the functionality of a chip but this technique would require extensive resource and time.

## 2.3 DSeRC: Design Security Rule Check

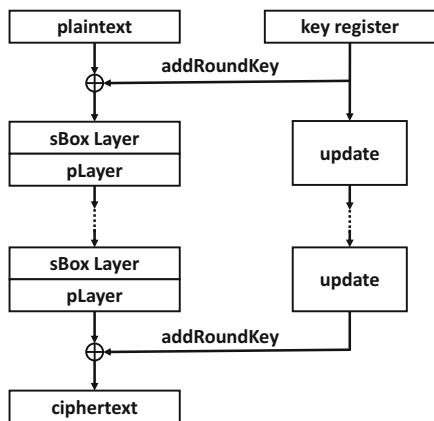
In order to identify and evaluate vulnerabilities associated with ICs, Design Security Rule Check (DSeRC) framework is to be integrated into the conventional digital IC design flow, as shown in Fig. 2.3. DSeRC framework will read the design files, constraints, and user input data, and check for vulnerabilities at all levels of abstraction (RTL, gate level, and physical layout level). Each of the vulnerabilities is to be tied with a set of rules and metrics so that each design's security can be quantitatively measured. At RTL, the DSeRC framework will assess the security of IPs which are either developed in-house or procured from third party (3P); and will provide feedback to design engineers so that the identified security issues can be addressed. After resolving the security vulnerabilities at RTL, the design will be synthesized to gate level and design-for-test (DFT) and design-for-debug (DFD) structures will be inserted. Then, DSeRC framework will analyze the gate-level netlist for security vulnerabilities. The same process will continue for the physical layout design as well. Through this process, the DSeRC framework will



**Fig. 2.3** DSeRC framework

allow the designers to identify and address security vulnerabilities at earliest design steps. This will significantly improve the security of ICs as well as considerably reduce the development time and cost by reducing the time-to-market constraint. Also, the DSeRC framework will allow the designer to quantitatively compare different implementation of same design and thereby allow the designer to optimize performance without compromising security.

The DSeRC framework will need some input from the designer. For example, the security assets need to be specified by hardware designers based on the target specifications of a design. The security vulnerabilities and the corresponding metrics and rules required for the development of the DSeRC framework are discussed in the following subsections.



(a)

```

module PRESENT_ENCRYPT (
    output [63:0] odat, // data output
    input [63:0] idat, // data input
    input [79:0] key, // key input
    input load, // data load
    input clk // clock
);

//-----wires, registers-----
reg [79:0] kreg; // key register
reg [63:0] dreg; // data register
...
// Load/reload key into key register
always @(posedge clk)
begin
    if (load)
        kreg <= key;
    else
        kreg <= kdat2;
end
end

```

(b)

**Fig. 2.4** Unintentional vulnerabilities created by design mistakes. (a) top-level description of PRESENT [20], (b) verilog implementation of PRESENT (<http://opencores.org/>)

## 2.3.1 Vulnerabilities

### 2.3.1.1 Sources of Vulnerabilities

Vulnerability in an IC means a weakness which allows an adversary to access the assets by performing some form of attack. Sources of the vulnerabilities in ICs are divided into following categories:

**Vulnerabilities Due to Design Mistakes** Traditionally the design objectives are driven by cost, performance, and time-to-market constraints; while security is generally neglected during the design phase. Additionally, security-aware design practices do not yet exist. Thus, IPs developed by different designers and design teams may present different set of vulnerabilities. We illustrate this further with a case study below.

Figure 2.4a shows the top-level description of PRESENT encryption algorithm [20]. A segment of its Verilog implementation is shown in Fig. 2.4b. We can see that the key is directly being assigned to the register, defined as “kreg” in the module. Although the encryption algorithm itself is secure, a vulnerability is unintentionally created in its hardware implementation. When this design is implemented, the “kreg” register will be included in the scan chain and an attacker can gain access to key through scan chain based attack [4].

