Leveraging Logic Masking to Facilitate Hardware Trojan Detection Methods

Arash Nejat

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : NANO ELECTRONIQUE ET NANO TECHNOLOGIES

Arrêté ministériel : 25 mai 2016

Présentée par

Arash NEJAT

Thèse dirigée par **Vincent BEROULLE** , Enseignant-chercheur, Grenoble INP
et codirigée par **David HELY**, Maitre de Conférence, Grenoble INP
préparée au sein du **Laboratoire Laboratoire de conception et d'intégration des systèmes**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Tirer parti du masquage logique pour faciliter les méthodes de détection des chevaux de Troie hardware

Leveraging Logic Masking to Facilitate Hardware Trojan Detection Methods

Thèse soutenue publiquement le **22 janvier 2019**,
devant le jury composé de :

Monsieur VINCENT BEROULLE
MAITRE DE CONFERENCES, GRENOBLE INP, Directeur de thèse
Monsieur DAVID HELY
MAITRE DE CONFERENCES, GRENOBLE INP, Co-directeur de thèse
Monsieur JEAN-LUC DANGER
PROFESSEUR, TELECOM PARISTECH, Rapporteur
Monsieur LIONEL TORRES
PROFESSEUR, UNIVERSITE DE MONTPELLIER, Rapporteur
Monsieur LILIAN BOSSUET
PROFESSEUR, UNIVERSITE JEAN MONNET - SAINT-ETIENNE, Examinateur
Monsieur GIORGIO DI NATALE
DIRECTEUR DE RECHERCHE, CNRS DELEGATION LANGUEDOC-ROUSSILLON, Examinateur

Table of Contents

List of Acronyms

ADC	Analog-Digital Converter
BEOL	Back-End-Of-Line
BFS	Breadth First Search
CAD	Computer-Aided Design
COTS	Commercial-Off-The-Shelf
CUTT	Circuit Under Trojan Test
DAC	Digital-Analog Converter
DfTr	Design for Trust
DSP	Digital Signal Processor
FBA	Fault-Based Analysis
FEOL	Front-End-Of-Line
FF	Flip Flop
FPGA	Field Programmable Gate Array
FPR	False Positive Rate
FSM	Finite State Machine
GDSII	Graphic Database System
HD	Hamming Distance
HDL	Hardware Description Language
HDP	Hardware Trojan Detection Probability
HT	Hardware Trojan
I/O	Input/ Output
IC	Integrated Circuit
IDDq	Leakage Current
IDDt	Transient Current
IP	Intellectual Property
IVG	Input Vector Generation
KG	Key-Gate
KI	Key-Input
KL	Kernighan–Lin
LOG SATT	Logic Simulation, Security, And Trust Analysis Tool
MSP	Maximum Shortest Path
MUX	Multiplexer

PCA	Power Consumption Analysis
PDA	Path Delay Analysis
PDF	Probability Distribution Functions
PI	Primary Input
PO	Primary Output
PSI	Pseudo Input
PSO	Pseudo Output
PUF	Physical Unclonable Function
PV	Process Variation
RO	Ring Oscillator
RS	Rare Signal
RTL	Register Transfer Level
SAL	Switching Activity Localization
SoC	System On Chip
SSA	Sum Of Switching Activity
TCA	Trojan To Circuit Activity
TCP	Trojan To Circuit Power
TDP	Trojan Detection Probability
TDR	Trust-Driven Retiming
TRNG	True Random Number Generator

List of Figures

List of Tables

Abstract

The ever-increasing complexity of integrated circuits (ICs) design and manufacturing has necessitated the employment of third parties such as design-houses, intellectual property (IP) providers and fabrication foundries to accelerate and economize the development process. The separation of these parties results in some security threats. Untrustworthy fabrication foundries are suspected of three security threats: hardware Trojans, IP piracy, and IC overproduction. Hardware Trojans are malicious circuitry alterations in IC layouts intended for sabotage objectives.

Some IC design modifications, known as Design-for-Trust (DfTr) have been proposed to facilitate Trojan detection methods or prevent Trojan insertion. In addition, key-based modifications, known as design masking or obfuscation, have been proposed to protect IPs/ICs from IP piracy and IC overproduction. They obscure circuits' functionality by modifying circuits such that they do not correctly work without being fed with a correct key.

In this thesis, we propose three DfTr methods based on leveraging the masking approach to hinder Trojan insertion. The first proposed DfTr method aims to maximize obscurity and simultaneously minimize the rare signal counts in circuits under masking. Rare signals barely have transitions during circuit operations and so the use of them causes hardware Trojans will not be easily activated and detected during circuit tests. The second proposed DfTr facilitates path delay analysis-based Trojan detection methods. Since the delay of shorter paths varies less than longer ones', the objective is to generate fake short paths for nets which only belong to long paths by repurposing the masking elements. Our experiments show that this DfTr method increases the Trojan detectability in modified circuits and also provides the advantages of masking methods. The aim of

the third DfTr method is to facilitate power-analysis-based Trojan detection. In a masked circuit by the proposed method, one has more control over the switching activity of the different circuit parts. For instance, one can target one part of the circuit, increase its switching activity, and simultaneously decrease the other parts' switching activity; consequently, if the target part includes an hardware Trojan, its switching activity and so power consumption rises, although the total power consumption of the circuit goes down due to low switching activity rates in most parts of the circuit. When the circuit consumes less power, the power measurement noise abates. The noise can disturb to observe Trojans' effects on the power consumption of Trojan-infected circuits.

In addition, in this thesis, we introduce a CAD tool that can run various masking algorithms on gate-level netlists. The tool can also perform logic simulation and estimate circuit area, power consumption, and performance at the gate level.

Chapter 1

Introduction

During the last two decades, a fabless business model has emerged into semiconductor industries due to costly fabrication processes of integrated circuits (ICs) [1]. In this model, semiconductor companies outsource the manufacturing part of their designs to third fabrication foundries, shortly called *fabs*. On the other hand, time-to-market and design complexity issues associated with today's system-on-chips (SoCs) have convinced design houses to employ predesigned circuit blocks, called *intellectual properties* (IPs) [1]. However, the separation of design houses, IP developers, and fabs in the fabless model has economic advantages, it entails serious security challenges. For instance, design houses may illegally sell IPs that they purchased from IP developers. Likewise, untrustworthy fabs can also extract IPs from their costumers' layouts and then pirate them. In addition, they can clone or overproduce the layouts [2]. In literature, this threat is known as IP piracy. Untrustworthy IP developers and fabs are suspected of another security threat in which malicious circuits, known as *hardware Trojans*, are added to IPs or fabricated ICs for sabotage objectives. These issues are some security threats in the fabless business model. For each threat, detection and prevention methods have been

proposed in many research work [3]. The focus of this dissertation is on detecting and preventing hardware Trojan and IP piracy threats that may happen in untrustworthy fabrication foundries.

1.1 Hardware Trojan

Hardware Trojans can cause secret information leakage, malfunctions in specific circumstances, or performance downgrade in Trojan-infected ICs [4]. Regarding ICs' application, design, and fabrication technology, various hardware Trojans can be designed. Therefore, different detection methods have been proposed. These methods are based on either *side-channel analysis* or *logic testing*. The former one is the inspection of circuit parameters like power consumption, and logic testing is based on conventional functional/structural tests and includes generating and applying proper input patterns and then observing their results [5]. Any abnormality in measured side-channels or captured outputs can warn the presence of a hardware Trojan.

Side-channel-analysis-based detection methods face two important challenges: process and environmental variations. Process variations happen during IC manufacturing and cause variations in some transistor characteristics such as channel length and oxide-thickness. Environmental variations happen while a circuit is working and induce changes in the operating environment of the circuit such as temperature and supply voltage. Trojans effects on side-channels may be undetectable among effects of process and environmental variations. Logic-testing-based detection methods are also a challenging task because one must generate proper test vectors to catch Trojans effects on the outputs; and skillful Trojan attackers try to use low-controllable and low-

observable signals for Trojan activation and mission, respectively.

The challenges in Trojan detection impose some design modifications on designers to integrate security and trust into fabricated ICs. This approach is entitled Design-for-Trust (DfTr) [6]. Two main categories of DfTr approaches are 1) preventing Trojan insertion that hinders Trojan attackers for inserting hard-to-detect Trojans and 2) facilitating Trojan detection that boosts logic testing and side-channel analysis-based Trojan detection methods. More details and examples of Trojan characteristics, taxonomies, detection methods, and DfTr are presented in Section 2.

1.2 IP Piracy

Untrustworthy design houses and fabs are suspected of the IP piracy threat. One prevention approach for this threat is *design masking* that aims at obscuring the original functionality of a circuit by adding some inputs, gates, and flip-flops to the circuit. A masked circuit correctly functions only if it is fed with a correct specific value, like a correct key. In other words, if someone applies any incorrect key to a masked circuit, it malfunctions. Only authorized users (i.e., IP or IC owners) must have this correct key. Therefore, the piracy of a masked circuit is meaningless as long as the key is not revealed.

Authors have used different names for the mentioned approach, such as *obfuscation*, *encryption*, *locking*, and *masking*. The authors in [11] discussed that the term masking is more accurate than other ones. In this dissertation, hereafter, *state masking* and *logic masking* will be used to refer to the approach at register transfer level (RTL) and gate level, respectively.

1.2.1 State Masking

The first step of design masking is *state masking* that modifies the finite state machine (FSM) of a circuit at RTL [7]. The modified FSM has the original states of the circuit plus some added fake ones. It starts from a new start state on power-up, and one needs to know a specific inputs sequence, as the key, to pass the fake (masked) states and reach the original start state.

In order to interlock the original and fake states, designers have to make several transitions from original states to fake ones, and vice versa [8]. For this aim, one can synthesize the modified FSM to a gate-level netlist (non-optimized and not technology-mapped netlist) and then perform logic masking on the combinational part.

1.2.2 Logic Masking

The design modifications in logic masking usually include adding some inputs and XOR/XNOR gates to the combinational part of a circuit at the gate level. These inputs and gates are usually called *key-input* and *key-gate* in the literature. Applying incorrect value (key) to the key-inputs results in faults at the output of some of the added key-gates. These faults may be propagated to the primary outputs, and then make failures.

Logic (and state) masking methods can hinder Trojan insertion as well, however, their first objective is to prevent IP piracy. Indeed, Trojan attackers cannot easily have enough knowledge about masked circuits. In other words, they need to know the correct key of a masked circuit in order to insert a well-designed hardware Trojan. For instance, without knowing the correct key, an attacker may insert a hardware Trojan in a masked circuit

somehow the Trojan may be activated only when the circuit is fed with an incorrect key [9].

1.3 Major Contributions

This doctoral dissertation presents discussions about IP piracy and Trojan insertion in untrustworthy fabs and how logic masking can counter Trojan threats. The basic objective and attitude in this work are to leverage logic masking to hinder Trojan insertion and facilitate Trojan detection methods based on logic-testing and side-channel analysis. Accordingly, the main contributions of this dissertation include three proposed approaches for three objectives, as follows:

1. Approach 1: Advantaging logic Masking to amELiorate functIOnal/stRuctural testing-bAsed Trojan detectION (*AMELIORATION*).

2. Approach 2: Employing logic maSking to faCilitate pAth deLay Analysis-based Trojan detectION (*ESCALATION*).

Approach 3: REstricting Switching acTivity using LOgic masking to improve poweR Analysis-based Trojan DetectION (*RESTORATION*).

For each proposed approach, logic masking methods and algorithms will be proposed and run on some circuit benchmarks and then the achievement will be measured by comparing both the masked and original circuits. In addition, the quality of masking in some circuits masked by different logic masking algorithms will be measured according to two main criteria: 1) the Hamming distance between correct and incorrect output bits of a masked circuit while applying correct and incorrect keys [12], 2) the number of

mismatch points between an original circuit and its peer masked circuit [9]. First criteria can be measured by applying many input vectors and keys (to the primary inputs and key-inputs, respectively) and then comparing the output vectors of the masked and original circuit, for each applied input vectors. For the second criteria, formal verification tools can be used to compare the original circuit with its peer masked circuit. In the following, each proposed approach is briefly explained.

1.3.1 AMELIORATION

The first proposed approach concerns rare (low-controllable) signals that barely have transitions during the circuit operation either in the normal mode or test mode. A masked circuit by the proposed approach has two challenges for Trojan attackers: 1) lack of knowledge about the original functionality of the being attacked circuit, 2) shortage of rare signals for design Trojan activation mechanisms. In normal circuits, Trojan attackers can easily find rare signals. The use of rare signals causes hardware Trojans will not be easily activated during the logic tests or normal operations [10]. In masked circuits, finding rare signals is difficult, because the key-gates and key-inputs affect the transition rate of the signals. Indeed, the transition rate (and controllability) of key-inputs, like that of the primary inputs, is máximum. Each key-input is directly connected to one input of a key-gate. Thus, the transition rate and controllability of some signals would increase if key-gates are preceded them.

AMELIORATION aims at mystifying circuit original functionality and removing rare signal, simultaneously. It checks all signals of a circuit to find the best ones, according to the aims, for inserting on XOR/XNOR key-gates. To find such signals, a security metric

is proposed that employs "high balanced Hamming distance achievement" and "rare signals elimination". It should be noted that if there is no rare signal in a circuit, the chance of the Trojans activation is increased. The more Trojan activation chance is, the more faults caused by the Trojan will be observed. To avoid this, a Trojan attacker must make rare signals by performing logic operations (such as AND or OR) on normal signals. This production of rare signals increases his Trojan size; therefore, the Trojan affects more on side-channel parameters, such as power consumption or paths' delay [3].

In this work, an algorithm based on approach AMELIORATION has been implemented; and its efficiency has been evaluated in a toolset has been designed and developed using c# programming language. It gets the gate-level netlist of a circuit, performs logic simulations, and applies the proposed approach. The simulation results on ISCAS-85 circuit benchmarks show that the proposed method offers high quality of logic masking while also significantly reduces the number of rare signals. Furthermore, the toolset extracts and reported the amount of area, power consumption, and delay overheads at the gate level.

1.3.2 ESCALATION

The second proposed approach masks the circuit functionality while improving the efficiency of path delay-based Trojan detection methods. Such methods can detect a Trojan if its effects on the delay of the paths of the Trojan-infected circuit are perceivable from among process variation effects. Shorter paths have less delay variation that longer ones, proven in [13]. Thus, apart from logic masking benefits, the objective of ESCALATION is to generate short paths using key-gates and key-inputs for nets that only

belong to long paths.

It should be noted that if all nets of a circuit at least belong to one enough-short path, it will be easier for defenders to detect any potential Trojans. In this condition, in order to hide Trojan delay effects, attackers can increase the drive strength and capacity load of the cells which are before and after the targeted Trojan. These changes increase the Trojan power consumption, and so increase the success of power-analysis-based Trojan detection methods.

An algorithm based on this approach will be implemented in this work. The masked circuits by this algorithm were also synthesized, placed, and routed in order to validate the Trojan detection efficiency of the proposed algorithm at the layout level. The obtained results show that ESCALATION can improve Trojan detection probability in path delay-based detection methods. In addition, the ability of the implemented algorithm to mask circuit functionality will be evaluated based on Hamming distance, as the meter. The results show that the implemented algorithm achieves better Hamming distance than the random masking, presented in Chapter 2.

1.3.3 RESTORATION

The third proposed approach concerns the proportion of Trojan power to circuit power. The power consumption of hardware Trojans will be observable if they raise the power consumption of Trojan-infected circuits to a value more than the expected one. The expected power consumption is different in different states of the circuit. Moreover, the power of two instances of a circuit in two ICs and in the same circuit state are different due to process variations. In circuitry situations that a circuit uses more power (e.g.

because of more internal cells activities), the power variation is more than that of the circuit uses less power, proven in [14]. Thus, in order to increase the success chance of power-analysis-based Trojan detection methods, one has to increase Trojan power and simultaneously decrease circuit power.

Trojans power will probably rise by increasing the switching activity of their host circuit, but it increases the total power consumption of circuit, hence, the circuit power variations are increased. To solve this problem, one must localize switching activity; it means that the switching activity of one small part of IC must be increased while that of other IC parts simultaneously decreased [14].

To localize switching activity in special parts of a circuit, RESTORATION divides the circuit into regions, and then adds one key-gate to each one. In each region, all of the key gates are controlled by one key input, and so the number of key inputs is equal to the number of regions.

An algorithm based on RESTORATION approach was implemented in this work. The algorithm gets the gate-level netlist of a combinational circuit; since there is no physical information in this level, the algorithm runs a portioning algorithm and assigns one key-gate to each partition. The experiment results in this level show that RESTORATION well localizes switching activity for each partition; consequently it can improve Trojan detection probability in power-analysis-based detection methods, as matter of fact that physical synthesis tools in the placement step try to place cells of a partition, which are highly logical-correlated, close to each other with the aim of decreasing costs of routing (wire lengths and so power and performance).

1.4 Dissertation Outline

The rest of this dissertation is organized as follows: Section 2 reviews literature related on both IP piracy and hardware Trojan. Section 3 details the ALEMORATION approach and introduces a CAD tool. Section 4 presents and evaluates the ESCALATION approach and an algorithm based on it. Section 5 explains the RESTORATION approach and proposes a simple algorithm based on that. In addition, it presents the primary results of the algorithm. Finally, Section 6 summarizes the dissertation and discusses future perspectives.

Chapter 2

Background on Hardware Security

2.1 Introduction

Due to the expensive production cost of today's SoCs, the fabless business model has progressively become the main model in the semiconductor industry over the last decade [1, 15]. In this model, semiconductor companies prefer to only design their SoCs layout and then outsource them to a third-party manufacturer, shortly called *fab* [16]. Moreover, semiconductor companies purchase IPs and use them in their SoCs design, in order to decrease the time-to-market and design complexities [16]. Fig. 2.1 shows a SoC including different blocks. These blocks may be purchased from different IP suppliers across various countries. Although the fabless model reduces the final cost of the SoCs production, it faces security challenges among its involved parties [17]. In the following sections, security threats that may happen by involved different third-parties during the SoC life cycle are presented.

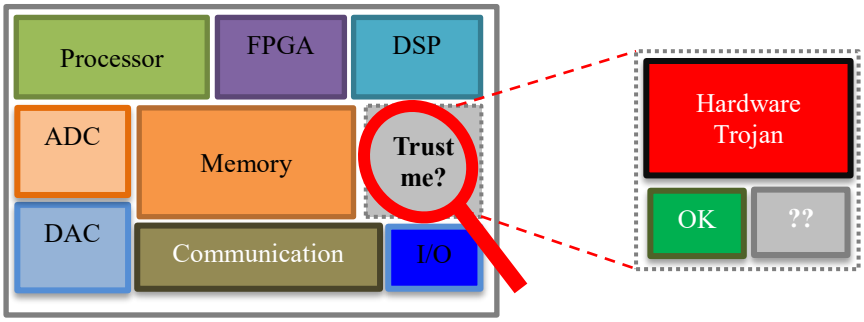


Fig. 2.1 A SoC including trusted, Trojan-infected, and suspicious IPs [18, 19]

2.1.1 SoC Life Cycle

The life cycle of SoCs includes six periods [18, 19], shown in Fig. 2.2.a. In the first period, the IP design period, IP suppliers prepare IPs which could be further integrated into SoCs. During designing an IP, its developer may buy and use IPs from other suppliers. IPs are offered in three forms: 1) synthesizable RTL specifications, 2) generic gate-level netlists, and 3) placed and routed transistors in layout formats (like GDSII) [20].

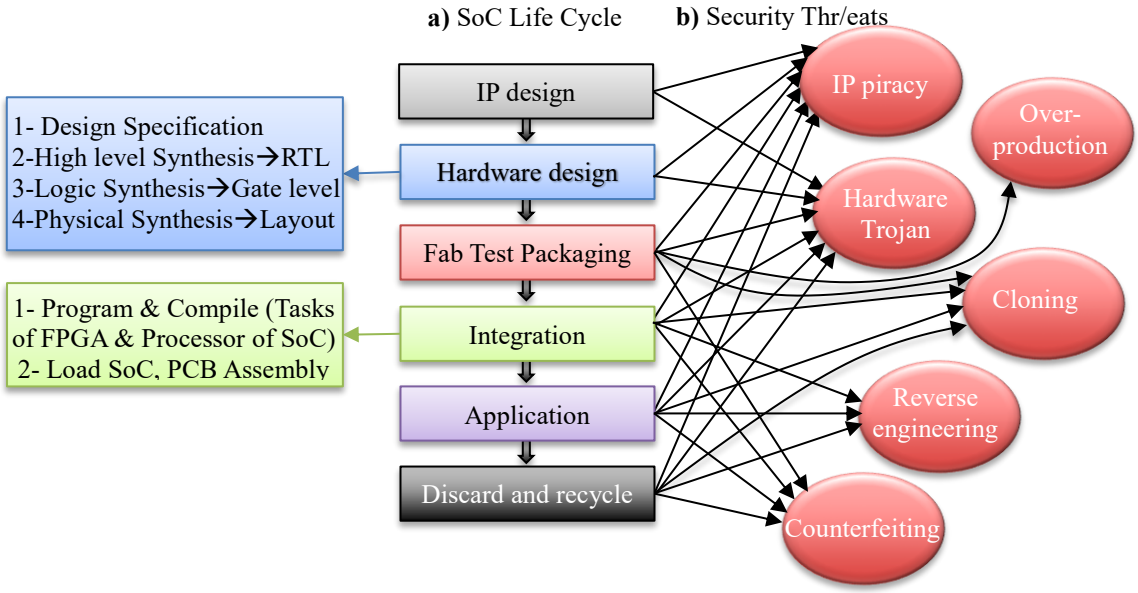


Fig. 2.2 a) Six periods of the SOC life cycle and **b)** possible security threats [18-19]

The second period of the SoC life cycle is the hardware design period. In this period, a design house starts to design the hardware part of a SoC, from a high-level description to the layout level. The hardware design flow includes some usual steps, shown in the left side of Fig. 2.2.a [19]. At first, the design house specifies the description of the intended SoC. According to this specification, the design house needs to prepare different circuit blocks. They implement some of these blocks; the other blocks have to be purchased from IP suppliers. During the implementation of the blocks, the design house has to also decide about the required technology and perform logical and physical synthesis. As industrial logical/physical synthesis tools are expensive, they may decide to outsource the RTL/gate-level specifications of the intended blocks (besides the purchased IPs at these levels) to another company to generated technology-mapped netlists for each block [21]. Similarly, these netlists may be again outsourced to other third parties in order to perform physical syntheses and obtain their layouts, seen in Fig. 2.2.a. Finally, all the blocks are gathered to obtain the layout level specification of the SoC.

The third period of the SoC life cycle is the fabrication period in which a SoC layout and information for its tests are outsourced to a fab [22]. The fab may either test the fabricated instances (dies) of the SoC layout by own or ask another company to do it [23]. The goal of these tests is to find circuit defects, happened during the manufacturing, and separate the defective dies from the well-fabricated ones. Another third party probably contributed to the fabrication period is packaging companies. They package each the-tests-passed die into a complete chip. Finally, the fab delivers back the chips to the design house.

The fourth period is when a design house sells a SoC chip to their clients who ordered the SoC and intended to use it [24]. The clients program the processor and reconfigurable parts of the SoC and then integrated it in their board-level design as the final product,

which offers services to end users, like a smartphone.

The fifth and sixth periods are the application and discard periods, respectively [25]. In the application period, a SoC works inside a final product. In the discard period, a final product is no longer useful and end users throw it away. However, in these periods, a SoC is one part of a final product and normal end users do not have direct access to the SoC, some security threats are possible from professional end users [26]. In the following, security threats may happen in various third parties in the different period of the SoC life cycle are briefly presented.

2.1.2 Security Threat Models

As discussed earlier, during the SoC life cycle, there are various security threats that may happen in involved third parties. Some of these threats are shown in Fig. 2.2.b.

Security threats begin from the first period of the SoC life cycle. In this period rogue IP designers are suspicious. Two threats in this period are hardware Trojan and IP piracy [2, 3, 17, 20]. Hardware Trojans are malicious circuitry components stealthily inserted in a circuit for sabotage objectives [4, 5, 22]. IP piracy happens if untrustworthy parties buy IPs, take advantage of the access to the purchased IPs, and illegally resell them.

In the hardware design period, any disloyal employee of the design house or employing CAD tools (in either the design house or third parties that perform logical and physical syntheses) may add hardware Trojans to the SoC design [21, 27]. In addition, untrustworthy design houses are suspected of IP piracy [28].

Other security-critical third parties are untrustworthy fab, test, and packaging

foundries. Fabs can overproduce a SoC that they receive its layout from their clients for fabrication [23]. They can also make *counterfeit* SoC using two components: (1) a new package different from the original ordered package and (2) the overproduced dies from the ordered SoC layout [25]. Fabs can sell overproduced and counterfeit SoCs in black markets under the original price because they have not paid any for designing the SoC [29]. They can also clone the whole SoC layout or extract and pirate some IPs from it. Fabs can make small changes into the layout of a SoC and insert hardware Trojans [4, 22].

The security threats of untrustworthy fabs exist in test and packaging foundries with small differences [23, 25]. Test and packaging foundries have access to the fabricated dies of a SoC received from a fab for testing and packaging; thus, in order to obtain the layout of the SoC, they need to perform reverse engineering (including delayering, imaging and image processing [26]) on the layout of the fabricated dies. They can clone the obtained layout or extract and pirate its IPs [28]. In addition, they can insert hardware Trojans into the obtained layout and send it for fabrication [4, 22]. Moreover, packaging foundries can insert hardware Trojans besides to the tested dies into the ordered package. In this case, they do not need to change the layout and fabricate the new one [30].

During the application period, end users can obtain the die of a SoC by encapsulating its package [26]. Thereafter, they can insert hardware Trojans besides the obtained die into a new package. Moreover, cloning and IP piracy will be possible if they can perfume reverse engineering on the obtained die [1, 31]. Alternatively, they can remark the packages of used, aged, or recycled SoCs and make counterfeit ones [25]. Similarly, all these threats are possible in the recycle period [25, 31].

For all these security threats, detection and prevention methods have been proposed.

in the followings sections in order to make the readers familiar with some related topics to this dissertation, more details about IP protection methods against IP piracy and hardware Trojans taxonomies, Trojan models, Trojan detection and prevention methods are presented.

2.2 IP Protection

IP piracy threat in the different periods of the SoC life cycle forces IP designers to use IP protection methods [7-10, 12, 31-43]. Researchers classify these methods in two categories, passive and active ones, based on whether or not they can prevent IP piracy [33]. Passive methods cannot prevent IP piracy because, unlike the active ones, they do not add the lock/unlock feature to IP design [33]. A protected IP incorporating this feature would correctly operate when a correct value, as a key, is applied to. Thus, pirates need to figure out the correct key in order to reach the correct functionality of the modified IP. In the following, some of the passive and active methods with their accurate terminology are presented and shown in the figures 2.3-2.9.

2.2.1 Passive IP Protection Methods: Watermark, Obfuscation, and Camouflage

A digital *watermark* is a designer signature that is embedded into an IP in order to prove the designer's ownership [34]. Adding watermark to IP is a passive method because it cannot prevent IP piracy. Owners of such IPs can present the presence of their watermarks in pirated IPs and claim their rights. Figure 2.3 shows a watermark at RTL whose states

are added to the original states and reachable by specific input vectors. Note, if pirates can detect and then remove a watermark, its owners would not be capable to prove their ownership.

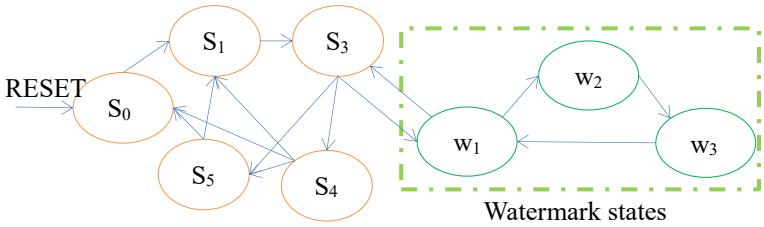


Fig. 2.3 An embedded watermark that includes three states and is reachable in one state (S_3) of the host circuit

The term *obfuscation* refers to modify the design or specification of an IP without making any change in the IP functionality [11, 35]. For example, a designer can keep some redundant logic gates and don't-care inputs (which logic optimization algorithms usually remove) in a netlist, in order to make the understanding of the netlist functionality difficult. Another example is to modify an IP specification in order to reduce its readability, i.e. changing the name of variables and signals and clearing the comments in the Hardware Description Language (HDL) files of an IP specification. Figure 2.4 shows an obfuscator. As seen in the examples and figure, obfuscated IPs are piratable, and one can use them as a black box; thus, obfuscation is a passive IP protection method.

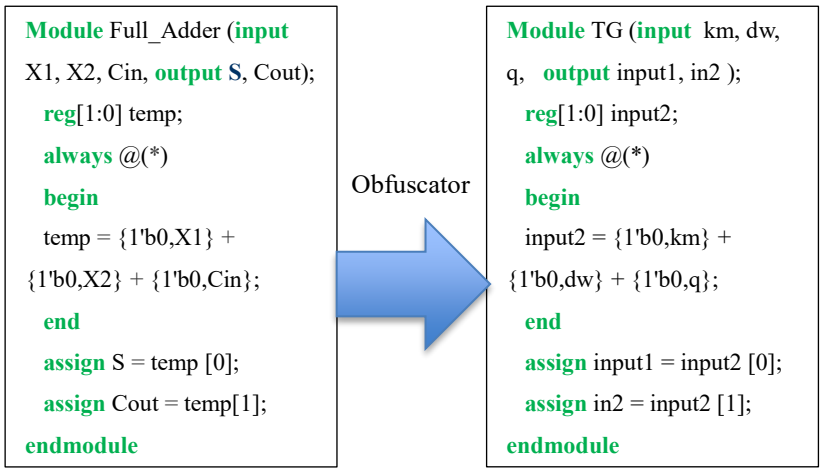


Fig. 2.4 An obfuscator that renames inputs, outputs, variables.

Camouflage or look-alike gates protect protects ICs against chip-level reverse engineering [36]. The layout-level design of a camouflage gate has contacts in different layout layers; some of them are called *true contacts* that make the logic functions of the cell, other ones called *dummy contacts* and only make an identical layout-level appearance with other gates [36]. Dummy contacts in a camouflage gate are true ones for other gates. For example, Fig. 2.5.a and 2.5.b show a normal layout design of NAND and NOR gates, respectively. In addition, their camouflaged design is shown in Fig. 2.5.c and 2.5.d. As seen in these figures, the first two figures have a different design; however, the third and fourth figures look the same.

As mentioned, professional end users in the application period can perform reverse engineering on a SoC to obtain its layout (and then its IPs). It includes encapsulating the SoC package, delayering the SoC die, imaging each layer of the die, and then using the images to make the transistor level netlist of the SoC. If a SoC is designed using camouflage gates, the image of each layer, obtained from reverse engineering process, contains true and dummy contacts that appear the same look from the top and so make the logic functionality of the gates unknown.

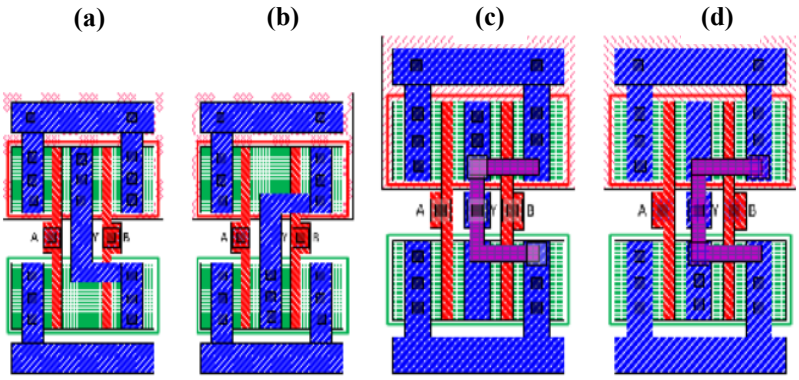


Fig. 2.5 Regular layouts of 2-input **(a)** NAND and **(b)** NOR gates, camouflaged layouts of 2-input **(c)** NAND and **(d)** NOR gates [36].

