INVITED PAPER

Regaining Trust in VLSI Design: Design-for-Trust Techniques

This paper surveys how concepts in VLSI test can be adopted in the context of hardware security and trust and design-for-trust techniques.

By Jeyavijayan Rajendran, *Student Member IEEE*, Ozgur Sinanoglu, *Member IEEE*, and Ramesh Karri, *Senior Member IEEE*

ABSTRACT | Designers use third-party intellectual property (IP) cores and outsource various steps in their integrated circuit (IC) design flow, including fabrication. As a result, security vulnerabilities have been emerging, forcing IC designers and end-users to reevaluate their trust in hardware. If an attacker gets hold of an unprotected design, attacks such as reverse engineering, insertion of malicious circuits, and IP piracy are possible. In this paper, we shed light on the vulnerabilities in very large scale integration (VLSI) design and fabrication flow, and survey design-for-trust (DfTr) techniques that aim at regaining trust in IC design. We elaborate on four DfTr techniques: logic encryption, split manufacturing, IC camouflaging, and Trojan activation. These techniques have been developed by reusing VLSI test principles.

KEYWORDS | Design automation; design for testability; security

I. INTRODUCTION

Typically, an integrated circuit (IC) comprises a wide variety of components, including from digital, analog, photonic to microfluidic [1]. Sensors, actuators, and biological interfaces are also being integrated into these ICs. On the one hand, IC designs have been enabled by advances in mixed system integration and the increase in the wafer sizes (currently about 300 mm and projected to be 450 mm by 2018 [1]), reducing the cost per IC. On the other hand, support for multiple capabilities and mixed technologies has increased the cost of owning an advanced foundry. For instance, the cost of owning a foundry will be $5 billion in 2015 [2]. Consequently, only high-end commercial foundries now manufacture high-performance, mixed system ICs, especially at the advanced technology nodes [3]. Absent the economies of scale, many of the design companies cannot afford owning and acquiring expensive foundries, and hence, outsource their fabrication process to these "one-stop-shop" foundries.[1]

While such globalization of IC design flow has successfully ameliorated the design complexity and fabrication cost problems, it has led to several security vulnerabilities [4]. If a design is fabricated in a foundry that is not controlled by the (fabless) design house, reverse engineering, malicious circuit insertion/modification, and intellectual property (IP) piracy are possible [3]. A rogue element in the foundry or a malicious user can reverse engineer the functionality of an IC/IP, and then steal and claim ownership of the IP [5]. An untrusted IC foundry may overbuild ICs and illegally sell these excess parts [6], [7]. Rogue elements in the foundry may insert malicious circuits (hardware Trojans) into the design unbeknownst to the designer [8], [9]. Finally, an attacker can reverse engineer an IC by depackaging and delayering it, imaging the individual layers, stitching the images, and extracting the netlist. This way, he/she can steal an IP or reveal competitor's trade secrets. Because of these attacks, the semiconductor industry loses several billions of dollars annually [10]. The underlying reason is that the designers

[1]Companies that do not own their foundry are termed "fabless" design houses.

Manuscript received October 25, 2013; revised May 30, 2014; accepted June 14, 2014. Date of publication July 15, 2014; date of current version July 18, 2014. This work was supported in part by the Semiconductor Research Corporation and Advanced Technology Investment Company (SRC-ATIC) under Grant 2013-HJ-2440-S4; the New York University/New York University Abu Dhabi (NYU/NYU-AD) Center for Research in Information Security Studies and Privacy (CRISSP); the National Science Foundation, Computing and Communication Foundations (NSF-CCF) under Grant 1319841; the NSF-CI-ADDO-NEW under Grant 1059328; and the U.S. Army Research Office (ARO) under Grant W911NF-13-1-0272.

J. Rajendran and **R. Karri** are with the Electrical and Computer Engineering Department, Polytechnic School of Engineering, New York University, Brooklyn, NY 11201 USA (e-mail: jv.ece@nyu.edu; rkarri@nyu.edu).

O. Sinanoglu is with the New York University-Abu Dhabi, Abu Dhabi 129188, United Arab Emirates (e-mail: os22@nyu.edu).

Digital Object Identifier: 10.1109/JPROC.2014.2332154

0018-9219 © 2014 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

have minimum control over their design in this distributed, and potentially, vulnerable design and fabrication flow.