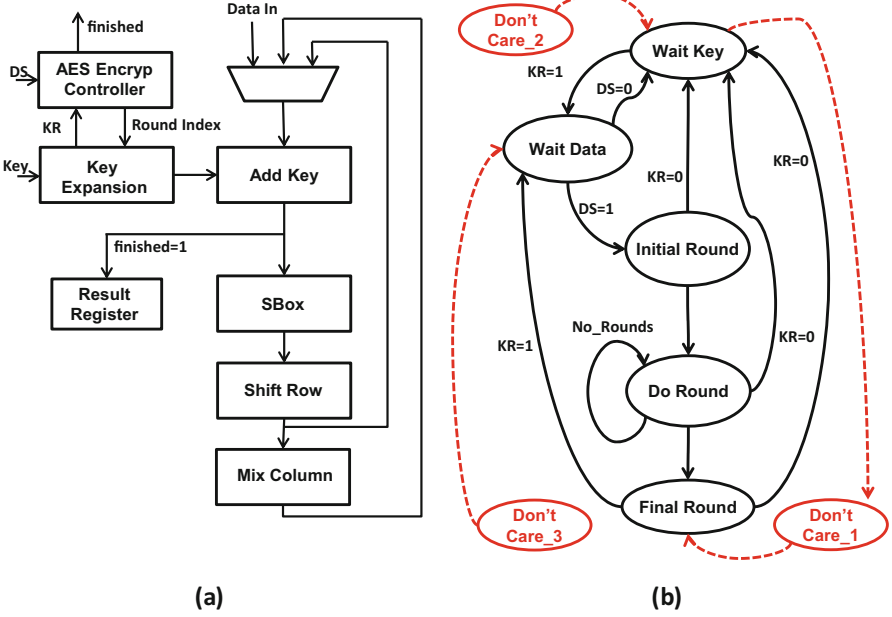
Also, different implementation style of a same algorithm can have different levels of security. In a recent study [21], two AES SBox architectures, PPRM1 [22] and Boyar-Peralta [23], have been analyzed to evaluate which design is more susceptible to fault injection attack. The analysis showed that P-AES is more vulnerable to fault injection attack than the B-AES architecture.

**Vulnerabilities Introduced by CAD Tools** In the IC design process, CAD tools are extensively used for synthesis, DFT insertion, automatic place and route, etc. These tools are not equipped with understanding of the security vulnerabilities in ICs and can therefore introduce additional vulnerabilities in the design. As an example, synthesis tools can create new vulnerabilities in a design when the tool synthesizes the design from RTL to gate level. In the RTL specification of a finite state machine (FSM) there are don't-care conditions where the next state or the output of a transition is not specified. During the synthesis process, the synthesis tool tries to optimize the design by introducing deterministic states and transitions for the don't-care conditions. The introduction of don't-care states and transitions by the CAD tool can create vulnerability in the circuit by allowing a protected state to be illegally accessed through the don't-care states and transitions [8].

We use the controller circuit of an AES encryption module (<http://opencores.org/>) as another case study to demonstrate the vulnerability introduced by the CAD tools. The state transition diagram of the FSM shown in Fig. 2.5b implements the AES encryption algorithm on the data path shown in Fig. 2.5a. The FSM is composed of 5 states and each of these states controls specific modules during the ten rounds of AES encryption. After ten rounds, the “Final Round” state is reached and the FSM generates the control signal *finished* = 1 which stores the result of the “Add Key” module (i.e., the ciphertext) in the “Result Register.” For this FSM, the “Final Round” is a protected state because if an attacker can gain access to the “Final Round” without going through the “Do Round” state, then premature results will be stored in “Result Register,” potentially leaking the secret key. Now, during the synthesis process if a don't-care state is introduced that has direct access to a protected state, then it can create vulnerability in the FSM by allowing the attacker to utilize this don't-care state to access the protected state. Let us consider that the “Don't-Care\_1” state shown in Fig. 2.5b is introduced by the synthesis tool and this state has direct access to the protected state “Final Round.” Introduction of “Don't-Care\_1” state represents a vulnerability introduced by the CAD tool because this don't-care state can facilitate fault and Trojan based attack. For example, an attacker can inject a fault to go to state “Don't Care\_1” and access the protected state “Final Round” from this state. The attacker can also utilize the “Don't Care\_1” to implant a Trojan. The presence of this don't-care state gives the attacker a unique advantage because this state is not taken into consideration during validation and testing; therefore it is easier for the Trojan to evade detection.