2.2.2 IP Piracy Prevention by Active IP Protection Methods

As mentioned, active methods prevent IP piracy because they modify an IP such that it correctly functions when a correct key is applied to. According to the abstraction level of such methods, RTL or gate-level, authors have used different names for such as logic encryption [12], state/logic obfuscation [8, 9], logic locking [37], logic masking [38, 40]. Sometimes, these terms have been used interchangeably.

The use of the term *encryption* is accurate when design modifications include embedding a crypto processor into a design [11]. As shown in Fig. 2.6.a, the original inputs of a circuit under modifications are fed by the outputs of a crypto processor. Similarly, one can use a crypto-processor at the outputs of a circuit; thus, the original functionality of the circuit is encrypted, as shown in Fig. 2.6.b. Such IPs have wrong outputs if the crypto-processor gets incorrect keys. This method can protected IPs from end users in the application period; however, such IPs are piratable in untrustworthy fabs because attackers can either separate the embedded crypto-processor or write a known value, as the key, to a specific memory that is designed to keep the correct key of the crypto-processor. Figure 2.6 shows that if attackers write such a key, then they can easily use it to decrypt the functionality of the IP. The requirement hardware including a tamper-resistance memory is described in the following subsection.

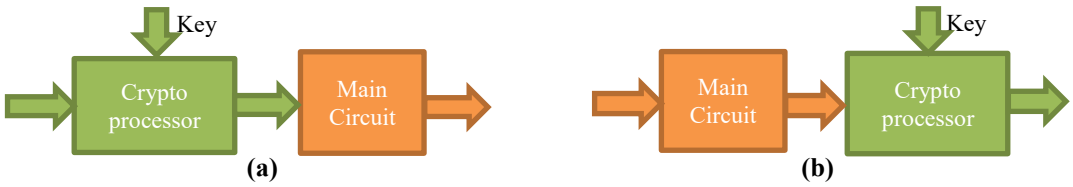


Fig. 2.6 circuit encryption using a crypto processor in **(a)** inputs, **(b)** outputs.

The term *masking* refers to a prevention approach against IP piracy, which aims to obscure the original functionality of a circuit by adding some gates, state elements, and extra inputs. A masked circuit has two modes: a functional mode and a masked mode. It works like the one before masking modifications in the functional mode. In the masked mode, the masked circuit malfunctions. In order to reach the functional mode, one needs to apply a correct value(s) to the masked circuit inputs. In other words, this value acts like a key and so long as only authorized users (i.e., IP or IC owners) know this value, IP piracy prevention is achieved.

Masking modifications usually include 1) adding some fake states to the original FSM of a circuit at RTL and then 2) adding some gates to the combinational part of the circuit at the gate level. The modification at RTL is named *state obfuscation* in literature, i.e. in [7, 8]. According to the obfuscation definition, the term *state masking* specifies this modification better than that of state obfuscation. The modification at gate level has been referred in different ways such as *logic obfuscation*, *logic encryption, logic masking,* or *logic locking*. The authors in [11] discussed that these modifications do not have the features of encryption and obfuscation, thus, these terms cannot well indicate this modification; but, nonetheless, the term logic masking and logic locking are significant and have been used interchangeably in the literature. Hereafter, we use the term logic masking to refer to this modification. In the following state masking and logic masking are presented with more details.

2.2.3 IP Masking

2.2.3.1 State Masking

State masking conjoins a fake FSM and the FSM of a circuit into a new one, at RTL [7-9]. Indeed, the new FSM represents a new circuit incorporating a functional and a mask (fake) mode in which the circuit functionality is the same as the original circuit and meaningless, respectively. Upon power-up, the new FSM starts from a fake start state; and one needs to know a specific inputs sequence to pass the circuit through the start state of the original FSM. An example is shown in Fig. 2.7, the circuit starts from S^0_M which belongs to the mask mode (fake FSM), and the start state of the original circuit (S^0_O) is reachable from S^0_M by applying the key sequence K_1, K_2, K_3, and K_S to the circuit's inputs.

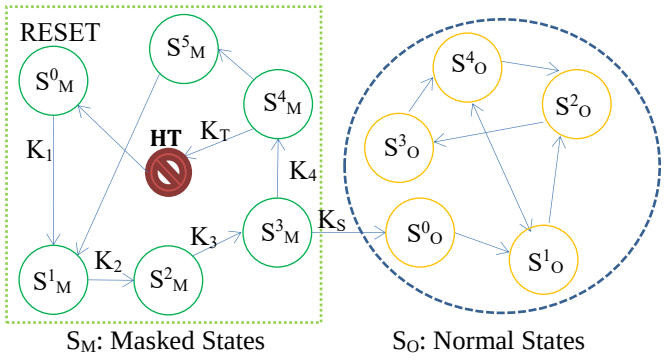


Fig 2.7 FSM structure of a masked sequential circuit; an inserted hardware Trojan (HT) will be active by a specific input (K_T) in the added FSM

It is noteworthy that if there is only one transition between the fake and the original FSM, like what is seen in Fig. 2.7, an attacker can infer the fake states; one can randomly apply many input vectors to the circuit and figure out that the circuit never goes to some states [8]. To tackle this issue, a designer must interlock the fake and original FSM such that the merged FSM has three features. First, there must be several

transitions between the fake and original FSM. Second, if the circuit goes to the original states using incorrect keys, it must generate incorrect outputs; and third, in such cases, the circuit must return to states of the fake mode [8].

In order to well interlock the fake and original FSM including the mentioned features, a designer must add some extra inputs and transitions between these two FSMs. It can be performed at RTL or gate level [41]. A designer should have accurate information about the states and transitions in the original FSM, at RTL. At gate level, these modifications are implementable using logic masking. The authors in [41] showed that the logic masking approach results in a higher level of integration in between the original and fake FSMs. Another advantage of logic masking is that there is no need to know about the original FSM and circuit behavior, thus, it can be performed by another design group.

2.2.3.2 Logic Masking

The majority of proposed logic masking methods modifies the combinational part of the circuit under masking at the gate level [40]. They are based on two steps. The first step, which is common in all methods, is a random selection between XOR and XNOR gate (the so-called *key-gate*). For each added key-gate, designers also add one input (the so-called *key-input*) and connect it to one input of the key-gate. The selected key-gate determines the correct key value. For XOR (XNOR) the correct key is 0 (1). The key-inputs increases the truth table size of the modified circuit and the key-gate produces wrong outputs while the key input is not derived by the correct key. A circuit including one XOR (K_1) and one XNOR (K_2) key-gate is shown in Fig 2.8; the correct key for key-gates K_1 and K_2 is ‘0’ and ‘1’, respectively.

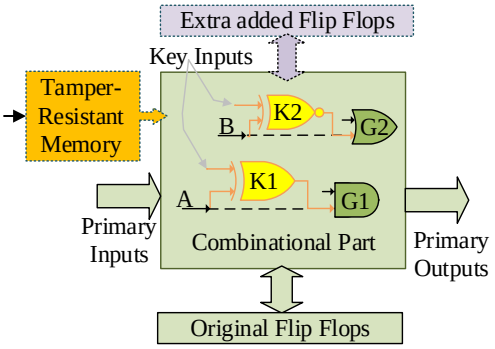


Fig. 2.8 Mealy machine view of a masked circuit, including two key-gates and key-inputs, and a tamper-resistant memory which drives key-input

The second step is to choose a net from the combinational part of the circuit, then connect to another input of an added key-gate [40]. Instead of the selected net, the key-gate output derives the cell(s) that has (have) been derived by the net. Different algorithms can implement this selection process taking into account various objectives. Figure 2.8 shows two nets A and B selected for the logic masking modifications. In the original circuit, these nets (dashed lines) drive an input of gates G_1 and G_2. The output of K_1 and K_2 drive these inputs, in the masked circuit.

For the first time, the authors of [42] proposed an algorithm for random net selection among circuit nets for inserting key-gates. This approach is termed random masking. Whereas this approach is easy-to-implement, the quality of logic masking is guaranteed.

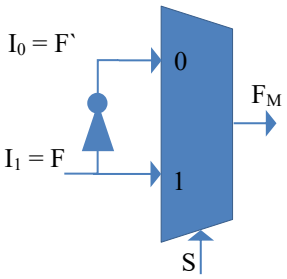
In [9] the authors proposed a logic masking algorithm considering information about fan-in and fan-out cones of the circuit. Their objective is to maximize the mismatch points between the masked circuit and the original circuit when they are compared using formal methods.

In [43] the authors presented another logic masking method in which key-gates have more effect on each other. This method inserts key-gates such that attackers cannot use the circuit inputs to excite and propagate the effect of one key-gate and simultaneously prevent exciting and propagating the effect of other key-gates. Thus, this method hinders

attackers' efforts to reveal the key by feeding the circuit-specific inputs.

Using multiplexer primitive as a key-gate instead of XOR/XNOR has been proposed in [12, 40, 43, 44], as shown in Fig. 2.9. One of the two inputs of a multiplexer is randomly selected. This input defines the correct value of its key-input. Figure 2.9 shows the selected input is I_1 and so the correct key is '1'. Another input of the multiplexer must be connected to a net from the combinational part according to the masking algorithm. One can use the inversion of the selected net in the previous, as shown in Fig. 2.9. The multiplexer passes the correct input if its select input (the key-input) is fed by the correct value; otherwise, it passes a wrong value coming from another part of the circuit.

Fig 2.9 2-to-1 multiplexer cell as a key-gate and its truth table [43]



I_0	I_1	S	F_M
I_0	X	0	I_0
X	F	1	F

In all logic masking approaches, after key-gate insertion, designers must incrementally perform a logic optimization algorithm in order to solve key-gates in combinational parts [40]. For example, if one inverter gate, in the original circuit, is preceded by an XOR key-gate, a logic optimizer algorithm might convert them to an XNOR key-gate. As a result, the masked circuit has an XNOR key-gate which its correct key is '0', though the correct key value of an XNOR key-gate seems '1' at first glance. Thus, the reverse engineering of the masked layout cannot allow attackers to find the correct key value.

2.2.3.3 Criteria for Logic Masking

Researchers have proposed different criteria to measure masking quality in a modified circuit by logic masking. First, the authors in [9] proposed to use the number of mismatch points between a masked circuit and its original one. A mismatch point makes an inversion at its place. If the inversion arrives at the circuit outputs, consequently it produces a wrong output. Obviously, the output failure might depend on input vectors. As a result, the proportion failed and succeeded input vectors can be used as a metric for the masking qualification.

The simplicity of the mismatch point number as a logic masking criteria can be misleading because it only concerns failed outputs even with one wrong bit failure. If the fab that fabricated a masked layout finds an activated IC of the masked layout in the market, they can guess the correct key by observing the masked-layout output bits (for different input vectors) and comparing it with the correct output bits of the activated IC [43]. The correct output bits, in each trial of the attack, lead the attacker to find the correct key [45].

For this reason, an effective masking approach must produce nearly equal numbers of the correct and incorrect output bits when it is driven by the wrong keys [12]. In other words, Hamming distance (HD) between the correct and incorrect output bits (when a circuit is fed by the incorrect keys) should be near to 50%. The more balance HD, the more masking is obtained [12]. In this situation, the attacker can hardly deduce the correct key by incrementally improving partial correct keys. The authors in [12] also proved mathematically that in case of having 50% HD the attacker faces the most difficult situation for the key guesses. In addition, 0% HD means the masked circuit outputs are always correct and independent of the key value. Moreover, 100% HD means the outputs

and the key-inputs are correlated, but it is not proper for masking [12]. The reason is that the wrong output bits always equal the inversion of the correct value while the key-inputs are applied by the wrong keys.

To obtain a highly balanced Hamming distance (50%), the authors of [12] have proposed to employ the concept of fault excitation/propagation analysis. If a fault in a signal can affect nearly 50% of all circuit outputs, the signal is called a high-fault-impact signal. The high-fault-impact signals are the best choices for key-gate insertion. This method has been applied to ISCAS-85 benchmark circuits and worked well but could not achieve a Hamming distance of 50% for all circuits.

2.2.3.4 Hardware Requirements for IP Masking

Logic/state masking involves secure infrastructures (i.e. a tamper resistance memory to store the key, as shown in Fig. 2.8 [11]. The authors in [4] discussed even if the infrastructures are secure, the masking quality is an important security concern. For example, if the fab that fabricated a masked layout finds an activated IC of the masked layout in the market, the fab can guess the correct key by observing the masked-layout output bits (for different input vectors) and comparing it with the correct output bits of the activated IC [12]. The correct output bits, in each trial of the attack, lead the attacker to find the correct key [4]. For this reason, an effective masking approach must produce nearly equal numbers of the correct and incorrect output bits when it is driven by the wrong keys. In other words, Hamming Distance (HD) between the correct and incorrect output bits (when a circuit is fed by the incorrect keys) should be near to 50%. The more balance HD, the more masking is obtained [12]. In this situation, the attacker can hardly deduce the correct key by incrementally improving partial correct keys [12].

2.3 Hardware Trojan

2.3.1 Hardware Trojan Attributes and Taxonomies

As previously mentioned, hardware Trojans are malicious alterations in circuits intended for sabotage purposes [4-5, 19, 22]. In order to model Trojan threats and counter them, awareness about Trojans attributes is vital, because Trojan attackers are smart, skillful, and able to design many different hardware Trojans for each circuit. As a result, many classifications have been proposed so that in each of them an attribute of hardware Trojans is concerned [4].

Two important Trojans attributes are insertion phase and intention [4, 19, 46]. In Section 2.1, it was briefly explained who may insert hardware Trojan into SoCs in the different phases of the SoC life cycle. It was also mentioned that Trojans' intentions can be leak confidential information, cause unexpected faults and failures, make denial of service, or degrade some circuit features such as performance or power consumption (and consequently circuit lifetime) Depending on the intentions and insertion phases of hardware Trojans, attackers design them at different abstraction levels such as RTL, gate, or transistor level. Moreover, they may insert them into different places of a SoC such as processor, memory, clock grid, etc [19, 46]. The Trojan classifications based on insertion phase, intention, abstraction level, and location are summarized in Fig. 2.10, and extensively discussed [19, 46].

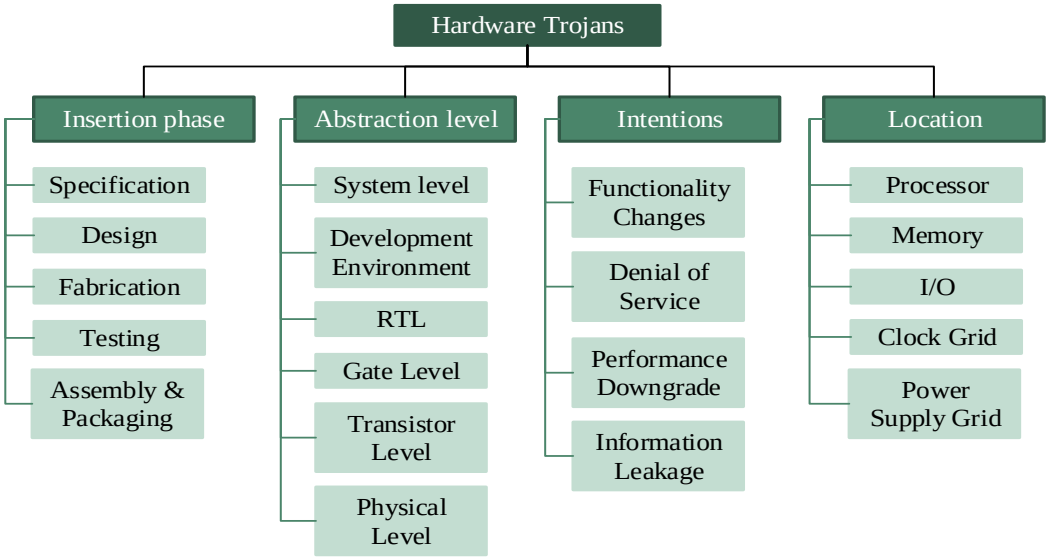


Fig 2.10 Four hardware Trojan taxonomies based on four Trojan attributes: insertion phase, abstraction level, intention, and location [46].

Hardware Trojans can be classified based on other attributes. Regarding four physical characteristics: structure, distribution, size, and type, hardware Trojans can be classified into different categories. Trojan attackers may change the layout in order to insert a hardware Trojan, or they may insert it in the package of SoC and not change the layout. Hardware Trojans can be small including few transistors or large like a macro gate module. Trojan attackers may tightly place these transistors and/or gates together or loosely distribute across a SoC die or package. Finally, hardware Trojans can be functional or parametric. Trojan attackers insert functional Trojans by adding/removing some transistors to/from a being attacked circuit. On the other hand, parametric Trojans include modifications in features of circuit nets and transistors, i.e. in order to leak confidential information through the circuit side-channels without making any change in the circuit functionality. These categories are summarized in Fig. 2.11 and well detailed in [47].

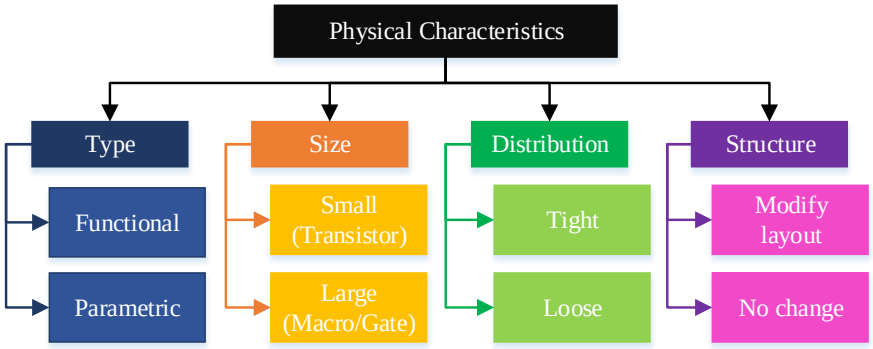


Fig 2.11 Trojan taxonomy based on physical characteristics [47]

Another important attribute for each hardware Trojan is their activation mechanism by which they are trigged [18-19, 46-47]. Regarding this attribute, hardware Trojans are either always active or conditionally active. The always-active Trojans are those that start their mission upon the circuit power-up, i.e. parametric hardware Trojans. On the other hand, conditionally active Trojans are the ones that get triggered if specific condition happen. External or internal events make these conditions.

Internal events can be specific digital values of the circuit flip-flops and signals, influenced by the circuit logic operations. They can be digital or analog values of some signals connected to the circuit sensors capturing physical in-circuit conditions. Activating external events can be specific values applying to the circuit inputs or peculiar physical environmental conditions sensed by the circuit sensors. Figure 2.12 shows the sum up of the Trojan taxonomy based on activation mechanisms, more explained in [47].

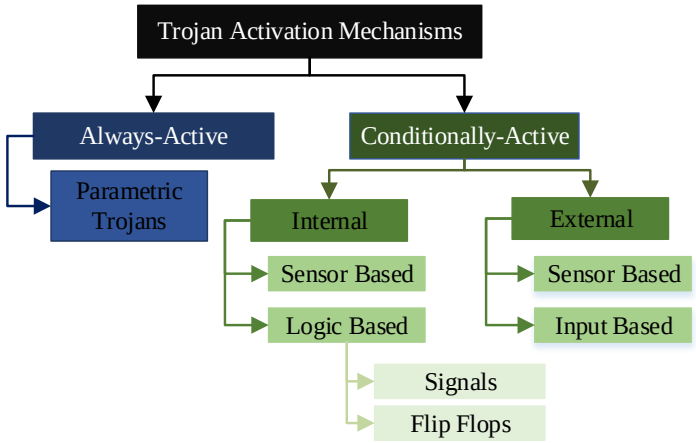
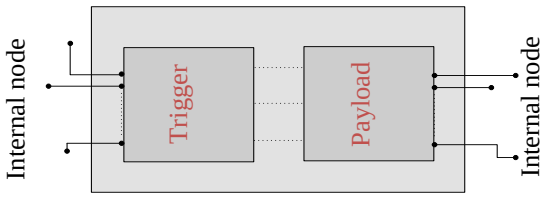


Fig 2.12 Trojan taxonomy based on activation characteristics [47]

2.3.2 Hardware Trojan Model

Regarding the Trojans diversity, proposing an efficient model like the stuck-at fault model in the test and testability discussions is impossible []. Nevertheless, an abstracted model has been proposed in which hardware Trojans in a simplified form can be considered including two parts: a *trigger* and *payload* part, as shown in Fig. 2.13 []. Many researchers have used this model in their proposed Trojan countermeasures.



2.13 Abstract Trojan model including the trigger and payload part [4]

The “trigger” is a mechanism like a simple comparator that activates a Trojan in a specific situation [4]. Please note that one can imagine trigger conditions in always-active hardware Trojans are met upon power-up [19]. Examples of such hardware Trojans are parametric ones. The “payload” performs malicious missions of a Trojan. Both the trigger and payload can be digital, analog, or a combination of them. In digital ones, they can categorize in sequential or combinational circuits. Taxonomy of different triggers and payloads are presented in Fig. 2.14 [18], and extensively discussed in [18-19].

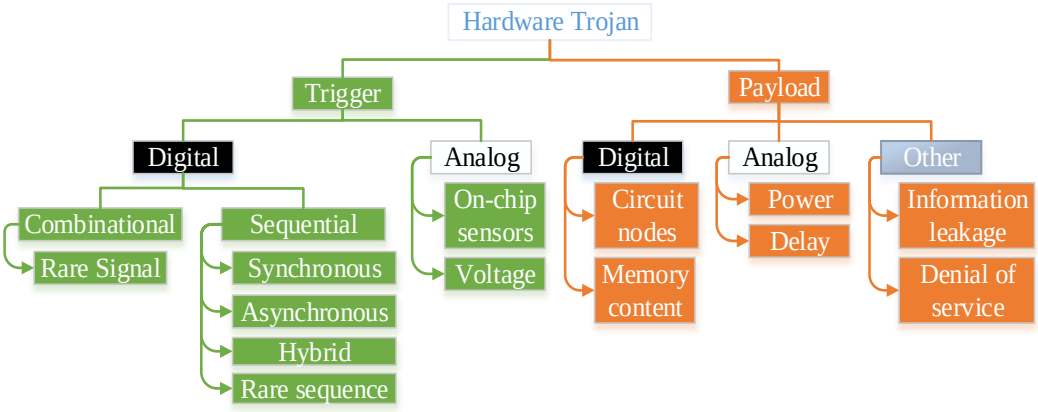


Fig 2.14 Taxonomy of Trojan triggers and payloads [18]

2.3.3 Hardware Trojan Attack Model

Hardware Trojans can enter into a SoC die if they are designed and inserted into a circuit block used in the SoC layout [28]. In other words, they contaminate SoC dies in three phases: (1) specification, design, and fabrication, more details in Section 2.1.2. The third parties involved in these phases are IP developers, design houses, and fabs. The authors in [22] thoroughly discussed that these third parties make seven attack models, listed in Table 2.1. In each model in order to detect hardware Trojans, defenders need a golden model to compare it with functional and physical characteristics of IPs, layout designs, or fabricated ICs bought from untrustworthy third parties. In the following, these seven attack models and their required golden model are briefly explained.

Table 2.1 Trojan attack model based on suspiciousness foundries that may insert hardware Trojans [22]

Model	IP developer	Design house	Fab	Comments
1	✓	✓	💣	Untrustworthy fab and packaging foundries may insert Trojans into SoC layouts/packages
2	✓	💣	✓	Untrustworthy design houses or their CAD tools may insert Trojans into SoC design
3	✓	💣	💣	Untrustworthy System Integrators with clients that order some circuits to design and fabrication
4	💣	✓	✓	Untrustworthy IP developer can insert hardware Trojans in their IPs
5	💣	✓	💣	The combination of *model 1* and 4 in which fabs and IP developers are suspected for fabless SoC design houses
6	💣	💣	✓	design houses and IP developers are suspected of Trojan attacks
7	💣	💣	💣	Untrustworthy design houses, IP developers, or fabs may insert Trojans in COTS components

💣 = untrustworthy ✓ = trustworthy

Attack model 1: a fabless design house considers this model in which they use trusted CAD tools and buy required IPs from trustworthy IP developers, but send their final layout to an untrustworthy fab. Attackers in the fab have full access to the layout and are able to perform reverse engineering on it in order to obtain the functionality and insert hard-to-detect hardware Trojans. After the fabrication, in order to detect hardware Trojans, the design house can use the functional and structural specification of the layout as a golden model. In addition, some circuitry parameters such as power consumption and timing information can be employed as a golden model; it will be discussed in details in the next section. Any abnormality in ICs' functionality, structure, and parameters alarms the presence of a hardware Trojans.

Attack Model 2: In this model, design houses are suspicious of Trojan attacks. They, or even any disloyal employee of theirs, can intentionally insert hardware Trojans into

the design or layout of an ordered SoC. Note that untrustworthy CAD tools used in design houses during the design period can be the source of Trojan attacks. Clients of such design houses may have some previously purchased trusted IP cores and would like to use them beside their ordered SoC in order to make its final layout. They send the final layout to a trustworthy fab. Thus, hardware Trojan detection must be done before sending the layout to fabs. In this model, the golden model is the specifications of designs ordered by clients to untrustworthy design houses.

Attack Model 3: This model is the combination of Attack Model 1 and Attack Model 2. It is about untrustworthy system integrators that their clients order some circuits to design and fabricate. Clients may have bought some trusted IP cores and would like to use them beside their orders. In this model, since untrustworthy system integrators design and fabricate circuits, the golden model is specifications of ordered deigns and trusted IPs.

Attack Model 4: This model is about semiconductor companies that, regardless of being fabless or not, need some IP cores in order to decrease costs and time-to-market. All fabricated ICs have a hardware Trojan if it exists in an IP used in the ICs layout. The golden model in this model is specifications of purchased IPs.

Attack Model 5: This model is the combination of Attack Model 1 and Attack Model 4, and includes all fabless design houses that buy IPs and fabrication services from untrustworthy third parties. As there are two hardware Trojan sources in this model, Trojan detection routines before and after fabrication must be performed based on the golden models required against Attack Model 1 and Attack Model 4.

Attack Model 6: In this model, design houses are suspicious of hardware Trojan attacks, like Attack Model 3 but unlike that, fabs and IP developers are trustworthy and

untrustworthy, respectively. Imagine, a client orders a custom-design to an untrustworthy design house. The client may have some IPs purchased from untrustworthy developers and want to use them beside the ordered custom-design to make the final SoC layout, and then send it to a trustworthy fab. In this model, hardware Trojan detection must be performed before fabrication; and the golden model for that is the specifications of custom-design and IPs ordered by clients to untrustworthy design houses and IP developers.

Attack Model 7: There are many commercial off-the-shelf (COTS) components in today's system. They are not as expensive as custom-designed circuits, but the use of these components raises serious security concerns such as hardware Trojan threat. Since clients may not know about designers of COTS components or used IPs in them (and their fabrication foundries in Attack Model 7), each of the three mentioned third parties solely or together can be suspicious of Trojan attacks. In this model, general specifications of COSTs can be used as a golden model.

As seen in Table 2.1, different Trojan models exist and only one model must be considered and zoomed in order to better deal. Attack Model 1 in which fabs are untrustworthy is concerned in this dissertation. In the following section, countermeasures against this model are briefly presented. Henceforth the terms hardware Trojan, Trojan countermeasure, golden model, and so on refer to such terms in Attack Model 1.

2.3.4 Hardware Trojan Countermeasures

Many researches have targeted countermeasures against Attack model 1. A taxonomy of such countermeasures, shown in Fig. 2.15, consist of two main categories: *Trojan*

detection and *design-for-trust* [4-6, 18, 22]. Broadly speaking, in the first category abnormalities in ICs' functionalities or side-channels are investigated to detect hardware Trojans. The second category aims at preventing Trojan insertion or facilitating Trojan detection [6]. In the following, these two approaches are more explained.

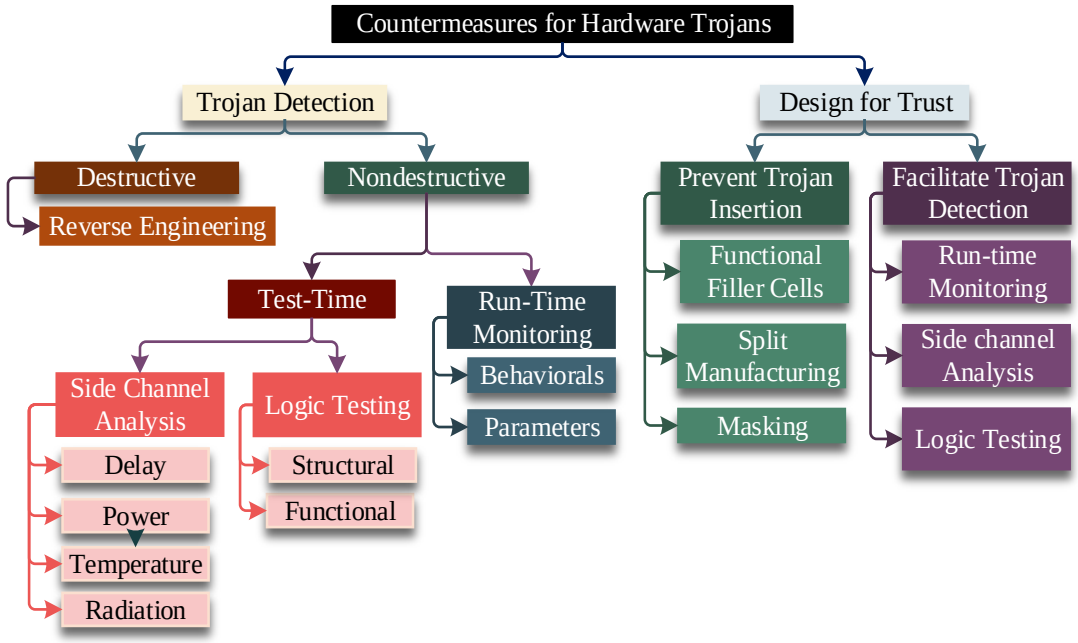


Fig 2.15 Taxonomy of Trojan countermeasures [22]

2.3.4.1 Hardware Trojan Detection

In order to detect hardware Trojans, the first idea coming in mind is to obtain fabricated IC layouts by performing reverse engineering and then compare them with original order layouts. Regardless of how much reverse engineering can be expensive, this approach cannot be a solution since it irreversibly destructs ICs under reverse engineering and attackers may insert hardware Trojans only in some of fabricated ICs [48]. Thus, nondestructive detection methods are much needed. Reverse engineering is helpful to obtain golden models [4]. We can measure side-channels from a few fabricated ICs; then,

reverse engineering make us sure the measurements are obtained from the genuine ICs.

Regarding the trigger-payload model, shown in Fig. 2.13, many nondestructive detection methods have been presented. Their approaches are classified into three categories, as shown in Fig. 2.15. One approach is to use conventional functional/structural tests in order to generate and apply proper input patterns to ICs under tests and, consequently, trigger possibly inserted Trojans and observe their effects on outputs. Detection methods based on this approach are known as *logic testing* [18, 49]. Another detection approach is *side-channel analysis* that aims to find Trojans effects on ICs parameters such as power consumption and paths delay [4-5]. Defenders must use detection methods based on these two approaches before the application period.

The third approach is *run-time monitoring* [50]. Designers can increase trustworthy in their SoCs using this approach. They usually design or program Trojan-monitoring modules for reconfigurable and programmable parts of a SoC. During the application period, these modules continually check behaviors or/and parameters of security-critical parts of the SoC. These behaviors and parameters can be the ones used in the logic testing or side-channel analysis.