While hardware security and trust is a relatively recent but an important problem, a somewhat similar yet fundamentally different problem of manufacturing defects has been extensively researched by very large scale integration (VLSI) test researchers for the last few decades. We leverage the knowledge gained from VLSI testing and apply it to hardware security. On the one hand, while manufacturing defects are natural and unintentional, the attacks outlined above are man-made, intentional, and meant to be hidden. On the other hand, many concepts in VLSI testing, such as justification and sensitization (see Section II), can be adapted in the context of hardware security. Inspired by the design-for-testability (DfT) solutions for better testability of manufacturing defects, one can develop design-for-trust (DfTr) solutions to detect and prevent these attacks.

In this paper, we shed light on the vulnerabilities in the VLSI design and fabrication flow and survey four DfTr techniques. These DfTr techniques are as follows.

1) *Logic encryption* implements a built-in locking mechanism on ICs to prevent reverse engineering and IP piracy by a malicious foundry and user, and hinder Trojan insertion by a malicious foundry.
2) *Split manufacturing* splits the layout and manufactures different metal layers in two separate foundries to prevent reverse engineering and piracy by a malicious foundry.
3) *IC camouflaging* modifies the layout of certain gates to deceive reverse engineers into obtaining an incorrect netlist, thereby, preventing reverse engineering by a malicious user.
4) *Trojan activation* inserts dummy scan flip-flops (dSFFs) to manipulate the transition probabilities and expose a Trojan that is hidden in low-activity regions.

When these techniques are applied arbitrarily with no consideration of the design structure, they fail to provide the required levels of security. Consequently, they are easy to break and provide confidential information to the attacker. We also show that by defining appropriate security metrics and incorporating techniques such as fault activation, sensitization, and masking, these DfTr techniques can be made stronger, providing the security levels needed to regain trust in hardware.

The paper is organized as follows. Section II describes the VLSI test principles. Section III elaborates on the four DfTr techniques, their security requirements, and the shortcomings of the naive approaches. Section IV explains how VLSI testing principles can help these DfTr techniques in meeting their security requirements. Section V provides a discussion on the DfTr techniques. Section VI concludes the paper.

II. BACKGROUND: TEST PRINCIPLES AND THEIR APPLICATIONS

In this section, we describe the VLSI test principles that we will use to develop DfTr techniques [14].

- Test principle 1 (fault excitation): A stuck-at-v fault at a site is excited when an input pattern justifies that site to $\overline{v}$.
- Test principle 2 (sensitization): A site is sensitized to an output if every side input of every gate on a path from the site to the output is justified to the noncontrolling value of the gate. Sensitization of an internal line l to an output O refers to the condition (values applied from the primary inputs to justify the side input of gates on the path from l to O to the noncontrollable values of the gates) which bijectively maps l to O, and thus, renders any change on l observable on O.
- Test principle 3 (fault propagation): The effect of a fault at a site propagates to an output if the input pattern excites the fault and sensitizes the faulty site to the output.
- Test principle 4 (fault masking): Multiple effects of the same excited fault or multiple excited faults mask each other when none of their effects manifest at the outputs, as the errors cancel out.
- Test principle 5 (controllability): It is a measure of difficulty of setting a wire to a desired value (logic 1 or logic 0).

III. BASELINE DfTr TECHNIQUES

In this section, we focus on four DfTr techniques: logic encryption, split manufacturing, IC camouflaging, and Trojan activation. For each of these techniques, we describe the threat model and different design approaches proposed in the literature, and introduce their security criteria and metrics. In addition, for each technique, we explain why naive approaches fail to meet the criteria, motivating VLSI-testing-inspired DfTr approaches.

A. DfTr1: Logic Encryption

Logic encryption hides the functionality and the implementation of a design by inserting additional gates, referred to as key gates, into the original design. In order for the encrypted design to exhibit its correct functionality (i.e., produce correct outputs), a valid key has to be supplied to the encrypted design (for example, loading the key to a tamper-proof on-chip memory [15]). Upon applying an incorrect key, the encrypted design will exhibit an incorrect functionality (i.e., produce incorrect outputs).