Additionally, during the synthesis process CAD tools flatten all the modules of the design together and try to optimize the design for power, timing, and/or area. If a secure module (e.g., encryption module) is present in an SoC, design flattening and the multiple optimization processes can lead to merging trusted blocks with those untrusted. These design steps, which the designer has little control of, can introduce vulnerabilities and cause information leakage [24].

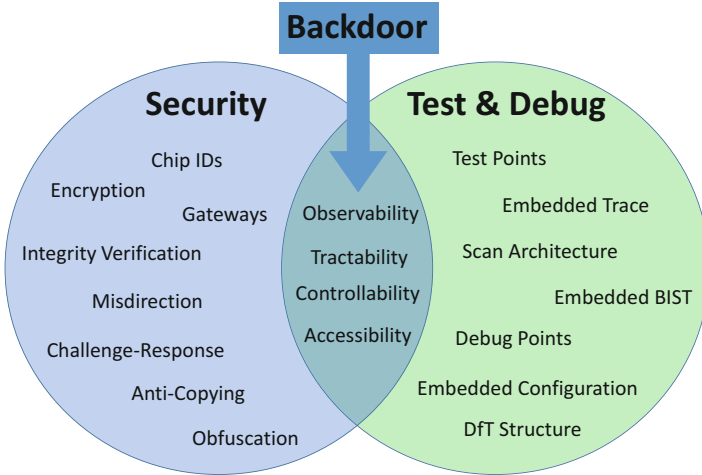
**Vulnerabilities Introduced by DFT and DFD Structure** High testability is important for critical systems to ensure proper functionality and reliability throughout their lifetime. Testability is a measure of controllability and observability of



**Fig. 2.5** Unintentional vulnerabilities created by CAD tools. (a) and (b) show data path and finite state machine (FSM) of AES encryption module. KR and DS stand for Key Ready and Data Stable signal, respectively; the red marked states and transitions represent the don't-care states and transitions introduced by the CAD tool

signals (i.e., nets) in a circuit. Controllability is defined as the difficulty of setting a particular logic signal to “1” or “0” and observability is defined as the difficulty of observing the state of a logic signal. To increase testability and debug, it is very common to integrate design-for-test (DFT) and design-for-debug (DFD) structures in a complex design. However, the increased controllability and observability added by DFT and DFD structures can create numerous vulnerabilities by allowing attackers to control or observe internal states of an IC [25].

In general, test and debug can be viewed as the opposite of security when it comes to accessing circuit internals, as shown in Fig. 2.6. Unfortunately, the DFT and DFD structures cannot be simply avoided in modern designs because of large amount of unexpected defects and errors that occur during the fabrication. Additionally, National Institute of Standards and Technology (NIST) requires that any design used in critical applications needs to be properly testable, both in pre- and post-manufacturing. Therefore, the DFT and DFD structures must be incorporated in ICs, though these structures may create vulnerability. Thus, it is necessary for the DSeRC framework to check whether any security vulnerability is introduced by the DFT and DFD.



**Fig. 2.6** Requirement for high quality test and debug contradicts security

### 2.3.1.2 Vulnerabilities at Different Abstraction Levels

For the development of DSeRC framework, each vulnerability needs to be assigned to one or multiple proper abstraction levels where it can be identified efficiently. Generally, an IC design flow will go through specification, RTL design, gate-level design and consequently physical layout design. DSeRC framework aims at identifying vulnerabilities as early as possible during the design flow because late evaluation can lead to a long development cycle and high design cost. Also, vulnerabilities in one stage, if not addressed, may introduce additional vulnerabilities during transitions from one level to the next. In this section, we categorize vulnerabilities based on the abstraction levels (see Table 2.1).

**Register Transfer Level (RTL)** The design specification is first described in a hardware description language (HDL) (e.g., verilog) to create the RTL abstraction of the design. Several attacks performed at the RTL have been proposed in literature. For example, Fern et al. [26] demonstrated that don't-care assignments in RTL code can be leveraged to implement hardware Trojans that leak assets. In this case, the don't-care assignments in RTL code is the source of vulnerability. Additionally, in the RTL, hardware Trojans are most likely to be inserted at hard-to-control and hard-to-observe parts of the code [27]. Identifying hard-to-control and hard-to-observe parts of the code can help designers assess the susceptibility of the design to Trojan insertion at the RTL.