It is noteworthy that there is no silver bullet to detect hardware Trojans [19]. Trojan detection in all the approaches is a challenging task because smart and skillful attackers can insert hard-to-trigged and well-designed Trojans, into ordered layouts in untrustworthy fabs.

In hard-to-trigged Trojans, low controllable signals are employed for the trigger part [17, 46]. Since exhaustive functional/structural tests for all possible inputs are infeasible and, furthermore, such Trojans and are activated in rare specific conditions and barely make a change in circuit outputs they may not be detected by Trojan detection methods

based on logic testing [18]. In addition, hardware Trojans that have analog triggers and/or payloads are undetectable by such methods. In fact, functional/structural tests have been proposed to detect simple manufacturing faults, not malicious alterations designed by expert attackers aiming special sabotage purposes [4].

Trojan attackers try to well design their hardware Trojans affecting side-channels as least as possible [6]. In today's big circuits, the effects of a well-designed Trojan, including a few cells, on side-channels are not easily distinguishable because side-channels do not have a fixed value and they vary due to process and environmental variations [4]. These variations are the challenges of Trojan detection methods based on side-channel analysis. Process variations happen during IC manufacturing and because of them, transistors in a fabricated IC have different physical parameters, such as channel length, oxide thickness, etc., and consequently, electrical parameters, such as threshold voltage, capacitance load, etc. [50]. Environmental variations occur due to changes in the environmental conditions of transistors, such as temperature, while they are working [50].

In order to better identify Trojan effects on side-channels, one can use some purposefully and intelligently generated test vectors [22]. Indeed, this approach is the combination of logic testing and side-channel analysis. For instance, the authors proposed to use test vectors generated for the traditional path delay fault in order to detect hardware Trojans [58]. Further details on the complementary use of logic testing and side channel are explained in the following sections.

Trojan Detection Flow in Methods Based on Combining Logic Testing and Side-channel Analysis

Such methods have four main steps, detailed in [52]. First, sufficient intelligent input vectors must be generated. This is very challenging because these vectors must be at least as possible and stimulate all of the circuit elements. For example, test vectors generated for the path delay fault detection can be useful for Trojan detection, but these vectors must cover a collection of paths in such a way that each net of the circuit belongs to at least one path of the collection [40]. The second step consists in applying the generated test vectors to the ICs and measuring the signals of targeted side-channels. These signals are employed as a fingerprint for each IC. When the targeted side-channel is paths' delays, one can use test vectors generated for path delay fault analysis in order to measure the delay of the selected paths. If the targeted side-channel is power consumption, the knowledge of the transient current (IDD_T) and leakage current (IDD_Q) test can be used to generate efficient test vectors (more explanations about power consumption components are provided later in section 2.3.4.1.3). The third step is to calculate the variation of the targeted side-channel. It is used as a fingerprint for each fabricated IC. The final step is to compare ICs' fingerprints with a golden model. This comparison shows the presence of a hardware Trojan if it shifts the measured side-channel out of the variation of the used golden model.

It is noteworthy that the required golden model in this approach is obtained in the third step if the variation of the targeted side-channel is calculated from a few genuine ICs (golden ICs), one can use it as a golden model. In order to find these genuine ICs, one can randomly select a few ICs and then, after measuring targeted side-channels, perform destructive reverse engineering [48]. Later researches have shown it is possible to avoid this reverse engineering [53]. For this purpose, different solutions have been proposed.

For instance, the authors in [54] proposed to embed some sensors into the ordered layout and using them in the fabricated IC alleviate the effect of process variations on the measured side-channel. Another approach, proposed in [55], is to obtain the (e.g. power) fingerprints of the different parts of a fabricated IC and then compare the fingerprints by each other to detect inserted hardware Trojans, regarding they cannot exist in the all parts of an IC, otherwise, they will be very big and easily distinguishable. These methods are instances of the approach *golden-free* Trojan detection [21].

Informally speaking, the number of published articles related to path delay or power consumption fingerprinting shows these two side channels have been quite interesting for researchers. One reason for this can be the presence of considerable experiences and knowledge in the "*test and testability"* topics usable for these two side channels. We will also consider them in our proposed methods, in Chapter 4 and 5. Therefore, in the following we review some Trojan detection methods based on path delay and power consumption fingerprinting.

PDA-Based Hardware Trojan Detection

For the first time, the authors in [56] tried to detect hardware Trojans using path delay fingerprints. They simulated the presence of process variation by assuming 15% randomly varying delay parameters for a 130 nm technology. Their simulation results show that their approach can detect the Trojans that they inserted into their benchmark circuits. However, one must take into account that in newer technologies the challenges of process variations are more serious.

The authors in [57] proposed to leverage a path delay measurement structure, named

shadow register, to detect hardware Trojans. The implementation of shadow register includes duplicating the clock signals and all the flip-flops (FFs) of the circuit. The input of each original FF feeds an added (shadow) FF. The shadow FFs capture their input using the added clock signal. The added and original clock signals have the same frequency, with a specific phase difference. The phase difference should be equal to the maximum path delay variation. In this case, if an FF and its shadow have two different values, one can suspect the presence of a hardware Trojan. The comparison between each FF and its shadow FF requires an additional XOR gate. The results show that the area overhead and resolution of this structure, as well as Trojan detectability, are very high.

The authors in [58] proposed to employ another structure with the same objective as [57]. This structure needs only one additional multiplexer for each FF in the circuit. The results show that this structure has less area overhead and lower delay-measurement accuracy in comparison to the shadow register- based measurement approach [57].

Cha et al. have proposed the use of a *ring oscillator* (RO) in addition to the shadow register structure [59]. In this work, an RO is used as a calibration device. Indeed, accurate calibration is necessary to remove (or at least decrease) process variation effects and then be able to detect Trojan side-channel effects.

A brief explanation is that process variations have two different components: die-to-die variation components that alter side-channels in each instance of an IC. With-in-die variation components that alter side-channels at different points of an IC [51]. For instance, due to die-to-die variations, the frequency of one RO is different in each instance of the IC. Die-to-die variations do not have any effect on ROs in different places of one IC. If due to die-to-die variations an RO in an IC is X percentage slower (or faster) than the expected value, one can be sure all ROs (with different sizes and structures) in the IC

are X percentage slower (or faster). Thus, one can calculate the die-to-die variation effect on path delay using one RO, and then be sure the rest of path delay variation is due to with-in-die variation or Trojan effects.

Shadow registers empower defenders to investigate Trojan in shorter paths. The results in [58] show that shorter paths are better choices than long ones for delay-based Trojan investigations. Due to with-in-die variations, the delays of two specific gates are different in two different places of an IC. The two gates differ less if they are close to each other [51]. If the gates of a path are placed very far from each other, they are more different and they create a long path. This is the reason for less delay variation in shorter paths. Shekarian et al. theoretically proved this fact [13]. Thus, in order to enhance the Trojan detection probability of path delay analysis (PDA)-based Trojan detection methods, shorter paths should be investigated instead of longer ones. The experiments in [58] also show that performance-driven technology mapping increases the success of PDA-based Trojan detection methods because it generates shorter paths.

ROs have also been used with the aim of Trojan detection based on PDA. In this method, all the circuit gates must belong to at least one RO. RO frequency deviation warns of the presence of Trojans. In order to have less area overhead, RO-embedding algorithms aim to cover all of the circuit gates with fewer ROs. Decreasing the number of ROs increases RO lengths. The authors in [60] proposed a delay chain, named REBEL, which bypasses circuit FFs in order to measure the delay of different parts of the circuit and then detect Trojans. Whereas REBEL and RO-based structures have less area overhead in comparison to the earlier-mentioned structures, they make long paths for investigating Trojan. Long paths (or long ROs) have more delay variations, therefore, their Trojan detection probability is lower.

PCA-Based Trojan Detection

Trojan detection methods based on power consumption analysis (PCA) have scalability issues. In other words, detecting small hardware Trojans including few gates in thousands-gate circuits by PCA is infeasible [19]. Note that the power consumption of hardware Trojans, like other circuits, includes two components: (1) dynamic and (2) static power. Dynamic power, which is the result of IDD_T, is caused by the charge and discharge of capacitances in a circuit while the circuit transistors switching. On the other hand, static power, which is the result of IDD_Q, is consumed when there is no transistor switching.

At first glance, it seems that static power analysis can easily detect hardware Trojans because their design is such that they are often dormant. However, as matter of fact, static power depends on the voltage threshold of transistor that is seriously affected by die-to-die variations. As a result, researchers have rather inspected dynamic power in their proposed PCA-based Trojan detection methods.

The authors in [61] proposed one of the first work about PCA-based Trojan detection. Their main idea was to perform PCA in each circuit under Trojan test (CUTT) by applying random input vectors. Then, they tried to calibrate the noise of process variations from the signal of interest (power consumption) using the Karhunen–Loeve expansion. They could detect hardware Trojans with sizes equal to or less than 1% of Trojan-infected circuits. This size for hardware Trojans in thousands-gate circuits is not considered small. In addition, they had synthesized their circuits by technologies above 100 nm in which process variation is not as serious as in newer technologies. The inability reasons for this method are (1) a globally measured current and (2) randomly generated input vectors were employed to analyze the power consumption.

In order to make up the weaknesses of this work, the authors in [62] proposed to

measure local currents from different power pads of a CUTT and analysis them individually, instead of analyzing the global current totally consumed by the whole circuit. They proved that cells placed near to a power pads drain more current from the pad rather than the farther ones.

As mentioned before, intelligently generated input vectors better results than random ones. One aim for generating such vectors is to perform switching activity localization, i.e. increase switching activity in one small part of the CUTT and decrease it in other parts. In [63], the authors aimed to localize switching activity by a classification method that categorizes randomly generated input vectors into different vector sets suited to different parts of the CUTT. In their proposed method, input vectors that increase switching activity only in one part of the CUTT are gathered in a set. This method is very challenging and sometimes infeasible because many input vectors make activities in different parts of the CUTT.

Some challenges of Trojan detection were briefly explained in the last two sections. Broadly speaking, In order to deal with them, researchers have proposed some design modifications. In the next section, further details about this approach are presented.

2.3.4.2 Design for Trust (DfTr)

Due to various challenges in Trojan detection methods, researchers have proposed purposeful design modifications in order to integrate security and trust in fabricated ICs [6]. Indeed, the main aims of such modifications are categorized into: (1) preventing Trojan insertion and (2) facilitating Trojan detection [4-6, 18-19]. For such modifications, terms *design-for-trust* (DfTr) [6] and *design-for-hardware-trust* [4] have been

interchangeably used in the literature. In this dissertation, we use the former term.

Preventing Trojan insertion DfTr methods include three categories: *functional cell filling* [64], *split manufacturing* [65], and *masking* [9], seen in Fig. 2.15 [22].

The main idea of the first approach is to fill the empty spaces of a die, which physical design CAD tools usually fill using non-functional filler cells, with functional cells. In such situations, Trojan attackers cannot find any space for hardware Trojans.

The second approach splits a layout into two parts: an advance and expensive part, *front-end-of-line* (FEOL), and an ordinary part, *back-end-of-line* (BEOL) [65]. Then, the layout designers send the FEOL and BEOL part to an untrustworthy and trustworthy fab, respectively. They must send both the fabricated parts to a trustworthy assembly foundry. The use of this approach hinders Trojan attackers since they do not have access to the whole layout; consequently, they cannot insert efficient hardware Trojans.

The third preventing Trojan DfTr approach is masking. As mentioned in Section 2.2.3, Trojan attackers cannot easily have enough knowledge about the functionality and structure of a masked circuit, if they do not know its correct key. Consequently, they have difficulties to design hard-to-detect and efficient hardware Trojans. For instance, without knowing the correct key, a hardware Trojan attacker may insert its hardware Trojan in a masked circuit such that it may become active only when the circuit is fed with incorrect keys [9]. Figure 2.8 shows a hardware Trojan that becomes active once the circuit is in a state that has been added to the original states of the circuit for the masking purpose. This state is always unreachable if the circuit holds the correct key.

It is noteworthy that DfTr methods that aim to facilitate Trojan detection consequently hinder Trojan attackers to insert hard-to-detect hardware Trojans. Such methods usually

concern one of three categories of nondestructive Trojan detection approaches. In Chapter 3, 4, and5, we will propose three DfTr approaches concerning logic testing, PDA-based, and PCA-based Trojan detection. Before the start of these chapters, for each of these three categories, examples are briefly introduced in the following:

DfTr Method Concerning Logic Testing-based Trojan Detection

The authors in [66] proposed to add some dummy flip-flops to a circuit in order to increase the controllability of low controllable signals. The authors in [10] aimed the same objective using AND/OR gates as key-gates. For this purpose, they found rare signals and then inserted AND (OR) gates one level-of-gate before signals that rarely have '0' ('1') values. The use of AND/OR gates as key-gates has a shortcoming. The problem is that attackers can easily guess the correct key. The correct key is always the non-controlling value for the AND/OR key-gate. This point will be more explained and illustrated by examples in Chapter 3.

DfTr Method Concerning PDA-Based Trojan Detection

As mentioned earlier, longer paths have more delay variability caused by process variations. Therefore, nets that only belong to long paths are vulnerable points in PDA-based Trojan detection. In order to improve such detection methods, the authors in [13] proposed a trust-driven retiming algorithm, named TDR [13]. TDR aims to reduce the number of vulnerable points. This research work will be more reviewed in Chapter 4, and their results will be compared with that of our proposed approach.

DfTr Method Concerning PCA-Based Trojan Detection

The authors in [14] proposed a DfTr to improve PCA-based Trojan detection methods. They proposed to employ some scan chains instead of having only one global chain. They consider a circuit under design (at the placement step in the physical design) as a collection of sub-circuits and then split the scan chain of the circuit to some sub-chains equal to the number of the considered sub-circuits. Each sub-chain is suited to a sub-circuit. Cells of each sub-chain are physically near to each other. Then, by applying random vectors to a sub-chain and a zero vector, in which all bits have '0'value, to the other ones, switching activity is increased in a targeted sub-circuits and decreased in other ones. In Chapter 5, we will aim the same objective by proposing (1) an input-vector generation method, (2) a logic masking method and (3) an algorithm that employs these two methods.

Chapter 3

Advantaging Logic Masking to Ameliorate Logic-Testing Based Trojan Detection

3.1 Introduction

It was mentioned in Chapter 2 that low transition rate (rare) signals are suitable options for employing in the trigger part of hardware Trojans (especially functional ones). In fact, in order to have very low observable effects on circuit outputs, hardware Trojans are designed such that they are triggered only in one (or few) specific occasional condition (s). Trojan attackers can make such a condition by combining a few rare signals.

In this chapter, we introduce a DfTr approach that leverages logic masking to reduce the number of rare signals in the circuit under masking; in other words, it aims to hinder difficult-to-trigger Trojans insertion and, consequently, increase the chance of Trojan activation. We called this proposed approach "AMELIORATION: Advantaging logic Masking to amELiorate functIOnal/stRuctural testing bAsed Trojan detectION". Regarding this approach, we proposed a logic masking algorithm that aims at mystifying circuits' original-functionality and removing rare signals. A circuit masked by AMELIORATION has two challenges for Trojan attackers: 1) lack of knowledge about

the original functionality of the circuit and 2) shortage of rare signals for designing hard-to-detect hardware Trojans. If there is no rare signal in a circuit, Trojan attackers must produce ones by performing specific logic operations on normal signals. The produced rare signals can then be employed for the activation of hardware Trojans. However, the production of rare signals increases the Trojans size; therefore, the Trojans affect the circuit side-channel parameters, such as power consumption or delay of paths, more [6]. Hence, removing rare signals hinders designing hard-to-detect hardware Trojans.

3.2 Contributions

The contributions of this chapter are as follows:

- We propose a logic masking algorithm that simultaneously concerns mystifying the original functionality and removing rare signals. It searches the whole circuit to find the best signals for inserting XOR/XNOR key-gates satisfying both the mentioned objectives.
- We propose a security metric that simultaneously employs "highly balanced Hamming distance achievement" and "rare signal elimination".
- We present the obtained Hamming distance for various combinational circuits, chosen from the ISCAS’85 benchmarks, to evaluate the efficiency of our proposed logic masking algorithm.
- We study the effects of logic masking on mystifying the functionality of sequential circuits. For this purpose, we run two logic masking algorithms on the combinational parts of sequential circuits chosen from ISCAS’89 benchmarks. The

increases in the sizes of the finite-state-machine (FSM) of the modified sequential circuits, including both the number of states and state transitions, are presented. This method can resist against SAT attack [18] because of the misleading behavior of the circuit for incorrect keys, which will be explained later in this paper.

- We present a CAD tool, called LOG-STAT (Logic simulation, Security, and Trust Analysis Tool) and uploaded in [68]. It is /designed and developed using the c# programming language. It takes a gate-level netlist and applies logic masking algorithms. The tool can perform logic simulation, calculate the obtained Hamming distance of masked combinational circuits, and estimate FSM extension (according to the increased number of the modified FSM's states and transitions as a result of the logic masking algorithm). LOG-STAT can also extract and report the area, power consumption, and delay overheads for the employed logic masking algorithm at the gate level.

3.3 Proposed Solution

As mentioned in the previous chapter, the shortcoming of using AND/OR gates as key-gates is that attackers can easily guess the correct key. For such key-gates, the correct key is always the non-controlling value. This problem does not exist when using XOR/XNOR key-gates. Figure 3.1 illustrates this point. In this figure, circuit 'a' includes three AND gate; assume that it is the original circuit. On the top of each signal in Fig. 3.1, a numerical pair shows the probabilities of having '0' and '1'. The original circuit output (O3) has a 6% probability of having '1'. Circuit 'b' in this figure is obtained by inserting an XOR key-gate into the original circuit. This key-gate increases the probability of having '1' at

the circuit output to 13%, and its correct key is ‘0’. The XOR key-gate in circuit ‘b’ is changed to an XNOR key-gate in circuit ‘c’ using De Morgan laws, but the correct key remains ‘0’. As observed in this circuit, an attacker cannot guess the correct key only by seeing an XOR or XNOR key-gate. Now, imagine that one uses an OR gate instead of the XOR key-gate in circuit ‘b’; then, circuit ‘d’ is obtained. Likewise, the use of De Morgan laws in circuit ‘d’ changes the added OR gate to NOR and AND gate in the circuits ‘e’ and ‘f’, respectively. The correct key for both the circuits ‘d’ and ‘e’ is ‘0’, equal to the noncontrolling value of the OR and NOR gates. The correct key for the circuit ‘f’ is changed, but it is still the noncontrolling value (of the AND gate, equal to ‘1’). As observed in the circuits (‘b’ and ‘c’) or (‘d’, ‘e’, and ‘f’), the use of De Morgan laws does not change the signal probability of a circuit’s nets. In addition, even if the use of OR, AND, NOR and NAND gates increases the signal probability slightly more than the use of XOR and XNOR key-gates, an attacker can easily neutralize them using their noncontrolling value as the correct key.

In order to solve the mentioned shortcoming, we offer a logic masking algorithm that employs our proposed security metric for inserting XOR/XNOR key-gates. This metric concerns both rare signal elimination and logic masking quality. The next section presents this security metric and logic masking algorithm in details.

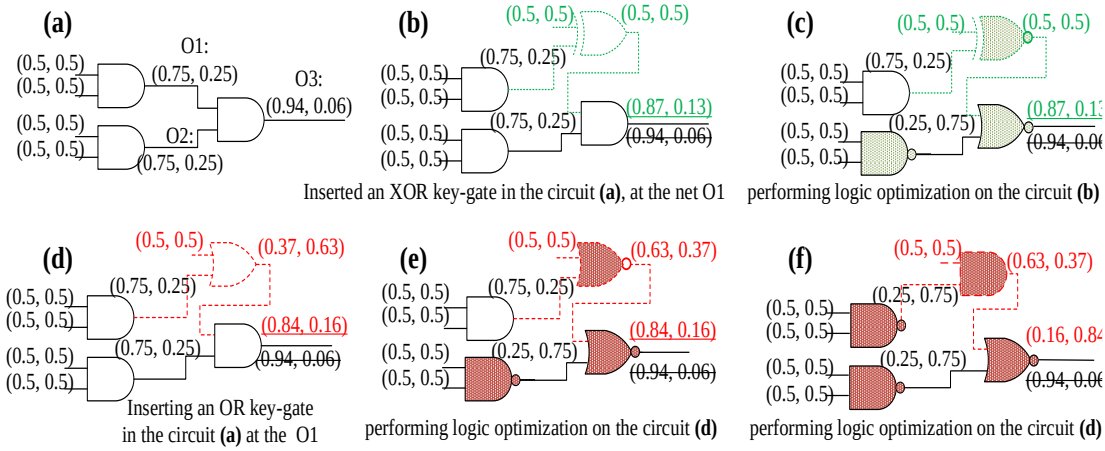


Fig. 3.1 A circuit with three AND gates; the numerical pair on the top of each signal shows its probability of being '0' and '1';

(a) original circuit, *PoB1(O_3) = 6%,

(b) masked circuit using an XOR key-gate in circuit **(a)** ➔ [PoB1(O_3) = 13%], CK^+ = 0,

(c) changed circuit using De Morgan laws in circuit **(b)** ➔ key-gate changed to XNOR, CK = 0,

(d) masked circuit using an OR key-gate in circuit **(a)** ➔ [PoB1(O_3) = 16%], CK = ^NCV = 0,

(e) changed circuit using De Morgan laws in circuit **(d)** ➔ key-gate changed to NOR, CK = NCV = 0,

(f) changed circuit using De Morgan laws in circuit **(d)** ➔ key-gate changed to AND, CK = NCV = 1,

* PoB1(x): Probability of Being '1' in signal x + CK: Correct Key ^NCV: Non-Controlling Value

3.3.1 Security Metric

The proposed security metric, named Security-Metric$_{HAHB}$ (HAHB: Hardware Trojan Avoidance & Hamming Distance Balanced), is measured based on Eq. 3.1:

$$\text{Security-Metric}_{HAHB} = \omega_1 HA + \omega_2 HB \tag{3.1}$$

$$HA = 1 - \frac{RS - Min_{(RS)}}{Max_{(RS)} - Min_{(RS)}}$$

$$HB = 2 * (0.5 - |0.5 - \text{Hamming distance}|)$$

where HA and HB are parameters defined for each signal of the circuit under masking. Parameter HA of a signal represents the number of rare signals (RS) that remain in the

circuit if this signal is used for inserting an XOR/XNOR key-gate. For all the signals in the circuit, one can obtain their RS parameter. A signal which has the greatest (smallest) RS is $Max_{(RS)}$ ($Min_{(RS)}$). The fraction in the HA formula is the simplest means to scale the range of RS to the range (0, 1). Similarly, the parameter HB corresponds to the percentage Hamming distance (HD) that would be achieved if a signal were used for inserting a key-gate. The closer the Hamming distance is to 50%, the higher HB is obtained. In Eq. 3.1, ω_1 and ω_2 are weights assigned to HA and HB, respectively, with $0 \leq \omega_1 \leq 1$, $0 \leq \omega_2 \leq 1$, and $\omega_2 + \omega_2 = 1$. In our experiments, we examined different values for these weights. Please note that to count rare signals, one needs a threshold for signal transition probability. The authors of [10] assumed that signals are rare if their transition probabilities are less than 0.01. We use the same assumption.

3.3.2 Greedy Algorithm

We use a greedy algorithm to insert key-gates. It inserts a key-gate into every signal in the circuit and then calculates Security-$Metric_{HAHB}$ for each signal. Then, it selects the best place according to the Security-$Metric_{HAHB}$. Algorithm 3.1 describes the whole process. It accepts a combinational circuit netlist and a key length as its inputs and returns a masked netlist and a Boolean vector that contains the correct key. The algorithm has two constants. The first one (RARE_THRESHOLD) is the threshold of assumed rare signals; this threshold has been explained in the previous section. The second constant is employed to allow determining the number of random test vectors.

Algorithm 3.1: AMELIORATION Algorithm (Inputs: *netlist*, k*eyLength,* **Output:** *maskedNetlist*, *correctKey* [k*eyLength*] **)**

Constant: RARE_THRESHOLD**,** TRANSITION_VARIABILITY_THRESHOLD*;*

1: **Levelize-netlist** *netlist; // O(N)*
2: **Generate-Test-Vectors (Return:** *inputVectors,* CorrectO*uputVectors***);** *// O(2N²)*
3: **For** *A = 1* **to** *A <= keyLength* **do** *//repeated K times*
4: **Select-Key-Gate** *KG***;** /*randomly between XOR or XNOR*/
5: **Save-Correct-Key** *correctKey* [A++]; /*corresponfing to KG*/
6: **Contact-One-Bit-With-A-Random-Value** to *inputVectors*;/*corresponding to *KG*/
7: **For each** net_B **in** *netlist* **do** *//repeated N times*
8: **Insert-key-gate** *KG* , net_B **;**
9: **Set-to-0** *transitionCounter of* all the nets**;**
10: **For** *C = 1* **to** *C <= vectorIndex* **do** *//repeated M times*
11: **Cycle-based-Simulation (***inputVectors [C],* **Return:** *faultyOutputVectos[C]***);** *//O(N)*
12: **Hamming-Distance-Calculation (***faultyOutputVectos, CorrectOutputVectors*; **Return:** *HD* **);** *//O(N)*
13: **Save** *HD* **to** net_B.HD**;**
14: **Set-to-0** *rareSignalCounter*;
15: **For each** net_D **in** *netlist* **do** *//repeated N times*
16: **If** (*transitonCounter of* net_D */ vectorIndex)* **<** RARE_THRESHOLD **do**
17: rareSignalCounter ++;
18: **Save** *rareSignalCounter* **to** net_B.RS
19: **Undo key-gate** *KG*/* from */ net_B**;**
20: **End for**
21: **For each** net_E **in** *netlist* **do** *//repeated N times*
22: **Calculate-*Security-Metric*** net_E; /*using net_E.*HD* and net_E.*RS* */
23: **Choose-Net** *ChosenNet*/*a net with the highest Security-$Metric_{HAHB}$*/;
24: **End for**
25: **Insert the selected key-gate** on *ChosenNet*;
26: **End For;**
27: **Return** *netlist, correctKey***;**

Function: Generate-Test-Vectors (Outputs: *inputVectors, ouputVectors* **)** */ /O(M * (N + N)) ∵ M ≈ N ➔ O (2N²)*

28: **Set-to-0** *vectorIndex*, *transitionCounter* for all the nets**;**
29: **Do:** *//O(k)*
30: **Generate-One-Random-Test-Vector** *testVector;*
31: **Save** *inputVectors [++vectorIndex] = testVector;*
32: **Cycle-based-Simulation (***testVector*, **Return:** *responseVector***);** *//O(N)*
33: **Save** *ouputVectors [vectorIndex] = responseVector;*

```
34:     For each net_i in netlist do //repeated N times
35:         Update-Transition-Rate of net_i; /* deviding TransitionCounter by vectorIndex */
36:         Update-Transion-Rate-Variability of net_i ;/* newTratransionRate - oldTratransionRate */
37:         Update maxTransitionRateVariability; /* keep the biggest transion rate variability */
38:     End For;
39:  While (maxTransitionRateVariability of netlist > TRANSITION_VARIABILITY_THERESHOLD);
40:Return inputVectors, ouputVectors;

Function: Cycle-based-Simulation (Input: testVector; Output: outVector ) //O(N)
41:  For each net_i in netlist do //repeated N times
42:     If net_i connected to a primary input do
43:         Assign the primary input value to net_i
44:     Else
45:         Uptade-Net-Value net_i /*accordign to logic function of the cell driving net_i */
46:     Update-Transition-Counter of net_i ; /*If the vale of net_i is changed */
47:End For
48:Return outVector;

Function: Hamming-Distance-Calculation (Inputs: vector1, vector2; Output:HD ) // O (M * P) ∵
M ≈ N, & P << M ➔ O( N)
 49: Set-to-0 HD;
 50: For each vector_i in vector1 do //repeated M times
 51:     For each bit_j in vector_i do //repeated P times
 52:         If vector1 [i][j] != vector2 [i][j] do
 53:             HD++;
 54: Return HD;
```

At first, the algorithm uses a function that logically levelizes the netlist (line 1) by traversing all the circuit nets from the primary inputs to the primary outputs and labeling all the circuit cells. The netlist has to be levelized because a cycle-based simulation is used a few steps ahead. The next function (line 2) is Generate-Test-Vectors that generates and applies random input test vectors to the circuit and stores the output vectors as the correct outputs. Any input vector being applied to the circuit changes the value of some nets. As the circuit is combinational, the switching activity (transition rate) for each net

is calculated by dividing the number of transitions of the net by the number of the applied input vectors. The transition rate of each net converges to a value when sufficiently many input vectors are applied to the circuit; afterward, more input vectors do not significantly affect the transition rate. If the change in the transition rate for all the nets is less than a threshold (TRANSITION_VARIABILITY_THRESHOLD), the function Generate-Test-Vectors ceases to generate new input vectors.

The main part of the algorithm is the loop between lines 3 and 27, which repeats until all key-gates are inserted. In each key-gate insertion round, a key-gate (KG) is randomly selected to be either XOR or XNOR, and its correct key value is saved in correctKey (lines 4-5). One bit, corresponding the inserted key-gate, is set with a random value and contacted to all the generated test vectors (line 6) because when a key-gate is inserted, its key-input is observed as a new primary input of the circuit. To find the best signal for KG, it is inserted into each signal, one by one, and the generated input vectors are simulated in the modified circuit (For structure line 7-20). Using the results of these simulations, two parameters, the HD and the number of rare signals (RS), are obtained for the net that KG is inserted on (lines 10-18). When HD and RS are obtained for a net, the algorithm undoes the inserted key-gate KG (line 19) and tries another signal. When HD and RS are obtained for all the nets, Security-Metric$_{HAHB}$ is calculated according to Eq. 1 for each net, and the net with the highest Security-Metric$_{HAHB}$ is chosen for KG (lines 21-25). This process repeats until all the key-gates are inserted.

This algorithm and the fault analysis-based (FAB) algorithm proposed in [12] are greedy and iterative algorithms; they have the same time complexity. The time complexity of each function and loop is shown in Algorithm 3.1. At the beginning of the algorithm, the time complexity of the levelize function has the order N, which equals the number of nets in the circuit. Afterward, there is the generating test vectors function,

which has an outer loop including an inner loop and the simulation function. As observed in Algorithm 1, the inner loop is repeated N times to check all the nets. The simulation function is order N because all nets of the netlist must be reassigned. The outer loop is finished when it observes the measuring parameter (transition rare of the nets) does not have significant changes. According to our experiments, the repeat of the outer loop depends on the circuit size and is less than or equal to N for TRANSITION_VARIABILITY equal to 0.01%. As a result, the time complexity of this loop is in order $2N^2$.

The first loop in Algorithm 3.1 is repeated K times, where K is the number of the key-gates. It has two loops, which are repeated N times. In one of these loops, for each net of the netlist, the simulation function (with the order N) repeats for the number of generated test vectors (almost equal to N, as mentioned above). Then, there is the Hamming distance calculation function, which is of the order N, as observed in Algorithm 1. As a result, the main loop in the algorithm is repeated K ($N^3 + N^2 + N$) times. K is negligible compared to N; because of the area overhead, the number of key-gates must be much less than the number of nets (i.e., 5% of the number of the nets). As a result, the maximum time complexity of the algorithm is calculated based on Eq. 3.2:

$$O(N + 2N^2 + KN^3 + KN^2 + KN) \rightarrow O(N^3) \tag{3.2}$$

It is noteworthy that in the algorithm, a simple logic simulation function with a zero-delay model is used. Since logic masking is performed at the gate level on the combinational circuits, this simulation is sufficient to calculate the Hamming distance. This method applies test vectors and observes the switching activity of the signal to calculate the signal probability. This simulation method is more efficient and less time

consuming than the statistical probability calculation [69]. Using the unit delay model or more accurate delay models is also more time-consuming than the zero-delay model [70].