Logic encryption of hardware does not mean encrypting the design file by a cryptographic algorithm; instead, it means encrypting the hardware's functionality. Researchers have previously used the term "logic obfuscation" [6],

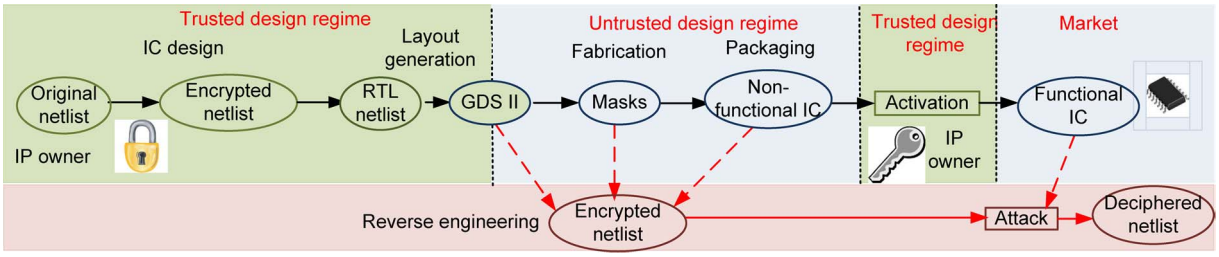

Fig. 1. ***Logic encryption in an IC design flow [7], [11]–[13]. The top part depicts an IC design flow enhanced with logic encryption capabilities to thwart IP piracy, IC overproduction, reverse engineering, and Trojan insertion [6], [7]. Before sending the design to an untrusted foundry, the designer encrypts the design using logic encryption techniques. The foundry then manufactures this encrypted design. On receiving the encrypted hardware, the IC designer activates it by applying the secret key and the IC is then sold in the market. The bottom part depicts the threat model for logic encryption. In the untrusted regime, an attacker can obtain the encrypted netlist from the IC design, or by reverse engineering the layout, mask, or a fabricated IC, and the functional IC from the market. Using this attack, the attacker can get a deciphered netlist and make pirated copies, overproduce ICs, or insert Trojans at "safe" places.***

[7] for this purpose. Obfuscation of a hardware hides only its functionality, but obfuscation does not prevent black-box usage (obtaining an output from a hardware on applying an input) [16]. Logic encryption prevents this black-box usage of the encrypted hardware in addition to hiding its implementation. Hence, we use the term "encryption," and not "obfuscation."

1) Threat Model: Logic encryption techniques can thwart an untrusted foundry from illegally copying, reverse engineering, and overproducing the IC design [6], [7]. Fig. 1 shows the threat model for logic encryption. The IP provider and the designer are trusted. The attacker is in the foundry. The designer encrypts the (security-critical) modules using the proposed technique, synthesizes them using trustworthy computer-aided design tools, and sends the generated layouts to the untrustworthy foundry. The foundry manufactures the encrypted IC and returns them to the designer. The designer then applies the secret key and makes the ICs functional. The designer or a trusted third-party performs functional validation and manufacturing testing on this functional IC. Once the ICs pass the tests, the functional ICs are packaged and sold. The secret keys for an IC are stored in a tamper-proof on-chip memory, and thus, are assumed to be unrecoverable by a malicious user.

We consider two types of attackers: a rogue element in the foundry and a malicious user. In the first type, the rogue element in the foundry has access to the layout files (in standard GDSII format [2]) given to him/her for manufacturing the IC. This attacker can pirate the IP and/or overproduce IC. In addition, this attacker can analyze the design to identify the structural behavior of the design, thereby finding safe places to insert Trojans. In the second type, a malicious user can buy the IC, depackage and delayer it, image the individual layers, stitch the images, and extract the gate-level netlist. This attacker can reverse engineer an IC and pirate the IP.

Logic encryption prevents these attacks by encrypting all or critical modules in a design. Since the design is encrypted by the designer, without the secret keys, the foundry cannot use any copies or overproduced ICs. Furthermore, it prevents an attacker from analyzing the structural behavior of the design, thereby hindering Trojan insertion. Additionally, if the target IC is sold in the market, an attacker can buy one of the functional ICs. In this way, an attacker can gain access to good input/output (I/O) pairs. Furthermore, the attacker can identify the inputs of key gates through structural analysis of the netlist.

2) Previous Work: Logic encryption can be broadly classified into two types: sequential and combinational. In sequential logic encryption, additional logic (black) states are introduced in the state transition graph [7], [11], [12]. The state transition graph is modified in such a way that the design reaches a valid state only on applying a correct sequence of key bits. If the key is withdrawn, the design, once again, ends up in a black state.

In combinational logic encryption, XOR/XNOR gates are introduced at random locations in the design to conceal its functionality [6]. One of the inputs of these inserted gates serves as the key input. One can configure these inserted gates as buffers or inverters using these key inputs. The values applied to these key inputs are the keys. To generate unique keys per IC, [17] calibrates the XOR/XNOR gates postfabrication by tweaking the threshold voltage of the gates and introducing process variation sensors into a circuit. To make the IC functional, these sensors have to be configured by applying the secret key. Instead of XOR and XNOR gates, one can also use multiplexers as key gates [18]. The select line of the multiplexers is connected to the key inputs.