In general, vulnerabilities identified at the RTL are comparatively easier to address. However, some vulnerabilities, e.g., how susceptible the design is to fault or side-channel attacks, are very challenging if not impossible to identify at this level.

**Gate Level** The RTL specification is synthesized into gate-level netlist using a synthesis tools, e.g., design compiler. At gate level, a design usually is represented with a flattened netlist hence, loses its abstraction; however, more accurate information about the implementation in term of gates or transistors are available. At gate level, hard-to-control and hard-to-observe nets can be used to design hard-to-detect Hardware Trojans [28]. Also, transition from RTL to gate level can introduce additional vulnerabilities by the CAD tools. Examples of vulnerabilities introduced by the tools have been discussed in Sect. 2.3.1. These vulnerabilities need to be analyzed at the gate level.

DFT and DFD structures are generally incorporated in the ICs at the gate level. Therefore, the vulnerabilities introduced by the test and debug structure (see Sect. 2.3.1) need to be analyzed at the gate level.

**Layout Level** Physical layout design is the last design stage before shipping the chip to fabrication, so all the remaining vulnerabilities should be addressed in this level. During layout design, the placement and routing phase gives information about the spatial arrangements of the cells and metal connections in the circuit. In the layout level, power consumption, electromagnetic emanations, and execution time can be accurately modeled. Therefore, vulnerability analysis of side-channel and fault-injection based attacks can be done very accurately at this level. Additionally, some vulnerability analyses, e.g., vulnerability to probing attack [29] can only be done at the layout level. However, any analysis done in this level is very time consuming compared to RTL and gate level.

### 2.3.2 Metrics and Rules

The vulnerabilities that have been discussed so far are to be tied with metrics and rules so that each design's security can be quantitatively measured (see Table 2.1). These rules and metrics of the DSeRC framework can be compared with the design rule check (DRC). In DRC, semiconductor manufacturers convert manufacturing specifications into a series of metrics that enable the designer to quantitatively measure a mask's manufacturability. For the DSeRC framework, each vulnerability needs to be mathematically modeled and the corresponding rule and metric need to be developed so that the vulnerability of a design can be quantitatively evaluated. As for the rules, there can be two types of rules; one type is based on *quantitative metric* and the other type is based on a *binary classification* (yes/no).

We present a brief description of some of the rules and metrics corresponding to the vulnerabilities shown in Table 2.1.

**Asset Leakage** As discussed in Sect. 2.3.1.1, vulnerabilities associated asset leakage can be unintentionally created by design-for-test (DFT), design-for-debug (DFD) structures, CAD tools, and/or by designer's mistake. These vulnerabilities cause violation of information security policies, i.e., confidentiality and integrity policies. Therefore, the metric for identifying these vulnerabilities is *confidentiality*

**Table 2.1** Vulnerabilities, metrics, and rules included in DSeRC

	Vulnerability	Metric	Rule	Attack (Attacker)
RTL Level	Dangerous Don't Cares	Identify all "X" assignments and check if "X" can propagate to observable nodes	"X" assignments should not be propagated to observable nodes	Hardware Trojan Insertion (Insider)
	Hard-to-control & hard-to-observe signal	Statement hardness and signal observability [27]	Statement hardness (signal observability) should be lower (higher) than a threshold value	Hardware Trojan (Insider)
	Asset leakage	Structure checking and information flow tracking	YES/NO: access assets or observe assets	Asset hacking (End user)
	...	...	...	...
Gate Level	Hard-to-control & hard-to-observe net	Net controllability and observability [28]	Controllability and observability should be higher than a threshold value	Hardware Trojan (Insider)
	Vulnerable Finite State Machine (FSM)	Vulnerability factor of fault injection ( $VF_{FI}$ ) & vulnerability factor of Trojan insertion ( $VF_{Tro}$ ) [8]	$VF_{FI}$ and $VF_{Tro}$ should be zero	Fault injection, Hardware Trojan (Insider, end user)
	Asset leakage	Confidentiality and integrity assessment [30]	YES/NO: access assets or observe assets	Asset hacking (End user)
	Design-for-Test (DFT)	Confidentiality and integrity assessment [30]	YES/NO: access assets or observe assets	Asset hacking (End user)
	Design-for-Debug (DFD)	Confidentiality and integrity assessment [30]	YES/NO: access assets or observe assets	Asset hacking (End user)
	...	...	...	...
Layout Level	Side-channel signal	Side-channel vulnerability factor (SVF) [31]	The SVF should be lower than a threshold value	Side-channel attack (End user)
	Micro-probing	Exposed area of the security-critical nets which are vulnerable to micro-probing attack [29]	The exposed area should be lower than a threshold value	Micro-probing attack (Professional attacker)
	Injectable Fault/Error	Timing violation vulnerability factor (TVVF) [21]	TVVF higher than a threshold means the implementation is insecure	Timing based Fault injection attack (End user)
	...	...	...	...