3.4 Experimental System Setup and Results

3.4.1 Experiment setups

As explained, Security-Metric$_{HAHB}$ used in the algorithm has two factors, ω_1 and ω_2. These factors should be assigned by the circuit designer according to the required IP piracy or hardware Trojan prevention level. They can have a value between 0 and 1 such that the sum of them is exactly 1. During experiments, we first used the values '1' and '0' mutually for each one. In these cases, the security metric can be calculated using Eq. 3.3 and Eq. 3.4:

$$\omega_1 = 0, \omega_2 = 1 \rightarrow \text{Security-Metric}_{HAHB} = HB \tag{3.3}$$

$$\omega_1 = 1, \omega_2 = 0 \rightarrow \text{Security-Metric}_{HAHB} = HA \tag{3.4}$$

n addition, we assigned 0.5 to both ω_1 and ω_2 during our experiments. Therefore, the security metric is measured based on Eq. 3.5:

$$\omega_1=\omega_2=0.5 \rightarrow \text{Security-Metric}_{HAHB}=0.5(HA + HB) \tag{3.5}$$

As mentioned, a toolset called LOG-STAT has been developed in C#; LOG-STAT reads a circuit at the gate level and runs different masking algorithms, including the algorithm proposed in this paper, in addition to the FAB algorithm [12] and random key-

gate insertion [42]. The ISCAS-85 [71] and ISCAS-89 [72] benchmarks were used for the experiments. Table 3.1 presents the details of the circuits masking.

Table 3.1 Details of the used benchmark circuits

Circuits	Input	Output	Gate	Flip-flop
c432	36	7	160	-
c499	41	32	202	-
c880	60	26	383	-
c1355	41	32	546	-
c1908	33	25	880	-
c3540	50	22	1669	-
c5315	178	123	2307	-
c6288	32	32	2406	-
c7552	207	108	3512	-
s510	19	7	211	6
s641	35	23	379	19
s838	34	1	446	32
s1238	14	14	508	18
s1494	8	19	647	6
s5378	35	39	2779	179

LOG-STAT reports the Hamming distance and the number of rare signals. We compare the results of the proposed algorithm and the FAB algorithm presented in [12]. We separately use values of (1, 0), (0, 1), (0.5, 0.5) for (ω_1, ω_2). Additionally, LOG-STAT estimates the area, power and delay overheads at the gate level.

To estimate the dynamic power for every single cell of a circuit, one can use Eq. (3.6) [73]:

$$\text{Dynamic Power} = C.V_{DD}^2 .f.P \tag{3.6}$$

In this equation, C is the average load capacitance of the cell, VDD is the power supply, f is the frequency of the circuit, and P is the probability of a signal transition from 0 to 1 at the output of the cell. Thus, the total dynamic power consumption (TP) for a sequential circuit with n cells can be estimated using Eq. 3.7.

$$\mathrm{TP} = \sum_{i=1}^{n} C_i . V_{DD}^2 . f . P_i . Fout_i \tag{3.7}$$

For combinational benchmark circuits, we replace $f.P_i$ by the number of transitions (T) at the outputs of the cells. Thus, TP for a combinational circuit with n cells can be estimated using Eq. 3.8.

$$\mathrm{TP} = \sum_{i=1}^{n} C_i . V_{DD}^2 . T_i . Fout_i \tag{3.8}$$

where T_i and $Fout_i$ are the numbers of transitions and fan-outs of the i^{th} cell of the circuit, respectively.

To estimate delay and area overheads, LOG-STAT uses the default values of NanGate 45-nm primitive cells [74] and the method proposed in [73]. These overheads do not depend on the masking algorithm. In fact, only the number of used key-gates affects these overheads.

3.4.2 Simulation Results

As discussed in Section 3.1, we can use different security metrics in the proposed algorithm. We proposed a security metric including two factors: highly balanced

Hamming distance achievement and rare signal elimination. They are associated with weights ω_1 and ω_2, respectively. In addition, we proposed three cases in which these weights are different. In the following, we present the results for each case, including the Hamming distance and number of rare signals. We also compare the results of each case with those obtained in related work. Then, in the fourth section, overheads are presented and discussed.

3.4.2.1 IP Piracy Robustness Evaluation

The Hamming distance would be the only concern if we set $\omega_1 = 0$ and $\omega_2 = 1$. The Hamming distance results after execution of the proposed algorithm using Security-Metric$_{HB}$ are shown in Fig. 3.2. This figure illustrates that the average Hamming distance increases with the number of inserted key-gates. In all benchmarks, the proposed algorithm succeeded at achieving a Hamming distance of 50%. Fig. 3.3 also illustrates the number of rare signals after each key insertion. As observed in this figure, since the Security-Metric$_{HB}$ only concerns the Hamming distance, rare signals are not eliminated.

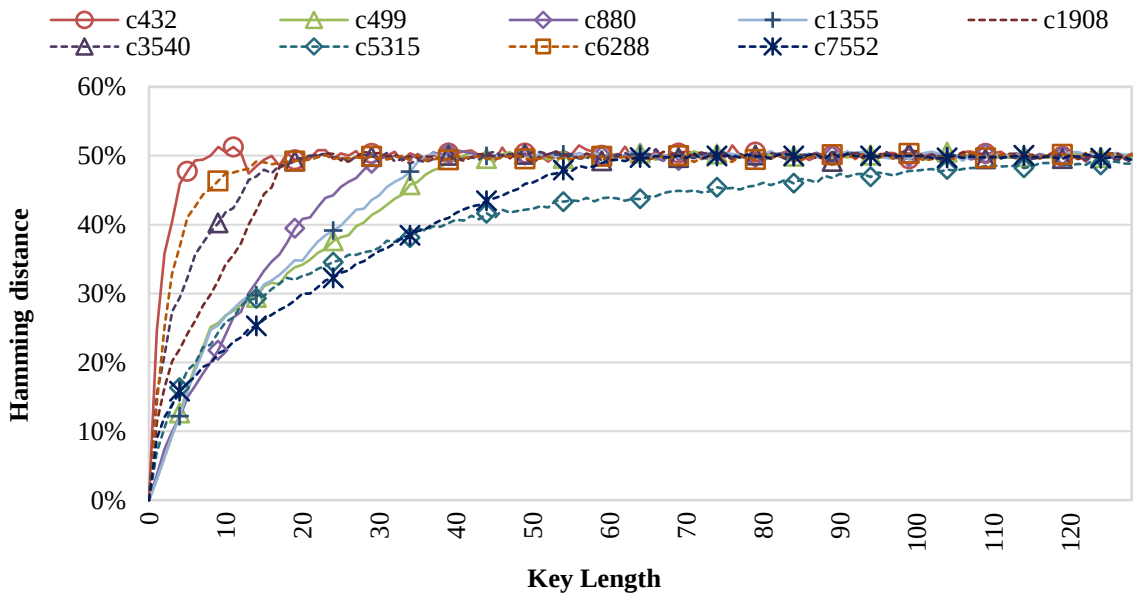


Fig. 3.2 Hamming distance of the output bits for each key length, using (ω_1, ω_2) = (0, 1)

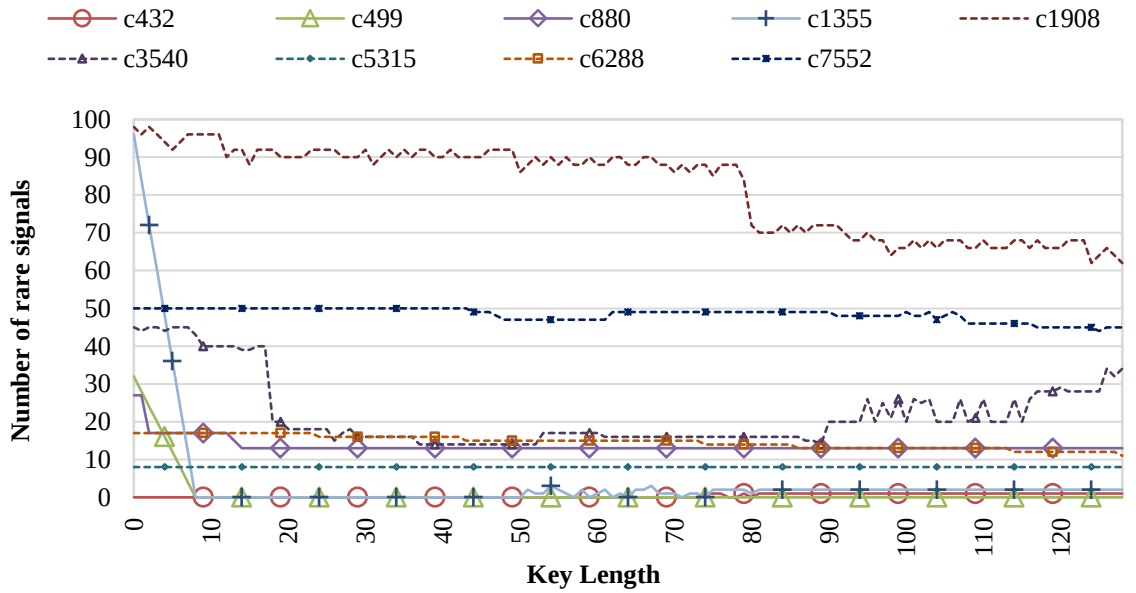


Fig. 3.3 The number of rare signals for each key length using (ω_1, ω_2) = (0, 1)

3.4.2.2 Hardware Trojan Avoidance Evaluation

The second case is when one is only concerned with Trojan threats and sets ω_1 and ω_2 to '1' and '0', respectively. As mentioned above, the proposed algorithm in this case only masks a circuit such that it will have as fewer as possible rare signals regardless of the

masking quality. Figure 3.4 presents the number of rare signals for each benchmark for different key lengths. This figure implies that for all the ISCAS-85 circuits, the proposed algorithm using Security-Metric$_{HA}$ has succeeded at reducing the number of rare signals to less than 10. The Hamming distance results for this case are depicted in Fig. 3.5. As expected, in this case, a highly balanced Hamming distance is not achievable.

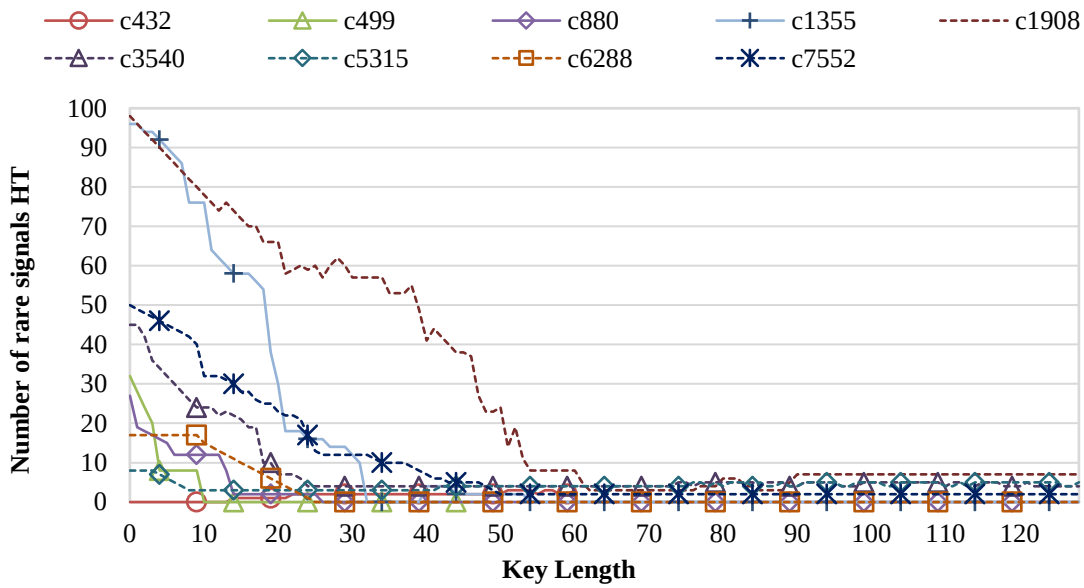


Fig. 3.4 The number of rare signals for each key length using (ω_1, ω_2) = (1, 0)

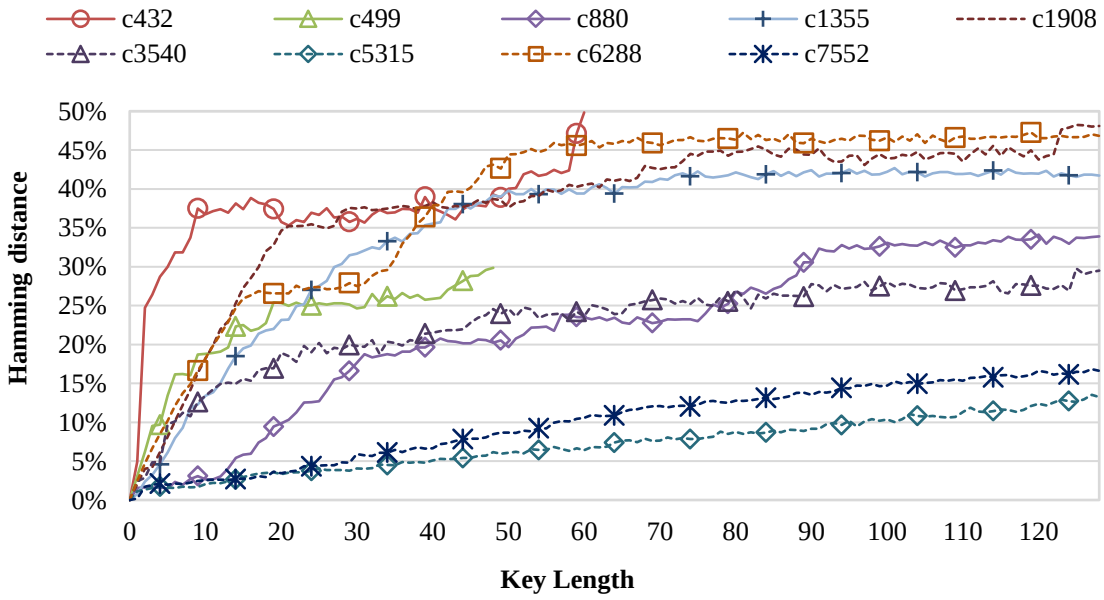


Fig. 3.5 Hamming distance of the output bits for each key length, using (ω_1, ω_2) = (1, 0)

3.4.2.3 Joint Evaluation of IP Piracy Protection and Trojan Avoidance

The Hamming distance and the number of rare signal results corresponding to the use of Security-Metric$_{HAHB}$ (with $\omega_1 = 0.5$ and $\omega_2 = 0.5$) in the proposed algorithm are depicted in Fig. 3.6 and Fig. 3.7, respectively. However, Security-Metric$_{HA}$ and Security-Metric$_{HB}$ present better results solely for either the balanced Hamming distance or rare signal elimination, in comparison to Security-Metric$_{HAHB}$; as inferred from the figures, Security-Metric$_{HAHB}$ is effective for both criteria. For instance, after inserting approximately 35 key-gates, fewer than 10 rare signals remained in most of the circuits (except for c7552 and c1908, which need 110 and 127 key-gates, respectively). In the same respect, 55 key-gates are sufficient to achieve a Hamming distance of more than 40% for the majority of the circuits (except for c7552 and c5315, which need 75 key-gates for this concern).

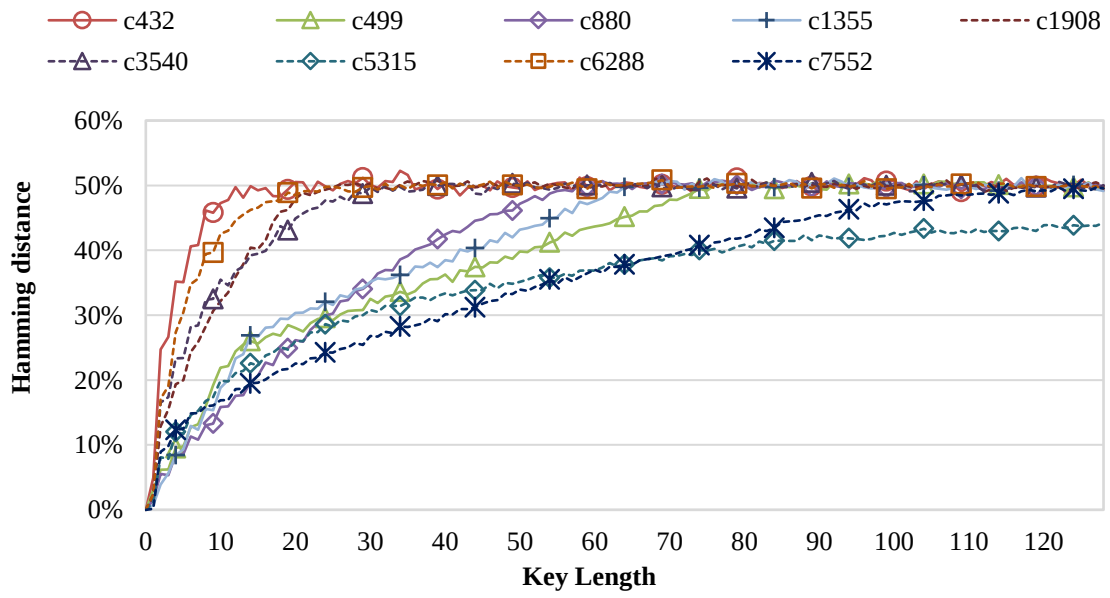


Fig. 3.6 Hamming distance of the output bits for each key length, using (ω_1, ω_2) = (0.5, 0.5)

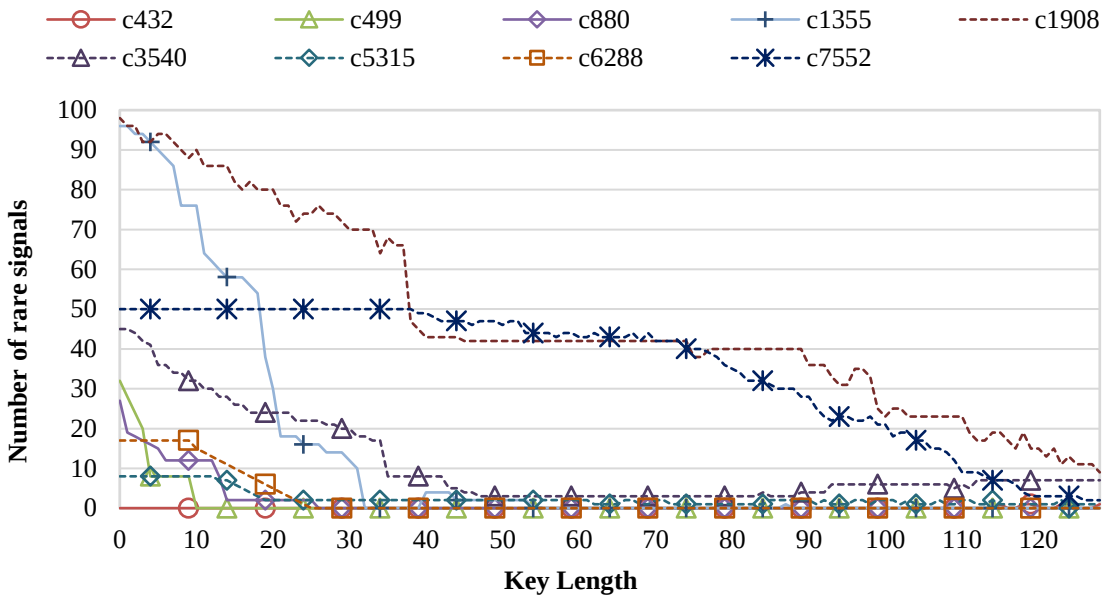


Fig. 3.7 The number of rare signals for each key length using (ω_1, ω_2) = (0.5, 0.5)

Table 3.2 provides a comparison between the Hamming distance results of the proposed algorithm (using Security-MetricHA_{HB} or Security-Metric$_{HB}$) and the FAB algorithm presented in [12]. The maximum achievable Hamming distance and the number of required keys to obtain this Hamming distance, presented in Table 3.2, show that the proposed algorithm using Security-Metric$_{HB}$ yields better results than the fault FAB algorithm for all the circuits, except for c499 and c880, for which the results are the same. In three circuits, c432, c1908, and c6288, Security-Metric$_{HAHB}$ requires slightly fewer key-gates than that of the FAB algorithm to reach 50% Hamming distance. In other circuits, the FAB algorithm requires fewer key-gates than Security-Metric$_{HAHB}$ to achieve a Hamming distance of 50%; moreover, in none of these circuits does it require any effort to reduce the number of rare signals.

Table 3.2 HD obtained by the FAB algorithm [12] and the proposed algorithm using security-metric$_{HB}$ and security-metric$_{HAHB}$

circuit	# of key-gates used to reach the most balanced HD			Most balanced HD		
	FAB	**HB**	**HAHB**	**FAB**	**HB**	**HAHB**
c432	17	8	12	50%	50%	50%
c499	40	40	74	50%	50%	50%
c880	28	28	58	50%	50%	50%
c1355	42	36	64	50%	50%	50%
c1908	28	19	23	50%	50%	50%
c3540	22	20	27	50%	50%	50%
c5315	97	122	112	44%	50%	44%
c6288	27	18	24	50%	50%	50%
c7552	89	64	112	46%	50%	50%
Average				**48.9%**	**50%**	**49.2%**

For the largest circuit, c7552, Security-Metric$_{HB}$ obtains a Hamming distance of 50% by inserting 64 key-gates; however, the upper bond of the achieved Hamming distance using FAB algorithm is 46% using 89 key-gates, whereas Security-Metric$_{HAHB}$ achieves Hamming distances of 46% and 50% by inserting 89 and 112 key-gates, respectively.

It is worth noting that in Table II, Security-Metric$_{HB}$ used more key-gates (122 ones) for c5315 to achieve a Hamming distance of 50%. However, the FAB algorithm and Security-Metric$_{HAHB}$ can obtain a maximum Hamming distance of 44% using 97 and 112 key-gates, respectively. Figure 3.3 shows that the use of Security-Metric$_{HB}$ results in a Hamming distance of 44%, like the FAB algorithm, but it needs to insert only 97 key-gates. For this case, neither the FAB algorithm nor Security-Metric$_{HAHB}$ in the proposed algorithm could reach a Hamming distance of 50% for c5315.

Figures 3.2, 3.4, 3.6 and 3.7 show the tradeoff between the numbers of used key-gates and the objectives of the proposed algorithm and security metric. However, sometimes the algorithm cannot improve this metric, even by accepting more key-gates. For instance,

as shown in Fig. 3.6, by 112 key-gates, we reach a Hamming distance of 44% in c5315, and adding more key-gates does not yield a better effect.

3.4.2.4 Overhead Results

Figures 3.8, 3.9 and 3.10 show the area, delay and power overheads, respectively, for all the key lengths used in the proposed algorithm incorporating Security-Metric$_{HAHB}$. LOG-STAT reported these figures. It uses the default value of the cell area/delay/power information listed in the open cell library of NanGate 45 nm [74]. The tool also uses Eq. 3.7 and 3.8 for the dynamic power overhead estimation. These figures show that as expected, there is a direct relation between the number of used key-gates and the overheads. The overheads increase by accepting more key-gates, usually regardless of the used key-gate insertion algorithm. This observation is also mentioned in previous work, such as [12].

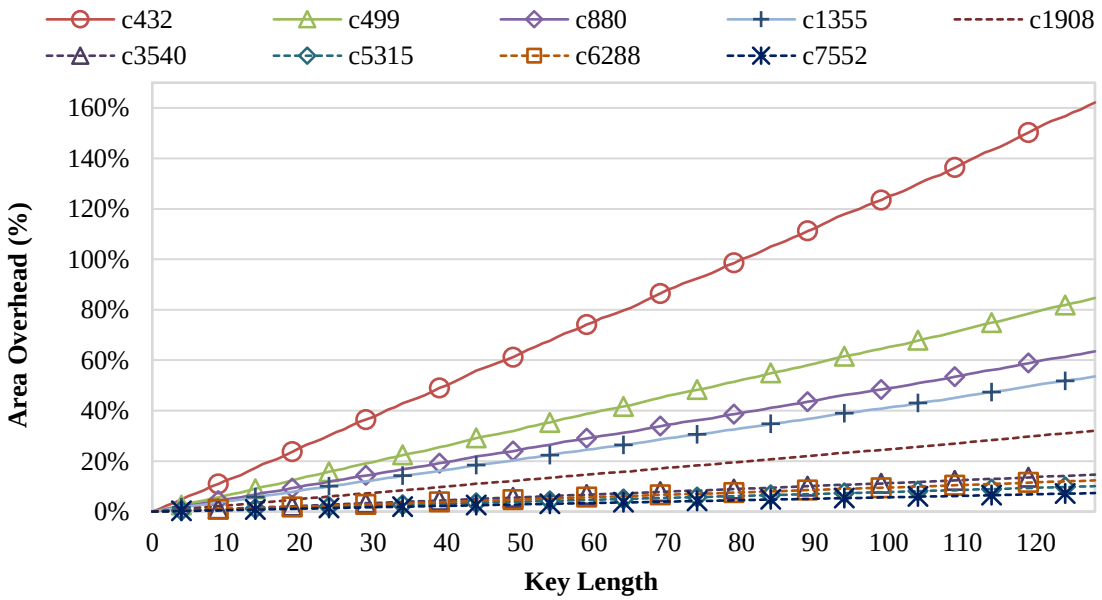


Fig. 3.8 Area overhead of the proposed algorithm incorporating Security-Metric$_{HAHB}$ for different key lengths

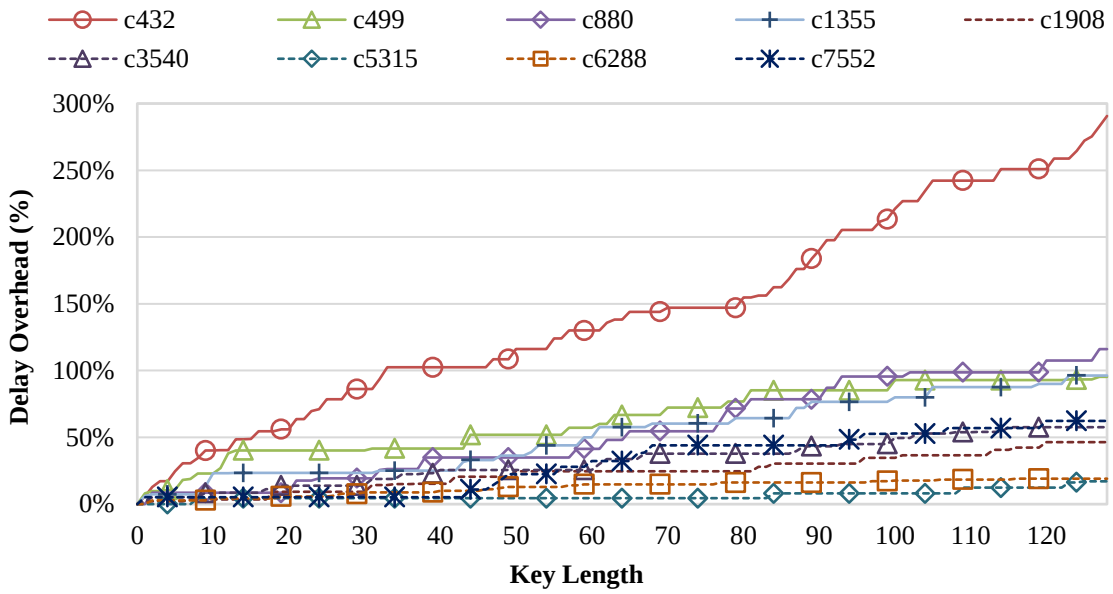


Fig. 3.9 Delay overhead of the proposed algorithm incorporating Security-Metric$_{HAHB}$ for different key lengths

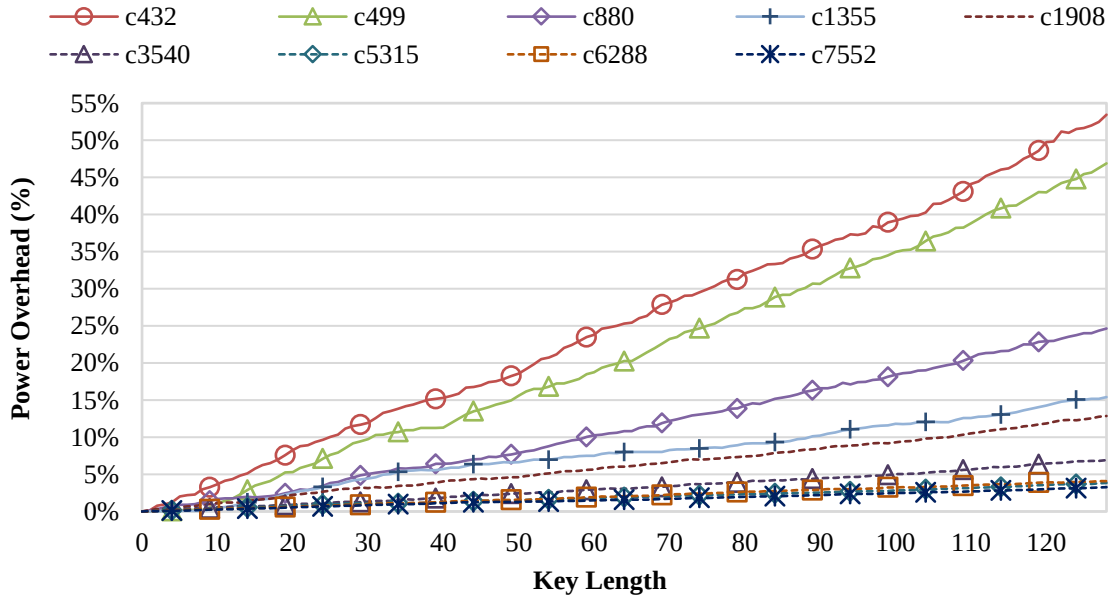


Fig. 3.10 Power overhead of the proposed algorithm incorporating Security-Metric$_{HAHB}$ for different key lengths

Table 3.3 gathers all data about the overheads for the two executions of the algorithm in which either of Security-Metric$_{HB}$ or Security-Metric$_{HAHB}$ are used. In both cases, the overheads are reported according to the shortest key length results for the closest

Hamming distance to 50%. The overheads only depend on the number of used key-gates. It is observed when the algorithm incorporating each security metric employs (almost) the same number of key-gate and, consequently, the overheads (e.g. c5315 and c6288). In addition, the area and power overheads for larger circuits (ones having more than 2k gates) are less than the ones for smaller circuits. This is also the case for the results of the FAB algorithm published in [12].

Table 3.3 Area, power, and performance overheads for each benchmark regarding the number of used key-gates to reach a hamming distance of 50%

Circuit	No. of key-gates used to reach the most balanced HD		Area overhead		Dynamic power overhead		Performance overhead	
	HB	HAHB	HB	HAHB	HB	HAHB	HB	HAHB
c432	8	12	9.9%	14.6%	2.9%	4.4%	34.3%	51.5%
c499	40	74	26.1%	48.3%	11.4%	20.9%	33.14%	61.3%
c880	28	58	14.7%	17.6%	5.1%	10.4%	15.17%	31.4%
c1355	36	64	14.9%	26.4%	5.5%	9.8%	25.05%	4.5%
c1908	19	23	4.7%	5.4%	2.1%	2.4%	5.65%	6.8%
c3540	20	27	2.3%	3.06%	0.9%	1.2%	7.98%	10.8%
c5315	112	112	8.5%	8.5%	3.1%	3.2%	8.7%	8.8%
c6288	22	24	2.1%	2.30%	0.7%	0.8%	5.91%	6.5%
c7552	64	112	3.6%	6.2%	1.54%	2.7%	5.27%	9.1%

The performance overhead is presented in Table 3.3, column 4. As observed in the table, the delay overhead is quite significant. This problem can be solved if designers only try to use nets belonging to noncritical paths.