Memory elements may also be inserted into the design [19]. The circuit will function correctly only when the memory elements are programmed correctly. One can

consider these programming bits as the secret keys, similar to that of logic encryption. However, the insertion of memory elements may incur significant power, area, and delay overheads.

3) Criteria: A logic encryption technique should satisfy two criteria: a) incorrect outputs should be produced on applying an incorrect key; and b) an attacker should not be able to retrieve the secret key.

Criterion 1 (50% Output Corruption): The objective of the defender (designer) is to prevent his/her IP from being copied by an attacker in the foundry and to prevent black-box usage. The attacker does not know the secret key used for encryption. Hence, he/she will apply a random key and in turn expect the module to become functional (i.e., to produce correct outputs). In the worst case, he/she has to apply all possible keys. If the key size is sufficiently large (say 128), then the number of possible key combinations will be large (2^{128}). Consequently, the brute force attack becomes computationally harder. Thus, the objective of the defender is to force the attacker to perform a brute force attack. To achieve this objective, the defender has to encrypt the module such that an attacker, with the knowledge of the publicly available logic encryption objectives and algorithms, is not able to obtain the correct outputs by applying an incorrect key. This can be done by minimizing the correlation between the corrupted and original outputs, thereby maximizing the ambiguity for the attacker.

The ambiguity for an attacker is maximum when 50% of the outputs are corrupted upon applying a random incorrect key (a mathematical derivation for this metric is provided in [18]). One can quantify this metric using the Hamming distance. The Hamming distance between the correct output and the output on applying a random incorrect key should be 50% on average. The Hamming distance is defined as the number of output bits that differ on applying a valid key versus an invalid key for the same input. Ideally, 50% of the output bits should differ on applying any invalid key for any input. In the case of logic encryption, the Hamming distance metric is formally defined as follows. For a circuit C which produces an output y for an input x on applying the valid key, logic encryption of C should satisfy

$$\frac{\sum_{x\in X}\sum_{k\in K;k\neq \text{valid key}} HD(y_{\text{valid key},x}, y_{k,x})}{2^{|x|}\times(2^{|k|}-1)\times|y|}\times 100\% = 50\% \quad (1)$$

where X is the set of all inputs, K is the set of all possible keys, and $y(k,x)$ is the output vector on applying the key k for the input x. $|x|$, $|y|$, and $|k|$ denote the size of the input, output, and key in bits, respectively.

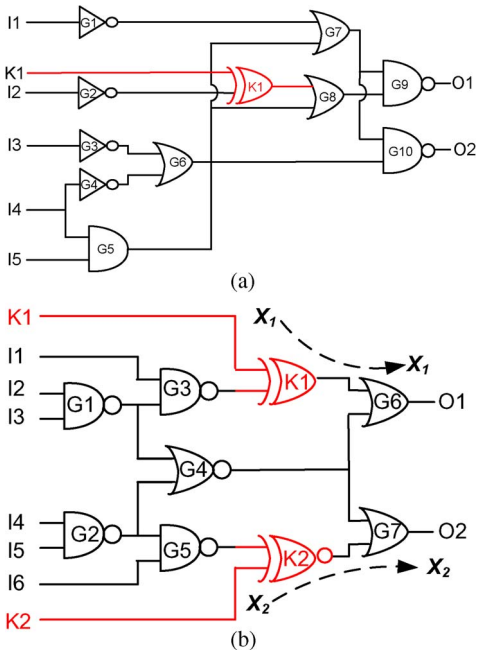

Fig. 2. ***Problems with naive logic encryption. (a) Circuit produces correct outputs for 20 out of the 32 input patterns even on applying an incorrect key to the key gate (K1). (b) An attacker can observe the key bits at the outputs by applying the input pattern 100000 [13], [20].***

Problem 1: Random insertion of key gates does not ensure an incorrect output is produced for an incorrect key, let alone 50% output corruption.