*and integrity assessment.* In [30], authors have presented a framework that validates if confidentiality and integrity policies are being maintained in the SoC. This framework is based on a novel concept of modeling an asset (e.g., a net carrying a secret key) as a fault and leveraging the known test algorithms to detect that fault. A successful detection of the fault indicates that there is flow of information from the asset to an observable point. The rule for this vulnerability will be whether the asset value can be propagated to any observable points. If the assessment result is YES, then there exists vulnerability in the design and the design is not secure.

**Vulnerable FSM** The synthesis process of a finite state machine (FSM) can introduce additional security risks in the implemented circuit by inserting additional don't-care states and transitions. An attacker can utilize these don't-care states and transitions to facilitate fault injection and Trojan attacks. In [8], authors have presented two metrics, named vulnerability factor of fault injection ( $VF_{FI}$ ) and vulnerability factor of Trojan insertion ( $VF_{Tro}$ ) to quantitatively analyze how susceptible an FSM is against fault injection and Trojan attacks, respectively. The higher the values of these two metrics are, the more vulnerable the FSM is to fault and Trojan attacks. For this vulnerability, the rule can be stated as follows; for an FSM design to be secured against fault injection and Trojan insertion attack, the values of  $VF_{FI}$  and  $VF_{Tro}$  should be zero.

**Micro-Probing Attack** Micro-probing is one kind of physical attack that directly probes the signal wires in order to extract sensitive information. This attack has raised serious concerns for security critical applications. In [29], authors have presented a layout-driven framework to quantitatively evaluate a post place-and-route design in terms of exposed area of the security-critical nets which are vulnerable to micro-probing attack. The larger the exposed area, the more vulnerable the SoC is to probing attack. Therefore, the rule for micro-probing vulnerability can be stated as follows, the exposed area to micro-probing should be lower than a threshold value.

**Susceptibility to Trojan Insertion** In [27], authors have presented a metric named “Statement Hardness” to evaluate the difficulty of executing a statement in the RTL code. Areas in a circuit with large value of “Statement Hardness” are more vulnerable to Trojan insertion. Therefore, the metric “Statement Hardness” gives the quantitative measure of a design’s susceptibility to Trojan insertion. Next is to define rule(s) to evaluate if the design is secure. For this vulnerability, the rule can be stated as follows; for a design to be secured against Trojan insertion attack, statement hardness of each statement in the design should be lower than  $SH_{thr}$ . Here,  $SH_{thr}$  is a threshold value that needs to be derived from the area and performance budget.

At gate level, a design is vulnerable to Trojan insertion which can be implemented by adding and deleting gates. To hide the effect of inserted Trojan, an adversary will target *hard-to-detect* areas of the gate-level netlist. *Hard-to-detect* nets are defined as, nets which have low transition probability and are not testable through well-known fault testing techniques (stuck-at, transition delay, path delay,

and bridging faults) [28]. Inserting a Trojan in *hard-to-detect* areas would reduce the probability to trigger the Trojan and thereby, reduce the probability of being detected during verification and validation testing. In [32], authors have proposed metrics to evaluate *hard-to-detect* areas in the gate-level netlist.

**Fault and Side-Channel Attacks** Authors in [21] have introduced Timing Violation Vulnerability Factor (TVVF) to evaluate the vulnerability of a hardware structure to setup time violation attacks which are one subset of fault-injection attacks. In [33], authors have proposed a framework named AMASIVE (Adaptable Modular Autonomous Side-Channel Vulnerability Evaluator) to automatically identify side-channel vulnerabilities of a design. Also, a metric named side-channel vulnerability factor (SVF) has been proposed to evaluate the ICs' vulnerability to power side-channel attacks [31].