3.5 Logic Masking Effects on Sequential Circuits

In Chapter 2, it has been introduced that the first step of sequential circuit masking is to conjoin a fake FSM and the original FSM of a circuit into a new one; this is practical to perform at the RTL level. In the second step, the states of the fake and original FSMs are interlocked. The final FSM must have several transitions between the fake and original FSM; in addition, if it goes to the original states by applying incorrect keys, it must generate incorrect outputs and return to the fake FSM's states. In Chapter 2, it was also mentioned that logic masking can be used as a solution to interlock the original and fake FSMs. For this purpose, one can synthesize the obtained FSM from the first step and then separate the combinational part and modify it using a logic masking algorithm.

The combinational part of a sequential circuit feeds the circuit's flip-flops and primary outputs. The outputs of this part have incorrect bits if the circuit was modified by a logic masking algorithm and one or some of the added key-inputs obtain incorrect values. Hence, such a circuit will incorrectly operate and have failures in two cases: 1) when one of its primary outputs is directly connected to one output of the combinational part and it has incorrect value; 2) an incorrect value for one output of the combinational part makes a fault in one or some flip-flops, and so this error will most likely bring the circuit to an incorrect state. As a result, a masked sequential circuit having (even) a few added key-inputs and being applied incorrect values is very likely to operate incorrectly. Thus, in this work, instead of reporting the failure rate, two other criteria are chosen to present the effects of logic masking on sequential circuits. The proportions of the number of states and transitions in a masked sequential circuit before and after performing logic masking are the two criteria used to demonstrate the effects of logic masking.

Incorrect states, which happen due to incorrect keys, either belong to the original and

fake FSMs or are new ones that never happen in the circuit reached from the first step. In other words, the effect of added key-gates may make some new states that were never reached before the key-gate insertion because of the circuit logic. The increase in the difference between the number of states before and after performing logic masking makes the final circuit more obscure.

As mentioned in Chapter 2, the SAT attack is powerful for finding the correct key for a masked combinational circuit. Nevertheless, sequential circuits masked by the use of logic masking, such as the method explained above, can effectively defend against the SAT attack. The reason is that for some keys, the circuit temporarily passes through correct states with incorrect outputs or vice versa.

To see the mentioned criteria caused by logic masking algorithms in this work, we chose the random and proposed logic masking algorithms and applied them to the combinational parts of gate-level sequential circuits chosen from the ISCAS-89 benchmark. The random logic masking algorithm, as mentioned in Chapter 2, randomly inserts key-gates into a circuit. LOG-STAT has implemented this algorithm, although it does not distinguish and protect the clock and reset signals from key-gate insertion. The implementation of the proposed algorithm for the sequential circuit in LOG-STAT includes separating the combinational part of sequential circuits and considers each flip-flop as a pseudo primary input and output. Then, the algorithm inserts key-gates according to the calculation of Hamming distance and the number of rare signals considering pseudo primary inputs/outputs such as normal primary input/outputs. Finally, when all key-gates are inserted, the pseudo primary inputs/outputs are replaced by flip-flops.

To estimate the number of states and transitions, one can traverse all the FSM, but it is

infeasible even in normal size circuit because the number of states exponentially increased by the number of F.Fs. Another solution, which is fast and easy-to-implement, is to apply many random input vectors. In this work, 10000 random vectors are first selected and then applied to a masked circuit and its original circuit. Figures 3.11 and 3.12, respectively, show the proportion of the number of states and transitions before and after performing the proposed logic masking. Figures 3.13 and 3.14 show the same result before and after performing the random masking algorithm.

These figures show that both of the algorithms have almost the same results. As observed in these figures, these algorithms for all of the circuits – except the smallest circuit, s510 – double the number of states and transitions by inserting a few key-gates (less than 10). The numbers of states and transitions are easily increased by as much as 200% using the random logic masking algorithm, which is very fast to implement, and a maximum of 32 key-gates are used for all of the circuits except for s510. As a result, logic masking can be an appropriate solution to mask sequential circuits.

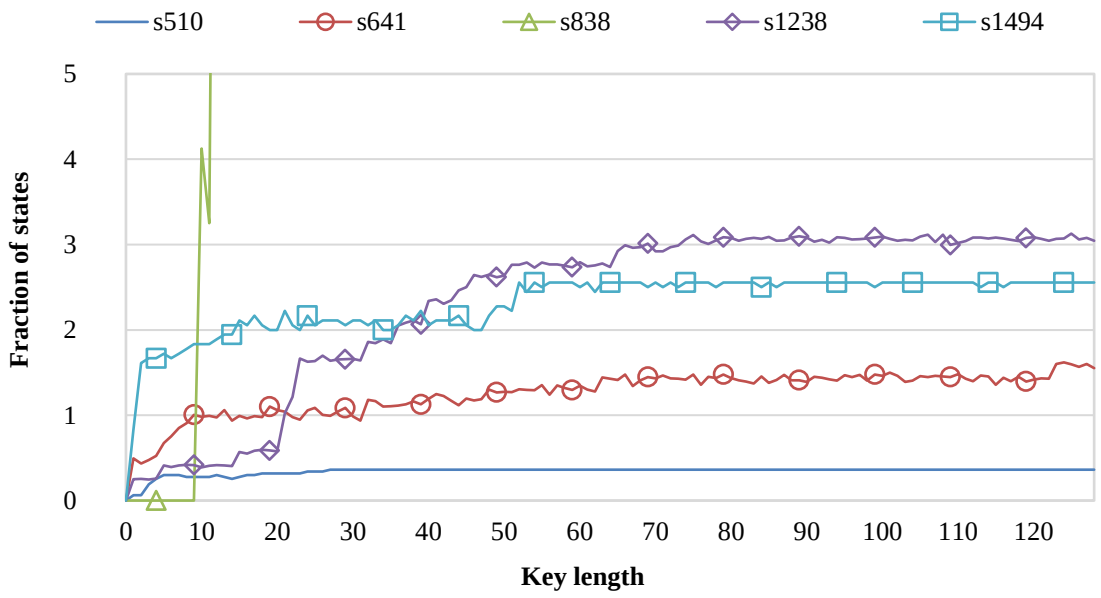


Fig. 3.11 The fraction of the number of states before and after performing the proposed logic masking algorithm along with Security-Metric$_{HAHB}$ for each key lengths

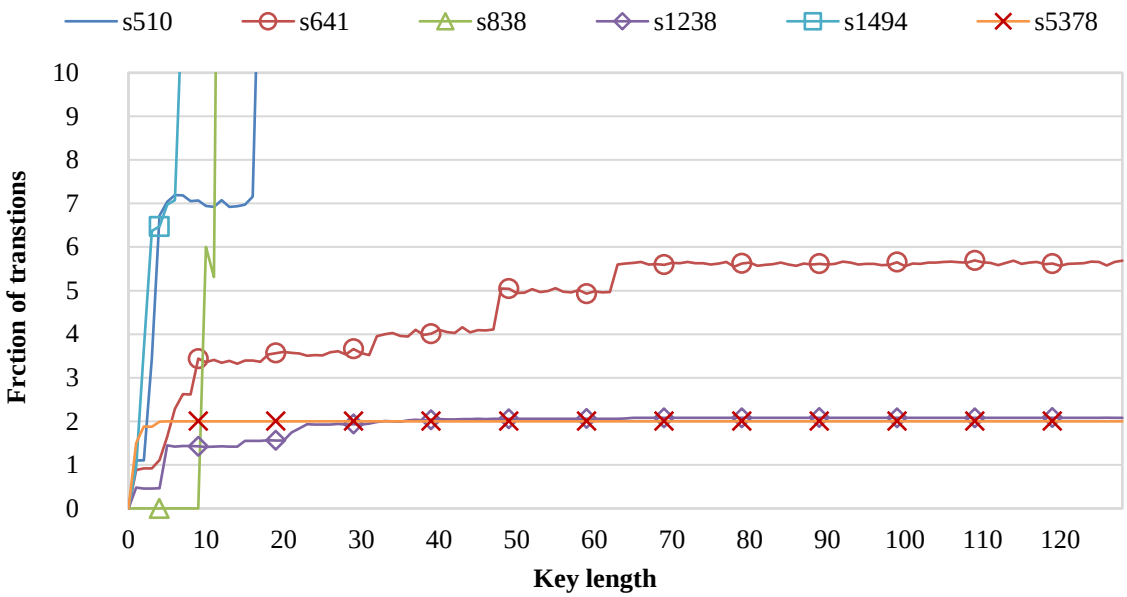


Fig. 12 The fraction of the number of transitions before and after performing the proposed logic masking algorithm along with Security-Metric$_{HAHB}$ for each key lengths

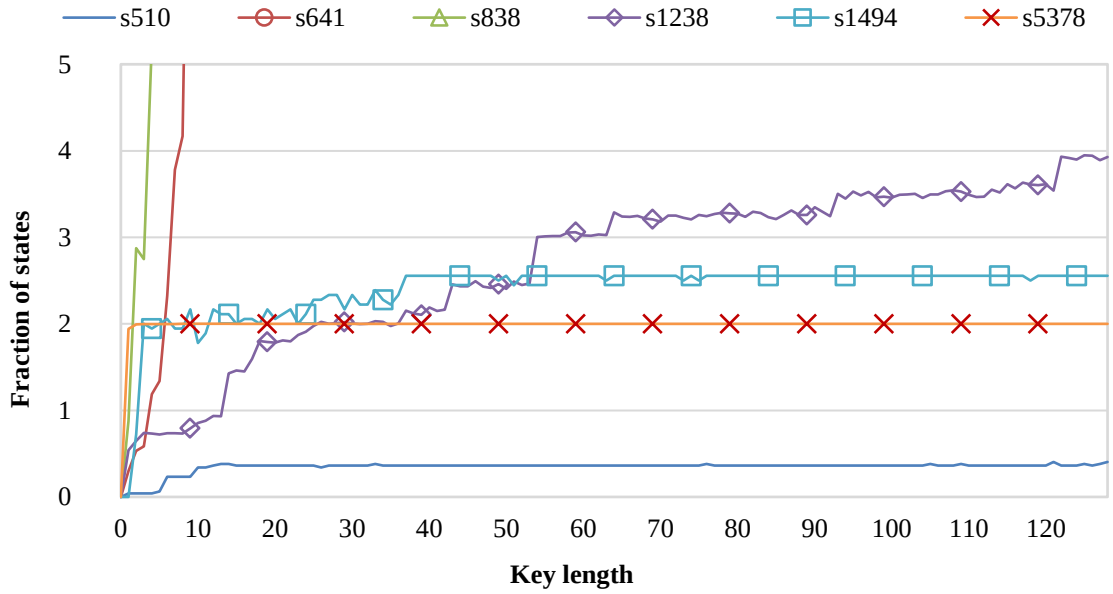


Fig. 13 The fraction of the number of states before and after performing the random logic masking algorithm

Please note that one can develop efficient logic masking algorithms in terms of the increase in the number of states and state transitions. In this work, we only tried to present a basic view of the effects of the logic masking on sequential circuits. Our future work

includes comparing the costs and benefits of interlocking of the fake and original FSM at the RTL and the use of logic masking at the gate level.

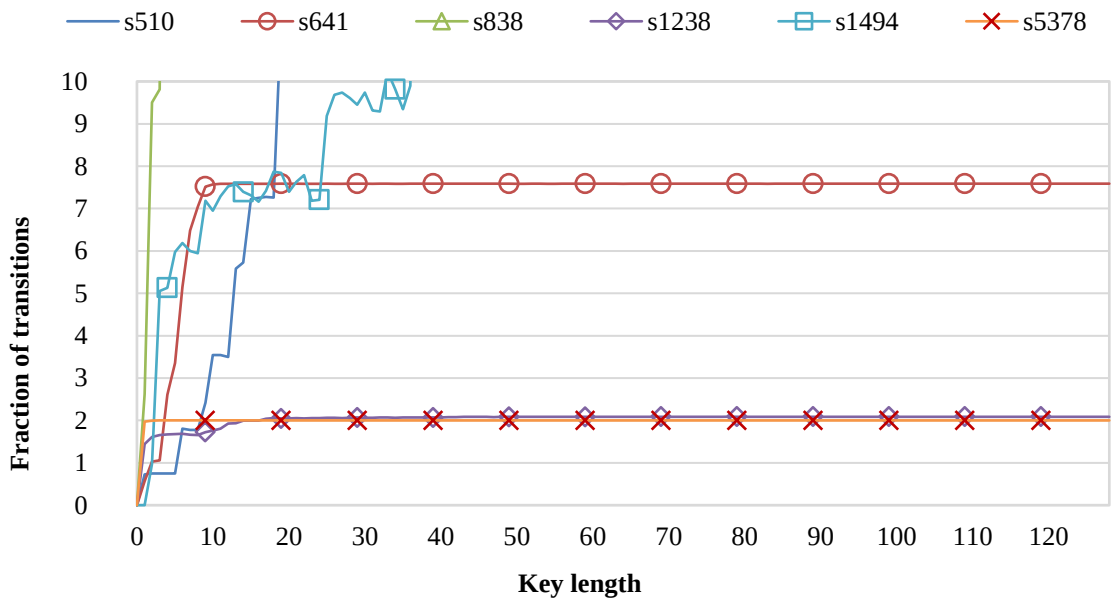


Fig. 14 The fraction of the number of transitions before and after performing the random logic masking algorithm

3.6 Conclusions

In this chapter, we proposed a security metric for logic masking algorithms. It has two different objectives: 1) highly balanced Hamming distance achievement and 2) Trojan avoidance, by eliminating rare signals. Assigning different weights to these objectives can make several benefit functions for logic masking algorithms. In addition, we introduced an iterative and exhaustive logic masking algorithm employing a benefit function based on the proposed security metric. The algorithm, in each itself iteration, directly analyzes the effect of one key-gate on all the circuit signals under masking. Then, the algorithm selects the most appropriate signals for key-gate insertion, according to a

benefit function in that each of the mentioned objectives has its own weight.

We ran the proposed algorithm including a benefit function in which both the objectives have the same weights. In addition, we ran the algorithm two more times such that in each of them, it employed a benefit function including one of the objectives. Then, we compared the results of these three cases with the FAB proposed in [12]. The results and comparisons show that the proposed algorithm employing the benefit function with equal weights can significantly approach the ideal output balanced Hamming distance (50%), more or less similar to the FAB, and effectively make Trojan insertion difficult by removing rare signals.

We analyzed the logic masking effects on the number of states and transitions of sequential circuits. The results show that even a few key-gates inserted in the combinational part of sequential circuits easily double the numbers of states and transitions.

In addition, we presented a CAD tool in this chapter. We developed a logic simulator and the experiments of this chapter in the tool. Moreover, we measured the area, performance and power overheads of different logic masking algorithms at the gate level by functions implemented in the coding environment of LOG-STAT. These measurements showed that the overheads do not depend on the masking algorithm. In fact, only the number of used key-gates affects these overheads.

Chapter 4

Employing Logic Masking to Facilitate Path Delay Analysis-Based Trojan Detection

4.1 Introduction

As mentioned in the previous chapters, the second benefit of logic masking methods and algorithms is to hinder hardware Trojan insertion, thus they are considered as DfTr approaches. In this chapter, we introduce a new DfTr approach that leverages logic masking to facilitate path delay analysis (PDA)-based Trojan detection methods. We called this approach "ESCALATION: Employing logic maSking to faCilitate pAth deLay Analysis based Trojan detectION". Regarding this approach, we propose a logic masking algorithm using XOR/XNOR gates and multiplexer cells as key-gate. It masks the functionality of circuits while improving the efficiency of PDA-based Trojan detection methods. Apart from logic masking benefits, the objective of the proposed algorithm is to generate fake short paths for nets that only belong to long paths, because shorter paths have smaller delay variations [13, 57]. This point will be discussed later in this chapter.

4.2 Contributions

The contributions of this chapter are as follows:

- We detail the ESCALATION approach and an algorithm based on that along with three ideas for its improvement. The ideas are explained and illustrated with examples.
- We report the improvement of Trojan detection probability (TDP) obtained by the proposed algorithm, for technology-mapped circuits.
- We compare the masking quality of the proposed approach with one masking algorithm proposed in [9], namely HARPOON.
- We compare the results and overheads of the proposed algorithm with a trust-driven retiming algorithm (TDR) proposed in [13]. TDR has the same objective as the proposed algorithm, but it does not make any changes in the original functionality. The comparisons are performed at the gate level because the TDR results and overheads are reported at the gate level.
- We validate the Trojan detection efficiency of the proposed algorithm at the layout level. Our experimental results show that our algorithm improves TDP based on PDA. It also provides logic masking.
- We validate the logic masking efficiency of the proposed algorithm. We calculate the Hamming distance (HD) of the outputs of masked circuits. We use HD as a usual metric to compare the masking quality of our algorithm with that of previous work. The results show that the proposed algorithm gives better HD results than the *random masking,* presented in Chapter 2.

4.3 ESCALATION Based Algorithm

A hardware Trojan can increase the delay of some paths in its host circuit. These paths can include either the payload part of the hardware Trojan or the nets that drive the trigger part. Note that most of the nets in a circuit usually belong to more than one path. A hardware Trojan delay effects are less detectable if it is investigated among long paths, because of the presence of process variation. Consequently, a net cannot be a suitable choice for Trojan attackers if the net belongs to at least one short path. In other words, the shortest-path passing a net is the best option to investigate potential hardware Trojans interacting with the net. This path selection approach can be used in all PDA-based Trojan detection methods. However, there may be nets whose shortest path is not short enough for high Trojan detection probability. As a result, in ESCALATION, we aim to generate short fake paths using key-gates for such nets.

4.3.1 Vulnerable points in path delay analysis-based Trojan Detection

As mentioned in Chapter 2, Vulnerable points in path delay-based Trojan detection are nets that only belong to long paths. For each net (N) of the circuit (C), there are some paths (P) that pass the net (N). Accordingly, the definition of the most vulnerable net (MVN) of a circuit, with n net, is obtained by *f* function, as follows:

Def. 4.1:

$$SP(N) = Min \{\forall P \in C: P \text{ is passing } N\}$$

$$MSP(C) = Max \{SP_i(N_i) \mid SP_i \,\&\, N_i \in C, i = 1... n\}$$

$$f: MVN \mapsto MCP$$

$$f = \{\exists N: N \in \text{the nets of MSP}, N \notin \text{the nets of other } SP_i\}$$

In Def. 4.1, the MSP of a circuit is the maximum (longest) path of the shortest-paths (SP). In other words, MSP is the shortest path of the most vulnerable net, and its value is greater than that of the shortest path of other nets.

4.3.2 Motivations

A trust-driven retiming algorithm as a DfTr approach, proposed in [13] and named TDR, was introduced in Chapter 2. The TDR algorithm reduces the MSP value by adding extra FFs. It increases the TDP of PDA-based Trojan detection methods but since the functionality of the circuit under retiming is not changed and no logic/state masking is generated, it does not prevent IP/IC piracy. ESCALATION, on the other hand, does protect circuits against IP/IC piracy as it uses the logic masking approach. It also uses the potential of masking to improve the TDP of PDA-based Trojan detection methods.

Like other logic masking methods, in ESCALATION approach, modifications are done at the gate level and in the combinational part of circuits. The key-gates used in ESCALATION are XOR/XNOR and MUX cells. The ESCALATION flow has three main

steps. In the first step, for each net, the shortest-path passing the net is found and selected. The second step is to sort the selected paths and find the longest one (MSP). The MSP has the least TDP among all the selected paths, and it includes the most vulnerable net. The third step is to insert a key-gate such that the most vulnerable net belongs to a new path shorter than the MSP. These three steps continue until all nets belong to at least one short-enough path or until all key-gates are inserted.

4.3.3 Examples

In order to show how ESCALATION works, and highlight some effective empirical hints, examples are presented below. The examples are at the gate level. The unit delay model is used at these examples. It is thus assumed that each cell in the netlist has 1 delay unit, and the interconnection delays are negligible.

Example 1: A path from a primary input (PI) to a primary output (PO) is shown in Fig. 4.1.a. Imagine that this path is the MSP of a circuit and the shortest path for net ‘N4’ (the most vulnerable net). In the original circuit, gate ‘G’ immediately precedes ‘PO1’; and net ‘N3’ connects the output of gate ‘C’ to an input of gate ‘D’. Thus, this path in the original circuit has 6 delay units and runs from primary input ‘PI1’, through the cells {A, B, D, E, F, G}, ending up at primary output ‘PO1’. In this figure, the designer has added the XOR ‘X1’ and the MUX ‘M1’ to make a shorter path for ‘N4’. The new path starts from key-input ‘KI2’ and bypasses gates ‘A’ and ‘B’. In addition, cells ‘F’ and ‘G’ are bypassed by ‘M1’ and a fan-out to ‘N5’, shown by the fan-out ‘F1’. As a result, the new path is generated with 4 delay units, running from ‘KI1’, through cells ‘X1’, ‘D’, ‘E’, and ‘M1’, and ending up at ‘PO1’.

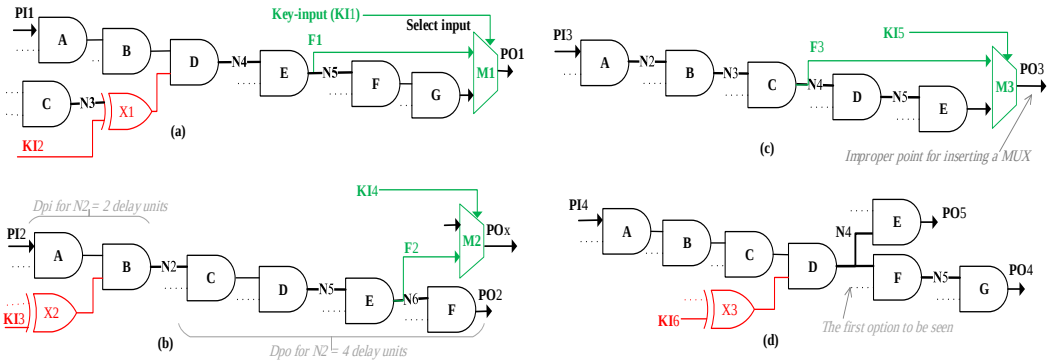


Fig. 4.1 Four paths from a primary input to a primary output. **A)** The shortest path of net N_4, from PI_1 to PO_1 in the original circuit, and from KI_2 to PO_1 in the masked circuit. **B)** The shortest path of nets N_2, and N_5, from PI_2 to PO_2 in the original circuit, an XOR and MUX of not use that cannot make shorter paths for N_2 and N_5, in the masked circuit. **C)** The shortest path for N_3 and N_5, from PI_3 to PO_3 in the original circuit, and an added MUX in the masked circuit that makes a shorter path for N_3 but not for N_5. **D)** The shortest path of net N_4, from PI_4 to PO_5, and the shortest path for N_5, from PI_4 to PO_4, in the original circuit. X_3 makes the shortest paths for both N_4 and N_5.

In order to efficiently insert XOR/XNOR or MUX key-gates, three important ideas are considered in the proposed algorithm:

1. Inserting an XOR/XNOR key-gate makes a shorter path if the delay of the nearest PI to the target net (Dpi) is greater than the delay of the target net to the nearest PO (Dpo) in the MSP. Otherwise, it is better to use a MUX key-gate.
2. Each inserted key-gate adds a delay unit (one level of gate) to all the paths that include the inserted key-gate. This may defeat the purpose. In order not to have this problem, the defender should avoid inserting key-gates in selected paths longer than the maximum accepted MSP.
3. There is often more than only one vulnerable net in the circuit. Thus, key-gate insertion should be done in such a way as to decrease the number of vulnerable nets as much as possible.

These problems and ideas are shown and illustrated in Fig. 4.1 and the following

examples.

Example 2: assume the path in Fig. 4.1.b, including {PI2, A, B, C, D, E, F, PO2}, is the MSP with 6 delay units in the original circuit. In addition, nets 'N2' and 'N5' are the most vulnerable nets. In this figure, 'X2' and 'M2' are added to make shorter paths for 'N2' and 'N5'. As shown, the insertion of 'X2' before 'B' does not generate any shorter paths for 'N2'. Likewise, the insertion of 'M2' before a PO and making a fan-out from 'N6' does not make any shorter paths for 'N5'. For a net like N2, in order for its Dpi to be smaller than its Dpo, a MUX must be used. Otherwise, like N5, an XOR/XNOR key-gate must be used.

Example 3: assume that the path in Fig. 4.1.c, including {PI3, A, B, C, D, E, PO3}, is the MSP with 5 delay units in the original circuit. In addition, nets 'N3' and 'N5' are the most vulnerable nets. In this case, the insertion of MUX 'M3' before PO3 and making a fan-out from 'N4' makes a shorter path for 'N3' including {PI3, A, B, C, M3, PO3}. It has 4 delay units. But this modification adds a delay unit to the shortest path of 'N5'. This path includes {PI3, A, B, C, D, E, M3, PO3} and has 5 delay units. As a result, such POs must not be used.

Example 4: assume that the path in Fig. 4.1.c, including {PI4, A, B, C, D, F, G, PO4}, is the MSP with 6 delay units in the original circuit. In addition, net 'N5' is the most vulnerable net. Also, assume that the shortest path of 'N4' has 5 delay units. It includes {PI4, A, B, C, D, E, PO5}. If 'X3' is inserted before 'F', a short path with 3 delay units, including {KI6, X3, F, G, PO4}, would be made for N5. The shortest path passing 'N¬4' would be the MSP with 5 delay units. As seen in Fig 4.1.d, 'X3' is inserted before 'E'; thus, the shortest paths for 'N5' and 'N4' have 4 and 3 delay units, respectively. They include {KI6, X3, D, F, G, PO4} and {KI6, X3, D, E, PO5}. The tip from this example is

that it is sometimes better to consider a few paths other than just the MSP. As seen in this example, it is better to insert the XOR key-gate before ‘E’ to solve the MSP problem, and the shortest path of ‘N4’ is a potential MSP.

4.3.4 Proposed Algorithm

In the following section, we explain an algorithm based on the ESCALATION approach. Algorithm 4.1 shows the pseudo code of the proposed algorithm. The algorithm takes a gate-level netlist, a number as the key lengths (the number of key-gates), and an integer number as the targeted MSP value (line 1). At first, the shortest path for each net of the circuit is found and stored in the set SIPSet (line 2). To find the shortest path for each net, a breadth-first search (BFS) is done from the target net to the primary inputs and primary outputs.

A structure is used to store the information on the shortest path (SIPInfo) for each net. It includes two pointers to the selected path (SP) and the target net (TN), and also two variables. One variable contains the delay from the net to its nearest PI (Dpi), and the other one contains the delay of the path from the net to its nearest PO (Dpo). The summation of these two variables is the delay of the path (Value). In order to find MSP, the paths in SIPSet must be sorted according to their delay (Value). MSP is stored in MSPSet (0). In addition, the potential MSPs are gathered in a set named MSPSet (line 3). The potential MSPs are the paths that belong to SIPSet and have delays greater than the targeted delay for the MSP value.

Algorithm 4.1. The proposed ESCALATION based algorithm

1-- **Inputs***: Netlist; KEYLENGTH; TARGET_MSP;* **Output***: Masked Netlist;*

```
2-- SIPSet ← Gather the shortest path for each net
3-- MSPSet ← Sort SIPSet then store MSP and potential MSPs
4-- counter ← 0
5-- While MSPSet.NotEmpty and counter < KEYLENGTH do
6--     DP = DeepSearch (TARGET_MSP, MSPSET [0]);
7--     If MSPSet[0]. Dpi > MSP Set[0]. Dpo then
8--         XOR/XNOR Insertion (DP, DR);
9--     Else
10--        MUX Insertion (DP, DR);
11--    End if
12--    DR = DeepRecalculate (TARGET_MSP, MSP);
13--    counter ++;
14--    SIPSet ← Gather the shortest path for each net
15--    MSPSet ←Sort SIPSet then store MSP and potential MSPs
16-- End while
Structure SPInfo
        *Path   SP;
        *net    TN;
        *int    Dpi ;
        *int    Dpo ;
        *int    value
End Structure
```

The next step of the algorithm is a loop that includes the key-gate insertion functions (lines 8 and 10). As shown in example 1, if Dpi is greater than Dpo, many nets belonging to the fan-in cone of the target net can be used to insert an XOR/XNOR key-gate. Likewise, there are many candidates for inserting a MUX key-gate in the fan-out of the targeted net if Dpo is greater than Dpi. Please note that there is no need to search all the nets in the cones. A BFS can be performed in the cones with the maximum DeepSearch according to (1) (line 6):

$$\text{DeepSearch} = \begin{cases} \text{TargetedMSP} - \text{Dpo} - 1 & \text{if Dpi} >= \text{Dpo} \\ \text{TargetedMSP} - \text{Dpi} - 1 & \text{if Dpi} < \text{Dpo} \end{cases} \tag{4.1}$$

The XOR insertion function (line 8) is then executed if Dpi is greater than Dpo; otherwise, the MUX insertion function (line 10) is executed. In the cones, the most appropriate net for key-gate insertion is a net that will remove more nets from MSPSet. This is shown in example 2. If an XOR/XNOR key-gate is to be inserted, Dpi should be recalculated for all nets that belong to the fan-out cone of selected net. Likewise, if a MUX key-gate must be inserted, Dpo should be recalculated for all nets that belong to the fan-in cone of the selected net. There is no need to recalculate all the nets in the cones. A BFS can be performed on the cones with the maximum DeepRecalculate according to (2) (line 11):

$$\text{DeepRecalculate} = \begin{cases} \text{TargetedMSP} - \text{Dpi} - 1 & \text{if Dpi} >= \text{Dpo} \\ \text{TargetedMSP} - \text{Dpo} - 1 & \text{if Dpi} < \text{Dpo} \end{cases} \tag{4.2}$$

After a key-gate is inserted, SIPSet is updated (line 14) and again MSP and the potential MSPs of the modified circuit are identified and stored in MSPSet (line 15). The loop is finished when MSPSet is empty or the number of inserted key-gates is more than the key-length (line 5).

The time and memory complexities of the proposed algorithm depend on the BFSs done to find the shortest path for each net. Please note that inside the loop, the BFSs are performed in order to find the most appropriate net for key-gate insertion, but the number of nets in the cones (considering DeepSearch and DeepRecalculate) is much less than the number of nets in the circuit, and it is thus ignorable. Other parts of the algorithm are fixed as well. Assuming we are working with an extracted graph of the circuit, BFS takes

O(b^(d+1)) time and memory [75], where b is the branching factor (it is equal to the number of nodes in the biggest logic cone), and d is the distance from the starting node (it is equal to the maximum logic-depth in the circuit). As a BFS must be done for each net and in each iteration, the order of time complexity of ESCALATION is O (nkb^(d+1)), where n and k are the number of nets in the circuit and the number of inserted key-gates, respectively. In a typical case, as n>>b>>d, the ESCALATION complexity order can be estimated as O (kn). This means that the time complexity of the ESCALATION algorithm increases linearly according to the size of the circuit.

4.3.5 Measuring ESCALATION Efficiency

In the previous section, we presented an algorithm for key-gate insertion based on the ESCALATION approach. Since ESCALATION has two aims, there are two criteria that must be considered. First, how much logic masking is obtained. This is the primary aim in all masking methods. Second, how much the TDP of PDA-based detection methods can be improved, an important goal of the ESCALATION approach. In addition, the key-gate overheads such as area and circuit performance must be noted.

4.3.6 Masking Quality

In Chapter 2, it was mentioned that the number of output failures can be a simple metric to measure masking quality. The more output failures there are, the more masking is obtained. However, a more accurate metric is the average of the HD of correct and incorrect output bits while the circuit is fed by the correct and wrong key vectors. As a

result, the masking quality of two circuits obtained by applying two different masking methods on one circuit can be evaluated by comparing these masking metrics.

4.3.7 TDP Calculation

In order to know the TDP improvement when PDA is used, the TDPs of the original circuit and of its masked circuits must be compared. Moreover, MSP has the least TDP among the selected shortest paths. Hence, the TDP of MSP can be considered as a metric.