Example 1: Let us consider the combinational logic encryption technique proposed in [6] and [7]. In this technique, XOR/XNOR gates are inserted at random locations. For instance, consider the C17 circuit shown in Fig. 2(a) encrypted with one XOR gate $K1$. The design will produce the correct output on applying the correct key value $K1 = 0$. On applying an incorrect key ($K1 = 1$), one expects incorrect outputs. For example, on applying the input pattern 01000, an incorrect output 00 is produced instead of the correct output 10. Unfortunately, the design produces correct outputs for most of the input patterns even on applying an incorrect key. For example, the input pattern 11100 produces the correct output 11 even when an incorrect key is applied. In fact, this design produces an incorrect output only for 12 out of the possible 32 input patterns. In other words, the design produces correct outputs for 62.5% (20) of the input patterns despite applying the incorrect key. Thus, this encryption procedure is weak as it does not ensure that incorrect outputs are produced for incorrect keys, let alone the 50% output corruption criterion.

Criterion 2 (Difficult-to-Break Encryption): A logic encryption should ensure the secrecy of the key even when an attacker has access to good I/O pairs.

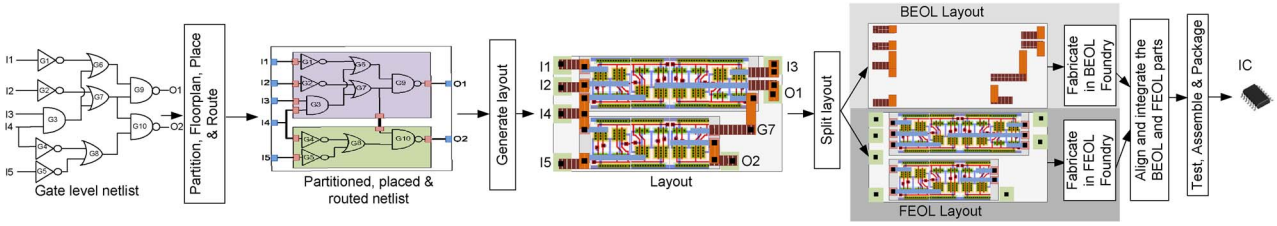

Fig. 3. ***Split-manufacturing-aware IC design flow. The layout obtained from the traditional design flow is split into two parts: the FEOL part containing the transistors and the lower metal layers, and the BEOL part containing the top metal layers. These parts are then fabricated in two different foundries. Either the designer or the BEOL foundry assembles the FEOL and BEOL wafers into the final IC [3], [21].***

Problem 2: Random insertion of key gates fails to ensure the secrecy of keys.

Example 2: Consider the encrypted circuit shown in Fig. 2(b). The key gates $K1$ and $K2$ are randomly inserted in the circuit. By applying the input pattern 100000, an attacker can observe the secret values of the key bits at the outputs $O1$ and $O2$. Thus, random encryption fails to ensure the secrecy of key bits.

B. DfTr 2: Split Manufacturing

Split manufacturing has been proposed to thwart a foundry from accessing the entire design information and from stealing the design or inserting Trojans into them [3], [21]. In split manufacturing [21], the layout of the design is split into two: the front end of line (FEOL) layers and the back end of line (BEOL) layers, which are then fabricated separately in different foundries, as illustrated in Fig. 3. The FEOL layers consist of transistors and lower metal layers (for example, metal 4 and lower layers [21]) and the BEOL layers consist of the top metal layers (for example, metal 5 and higher layers [21]). This corresponds to the partitioning of a gate level netlist into blocks where the transistors and wires inside a block form the FEOL layers, and the top metal wires connecting the blocks and the IO ports form the BEOL layers. Postfabrication, the FEOL and BEOL wafers are aligned, integrated, and tested [3], [21], [22].

The asymmetric nature of the metal layers facilitates split manufacturing. The top BEOL metal layers are thicker and have a larger pitch than the bottom FEOL metal layers. In one embodiment, the fabricated FEOL and BEOL wafers are integrated using electrical, mechanical, or optical alignment techniques and tested for defects by a system integrator [21]. In another embodiment, the FEOL wafer is fabricated in an advanced foundry and then sent to a trusted second foundry where the BEOL layout is built on top of it [21].

1) Threat Model: Split manufacturing may improve the security of an IC as the FEOL and BEOL layers are fabricated separately and combined postfabrication [3]. This prevents a single foundry (especially the FEOL foundry) from gaining full access to the IC. The attacker in the foundry that manufactures the FEOL wafer has the GDSII layout file of the design, and can reverse engineer it to obtain the gate-level netlist [5]. Thus, an attacker can always reverse engineer the FEOL parts of the circuits. However, he/she does not have information about the BEOL parts. Without the BEOL layers, however, the attacker can neither identify the "safe" places within a circuit to insert trojans nor pirate the designs. The attacker in the FEOL foundry gains knowledge about most of the design (the transistors and the lower metal layers) except for the missing BEOL connections. Once the attacker determines these missing BEOL connections, he/she can reconstruct the original design.