Note that the development of the metrics is a challenging task because ideally the metrics must be independent of attack model and application/functionality of a design. For example, an attacker can apply voltage starving or clock glitching based fault injection attacks to obtain the private key of AES or RSA encryption modules. The metric for fault injection needs to provide a quantitative measure of vulnerability for any design (AES or RSA) against any of these attacks (voltage starving or clock glitching). One strategy would be to first identify the root vulnerability that these attacks try to exploit. For this particular example, both voltage starving and clock glitching try to exploit the setup time violation, a common criteria for success of both attacks. Then the framework must evaluate the difficulty of violating setup time for a given design to gain access to the target security assets.

### 2.3.3 Workflow of DSeRC Framework

In this section we describe how the rules and metrics will be used to identify a vulnerability under DSeRC framework. Table 2.1 shows the vulnerabilities and their corresponding metrics and rules that our DSeRC framework covers. The example of PRESENT encryption algorithm will be used to illustrate the workflow of DSeRC framework. The designer will first give the RTL design files and the name of the asset (i.e., "key") as input to DSeRC framework. DSeRC framework will use the information flow tracking to analyze if the "key" can be leaked through any observation points (e.g., registers). As for this design the "key" can be observed through "kreg" register. Therefore, the framework will give a preemptive warning to the designer that if the register "kreg" is included in the DFT structure then, the key can be leaked through scan chain. It will be up to the designer to apply a countermeasure to address this vulnerability before moving to next level of abstraction. One possible countermeasure would be to exclude "kreg" from the scan chain.

After addressing this vulnerability, the design can be synthesized and DFT inserted. The designer will then give the synthesized gate-level netlist to DSeRC and the framework will use the *confidentiality assessment* [30] technique of DSeRC framework to analyze if the “key” can be leaked through any observable point. If the “key” cannot be leaked, then the DSeRC rule will be satisfied and the design can be considered to be secured against asset leakage. On the other hand, if the DSeRC framework identifies the “key” is being leaked to scan flip-flops (observable points), then the framework will raise flag and point the scan flip-flops which carry information about the key. The designer needs to address this vulnerability before moving to physical layout. One possible approach would to apply secure scan structure [34, 35] to counter the vulnerability introduced by the DFT structure.

Note that manually tracking the asset in an SOC to evaluate if it is being leaked through an observable point would be an exhaustive if not impossible task for the designer. On the other hand, DSeRC framework will be able to pinpoint the asset leakage paths allowing the designers to concentrate their analysis effort on those paths and make informed decision.

Some vulnerability can have the common factors, which provide an opportunity to merge multiple vulnerability analyses. As shown in Table 2.1, both asset leakage and fault injection need a metric to evaluate the observability of a net in the circuit structure. Therefore, observability analysis can be executed once for both vulnerability analyses.

While DSeRC may be a logical and necessary step in designing secure ICs, it would not eliminate the need for security subject matter expert. The DSeRC framework is intended to be an automated framework and therefore, may not take the application and use cases of an IC into consideration. For example, the Trojan observability metric will report the difficulty of observing each signal in the design. For Trojan detection, a high observability metric is desired. However, for an encryption module, a high observability metric for the private key will be a serious threat. Therefore, it would be upto the designer to interpret the results generated by the DSeRC framework.

## 2.4 Development of DSeRC Framework

### 2.4.1 Vulnerabilities, Metrics, and Rules

Vulnerability identification is crucial for the development of DSeRC framework. As more attacks are developed, the vulnerabilities exploited by these new attacks need to be identified. And as more vulnerabilities are discovered, their corresponding metrics and rules need to be developed as well. Given the diversity and vastness of security threats, it will take the combined effort of academic and industry researchers to develop a comprehensive set of vulnerabilities and corresponding rules and metrics for the DSeRC framework.

### ***2.4.2 Tool Development***

The DSeRC framework is intended to be integrated with the traditional IC design flow so that security evaluation can be made as an inherent part of the design process. This requires the development of CAD tools which can automatically evaluate the security of a design based on DSeRC rules and metrics. The tools' evaluation times need to be scalable with the design size. Also, the tools should be easy to use and the outputs generated by the tools need to be understandable by the design engineer.