In order to calculate the delay variability of MSP and hence TDP, a Golden reference is required. As mentioned in Section II, the Golden reference is the variation of the targeted side-channel without Trojan effects. It can be obtained by either the Golden ICs or Golden-Free approaches. Assume that the Golden reference for PDA is reached using one of these approaches and that it follows Gaussian distributions or normal probability distribution functions (PDF). We call it the "Golden PDF". The Golden PDF is constructed by "μ_p and σ_p", where μ_p is the mean and σ_p is the standard deviation, as shown in Fig. 4.2. Likewise, in the ICs under test, the PDF of path delay is named "Trojan-suspected PDF", and constructed with "μ_T and σ_T", where μ_T is the mean and σ_T is the standard deviation. The TDP is calculated by comparing the Trojan-suspected PDF and the Golden PDF.

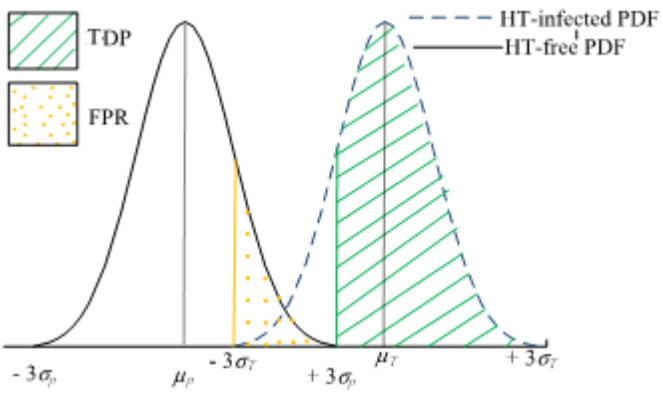


Fig. 4.2 The probability distribution functions of a path with a Trojan (dashed lines) and without any Trojan (solid lines). The area of the shaded part is equal to TDP with 0% FPR. The dotted area is FPR.

TDP is the probability that the Trojan-suspected PDF has a value of more than $+3\sigma_p$. In other words, TDP is equal to the area under the Trojan-suspected PDF between "$\mu_p + 3\sigma_p$" and "$\mu_T + 3\sigma_T$", as shown in Fig. 4.2. The TDP of a path can be formulated by Eq. 4.3:

$$TDP = Prob(a < x < b) = \int_a^b f_x(t)dt. \quad (4.3)$$

where 'x' is the path delay, and f_x is the PDF of the path delay. 'a' and 'b' are $\mu_p + 3\sigma_P$ and $\mu_T + 3\sigma_T$, respectively. According to the 3-sigma rule, and with 'b' approximated as infinity, we obtain:

$$TDP(\mu_p + 3\sigma_p < x < \infty) = \int_{\mu_p+3\sigma_p}^{\infty} f_x(t)dt. \quad (4.4)$$

In addition, according to PDF properties:

$$TDP(\mu_p + 3\sigma_p < x < \infty) = \int_{-\infty}^{\infty} f_x(t)dt - \int_{-\infty}^{\mu_p+3\sigma_p} f_x(t)dt.$$

$$= 1 - \int_{-\infty}^{\mu_p+3\sigma_p} f_x(t)dt. \quad = 1 - F_x(\mu_p + 3\sigma_p) \tag{4.5}$$

where F_x is the cumulative distribution function of Trojan-suspected PDF. Equation (4.5) can be used to calculate the TDP with less than 0.2% error. The use of the 3-sigma rule is the reason for this ignorable error. In fact, the Golden PDF can have a value higher than ($\mu_p + 3\sigma_p$) with less than 0.002 probability. In other words, two out of 1000 Trojan-free ICs are reported as Trojan-infected ICs. This fraction, illustrated by the dotted area in Fig. 4.2, is known as *false positive rate* (FPR).

FPR is the fraction of Trojan-free ICs that are reported as Trojan-infected ICs. A higher FPR can be accepted in order to have a higher TDP. For example, in Fig. 4.2, TDP is increased to 100% by accepting more FPR, the dotted area. To avoid the high FPR, more accurate and costly Trojan detection methods, such as layout image processing, must be used. There can be a tradeoff between the costs of FPR, other Trojan detection methods, and trustworthiness gained.

In order to get a 100% TDP, F_x in equation (4.5) should be zero. As a result, '$\mu_p+3\sigma_p$' in equation (4.5) should be less than 'μ_T-$3\sigma_T$' in Trojan-suspected PDF. FPR can be calculated by Eq. (4.6), similar to the TDP equation:

$$FPR = 1 - \int_{-\infty}^{\mu_T-3\sigma_T} g_x(t)dt. \tag{4.6}$$

where gx is the Golden PDF.

It is noteworthy that the interval changes of the Golden or Trojan-suspected PDF depend on die-2-die and with-in-die variation. For example, in 45 nm technology, they are 36% and 12% respectively [76]. These values together make Trojan detection very difficult. Fortunately, we can decrease them using some calibration structures. For instance, die-2-die effects can be removed from PDA using the method proposed in [57].

4.4 EXPERIMENTAL RESULTS

4.4.1 Experiment Setups and Assumptions

The experiments have been carried out on gate-level circuits from ISCAS-89 [72] and ISCAS-85 [71]. First, the circuits were elaborated by VERIFIC API [77]. Then the proposed algorithm (using the unit delay mode) was executed for different targeted MSP values. The algorithm was implemented using VERIFIC API and C++ programming. Afterward, all the modified circuits were synthesized by Design Compiler [78] and then placed and routed by SOC-Encounter [79]. The NANGATE 45 nm technology was used during the synthesis and physical design [74].

In order to perform a fair comparison between the proposed algorithm and previous works, we tried to use the same experiment flows and assumptions. We compare both MSP reduction and TDP improvement with the results of the TDR algorithm [13]. We also compare the logic masking quality of the algorithm with the [9] and [12], based on the number of output failures and HD. Finally, we report layout-level results.

4.4.2 TDP Results at Gate Level

Shekarian et al. used the unit delay model in their TDR algorithm [13]. They also reported the TDP improvement and MSP reduction at this level. In this model, zero correlation among the delay variation of cells is assumed, an unrealistic assumption. But the authors tried to make it acceptable by assuming a higher percent of delay variability. They assumed 60% cell delay variation due to process variation.

In order to compare the Trojan detection improvement (based on PDA) reached by the proposed ESCALATION algorithm and the TDR algorithm, four sequential circuits with different sizes are considered. Figures 4.3 and 4.4 give the results of the experiments. The first empirical observation shown by these experiments is that for each circuit there is a minimum MSP value that the proposed algorithms can reach. Going below this minimum value is impossible because adding new key-gates will create new vulnerable nets. Figure 4.3 shows the last four minimum reachable MSP values and their associated area overheads, shown in X and Y axes respectively. Note that the area overhead is calculated after synthesis. This figure shows that there is a direct link between MSP reduction and area overheads; a better MSP value (and therefore a better TDP) requires more key-gates and area overhead.

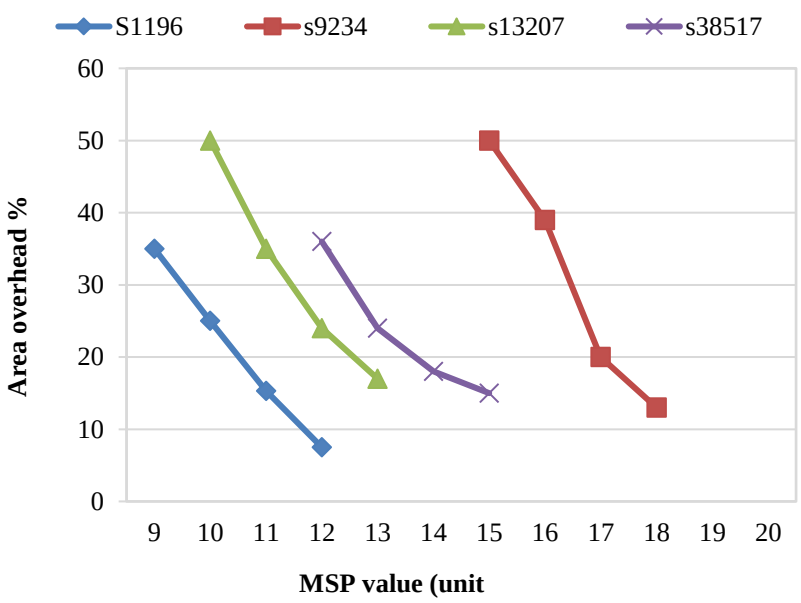


Fig. 4.3 The MSP value obtained (by ESCALATION) versus its required area overhead for four MSP values (the minimum reachable MSP value and three bigger values) in four sequential circuits.

Among all MSP values reachable by ESCALATION, the two MSP values for which ESCALATION makes area overhead as similar as possible to the area overhead made by TDR are selected and used in Fig. 4.4. The ESCALATION executions for obtaining these two MSP values are named ESCALATION1 and ESCALATION2 in Fig. 4.4. ESCALATION1 (2) needs a bit less (more) area overhead than TDR. Figure 4.4.a shows the MSP values obtained by these two ESCALATION executions and TDR. In Fig. 4.4.b, the area overheads of the two executions and TDR are compared. Shekarian et al. only reported the number of FFs added due to TDR execution as the area overhead [13]. In fact, the area overhead of the added FFs (including their area and required clock-route) is much bigger than the area overhead of key-gates. Thus, we calculate the TDR area overhead by multiplying the reported percentage of the added FFs by the percentage of the sequential area of the circuit. For example, for circuit s9234, the TDR algorithm adds 36 FFs, which equals 17% of the number of FFs in the circuit. As the sequential part of s9234 corresponds to 58% of the circuit area after synthesis, the modified circuit has a 9.9% area overhead. In Fig. 4.4.b, the area overheads of ESCALATION1 and

ESCALATION2 are prepared according to the area reports obtained by SYNOPSYS Compiler [78].

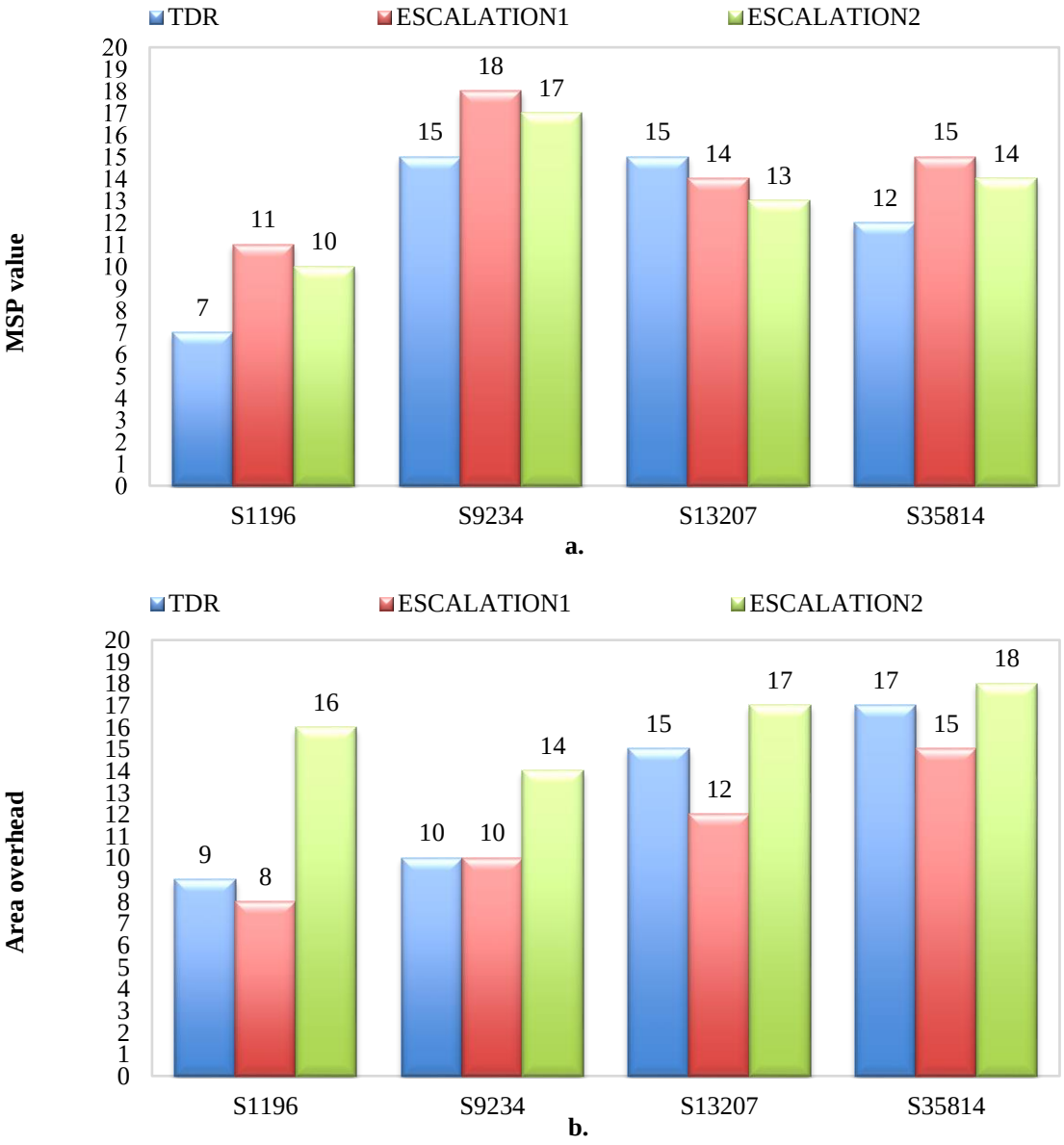


Fig. 4.4 Results of TDR and two executions of ESCALATION. ESCALATION1 (ESCALATION2) needs a bit less (more) area overhead than TDR. **a:** MSP values, **b:** area overheads

Figure 4.4 illustrates that the TDR algorithm achieves a slightly smaller MSP value with almost the same area overhead for the 3 smallest circuits; however, in one circuit

(s13207), the ESCALATION algorithm achieves a better MSP reduction with less area overhead. It is to be noted that heuristic algorithms do not always achieve the optimal result. The TDR algorithm is a heuristic one, and so it is difficult to understand why it does not always give better results than the ESCALATION algorithm.

Table 4.1 reports all the results gathered from the masked circuits by the ESCALATION algorithm with a 20% area overhead limitation. Columns 2, 3, and 4 show three variables for the original circuits: the MSP value, the TDP, and the required FPR for 100% TDP (RFPR), respectively. The same variables are reported for the masked circuit in Columns 5-7. The number of key-gates and the area overhead percentage are given in Columns 8 and 9, respectively. The area overheads are reported by Design Compiler. In two cases, before and after the ESCALATION algorithm execution, a Trojan (only one AND) is inserted in MSP and then TDP and RFPR are calculated. Comparing TDP and RFPR, in these two cases, shows that our algorithm can be an efficient DfTr approach. Indeed, the ESCALATION algorithm modified circuits in benefit of an average MSP relative reduction of around 60%. The average TDP absolute value is increased by around 34%. And the average RFPR absolute value is decreased by around 32%.

4.4.3 Masking quality

The number of output failures is a simple metric to qualify masking quality. Chakraborty et al. reported the masking quality of their algorithm, HARPOON, with this metric [9]. They used different percentages of area overheads: 5, 10, 15 and 20 percent. Thus, we executed our algorithm to obtain different MSP values. Then we select four MSP values which have similar area overheads close to the four considered area

overheads in [9]. In order to have a better comparison, we divide the number of output failures by the percentage of area overhead. This proportion is shown as normalized output failures in Fig. 4.5. In this figure, each circuit has an E or H label that stands for the ESCALATION and HARPOON algorithms, respectively. The results in Fig. 4.5 show that for each circuit, the proportion converges to the same number when the area overhead increases. It means that both algorithms have a similar logic masking quality for higher area overheads (with 15 or 20 percent area overheads for 2 circuits over the 3 studied circuits). For the area overheads less than 10 percent, the results are variable depending on the circuit. It is noteworthy that the ESCALATION algorithm has benefits for Trojan detection.

Table 4.1 MSP value, TDP and required FPR (to obtain 100% TDP) before and after performing the ESCALATION algorithm on sequential circuits, accepting 20% area overhead and assuming unit delay model and 60% cell delay variation

circuit	MSP value	TDP	RFPR	MSP value	TDP	RFPR	# of used key-gates	% of area overhead
	In the original circuit			by ESCALATION in the masked circuit				
S13207	35	10	91	14	38	63	741	17.7
S35814	38	8	93	13	41	60	297	17.2
S9234	43	5	96	16	31	70	165	22
S5378	21	23	78	11	51	50	109	21
S1423	20	22	78	7	80	21	67	21.5
S1196	18	24	77	10	55	46	31	21.1
Average	30	15	86	12	49	52		20

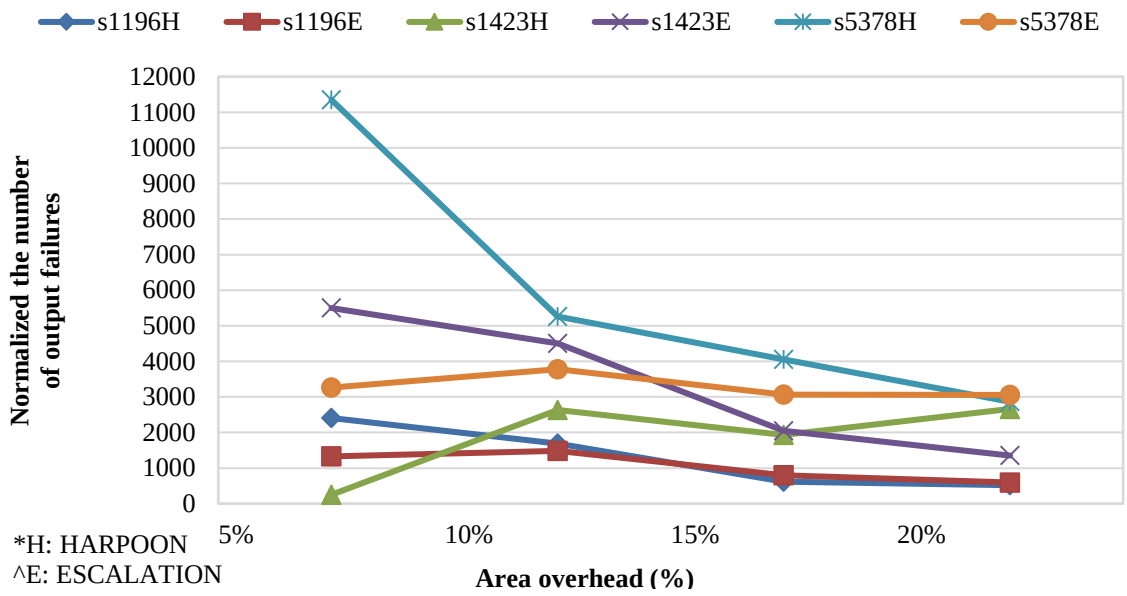


Fig. 4.5 Output failures versus area overhead for both HARPOON [9] and ESCALATION

In Table 4.2, we compared the average HD results reached by three different algorithms: random masking [42], our proposed algorithm, and the fault-based-analysis (FBA) algorithm [12]. As seen in the table, for 7 over 9 circuits, the results are better for our algorithm than with random masking. They gain on average 12% and 21%, respectively. The proposed algorithm cannot compete with the FBA algorithm because it only aims to reach high balanced HD. The ESCALATION algorithm does not have as much time and memory complexity as the FBA, a greedy algorithm. In the following section, we discuss how one can improve the HD results in the ESCALATION algorithm.

Table 4.2 Comparing Hamming Distance results of random masking, ESCALATION, and FBA algorithms

Circuit	# key-gate	Random	ESCALATION	FBA
C432	17	26%	37%	50%
C499	40	3%	22%	50%
C880	28	16%	17%	50%
C1355	42	13%	25%	50%
C1908	28	9%	25%	50%
C3540	22	15%	13%	50%
C5315	97	10%	20%	45%
C6288	27	24%	8%	50%
C7552	89	0.08	0.18	48%
Average	--	0.12	0.2 1	49%

4.5 Layout Level Validation

In Section 4.5, we used the same assumptions that the authors used at the gate level [13], to fairly compare with this work. Two assumptions in this work are 1) 60% cell delay variation, 2) no correlation among the delay variation of the cells in a path. In fact, there are components in the with-in-die variation that are physically-dependent and correlated [51]. The lack of layout-level information at gate level and RTL forces designers to use a simple delay and variation models as the authors have done [13].

In order to achieve more accurate results, we performed experiments at the layout level, post-placement and routing. The experiments include Trojan insertion in MSP and HPD calculation. The experiments consist of placing and routing masked circuits with SOC-Encounter [79]. The shortest path for each net and the MSP of each circuit are then found using TCL scripts. Afterward, an AND gate, as a Trojan, is inserted in MSP. Note that AND gate is the smallest functional Trojan that one can use. The MSP delays before and after Trojan inserting are obtained. The TDP of MSP is calculated according to the

formula (6).

In TDP, we consider 12% of path delay variation due to with-in-die variation according to [76]. In this work, the authors fabricated ICs with ROs with different lengths. The ROs were inserted in different locations of the layout design. The ICs were fabricated using different layout design styles, in the 45 nm technology. The reports in [76] show that there are around 36% and 12% of path delay variations due to die-to-die and with-in-die variation, respectively. Thanks to calibration methods, like [57], we can remove the effects of die-to-die variations from the PDA. In addition, defenders can use any structure for measuring path delay such as the ones given in [58, 57].

In Table 4.3, the MSP value for the original circuits and their masked circuits are presented in Columns 3 and 6. The masked circuits are obtained by the proposed algorithm. In addition, the TDP of MSP is reported in Columns 4 and 7. The table shows that 4 circuits do not need the MSP reduction, as the Trojan in their MSP is detectable almost 100%. The proposed algorithm improves the TDP by 23%. The TDP is averagely 10% in the masked circuits. This improvement needs to accept averagely 6% and 23% performance and area overhead, respectively. However, these overheads may look very much, but in reality, we do not need to mask the whole circuit. In the next section, we give examples and further explanations.

Regarding the power overhead of masking methods, it is noteworthy that masked circuits work in their functional mode by the correct key; thus, the added key-gates are totally transparent in this mode. They do not add any transitions in functional mode. Hence, they only have leakage power and do not add any dynamic power. The power overhead is not reported in this work because the static power of a few key-gates in 45 nm technology is negligible.

Table 4.3 TDP of MSP in the original and masked circuits; performance and area overhead in layout level

Circuit	Original circuit			Masked circuit			Perf overhead	# KG	Area overhead
	Perf	MSP	TDP	Perf	MSP	TDP			
C432	1.47 ns	0.45 ns	93%	1.47 ns	0.31 ns	100%	0%	17	16%
C449	0.79 ns	0.39 ns	100%	-	-	-	-	-	-
C880	1.06 ns	0.32 ns	100%	-	-	-	-	-	-
C1355	1.02 ns	0.40 ns	99%	-	-	-	-	-	-
C1908	1.08 ns	0.32 ns	100%	-	-	-	-	-	-
C3540	1.51 ns	0.47 ns	90%	1.66 ns	0.47 ns	90%	10%	22	13%
C5315	1.11 ns	0.45 ns	93%	1.25 ns	0.38 ns	99%	13%	97	19%
C6288	4.32 ns	1.23 ns	8%	4.55 ns	0.81 ns	30%	5%	27	3%
C7552	2.15 ns	0.77 ns	34%	2.19 ns	0.66 ns	51%	2%	89	21%
Average			64%			74%	6%		14%

4.6 Discussions

In this section, we discuss some noteworthy points about the ESCALATION approach and how it can have better results.

1. ESCALATION is a DfTr approach and hinders Trojan insertion in two ways. First, it masks a circuit, thus, Trojan attackers cannot have good knowledge about the original functionality of the circuit. Second, using the ESCALATION approach, if all nets of a circuit belong to at least one short-enough path, the circuit is more sensitive to Trojan delay effects. In such situations, in order to hide these effects, a Trojan attacker can increase the drive strength and capacity load of the cells that precede and proceed the Trojan. Cells having more drive strength and capacity load consume more power. As a result, increasing the strength and capacity load increases the success probability of PCA Trojan detection methods. Thus, PDA- and PCA-based Trojan detection methods must be combined.

2. The ESCALATION approach is based on logic masking, and it only modifies the

combinational part of circuits at the gate level. As explained in Section II, logic masking can be used as one step of masking an RTL netlist. It is used to change the state transition graph of an RTL netlist [80]. If we change the combinational part of a sequential circuit, then wrong keys create wrong values in the FFs and also POs. Output failures in a masked sequential circuit are the result of wrong keys and (consequently) wrong values in FFs. Thus, key detection in sequential circuits is more difficult than in simple combinational ones.

3. In order to have better TDP results in the ESCALATION approach, an ESCALATION-based algorithm can be implemented in design-abstraction levels lower than the gate level, such as after performing synthesis, placement, or routing. In the lower levels, there is more information about the delay components of nets and cells; thus, path delay calculation and finding the shortest path for the nets is more accurate (and certainly more complex) than at the gate level. If designers implement any logic masking method post-placement and routing, they must incrementally perform a logic optimization after inserting all the key-gates. This optimization is done in order to solve the key-gates in the combinational part of circuits. For example, if one inverter gate in the original circuit is preceded by an XOR key-gate, a logic optimizer algorithm might convert it to an XNOR key-gate. As a result, the masked circuit has an XNOR key-gate, for which the correct key is '0', although the correct key value of an XNOR key-gate would seem to be '1' at first glance.

4. In order to improve the HD results in the ESCALATION approach, both HD achievement and MSP reduction can be considered simultaneously. As mentioned in Section III, the objective of the proposed ESCALATION algorithm is only to reduce the MSP. In each iteration of the proposed algorithm, there might be more than one suitable net for our objective. Among these nets, other objectives can be investigated. For instance,

we have seen many times that there are a few nets for which key-gate insertion can make a shorter path for MSP. A net is then randomly selected, but one can investigate which one has a better effect on HD. This is sure to increase the time and memory complexity of the algorithm.

5. The area overheads reported in the previous section seem high; however, it must be taken into account that there is no need to mask a whole big circuit. Trojan and IP-piracy threats are important in the security-critical parts of circuits. A Trojan inserted in unmasked parts will not have critical effects. In systems-on-chips, there are many IPs that can be found freely everywhere; and so no one cares about them being stolen. A large circuit can therefore easily be partitioned into sub-circuits and only the security-critical parts masked. When this is the case, the area overhead is much less than the reports in this work. For instance, Rajendran et al. masked some small parts of a microprocessor (e.g. thread switch, DMA controller, FP unit, etc.) [12]. As shown in Table 3, in the worst case, we have 21% area overhead; definitely, this is a lot. But if the critical security part of a circuit, which should be masked, occupies just 10% of the circuit area, the area overhead is only 2.1%.

4.7 Conclusions

In this chapter, we presented a new DfTr approach, called ESCALATION, which leverages logic masking in order to enhance Trojan detection based on PDA. Its objective is to reduce the MSP value of the circuit. MSP value reduction is of major interest for Trojan detection: it increases the TDP of the most vulnerable net. Based on the ESCALATION approach, we proposed an algorithm that identifies the most vulnerable

net in the circuit and then inserts a key-gate before or after this net. According to the delay of the target net to the PIs or POs, an XOR or MUX key-gate is used by the algorithm.

Simple formulas for calculating both TDP and FPR have been proposed and proven. Using the formulas, TDP has been calculated considering a 60% cell delay variation at the gate level. Furthermore, in the layout level, TDP has been calculated considering 12% path delay variation. The layout level experiments and results show that the ESCALATION algorithm is capable of improving the TDP of the MSP by 35%.

In addition, the logic masking quality of the ESCALATION algorithm was investigated according to two metrics: the number of failed outputs and the HD of the output bits. We compared the ESCALATION algorithm to the HARPOON algorithm [9]. Experiments show that ESCALATION can reach a good level of logic masking quality, as good as HARPOON's, by accepting a bit more area overhead. Moreover, the HD of masked circuits using the ESCALATION algorithm was calculated. The results are much better than those attained by random masking [42]. However, they are not as good as the FBA results [12]. In addition, we have also discussed how to improve the HD of masked circuits obtained by the ESCALATION approach.

Chapter 5

Restricting Switching Activity Using Logic Masking to Improve Power Analysis-Based Trojan Detection

5.1 Introduction

Hardware Trojans lead up to increase power consumption in Trojan-infected circuits. However, this increase does not yield to desired results when one uses Trojan detection methods based on power consumption analysis (PCA). In fact, such methods have a scalability problem. In other words, detecting small hardware Trojans, i.e. including few gates, in thousands-gate circuits by PCA is infeasible. Thus, such circuits should be considered as a collection of small sub-circuits, and PCA must be individually performed for each one of them.

As observed in Chapter 3 and 4, one can employ logic masking as a DfTr approach in order to facilitate Trojan detection based on logic testing and path delay analysis. In this chapter, we introduce a DfTr approach improving PCA-based Trojan detection methods. We called this approach "RESTORATION: REstricting Switching acTivity using lOgic masking to improve poweR Analysis-based Trojan DetectION". Regarding this approach, we propose a new logic masking method and an algorithm to employ it. Apart from the

general benefits of logic masking, the main objective of the proposed logic masking method is to sort various sub-circuits of the circuit under Trojan test (CUTT), and consequently overcome the scalability problem of detection methods based on PCA. In addition, we propose a complementary solution to the mentioned problem. It is a circuit-partitioning aware input vector generation (IVG) method. These proposals will be discussed later in this chapter.

5.2 Contributions

The contributions of this chapter are as follows:

- We detail the RESTORATION approach and the new proposed logic masking method and algorithm. The method is explained and illustrated with examples.
- We insert hardware Trojan in the ISCAS-85 benchmark circuits and their counterparts masked by the proposed algorithm.
- We report how much the proposed algorithm besides the proposed IVG method beside the proposed IVG method can improve the proportion of Trojan switching activity to total circuit activity, the shortly so-called Trojan to circuit activity (TCA).
- We synthesize the benchmark circuits and masked ones and report how much the proposed algorithm besides the proposed IVG method can improve the proportion of Trojan power consumption to the total circuit power consumption, the shortly so-called Trojan to circuit power (TCP).

5.3 RESTORATION Based Algorithm

ICs' power consumption can be used as a signal in side-channel-based[1] Trojan detection. However, this signal varies because of process and environmental variations. Indeed, PCA results in detecting Trojans if it shows the power consumption of a Trojan-infected circuit is more than a specific maximum value that process and environmental variations can raise. The challenge is that in new modern technologies small hardware Trojans including few gates in thousands-gate circuits consume negligible power in comparison with overall circuit power consumption [4-5, 18-19, 22]. On the other hand, although the power consumption of a hardware Trojan will likely rise by increasing the switching activity of the Trojan-infected circuit, consequently, this action increases the total power consumption of the CUTT. Hence, its power variations are increased; because in circuitry situations that a circuit uses more power (e.g. because of more internal cells activities), the power variation is more than that of the circuit uses less power, proven in [62]. To tackle this problem, one must try to use DfTr methods concerning PCA-based Trojan detection.

As mentioned in Chapter 2, an approach to facilitate PCA-based Trojan detection methods is switching activity localization (SAL). It includes increasing switching activity in one small targeted sub-circuit of the CUTT and simultaneously decreasing that in all other sub-circuits. Therefore, it is beneficial in two ways. First, if there is a hardware Trojan in the targeted sub-circuit, its activity and, consequently, power consumption is increased. Second, the total power consumption of the CUTT and, consequently, its variation is decreased. Note that one must perform SAL for all the sub-circuits constituted the CUTT. Hence, instead of the CUTT all of its sub-circuits are individually placed under

[1] In Chapter 2, four steps of side-channel-based Trojan detection was described.