Example 3: Consider the 1985 International Symposium on Circuits and Systems (ISCAS-85) combinational logic benchmark circuit C17, shown in Fig. 4. It is partitioned into partition A (light colored) and partition B (dark colored). Typically, the wires within a partition (local wires except *Vdd* and clock) are assigned to lower metal layers. The wires that span the partitions and I/O ports are assigned to higher metal layers. This makes routing easier [24]. The nets connecting the input ports $I1$–$I5$ to the corresponding

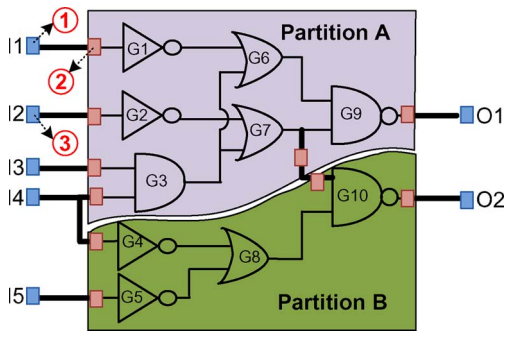

Fig. 4. ***Secure processor design flow to thwart insider attacks. The design team and the foundries are not trusted. The trusted security and integration teams perform logic encryption, integration, and Trojan detection (shaded blocks) by unlocking the design.***

inputs of the gates *G*1–*G*5 use the BEOL layers. The nets connecting the output of gates *G*9 and *G*10 to output ports *O*1 and *O*2, respectively, use the BEOL layers. The net that connects the output of *G*7 to one of the inputs of *G*10 also uses the BEOL layers. Without the BEOL connections, an attacker in the foundry does not know the connectivity among input ports, output ports, and partition pins.

2) Previous Work: In [25], an algorithm to select wires for the BEOL layers is provided. A formal notion of an attacker's inability to figure out the missing BEOL connections is provided. However, this approach has a significant performance overhead, potentially superseding the benefits of a high-end FEOL foundry. Furthermore, it overlooks the vulnerability of the design to proximity attack, when a design is synthesized by conventional physical design tools. One can also leverage the 3-D manufacturing technology where security-sensitive components can be placed in one layer and manufactured in a trusted low-end foundry, and other components of the design can be placed in another layer and manufactured in an untrusted high-end foundry [26]. Feasibility of split manufacturing for analog designs has been demonstrated [27].

Criterion 3 (Resilience to Proximity Attack): The attacker in the foundry should not be able to determine the missing FEOL connections from the BEOL connections.

Problem 3: Naive split manufacturing is vulnerable to proximity attack. This attack exploits the heuristic that floorplanning and placement (F&P) tools use to reduce the wiring (delay) between the pins to be connected [24]—place the partitions close by and orient the partitions. This heuristic of most F&P tools is a security vulnerability that can be exploited by an attacker in the FEOL foundry who does not have access to the BEOL layers. Thus, the attacker simply makes the missing connections between the two closest compatible[2] pins.

Example 4: Consider the locations of partition pins and IO ports of the F&P C17 benchmark as shown in Table 1. Consider the input port $P_{I1,\mathrm{IO,in}}$,[3] which is connected to pin $P_{I1,A,\mathrm{in}}$ in partition *A*. The locations of $P_{I1,\mathrm{IO,in}}$ and $P_{I1,A,\mathrm{in}}$ are $(0,6)$ and $(1,6)$, respectively. The distance between these two pins is 1 unit. Now, consider another input port $P_{I3,\mathrm{IO,in}}$. The distance between $P_{I3,\mathrm{IO,in}}$ and $P_{I1,A,\mathrm{in}}$ is 1.414 units. Thus, the closest possible pin to $P_{I1,A,\mathrm{in}}$ is $P_{I1,\mathrm{IO,in}}$. Hence, an attacker will connect these two pins in the netlist for the corresponding missing BEOL connection. Similarly, he/she can connect all the other partition pins with their closest pins and reconstruct the original design. An attacker can perform a proximity attack to recover most of the missing BEOL connections [23]. Thus, naive split manufacturing is vulnerable to the proximity attack.