### ***2.4.3 Development of Design Guidelines for Security***

To avoid some common security problems in early design stages, good design practices learned through experience can be used as guidelines. The design guideline is able to guide design engineers what to do (Do-s) and not to do (Don't-s) during the initial design. Do-s and Don't-s are very common in VLSI design and test domain, in order to improve the design quality and testability of faults [36]. These Do-s and Don't-s, such as initializable flip-flops, no asynchronous logic feedback, and no gating of clocks to scan flip-flops, are very straight-forward, but quite effective to make a design testable and thus save time and cost in design cycles. This is also applicable to hardware security. Taking the PRESENT in Fig. 2.4 as an example, if "kreg" does not become part of scan chain, the key leakage problem can be addressed at the design stage. However, to date, no comprehensive design guideline has been developed for hardware security. This can be a research direction where academia and industry can explore.

### ***2.4.4 Development of Countermeasure Techniques***

Knowledge of the vulnerabilities of a design is not sufficient to protect it. A series of low-cost countermeasure techniques are also needed for each vulnerability. One additional extension of the DSeRC framework will be to provide low-cost mitigation techniques for each vulnerability. The suggested countermeasure techniques will be a good hint to design engineers who lack of knowledge in hardware security. The improvement in security after applying the countermeasures can be measured by running the DSeRC check again.

As an example, in [8] authors have proposed a countermeasure to address the vulnerabilities introduced in FSMs by the synthesis tool or design mistake. In their proposed approach the state FFs are to be replaced by "Programmable State FFs." "Programmable State FFs" are defined as the state FFs which go to the Reset/Initial state if the protected state is tried to be accessed by any other state apart from the authorized states. The detailed architecture of the "Programmable State FFs" has been discussed in [8].

## 2.5 Conclusion

In this chapter, we have presented the basic concept of Design Security Rule Check (DSeRC) to analyze vulnerabilities in a design and consequently assess its security level at design stage. One of the main challenges for the development of the DSeRC framework is associated with mathematically modeling vulnerabilities for quantitative evaluation. This is because DSeRC framework needs to validate the security of a design regardless of the application of the design or attack models. Although DSeRC will not eliminate the need for security subject matter experts, it will, however, expedite the security analysis of ICs and SoCs, and increase the design engineer's awareness to the security issues of their designs.

## References

1. P.C. Kocher, Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems, in *Annual International Cryptology Conference* (Springer, Berlin/Heidelberg, 1996), pp. 104–113
2. P.C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *CRYPTO* (1999)
3. D. Hely et al., Scan design and secure chip [secure IC testing], in *Proceedings of the 10th IEEE IOLTS* (July 2004), pp. 219–224
4. J. Lee, M. Tehranipoor, C. Patel, J. Plusquellic, Securing scan design using lock and key technique, in *Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)* (2005)
5. B. Yang, K. Wu, R. Karri, Scan-based side-channel attack on dedicated hardware implementations of data encryption standard, in *Proceedings of International Test Conference* (2004)
6. E. Biham, A. Shamir, Differential fault analysis of secret key cryptosystems, in *CRYPTO* (1997)
7. C. Dunbar, G. Qu, Designing trusted embedded systems from finite state machines. *ACM Trans. Embed. Comput. Syst.* **13**(5s), 1–20 (2014)
8. A. Nahiyani, K. Xiao, K. Yang, Y. Jin, D. Forte, M. Tehranipoor, AVFSM: a framework for identifying and mitigating vulnerabilities in FSMs, in *Proceedings of the 53rd Annual Design Automation Conference* (ACM, 2016), p. 89
9. M. Tehranipoor, F. Koushanfar, A survey of hardware Trojan taxonomy and detection. *IEEE Des. Test Comput.* **27**, 10–25 (2010)
10. ARM inc., Building a secure system using TrustZone technology, [http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C\\_trustzone\\_security\\_whitepaper.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf)
11. K. Xiao, A. Nahiyani, M. Tehranipoor, Security rule checking in IC design. *Computer* **49**(8), 54–61 (2016)
12. Eric Peeters, SoC security architecture: current practices and emerging needs, in *Design Automation Conference (DAC)* (IEEE, 2015), pp. 1–6
13. P. Kocher, Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems, in *Advances in Cryptology* (1996), pp. 104–113
14. T. Korak, T. Plos, Applying remote side-channel analysis attacks on a security-enabled NFC tag, in *Topics in cryptology—CT-RSA 2013* (February 2013)
15. A. Das, J. Da Rolt, S. Ghosh, S. Seys, S. Dupuis, G. Di Natale et al., Secure JTAG implementation using Schnorr protocol. *J. Electron. Test. Theory Appl.* **29**(2), 193–209 (2013)