PCA-based Trojan surveillance.

Regarding the mention benefits of SAL, it can be an aim for the RESTORATION approach. In other words, a logic masking algorithm intending SAL is a DfTr method based on the RESTORATION approach. In this section, such an algorithm is proposed and explained. The modifications of this algorithm are done at the gate level and in the combinational part of sequential circuits, like other logic masking algorithms proposed in this dissertation. It uses a new logic masking method in which multiplexer (MUX) cells with a special insertion method and configuration are employed as key-gate. The proposed algorithm includes two main steps. First, it sorts a circuit under masking into sub-circuits, equal to the desired number of key-gates. Second, it inserts a MUX key-gate into each sub-circuit such that it has the most effects on SAL.

Before starting the description about the proposed algorithm, examples are presented below to explain the proposed partitioning aware IVG and logic masking method.

5.3.1 Partitioning Aware Input Vector Generation Method

Examples 1: Figure 5.1 shows circuit c17 selected from the ISCAS-85 benchmark circuits. The dotted lines in this figure sort two sub-circuits SC1 and SC2 obtained by a partitioning algorithm. The goal of such algorithms is to sort the graph of a circuit into disjoint partition[2] such that they have equal (or nearly equal) size and minimum interconnects in between. The circuit cells make the graph nodes and nets among them are the edges. In order to have SAL in SC1, one must apply random input vectors to the primary inputs entering to SC1 (Input1, Input2, and Input3) and simultaneously the all-

[2] In this chapter the terms *sub-circuit* and *partition* have been used interchangeably.

bits-zero vector to the primary inputs of SC2 (Input1and Input5). The same process is needed for SC2.

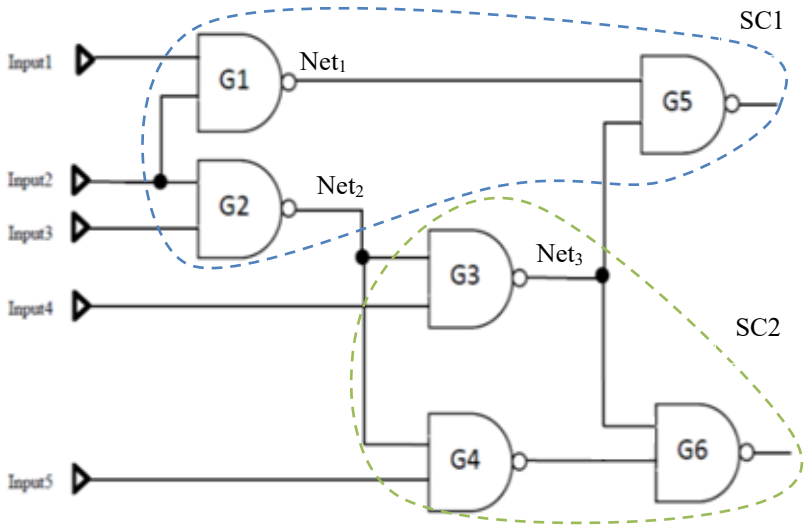


Fig 5.1 Circuit c17 partitioned to the sub-circuits SC1 and SC2.

Table 1 presents information about SAL in each sub-circuit in Fig. 5.1. Column 2 and 3 present the number of transitions happened in the target sub-circuit, and the other one, respectively. Column 4 presents the total number of transitions in circuit c17. These transitions happened when 1000 random input vectors are applied to the primary inputs of a sub-circuit and the primary inputs of another one get the zero vector. As observed in this table, transitions from a targeted sub-circuit pervade into another one. This is due to two nets (Net_2 and Net_3) interconnecting SC1 and SC2.

Table 5.1 The number and percentage of switching activity for each sub-circuit in Fig. 5.1 obtained by the circuit-partitioning aware IVG method

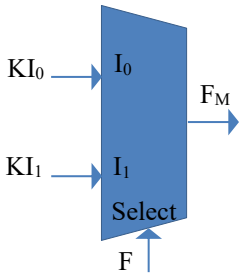
Sub-circuit	No. of transitions in the targeted sub-circuit	No. of transitions in non-targeted sub-circuits	Total No. of transactions	Percentage of localization
SC1	2801	1281	4082	69 %
SC2	2553	679	3232	79 %

5.3.2 A New Logic Masking Method

Regarding the problem of switching activity propagation from a target sub-circuit to other parts of a CUTT, a design hint is to insert key-gates in each sub-circuit such that filtrate possibly-propagated transitions from a targeted sub-circuit to other ones. For this propose, we proposed a new logic masking method. It uses 2-to-1 MUX cells as a key-gate. The required configuration[3] for MUX key-gate providing the mentioned benefit is shown in Fig. 5.2. As observed in this figure, a net of the circuit under masking (F) for inserting a key-gate on is connected to the select input of the MUX. The data inputs (I_0 and I_1) of the MUX operate like the key-input of an XOR kay-gate. In other words, the use of MUX kay-gate with this connection method has two key-inputs, unlike XOR kay-gate which has only one. As observed in the truth table in Fig. 5.2, this MUX functions like XOR key-gate with correct and incorrect key-input when its data inputs have “01” and “10” vectors, respectively. In addition, it locks its output to ‘0’and ‘1’ if one applies “00” and “11” vectors to “KI_0 KI_1”.

[3] Note this configuration is different with the one discussed in Chapter 2

Fig 5.2 2-to-1 multiplexer cell as a key-gate with the special configuration to filtrate switching activity propagation



KI_0	KI_1	F	F_M
0	0	X	0
0	1	X	X
1	0	X	X`
1	1	X	1

Example 2: Figure 5.3 shows circuit c17 accompanying with two MUX key-gates. They are inserted on Net_2 and Net_3 that connect sub-circuits SC1 and SC2. We call such nets pseudo input or output (PSI or PSO). In other words, a PSI (PSO) of a partition is a net coming from (going to) a cell in other partitions. Inserting a MUX key-gate on a PSI gives full controllability to the net. Hence, it is useful when a partition is the subject of SAL and especially if the partition does not have any primary input. Likewise, inserting a MUX key-gate on a PSO allows us to lock it to '1' or '0'. It does not let the switching activity of a targeted partition pervade to other partitions which must have low switching activity.

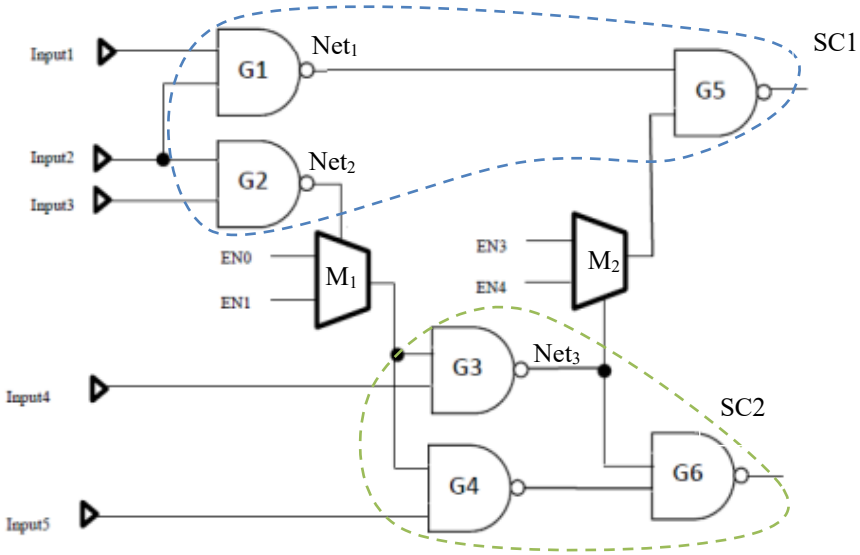


Fig 5.3 The masked version of circuit c17 using two MUX key-gates inserted on two nets interconnecting sub-circuits SC1 and SC2 in order filtrate the switching propagation form a partition to another one

Table 5.2 has the same fields as Table 1. It presents the results of applying 1000 random vectors to primary inputs and key-inputs of each sub-circuit shown in Fig. 5.3. As observed in the fourth column of Table 5.2, one can reach 100% SAL in both the sub-circuits using these MUX key-gates and the partitioning aware IVG method.

Table 5.2 The number and percentage of switching activity for each sub-circuit in Fig. 5.3 obtained by the IVG based on the partitioning information

Sub-circuit	No. of transitions in the targeted sub-circuit	No. of transitions in non-targeted sub-circuits	Total No. of transactions	Percentage of localization
SC1	3822	0	3822	100 %
SC2	3232	0	3232	100 %

Example 3: Figure 5.4 shows the same circuit as Fig. 5.1; however, infected by a hardware Trojan including two NAND gates, the red (striped) cells. The results of applying 1000 random input vectors to all the circuit primary inputs are gathered together in Table 5.3. Column 1 and 2 in this table reports the number of transitions in the hardware Trojan and the total transition number in the circuit. As observed, 21% TCA is obtained, shown in the third column. One can easily increase this ratio to 32 % using the circuit-partitioning aware IVG method, discussed in the previous example. Table 5.3 present the result of this method.

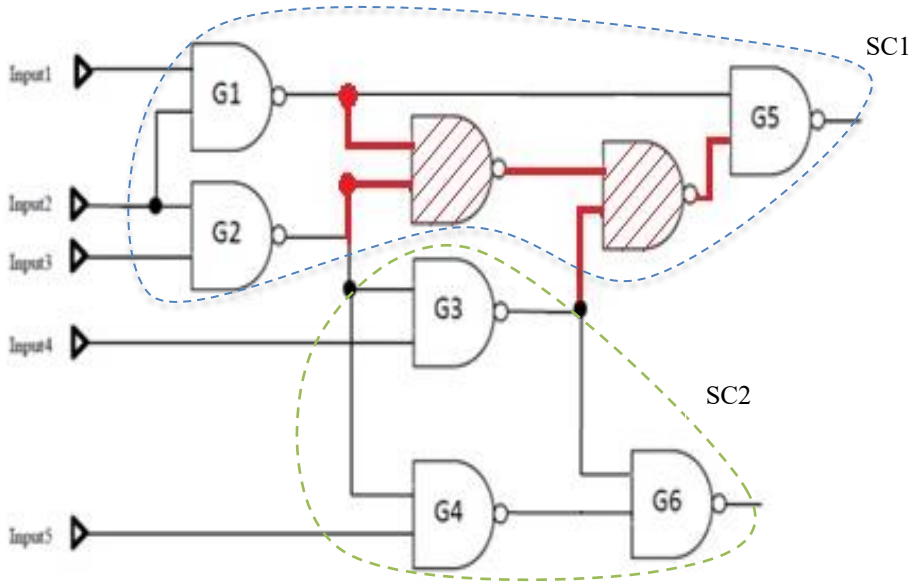


Fig 5.4 Circuit c17 infected by a hardware Trojan including two NAND gates and partitioned to the sub-circuits SC1 and SC2

Table 5.3 TCA and the number of switching activity in the circuit of Fig. 5.4 and its hardware Trojan obtained by applying random input vectors to all the primary inputs and key-inputs

No. of the Trojan transitions	Total No. of the circuit transactions	TCA
1578	7445	0.21

Table 5.4 TCA and the number and percentage of switching activity for each sub-circuit in Fig. 5.4 obtained by the circuit-partitioning aware IVG method

Sub-circuit	No. of the Trojan transitions	Total No. of the circuit transactions	TCA
SC1	1656	5230	0.32
SC2	0	2685	0

Example 4: Figure 5.5 shows the same circuit as Fig. 5.3; however, infected by a hardware Trojan including two NAND gates, the red (striped) cells. The use of the proposed logic masking and the circuit-partitioning aware IVG method in this circuit increases TCA by 52%. It is shown in Table 5.5. The comparison of this table and Table 5.4 shows that the use of the MUX key-gates improves TCA 20 %.

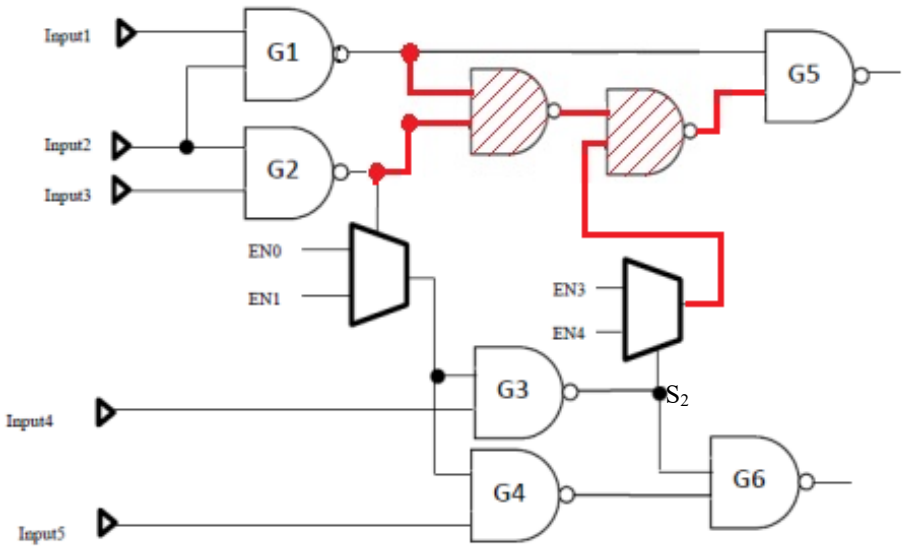


Fig. 5.5 Circuit c17 masked using two MUX key-gates and infected by a hardware Trojan including two NAND gates.

Table 5.5 TCA and the number and percentage of switching activity for each sub-circuit in Fig. 5.4 obtained by the input vector generation based on the partitioning information

Sub-circuit	No. of the Trojan transitions	Total No. of the circuit transactions	TCA
SC1	1985	3822	0.52
SC2	0	2626	0

5.3.3 Proposed Algorithm

In the following section, we explain the proposed algorithm. Its pseudocode is shown in Algorithm 5.1. The algorithm takes a gate-level netlist and a number (KEYLENGTH) as the key lengths, in line 1. Then, it partitions the netlist to some sub-circuits using the Kernighan–Lin (KL) partitioning algorithm, in line 2. The number of partitions (N) is calculated based on Eq. 5.1:

$$N = 2^{\text{floor}\,(\log_2 \text{KEYLENGTH})} \tag{5.1}$$

The reason for choosing this equation for N is that the implemented KL function in the proposed algorithm is only able to partition a circuit to numbers powers-of-2 (i.e. 2,4, 8,16,32 etc.). The time complexity of this function is O (n^2log n), where n is the number of cells in the circuit under partitioning.

Some parts of the data structure used for the partitions' implementation are shown in Algorithm 5.1. One part of them is a structure in which information about each partition is stored. This information includes the number of primary input (PI), the number of primary output (PO), and the sum of the switching activity (SSA) of a partition. The later field shows the total number of transitions when a partition is targeted to increase its switching activity.

The next step is to generate sets of input vectors by the proposed method (line 3). Each set is suited to one sub-circuit and corresponds to increase switching activity into a sub-circuit and decrease that of other ones. Input vectors in a set suited to a sub-circuit have two parts. One part has a random value in each vector and feeds the primary inputs of the sub-circuit. Another part has a fixed value in all vectors of the set and decreases switching activity sub-circuits not targeted by the set. These vectors are simulated in line4. The simulator was detained in Chapter2. During the simulation, SSA is calculated for each partition and later in the algorithm is used to decide a MUX kay-gates must be inserted in a partition or not. This point will be explained letter in this section.

Algorithm 5.1. The proposed RESTORATION based algorithm

```
1  Inputs: netlist; KEYLENGTH; Output:  maskedNetlist;
2  KL-Partitioning (netlist, KEYLENGTH, Return: partitions)
3  Input-Vector-Generation (partitions, RETURN: inputVectors);
4  Simulation (partitions, inputVectors);
5  KeyCounter = 0;
6  While ()
7     Sort-Partitions-by-PI;
8     For each partition in partitions do
9        MUX-Insertion-on-Pseudo-Primary-Input (partition);
10       Update-Input-Vector;
11       Update-Partition-Info;
12       keyCounter ++ ;
13       If keyCounter == KEYLENGTH do
14          return maskedNetlist;
15    End For
16    Sort-Partitions-by-SSA;
17    For each partition in partitions do
18       MUX-Insertion-on-Pseudo-Primary-Output (partition);
19       Update-Input-Vector;
20       Update-Partition-Info;
21       keyCounter ++ ;
22       If keyCounter == KEYLENGTH do
23          return maskedNetlist;
24    End For
25 End while
```

```
Structure Partition : partitions[]
*cell          cells[];
*net           nets[];
*PartitionInfo partitionInfo;
End Structure
```

```
Structure PartitionInfo
*Partition  partition
*int        PI;  // number of primary input
*int        PO;  // number of primary input
*int        SSA; // Sum-ofSwitching-Activity
End Structure
```

Afterward, the algorithm enters and remains into its main loop until all key-gates are inserted. The desired number of key-gates equals 5% of the number of cell in the circuit under masking. In this loop, first, the partitions are ascendingly sorted based on their PI,

in line 7. Thus, less PI makes more priority to get a MUX key-gate. Second, a MUX key-gate is inserted on a PSI of a partition, in line 9. In order to insert a MUX key-gate on a PSI of a partition, the function in line 9 checks all options. It inserts a MUX key-gate on a PSI, simulates some input vectors on the primary inputs of the partition (including the two inputs of the inserted MUX), and finally store the sum of all the transitions (SSA) happened in the partition during the simulation. Then it undoes the inserted MUX and inserts it on another PSI. This process is repeated until all of the PSI are checked. Then, the function selects the one that has the greatest SSA.

After a MUX insertion, the algorithm updates the generated input vectors and partitions' information, respectively in line 10 and 11. When a MUX key-gate is inserted, a two-bit-vector corresponding to the MUX key-inputs must be contacted to all the generated input vectors. The value of this vector is randomly selected between "00" or "11" for each input vector of the sets that the MUX output enters their suited sub-circuits. Contrariwise, the value of this vector is fixed to "00" for all the input vectors of the set that one of its primary output feeds the select input of the MUX. The update of the partition information includes increasing one unit to the PI field of the partition and reload the new value of SSA obtained during the simulation.

When inserting MUX key-gate on a PSI is done for all the partitions, the algorithm descendingly sorts them based on their SSA. Partitions that have higher SSA probably propagates more switching activity while targeting to increase switching activity. Hence, inserting a MUX key-gates on PSOs of such partitions is useful. Among all PSOs, the algorithm selects the one with the greatest logic fan-out cone. Because in such nets, there is more chance for propagating switching activity.

After inserting a MUX kay-gate, the algorithm updates the generated input vectors

and partitions' information, respectively in lines 19 and 20. When for each partition one MUX key-gate is inserted on their PSO, the second '*for'* structure is finished. Then, if the number of inserted key-gates is less than *KEYLENGHT*, the *while* loop starts from the beginning again and continues until all key-gates are inserted. This case may happen because KEYLENGHT is always greater than or equal to the number of madden partition (note Eq. 5.1)

5.4 Experiments and Results

5.4.1 Experiments Flow

The flow of our experiments is as follows:

1. We implemented the proposed masking algorithm and circuit-partitioning aware IVG method in the coding environment of LOG-STAT, our developed CAD tool presented in Chapter 3.
2. We masked the ISCAS-85 benchmark circuits by the implemented algorithm. The number of used key-gate for each circuit less than %5 of the number of cells in the circuits. For each masked circuit, sets of input test vectors are generated.
3. We made two copies from the netlist of each masking circuit. Into the first and second copy, we inserted hardware Trojan T1 and T4, respectively. These hardware Trojans are combinational comparators, as seen in Fig. 5.6. We use the lowest active signals of the circuit to feed the inputs (trigger) of the hardware Trojans.
4. In the logic simulator of LOG-STAT, we applied the generated test vectors to each Trojan-infected copy of the masked circuits and calculated TCA for each one.

5. We synthesized the Trojan-free and Trojan-infected masked circuits obtained in Step 2 and 4. Design Compiler [78] and NANGATE 45 nm [74] were used in this step.
6. We imported the synthesized circuits and test vector sets to Power-Compiler [81] in order to calculate the circuits' power consumption.

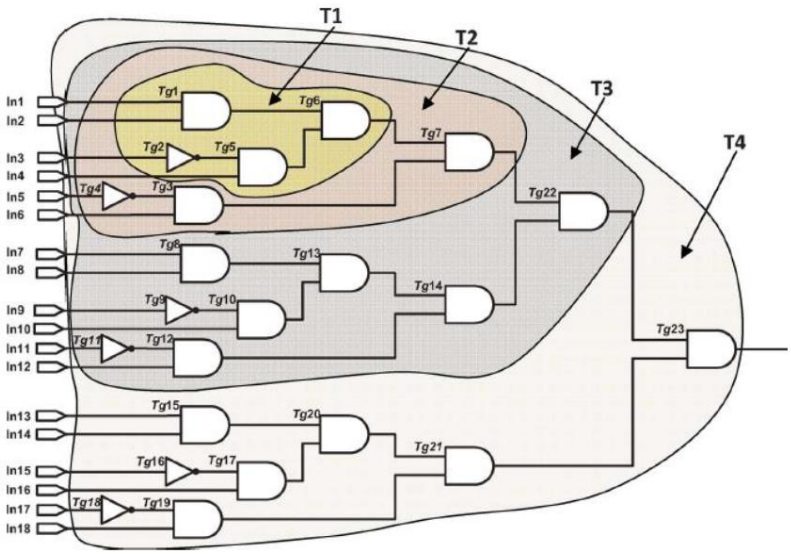


Fig. 5.6 Trojan benchmarks proposed in [14]

5.4.2 Results

In this section, we present the results for the circuits c432 and c882 with lots of details. Hereafter, we use a standard to name the presenting circuits:

"X-Y-Z-circuit name"

where X shows the circuit is original (O) or masked (M); Y shows the circuit is Trojan-free (TF) or T1-infected (T1) or T4-infected (T4); Z shows the circuit is synthesized (S) or not (NS); and finally, the circuit name, e.g. c432.

5.4.2.1 Circuit c432

Table 5.6 shows the number of transitions in circuit O-TF-NS-c432 using: (1) the random and (2) the proposed circuit-partitioning aware IVG method. The last row in this table shows the percent of transition decrease by the latter method using a different number of the partitions. As observed, the transitions happened in this circuit is decreased by 96 % using the proposed IVG method.

Table 5.6 The number of transitions in circuit O-TF-NS-c432 using the random and partitioning aware IVG methods

	Random IVG	**Partition based IVG**		
		4 Partitions	**8 Partitions**	**16 Partitions**
No. of transitions	104096	39543	39543	39543
% of transition decrease by partitioning aware IVG		62	94	96

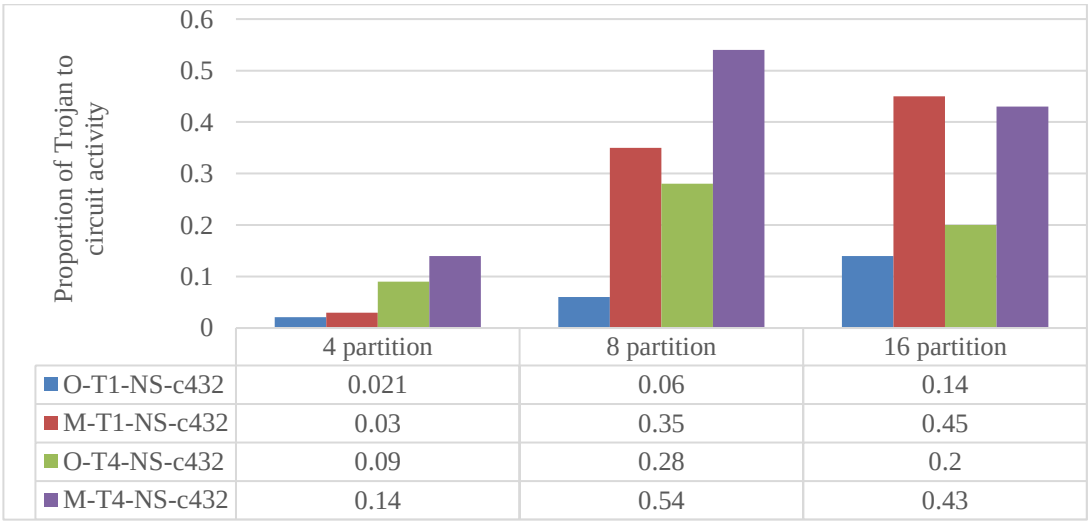
Table 5.7 shows the power consumption in circuit O-TF-S-c432 using the random and partitioning aware IVG methods. The last row in this table shows the percent of power decrease by the latter method using a different number of the partitions. As observed, the power consumed in this circuit is decreased by 83 % using the proposed IVG method.

Table 5.7 The power consumption in circuit O-TF-S-c432 using the random and partitioning aware IVG methods

	Random IVG	**Partition based IVG**		
		4 Partitions	**8 Partitions**	**16 Partitions**
Power consumption (uW)	9.59	5.54	2.32	1.63
% of power decrease by partitioning based IVG		42	75	83

Figure 5.7 shows TCA for four circuits O-T1-NS-c432, M-T1-NS-c432, O-T4-NS-c432, and M-T4-NS-c432. The TCA results in this figure are obtained using the proposed partitioning aware IVG method. Four, eight, and sixteen partitions are provided and their results are shown in the second, third, and fourth column, respectively. As observed in this figure, the use of the proposed logic masking algorithm improves the TCA results for all the case. Another point in this figure is that in the T4-infected circuits the TCA obtained by making eight partitions is better than that of sixteen partitions; the reason is that the partitions in the latter case are very small. During the experiments, we have seen that there is a threshold for partitioning and making more than that cause losing the benefit. This point has been observed for all the circuits during the experiments.

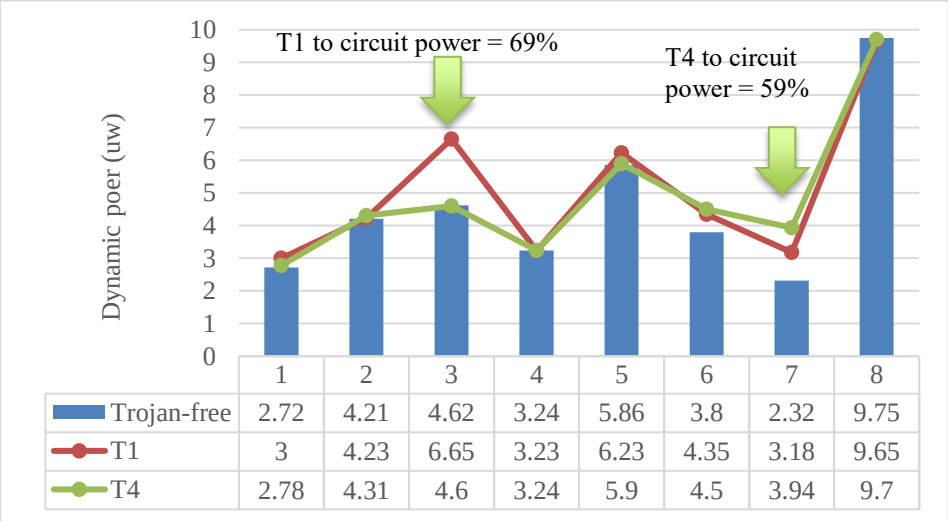
As the best case of SAL for circuit c432 happens when eight partitions are used, the power consumption of three synthesized circuits M-TF-S-c432, M-T1-S-c432, and M-T4-S-c432 is measured eight times such that in each of these measurements one partition is targeted using the partitioning aware IVG method. The results are shown in Fig. 5.7.



	4 partition	8 partition	16 partition
O-T1-NS-c432	0.021	0.06	0.14
M-T1-NS-c432	0.03	0.35	0.45
O-T4-NS-c432	0.09	0.28	0.2
M-T4-NS-c432	0.14	0.54	0.43

Fig. 5.7 Obtained TCA in four circuits O-T1-NS-c432, M-T1-NS-c432, O-T4-NS-c432, and O-T4-NS-c432 using the partition-based IVG methods

Two points are highlighted in Fig 5.8. In these points, TCP is maximum, equal to 69 % and 59 %, respectively for M-T1-S-c432 and M-T4-S-c432. The maximum TCP captured during our experiments for O-T1-S-c432 and O-T4-S-c432 using the random IVG method have been 4% and 6 %, respectively. As observed, the use of partitioning aware IVG method in masked circuits shows up the presence (and even the location) of small hardware Trojans.



	1	2	3	4	5	6	7	8
Trojan-free	2.72	4.21	4.62	3.24	5.86	3.8	2.32	9.75
T1	3	4.23	6.65	3.23	6.23	4.35	3.18	9.65
T4	2.78	4.31	4.6	3.24	5.9	4.5	3.94	9.7

Fig. 5.8 PCA in circuits M-TF-S-c432, M-T1-S-c432, and M-T4-S-c432 using the partitioning aware IVG methods and individually targeting eight made partitions

5.4.2.2 Circuit c880

Table 5.8 shows the number of transitions in circuit O-TF-NS-c880 using: (1) the random and (2) the proposed partitioning aware IVG method. The last row in this table shows the percent of transition decrease by the latter method using a different number of the partition. As observed, the transitions happened in this circuit is decreased by 94 % using the partitioning aware IVG method.

Table 5.8 The number of transitions in circuit O-TF-NS-c880 using the random and partitioning based IVG methods

	Random IVG	**Partition based IVG**			
		No. of made partitions			
		4	**8**	**16**	**32**
No. of transitions	150236	56327	29367	17843	7641
% of transition decrease by partitioning based IVG		62	80	88	94

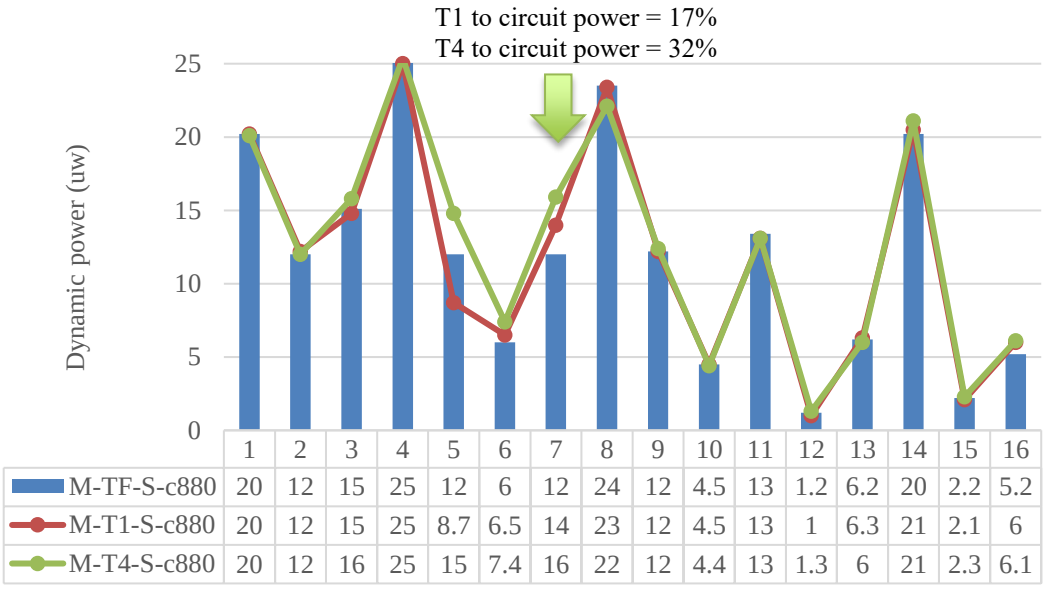
Table 5.9 compares the power consumption in the circuits O-X-S-c880 (note: X shows the circuit is Trojan-free or T1-infected or T4-infected) using the random IVG method, and M-X-S-c880 using the partitioning aware IVG method. The latter three circuits are masked using the proposed logic masking algorithm in which the circuits are partitioned to sixteen ones. As observed, the power consumption in the masked circuit while using the partition aware IVG method is decreased by 80-90 %.