Table 1 Proximity Attack on Split Manufacturing. *X*–*Y* Coordinates of the Pins in Partitions *A* and *B* and IO Ports of F&P C17 Design Shown in Fig. 4. The Coordinates Are Shown as Absolute Units for Ease of Understanding

Partition A		Partition B		Input & Output	
Pin	XY location	Pin	XY location	Port	XY location
$P_{I1,A,in}$	(1,6)	$P_{I4,B,in}$	(1,2)	$P_{I1,IO,in}$	(0,6)
$P_{I2,A,in}$	(9,5)	$P_{I5,B,in}$	(1,0)	$P_{I2,IO,in}$	(10,6)
$P_{I3,A,in}$	(1,5)	$P_{G7,B,in}$	(7,2)	$P_{I3,IO,in}$	(0,5)
$P_{I4,A,in}$	(1,4)	$P_{G10,B,out}$	(7,0)	$P_{I4,IO,in}$	(0,4)
$P_{G9,A,out}$	(9,4)			$P_{I5,IO,in}$	(0,0)
				$P_{O1,IO,out}$	(10,5)
				$P_{O2,IO,out}$	(8,0)

C. DfTr 3: IC Camouflaging

Camouflaging is a layout-level technique that hampers an attacker from extracting a gate-level netlist of a circuit from the layout through imaging different layers. In one embodiment of IC camouflaging, dummy contacts are used [28]. Contacts are conducting materials that connect two adjacent metal layers or a metal layer 1 and poly. They pass through the dielectric that separates the two connecting layers. While a conventional contact (true contact) has no gap, a dummy contact has a gap in the middle and fakes a connection between the layers, as shown in Fig. 5. Such dummy contacts are used to design standard cells that look alike irrespective of their functionality. For example, NAND and NOR standard cells can be designed to look alike using dummy contacts. When deceived into incorrectly interpreting the functionality of the camouflage gate, the attacker may extract a netlist that is different from the original netlist.

1) Threat Model: To prevent reverse engineering, a designer camouflages certain gates in the design.[4] For example, the OR gate *G*7 in Fig. 6 is camouflaged. This design with camouflaged gates is then manufactured at a foundry. The manufactured IC is sold in the market. An attacker can reverse engineer an IC by depackaging, delayering, imaging the layers, and extracting the netlist. However, in the extracted netlist, the functionality of the camouflaged gates is unknown. For example, in Fig. 5, the functionality of *G*7 is unknown and an attacker assigns an arbitrary two-input function to it. Consequently, an attacker may obtain an incorrect netlist. Additionally, if the target IC is sold in the market, an attacker can buy one of them. In this way, an attacker can gain access to good I/O pairs.

2) Previous Work on IC Camouflaging: An IC camouflaging technique can leverage unused spaces in an IC and fill

[2]Two pins are compatible if one pin is the output of a gate or an input port, and the other pin is an input of a gate or an output port.

[3]$P_{\mathrm{Net,Partition,Direction}}$ denotes a partition pin or an IO port. Net is the name of the wire in the original design. Partition represents either the partitions *A* or *B* or the IO port. Direction of a pin can be either in or out.

[4]A designer does not camouflage all the gates in the design because of the power, area, and delay overhead of the camouflaged gates.

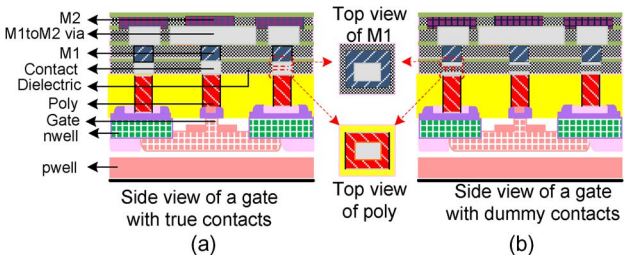

Fig. 5. ***Dummy contact-based IC camouflaging. (a) A logic gate with true contacts. (b) A logic gate with dummy contacts. The top view, which is used by an attacker in reverse engineering, is identical for both true and dummy contacts [28].***

them with standard cells [29]. The outputs of these filler cells will not drive any active logic. Hence, an attacker can identify and discard them while extracting the netlist. One can also camouflage a design by using programmable standard cells [30]. Postfabrication, these cells will be programmed using a control input. However, such control inputs have to be stored on-chip which requires a tamper-proof nonvolatile memory. Similar to introducing dummy contacts, a designer can also create dummy channels, which will result in nonfunctional transistors [31] and can be used to deceive an attacker.

3) Criteria for IC Camouflaging: An IC camouflaging technique should satisfy two criteria: a) incorrect outputs should be produced on attempts to try the incorrect one of many possible functionalities of a camouflaged gate; and b) an attacker should not be able to retrieve functionality of the camouflaged gates.