16. S. Chen et al., Security vulnerabilities: from analysis to detection and masking techniques. *Proc. IEEE* **94**(2), 407–418 (2006)
17. S.P. Skorobogatov, Semi-invasive attacks - a new approach to hardware security analysis, in Technical Report UCAM-CL-TR-630. University of Cambridge Computer Laboratory, April 2005
18. M. Tehranipoor, C. Wang, *Introduction to Hardware Security and Trust* (Springer, New York, 2011)
19. M.A. Harris, K.P. Patten, Mobile device security considerations for small-and medium-sized enterprise business mobility. *Inf. Manage. Comput. Secur.* **22**, 97–114 (2014)
20. A. Bogdanov et al., PRESENT: an ultra-lightweight block cipher, in *Cryptographic Hardware and Embedded Systems* (2007)
21. B. Yuce, N. Ghalaty, P. Schaumont, TVVF: estimating the vulnerability of hardware cryptosystems against timing violation attacks, in *Hardware Oriented Security and Trust* (2015)
22. S. Morioka, A. Satoh, An optimized s-box circuit architecture for low power AES design, in *Cryptographic Hardware and Embedded Systems* (2003)
23. J. Boyar, R. Peralta, A small depth-16 circuit for the AES s-box, in *Proceedings of Information Security and Privacy Research* (2012)
24. T. Huffmire et al., Moats and drawbridges: an isolation primitive for reconfigurable hardware based systems, in *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (2007)
25. J. Rolt et al., Test versus security: past and present. *IEEE Trans. Emerg. Top. Comput.* **2**(1), 50–62 (2013)
26. N. Fern, S. Kulkarni, K. Cheng, Hardware Trojans hidden in RTL don't cares - automated insertion and prevention methodologies, in *International Test Conference (ITC)* (2015)
27. H. Salmani, M. Tehranipoor, Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level, in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (2013), pp. 190–195
28. H. Salmani, R. Karri, M. Tehranipoor, On design vulnerability analysis and trust benchmarks development, in *Proceedings of IEEE 31st International Conference on Computer Design (ICCD)* (2013), pp. 471–474
29. Q. Shi, N. Asadizanjani, D. Forte, M. Tehranipoor, A layout-driven framework to assess vulnerability of ICs to microprobing attacks, in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2016), pp. 155–160
30. G. Contreras, A. Nahiyani, S. Bhunia, D. Forte, M. Tehranipoor, Security vulnerability analysis of design-for-test exploits for asset protection, in *Asia and South Pacific Design Automation Conference* (2017, to appear)
31. J. Demme et al., Side-channel vulnerability factor: a metric for measuring information leakage, in *39th Annual International Symposium on Computer Architecture* (2012), pp. 106–117
32. M. Tehranipoor, H. Salmani, X. Zhang, *Integrated Circuit Authentication: Hardware Trojans and Counterfeit Detection* (Springer, Cham, 2013)
33. S.A. Huss, M. Stottinger, M. Zohner, AMASIVE: an adaptable and modular autonomous side-channel vulnerability evaluation framework, in *Number Theory and Cryptography* (Springer, Berlin, Heidelberg, 2013), pp. 151–165
34. J. Lee, M. Tehranipoor, C. Patel, J. Plusquellic, Securing designs against scan-based side-channel attacks. *IEEE Trans. Dependable Secure Comput.* **4**(4), 325–336 (2007)
35. J. Lee, M. Tehranipoor, J. Plusquellic, A low-cost solution for protecting IPs against scan-based side-channel attacks, in *VLSI Test Symposium* (2006)
36. M. Bushnell, V. Agrawal, in *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits* (Springer, New York, 2000)

Hardware IP Security and Trust

Mishra, P.; Bhunia, S.; Tehranipoor, M.M. (Eds.)

2017, XII, 353 p. 131 illus., 78 illus. in color., Hardcover

ISBN: 978-3-319-49024-3