Table 5.9 The power consumption of the circuits synthesized from c880 "original or masked" \ "Trojan-free or Trojan-infected"

Original or masked / **Trojan-free or infected**	**O-X-S-c880**	**M-X-S-c880 (Masked using 16 partitions)**
Trojan-free (TF)	80.8 uW	12
T1-infected (T1)	81.2 uW	8.7
T4-infected (T4)	83 uW	15.9

During the experiments, it was seen that the best case of TCP for circuit c880 happens when sixteen partitions are used. Therefore, we provide the power consumption of three synthesized circuits M-TF-S-c880, M-T1-S-c880, and M-T4-S-c880 in Fig. 5.9. One point is highlighted in this figure. In this point, TCP is maximum, equal to 17% and 32

%, respectively for M-T1-S-c880 and M-T4-S-c880. The maximum TCP captured during our experiments for O-T1-S-c880 and O-T4-S-c880 using the random IVG method have been 0.5 % and 2 %, respectively. They are calculable by the information of Table 5.9. As observed, the use of the proposed masking algorithm and partitioning aware IVG method increases TCP from 0.05% and 2% to 17% and 32%, respectively for T1-infected and T4-infected circuits.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
M-TF-S-c880	20	12	15	25	12	6	12	24	12	4.5	13	1.2	6.2	20	2.2	5.2
M-T1-S-c880	20	12	15	25	8.7	6.5	14	23	12	4.5	13	1	6.3	21	2.1	6
M-T4-S-c880	20	12	16	25	15	7.4	16	22	12	4.4	13	1.3	6	21	2.3	6.1

Fig. 5.9 PCA in circuits M-TF-S-c880, M-T1-S-c880, and M-T4-S-c880 using the partitioning-based IVG methods and individually targeting sixteen made partitions

5.4.2.3 Other Circuits

Figure 5.10 shows the proportion of the power consumption in T1-infected and T4-infected circuits to the original ISCAS-85 benchmark circuits (except circuit c17). These circuits were synthesized and their power consumptions were measured by applying input vectors obtained by the circuit-partitioning aware IVG method. The number of used

partitions is mentioned besides the name of each circuit in this figure. As observed, for the smaller hardware Trojan (T1) minimum 20 % TCP is obtained (for circuit c2670). For the bigger hardware Trojan, the results are better. In this case, a minimum 30% TCP is obtained (for circuit c5315).

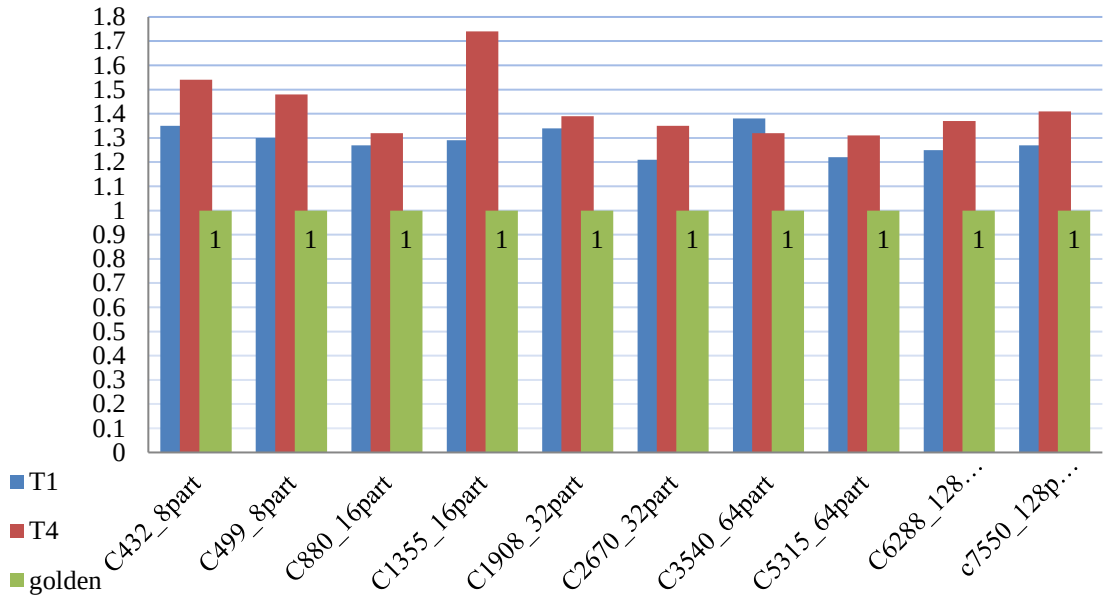


Fig. 5.10 power difference between golden and HT infected designs

5.5 Conclusions

In this chapter, in order to deal with the scalability problem of PCA-based Trojan detection methods, we introduced a new DfTr approach that leverages logic masking to facilitate such methods. We proposed a new logic masking method that instead of traditional XOR/XNOR key-gates uses MUX cells with a special insertion method and configuration. This method provides all the benefits of XOR/XNOR key-gates, including the increase of switching activity (controllability) and obscuring design functionality; moreover, one can configure MUXs inserted by this method to lock their outputs to ‘0’ or

‘1’. This option is useful to localize switching activity.

We proposed an algorithm that employs the proposed logic masking method. We implemented it in the coding environment of LOG-STAT. Then, we ran it on the ISCAS-85 benchmark circuits. The results showed that the proposed algorithm could significantly localize switching activity. In addition, we inserted two hardware Trojans in the benchmark circuits and their counterparts masked by the proposed algorithm. Then, we measured the proportion of Trojan to circuit activity. Moreover, we synthesized these circuits and measured their power consumption. For this power measurement, we needed input vectors. We generated them by two methods: (1) random IVG and (2) circuit-partitioning aware IVG. The results of these vectors showed that Trojan to circuit power in normal circuits using random vectors is negligible; however, the use of the proposed logic making algorithm and the partitioning aware vector generation improves TCP minimum 20% and maximum 70%.

Chapter 6

Conclusion, Ongoing, and Future Work

6.1 Conclusion

During the last two decades, most of the semiconductor companies have been established based on the fabless business model, which includes independent third parties such as design houses, IP developers, and fabrication foundries (fabs), etc. in different countries. This model gains economic advantages, though it entails serious security challenges caused by untrustworthy third parties. For instance, untrustworthy fabs can pirate IPs by separating them from the layout and illegally reselling in the black market. Another security threat that untrustworthy fabs are suspected of doing is to insert hardware Trojans into the layout. Hardware Trojans are malicious circuitry alterations performed on a circuit for sabotage purposes.

In this work, we focused on detection and prevention methods against hardware Trojan and IP piracy threat that may happen in untrustworthy fabs. We detailed these two threats and some detection/prevention methods against them in Chapter 2. Moreover, in Chapter 2, we briefly introduced the different periods of SoC life cycle

and security threats in each period.

Concerning the focused security threats, the main idea in this work was to leverage logic masking as an IP-piracy prevention method to hinder Trojan insertion or facilitate Trojan detection. Accordingly, we proposed three approaches and algorithms in Chapters 3, 4, and 5.

First, in Chapter 3, we proposed the AMELIORATION approach that is the leverage of logic masking to facilitate logic testing-based Trojan detection. Regarding this approach, we proposed a security metric and employed it in a logic masking algorithm. The metric simultaneously concerns "highly balanced Hamming distance achievement" and "rare signal elimination". The results in Chapter 3 showed that the proposed algorithm employing the security metric effectively hinders Trojan insertion by removing rare signals. It can also significantly approach the ideal output balanced Hamming distance (50%), more or less similar to the FAB algorithm, proposed in [12].

Second, in Chapter 4, we proposed the ESCALATION approach that is the leverage of logic masking to facilitate PCA-based Trojan detection in Chapter 4. Regarding this approach, we proposed a logic masking algorithm that employs XOR/XNOR gates and multiplexer cells (MUX) as key-gate. The objective of the proposed algorithm is to generate fake short paths for vulnerable nets, the ones that only belong to long paths because shorter paths have smaller delay variations [13]. The layout level experiments in this chapter showed that the proposed algorithm can improve 35% the probability of detecting a hardware Trojan including only one cell in the worst case of process variations.

Third, in Chapter 5, we proposed the RESTORATION approach that is the leverage of logic masking in order to facilitate PCA-based Trojan detection methods. Regarding

this approach, we proposed a logic masking method and algorithm. The proposed logic masking method instead of traditional XOR/XNOR key-gates utilizes MUX with a special insertion technique. In this methods, MUXs can function like XOR/XNOR key-gates or block their outputs to ‘0’ or ‘1’ according to their configuration. The proposed algorithm employing this method aims to increase switching activity localization in masked circuits. The results in this chapter showed that the use of the proposed logic making algorithm and input vector generation method can improve Trojan to circuit power ratio minimum 20% and maximum 70%.

In addition, in this work, we analyzed logic masking effects on the number of states and transitions in the FSM of sequential circuits. The results showed that even a few key-gates inserted in the combinational part of sequential circuits easily double the numbers of states and transitions.

Moreover, we developed and presented a CAD tool in this work. We used it during different experiments. It includes a logic simulator and functions to calculate the area, performance, and power of both combinational and sequential circuits at the gate level. It also has a function to calculate Hamming distance in masked circuits.

The main conclusion perceived from the experimental results of this work is that logic masking can be employed as an efficient DfTr, i.e. it has great potential to hinder Trojan insertion or facilitate Trojan detection methods, apart from the primary benefit of logic masking, which is to obscure design functionality.

6.2 Ongoing and Future Work

Some of logic masking capabilities for employing as a DfTr method were observed in this work; however, the assumptions, experiments, and evaluations were simple, and sometimes not based on precise models. In order to achieve more accurate results and deeper insights concerning the application of logic masking as a DfTr method, some ideas and experiments are under implementation or can be done in future; and we briefly explain some of them in the following.

6.2.1 A New AMELIORATION-Based Algorithm

As mentioned in Chapter 3, the proposed algorithm based on the AMELIORATION approach is a greedy algorithm. The time complexity of this algorithm is in order N^3, where N is the number of nets in the circuit under masking. This time complexity is a disadvantage for the proposed algorithm in the case the circuit under masking includes a few hundred thousand gates. To deal with this problem, we need heuristic algorithms. For instance, one can partition a big circuit to sub-circuits and then for each sub-circuit run the proposed algorithm.

6.2.2 Two New ESCALATION-Based Algorithms

In Chapter 4, we proposed an algorithm based on the ESCALATION approach. It finds vulnerable nets and then inserts key-gates on their fan-in or fan-out cone to make short fake paths for them. In other words, the algorithm does not consider the Hamming distance effect of inserting key-gates. Consequently, as observed in the result section of

Chapter 4, the algorithm cannot compete on Hamming distance achievement with the AMELIORATION-based or FAB algorithm [12]; although its results were better than that of random masking. This shortcoming of the first ESCALATION-based algorithm is the motivation for one of our future work. Accordingly, we propose the second ESCALATION-based algorithm that concerns two objectives while inserting a key-gate: (1) "highly balanced Hamming distance achievement" and (2) “fake short paths creation” for vulnerable nets. The algorithm finds vulnerable nets, identical to the first ESCALATION-based algorithm. Next, it consecutively targets one of the found vulnerable nets. For each targeted net the algorithm makes a list including the nets logically situated a few level-of-gates ahead/behind in the fan-in/fan-out cone of the targeted net. Then, the algorithm inserts a key-gate on a net of the list and analyzes (1) the Hamming distance effect of the inserted key-gate and (2) the new fake short paths created by the key-gate. Afterward, the algorithm undoes the inserted key-gate and selects another net from the list. When all the nets of the list are analyzed, the algorithm selects the best one according to its objectives.

Another weakness of the first ESCALATION-based algorithm is that it uses the unit delay model to calculate the delay of paths in the circuit under masking. However, the layout level experiments including circuits masked by this algorithm showed that the algorithm improves Trojan detection probability in PCA-based Trojan detection methods; as matter of fact, the unit delay model is not precise. It is due to that a placement[1] algorithm may place the cells of a path far from each other and, consequently, long nets are routed in between these cells. Long nets have more delay. As a result, a path including few cells is not considered a long path in the proposed algorithm, although after placement and routing it can be long enough to hide delay

[1] One of the physical design steps in the ASIC design flow.

effects of a hardware Trojan. This is the motivation for another of our future work. We proposed the third ESCALATION-based algorithm that employs a more accurate delay model. This algorithm takes a placed (or placed and rout) netlist. In the first step, it calculates the paths' delay, and regarding that decides which nets are vulnerable and need a key-gate. Then, in the second step, the algorithm consecutively targets one of the found vulnerable nets. For each targeted net the algorithm physically places a key-gate near to a cell connected to the targeted net. It also logically inserts the place key-gate on the fan-in or fan-out cone of the targeted net. It is noteworthy that we can develop the third algorithm such that it aims both the objectives of the second algorithm.

The mentioned algorithms above seems very promising and probably results better than the first ESCALATION-based algorithm. These algorithms are under implementation.

6.2.3 A New RESTORATION-Based Algorithm

Concerning the first RESTORATION-based algorithm proposed in Chapter 5, one of our future work is to implement another logic masking algorithm that aims to localize switching activity in a placed circuit. It uses information about the placement and switching activity of the circuit cells.

In short, the algorithm that we proposed in Chapter 5 and our new proposed algorithm operate like each other in all respects except one point. Both the algorithms use 2-to-1 multiplexers (MUX) as key-gate, the MUX insertion technique, and net-selection criteria for inserting MUXs. The only difference between these two algorithms is that the former one considers the circuit under masking as a collection of sub-circuits

that a partitioning algorithm depicts their boundaries and interconnections; the latter proposed algorithm considers the circuit under masking as a collection of sub-circuits that have (almost) equal dimensions and are allocated to different physical regions of the circuit.

It is noteworthy that regards the proposed new algorithm masks placed circuits, it includes one more step than the algorithm in Chapter 5. Indeed, once a MUX key-gate is logically inserted on a PSI[2] (PSO) of a sub-circuit, then the algorithm should physically place this MUX in one empty space in the region of the sub-circuit. Among different options in this step, we can easily choose an empty space randomly; or with more attention, we can choose an empty space that can be a suitable option for Trojan attackers.

The last idea that we discuss in this work is to concern the effect of MUX key-gates on the Hamming distance of the circuits masked by the RESTORATION-based algorithm in Chapter 5. For this purpose, we need to define a new security metric and employ it in the algorithm, like what we did in Chapter 3. This security metric must deduce that inserting MUX key-gates on which nets will simultaneously satisfy both the objectives: (1) highly balanced Hamming distance achievement and (2) high percent of switching activity localization. A solution for formulating this security metric is to insert a MUX key-gate on a PSI (PSO) of a sub-circuit and calculate two parameters for the PSI (PSO): (1) the Hamming distance and (2) switching activity ratio in the modified circuit; then we must undo the inserted MUX and select another PSI (PSO); and repeat this insertion/calculation until all the PSI (PSO) of the sub-circuits are investigated. Afterward, the best PSI (PSO) according to the importance of the adjectives can be

[2] A pseudo input (PSI) of a sub-circuit is a net that comes from other sub-circuits and enters to the sub-circuit. Similarly, a pseudo output (PSO) of a sub-circuit is a net that exits from the sub-circuit and enters to other sub-circuits.

selected, similar to the Eq. 3.1.

At the end of this dissertation, we must remark that some of the proposed algorithms and ideas that we proposed in this section are not complete and well-matured; and they need more discussions and thinking but, nonetheless, they simultaneously employ the physical and logical information of placed (and routed) circuits. Therefore, probably they result better than the situation in where no physical information exists.

Chapter 7

Bibliography

[1] Guin, U., DiMase, D. and Tehranipoor, M., 2014. *Counterfeit integrated circuits: detection, avoidance, and the challenges ahead.* Journal of Electronic Testing, 30(1), pp.9-23.

[2] Guin, U., Shi, Q., Forte, D. and Tehranipoor, M.M., 2016. *FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs.* ACM Transactions on Design Automation of Electronic Systems (TODAES), 21(4), p.63.0

[3] Jin, Y., 2015. *Introduction to hardware security.* Electronics, 4(4), pp.763-784.

[4] Tehranipoor, M. and Koushanfar, F., 2010. *A survey of hardware trojan taxonomy and detection.* IEEE design & test of computers, 27(1).

[5] Li, H., Liu, Q. and Zhang, J., 2016. *A survey of hardware Trojan threat and defense.* Integration, the VLSI journal, 55, pp.426-437.

[6] Rajendran, J., Sinanoglu, O. and Karri, R., 2014. *Regaining trust in VLSI design: Design-for-trust techniques.* Proceedings of the IEEE, 102(8), pp.1266-1282.

[7] Chakraborty, R.S. and Bhunia, S., 2010, January. *RTL hardware IP protection using key-based control and data flow obfuscation.* In VLSI Design, 2010. VLSID'10. 23rd International Conference on (pp. 405-410). IEEE.

[8] Dofe, J. and Yu, Q., 2018. *Novel dynamic state-deflection method for gate-level design obfuscation.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 37(2), pp.273-285.

[9] Chakraborty, R.S. and Bhunia, S., 2009. *HARPOON: an obfuscation-based SoC design methodology for hardware protection.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28(10), pp.1493-1502.

[10] Dupuis, S., Ba, P.S., Di Natale, G., Flottes, M.L. and Rouzeyre, B., 2014, July. *A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans.* In On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International (pp. 49-54). IEEE.

[11] Colombier, B., Bossuet, L. and Hély, D., 2016. *From secured logic to IP protection.* Microprocessors and Microsystems, 47, pp.44-54.

[12] Rajendran, J., Zhang, H., Zhang, C., Rose, G.S., Pino, Y., Sinanoglu, O. and Karri, R., 2015. *Fault analysis-based logic encryption.* IEEE Transactions on computers, 64(2), pp.410-424.

[13] Shekarian, S.M.H. and Zamani, M.S., 2015. *Improving hardware Trojan detection by retiming.* Microprocessors and Microsystems, 39(3), pp.145-156.

[14] Salmani, H. and Tehranipoor, M., 2012. *Layout-aware switching activity localization to enhance hardware Trojan detection.* IEEE Transactions on Information Forensics and Security, 7(1), pp.76-87.

[15] Villasenor, J., 2013. *Compromised by design?: Securing the defense electronics supply chain.* Center for Technology Innovation at Brookings.

[16] Johnson, B., Freeman, D., Christensen, D., Wang, S.T., *Market Trends: Rising Costs of Production Limit Availability of Leading-Edge Fabs.* GARTNER, INC. http://www.gartner.com/DisplayDocument?doc_cd=238123. Accessed 1 Sept 2012.

[17] Rostami, M., Koushanfar, F. and Karri, R., 2014. *A primer on hardware security: Models, methods, and metrics.* Proceedings of the IEEE, 102(8), pp.1283-1295.

[18] Bhunia, S., Hsiao, M.S., Banga, M. and Narasimhan, S., 2014. *Hardware Trojan attacks: threat analysis and countermeasures.* Proceedings of the IEEE, 102(8), pp.1229-1247.

[19] Vosatka, J., 2018. *Introduction to Hardware Trojans.* In The Hardware Trojan War (pp. 15-51). Springer, Cham.

[20] Mishra, P., Tehranipoor, M. and Bhunia, S., 2017. *Security and trust vulnerabilities in third-party IPs.* In Hardware IP Security and Trust (pp. 3-14). Springer, Cham.

[21] Potkonjak, M., 2010, June. *Synthesis of trustable ICs using untrusted CAD tools.* In Design Automation Conference (DAC), 2010 47th ACM/IEEE (pp. 633-634). IEEE.

[22] Xiao, K., Forte, D., Jin, Y., Karri, R., Bhunia, S. and Tehranipoor, M., 2016. *Hardware Trojans: Lessons learned after one decade of research.* ACM Transactions on Design Automation of Electronic Systems (TODAES), 22(1), p.6.

[23] Rahman, M.T., Forte, D., Shi, Q., Contreras, G.K. and Tehranipoor, M., 2014, October. *CSST: Preventing distribution of unlicensed and rejected ICs by untrusted foundry and assembly.* In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on (pp. 46-51). IEEE.

[24] Ray, S., Bhunia, S. and Mishra, P., 2017. *Security validation in modern SoC designs.* In Fundamentals of IP and SoC Security (pp. 9-27). Springer, Cham.

[25] Tehranipoor, M., Salmani, H. and Zhang, X., 2014. *Counterfeit ICs: Taxonomies, assessment, and challenges.* In Integrated Circuit Authentication (pp. 161-178). Springer, Cham.

[26] Quadir, S.E., Chen, J., Forte, D., Asadizanjani, N., Shahbazmohamadi, S., Wang, L., Chandy, J. and Tehranipoor, M., 2016. *A survey on chip to system reverse engineering.* ACM journal on emerging technologies in computing systems (JETC), 13(1), p.6.

[27] Jyothi, V. and Rajendran, J.J., 2018. *Hardware Trojan Attacks in FPGA and Protection Approaches.* In The Hardware Trojan War (pp. 345-368). Springer, Cham.

[28] Dutta, R.G., Guo, X. and Jin, Y., 2017. *IP trust: the problem and design/validation-based solution.* In Fundamentals of IP and SoC Security (pp. 49-65). Springer, Cham.

[29] *Top 5 Most Counterfeited Parts Represent a $ 169 Billion Potential Challenge for Global Semiconductor Market.* http://press.ihs.com/press-release/design-supply-chain/top-5-mostcounterfeited-parts-represent-169-billion-potential-cha

[30] Tehranipoor, M. and Wang, C. eds., 2011. *Introduction to hardware security and trust.* Springer Science & Business Media.

[31] Guin, U., Huang, K., DiMase, D., Carulli, J.M., Tehranipoor, M. and Makris, Y., 2014. *Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain.* Proceedings of the IEEE, 102(8), pp.1207-1228.

[32] Forte, D., Bhunia, S. and Tehranipoor, M.M. eds., 2017. *Hardware protection through obfuscation.* Springer International Publishing.

[33] Koushanfar, F., 2012. *Hardware metering: A survey.* In Introduction to Hardware Security and Trust (pp. 103-122). Springer, New York, NY.

[34] Oliveira, A.L., 1999, June. *Robust techniques for watermarking sequential circuit designs.* In Proceedings of the 36th annual ACM/IEEE design automation conference (pp. 837-842). ACM.

[35] Meyer-Bäse, U., Castillo, E., Botella, G., Parrilla, L. and Garcia, A., 2011, June. *Intellectual property protection (IPP) using obfuscation in C, VHDL, and verilog coding.* In Independent Component Analyses, Wavelets, Neural Networks, Biosystems, and Nanoengineering IX (Vol. 8058, p. 80581F). International Society for Optics and Photonics.

[36] Rajendran, J., Sam, M., Sinanoglu, O. and Karri, R., 2013, November. *Security analysis of integrated circuit camouflaging.* In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 709-720). ACM.

[37] Plaza, S.M. and Markov, I.L., 2015. *Solving the third-shift problem in IC piracy with test-aware logic locking.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34(6), pp.961-971.

[38] Nejat, A., Hely, D. and Beroulle, V., 2016, July. *Reusing logic masking to facilitate path-delay-based Hardware Trojan detection.* In On-Line Testing and Robust System Design (IOLTS), 2016 IEEE 22nd International Symposium on (pp. 191-192). IEEE.

[39] Samimi, S.M.S., Aerabi, E., Nejat, A., Fazeli, M., Hely, D. and Beroulle, V., 2016, October. *High output hamming-distance achievement by a greedy logic masking approach.* In 2016 IEEE East-West Design & Test Symposium (EWDTS) (pp. 1-4). IEEE.

[40] Nejat, A., Hely, D. and Beroulle, V., 2018. *ESCALATION: Leveraging Logic Masking to Facilitate Path-Delay-Based Hardware Trojan Detection Methods.* Journal of Hardware and Systems Security, pp.1-14.

[41] Chakraborty, R.S. and Bhunia, S., 2017. *State Space Obfuscation and Its Application in Hardware Intellectual Property Protection.* In Hardware Protection through Obfuscation (pp. 189-220). Springer, Cham.

[42] Roy, J.A., Koushanfar, F. and Markov, I.L., 2008, March. *EPIC: Ending piracy of integrated circuits.* In Proceedings of the conference on Design, automation and test in Europe (pp. 1069-1074). ACM.

[43] Rajendran, J., Pino, Y., Sinanoglu, O. and Karri, R., 2012, June. *Security analysis of logic obfuscation.* In Proceedings of the 49th Annual Design Automation Conference (pp. 83-89). ACM.

[44] Nejat, A., Hely, D. and Beroulle, V., 2015, December. *Facilitating side channel analysis by obfuscation for Hardware Trojan detection.* In Design & Test Symposium (IDT), 2015 10th International (pp. 129-134). IEEE.

[45] Xie, Y. and Srivastava, A., 2016, August. *Mitigating sat attack on logic locking.* In International Conference on Cryptographic Hardware and Embedded Systems (pp. 127-146). Springer, Berlin, Heidelberg.

[46] Karri, R., Rajendran, J., Rosenfeld, K. and Tehranipoor, M., 2010. *Trustworthy hardware: Identifying and classifying hardware trojans. Computer*, *43*(10), pp.39-46.

[47] Banga, M. and Hsiao, M.S., 2018. *Hardware IP Trust.* In The Hardware Trojan War (pp. 75-100). Springer, Cham.

[48] Bao, C., Xie, Y., Liu, Y. and Srivastava, A., 2018. *Reverse Engineering-Based Hardware Trojan Detection.* The Hardware Trojan War, pp.269-288.

[49] Govindan, V. and Chakraborty, R.S., 2018. *Logic Testing for Hardware Trojan Detection.* In The Hardware Trojan War (pp. 149-182). Springer, Cham.

[50] Narasimhan, S., Yueh, W., Wang, X., Mukhopadhyay, S. and Bhunia, S., 2012. *Improving IC security against Trojan attacks through integration of security monitors.* IEEE Design & Test of Computers, 29(5), pp.37-46.

[51] Blaauw, D., Chopra, K., Srivastava, A. and Scheffer, L., 2008. *Statistical timing analysis: From basic principles to state of the art.* IEEE transactions on computer-aided design of integrated circuits and systems, 27(4), pp.589-607.

[52] Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P. and Sunar, B., 2006, *Trojan Detection using IC Fingerprinting*, IBM Research Report.

[53] Davoodi, A., 2018. Golden-Free Trojan Detection. In *The Hardware Trojan War* (pp. 203-215). Springer, Cham.

[54] Kelly, S., Zhang, X., Tehranipoor, M. and Ferraiuolo, A., 2015. *Detecting hardware trojans using on-chip sensors in an asic design.* Journal of electronic testing, 31(1), pp.11-26.

[55] N. Yoshimizu, *Hardware trojan detection by symmetry breaking in path delays,* in IEEE International Symposium on Hardware-Oriented Security and Trust (2014), pp. 107–111

[56] Jin, Y. and Makris, Y., 2008, June. *Hardware Trojan detection using path delay fingerprint.* In Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on(pp. 51-57). IEEE.

[57] Cha, B. and Gupta, S.K., 2013, March. *Trojan detection via delay measurements: A new approach to select paths and vectors to maximize effectiveness and minimize cost.* In Proceedings of the conference on design, automation and test in Europe (pp. 1265-1270). EDA Consortium.

[58] N ejat, A., Shekarian, S.M.H. and Zamani, M.S., 2014. *A study on the efficiency of hardware Trojan detection based on path-delay fingerprinting.* Microprocessors and Microsystems, 38(3), pp.246-252

[59] Rajendran, J., Jyothi, V., Sinanoglu, O. and Karri, R., 2011, May. *Design and analysis of ring oscillator based Design-for-Trust technique.* In VLSI Test Symposium (VTS), 2011 IEEE 29th (pp. 105-110). IEEE.

[60] Saqib, F., Ismari, D., Lamech, C. and Plusquellic, J., 2015. *Within-die delay variation measurement and power transient analysis using REBEL.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(4), pp.776-780.

[61] Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P. and Sunar, B., 2007, May. *Trojan detection using IC fingerprinting.* In Security and Privacy, 2007. SP'07. IEEE Symposium on(pp. 296-310). IEEE.

[62] Aarestad, J., Acharyya, D., Rad, R. and Plusquellic, J., 2010. *Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad* IDD_Q. IEEE Transactions on information forensics and security, 5(4), pp.893-904.

[63] Huang, Y., Bhunia, S. and Mishra, P., 2016, October. *MERS: statistical test generation for side-channel analysis based Trojan detection.* In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 130-141). ACM.

[64] Ba, P.S., Dupuis, S., Palanichamy, M., Di Natale, G. and Rouzeyre, B., 2016, July. *Hardware Trust through Layout Filling: a Hardware Trojan Prevention Technique.* In VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on(pp. 254-259). IEEE.

[65] Yang, P.L. and Marek-Sadowska, M., 2016, November. *Making split-fabrication more secure.* In Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on(pp. 1-8). IEEE.

[66] Salmani, H., Tehranipoor, M. and Plusquellic, J., 2012. *A novel technique for improving hardware trojan detection and reducing trojan activation time.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 20(1), pp.112-125.

[67] Subramanyan, P., Ray, S. and Malik, S., 2015, May. *Evaluating the security of logic encryption algorithms.* In Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on (pp. 137-143). IEEE.

[68] Samimi, S.M.S., Nejat, A., Aerabi, E., Beroulle, V., Hely, D. and Fazeli, M., 2018, *LOG-STAT: Logic simulation, Security, and Trust Analysis Tool* [Online] Available: https://raw.githubusercontent.com/arashnejat/Logic-simulationSecurity-and-Trust-AnalysisTool/master/Logic%20simulation%2C%20Security%2C%20and%20Trust%20Analysis%20Tool.zip

[69] Ercolani, S., Favalli, M., Damiani, M., Olivo, P. and Ricco, B., 1989, April. *Estimate of signal probability in combinational logic networks.* In European Test Conference, 1989., Proceedings of the 1st (pp. 132-138). IEEE.

[70] Saleh, R.A., Jou, S.J. and Newton, A.R., 2013. *Mixed-mode simulation and analog multilevel simulation* (Vol. 279). Springer Science & Business Media.

[71] M. Hansen, H. Yalcin and J. Hayes, *Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering,* IEEE Design & Test of Computers, vol. 16, no. 3, pp. 72-80, 1999.

[72] Brglez, F., Bryan, D. and Kozminski, K., 1989, May. *Combinational profiles of sequential benchmark circuits.* In Circuits and Systems, 1989., IEEE International Symposium on (pp. 1929-1934). IEEE.

[73] Fazeli, M., Ahmadian, S.N., Miremadi, S.G., Asadi, H. and Tahoori, M.B., 2011, March. *Soft error rate estimation of digital circuits in the presence of multiple event transients (METs).* In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011* (pp. 1-6). IEEE.

[74] NanGate – The Standard Cell Library Optimization Company, [Online]. Available: http://www.nangate.com/

[75] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (2003). *Artificial intelligence: a modern approach* (Vol. 2, No. 9). Upper Saddle River: Prentice hall.

[76] Pang, L.T., Qian, K., Spanos, C.J. and Nikolic, B., 2009. *Measurement and analysis of variability in 45 nm strained-Si CMOS technology.* IEEE Journal of Solid-State Circuits, 44(8), pp.2233-2243.

[77] Verific Design Automation Inc., [Online]. Available: http://www.verific.com

[78] SynopsysDesign Compiler, [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx

[79] Cadence SOC Encounter, [Online]. Available: https://www.cadence.com

[80] Chakraborty, R.S. and Bhunia, S., 2011. *Security against hardware Trojan attacks using key-based design obfuscation.* Journal of Electronic Testing, 27(6), pp.767-785.

[81] Synopsys Power Compiler, [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/power-compiler.html