Criterion 4 (Output Corruption): The objective of the defender (designer) is to prevent his/her IP from being copied by an attacker in the foundry and to prevent black-box usage. The attacker does not know the functionality of the camouflaged gates. Hence, he/she will try one of the many possible functionalities of each camouflaged gate and in turn expect the design to become functional (i.e., to produce correct outputs). In the worst case, he/she has to perform a brute force attack by trying out all possible functionalities of all camouflaged gates. The objective of the defender is to force the attacker to perform a brute force attack. Therefore, the defender needs the camouflaged design to produce incorrect outputs on incorrect functionality assignment to camouflaged gates. A defender has to camouflage the design such that an attacker, with the knowledge of the publicly available IC camouflaging objectives and algorithms, is not able to obtain the correct outputs by trying an incorrect functionality. This can be done by minimizing the correlation between the corrupted and original outputs, and thus by maximizing the ambiguity for the attacker similar to logic encryption and split manufacturing. The optimal point is again where 50% of the outputs are corrupted upon trying an incorrect functionality.

Problem 4: Random selection of gates for IC camouflaging fails to ensure the production of incorrect outputs for an incorrect functional assignment to camouflaged gates.

Example 5: Consider the circuit shown in Fig. 7. The camouflaged gate C1 can implement one of {XOR, NAND, NOR}. This circuit produces incorrect outputs for four out of the 16 possible input patterns, because the output $C1$ does not corrupt the output $O1$ for most of the input patterns.

Criterion 5 (Difficult-to-Break Camouflaging): A camouflaging technique should ensure that the functionalities of the camouflaged gates should not be resolvable, even when an attacker has access to good I/O pairs.

Problem 5: Random selection of gates for IC camouflaging fails to ensure that the functionalities of the camouflaged gates are hardly resolvable.

Example 6: Consider the camouflaged gate $C1$ in Fig. 8. The functionality of $C1$ can be resolved to be XOR by applying 010XXXX at the inputs. This input pattern will justify the inputs of $C1$ to 00 and sensitize the output of $C1$ to $O1$. If $O1$ is 0, then the functionality of $C1$ is resolved as XOR. Otherwise, the functionality of $C1$ can be resolved to

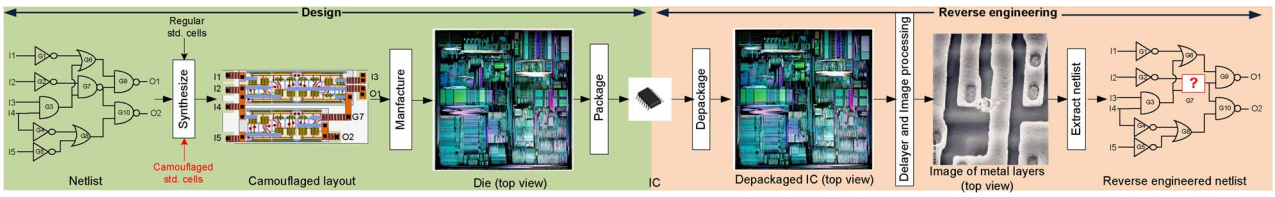

Fig. 6. ***IC camouflaging in an IC design flow. A design is synthesized into layout by using both regular standard cells and camouflaged standard cells. This layout is manufactured to obtain the IC. An attacker buys the camouflaged IC and depackages it. He/she delayers and images the metal layers and transistors. He/she then obtains the gate-level netlist by processing those images. However, the functionality of camouflaged cells cannot be resolved. For example, the functionality of G7 cannot be resolved [41].***

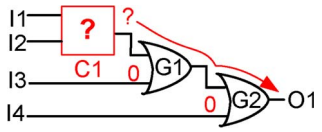

Fig. 7. *Problem with IC camouflaging. Camouflaged gate C1 can be one of {XOR, NAND, NOR}. On incorrectly assigning the functionality of the camouflaged gate, wrong outputs are produced for only four out of 16 possible input patterns [41].*

be NOR by applying 110XXXX at the inputs. This input pattern will justify the inputs of C1 to 10 and sensitize the output of C1 to O1. If O1 is 0, then the functionality of C1 is resolved as NOR. Otherwise, the functionality of C1 is resolved as NAND. Thus, a naive selection of gates for camouflaging may not offer security.

D. DfTr 4: Trojan Activation

Hardware Trojans inserted into a design at a foundry can be identified from their malicious behavior or by their additional power consumption [4], [32]–[34]. However, most of the Trojans remain dormant and they get activated only in rare conditions. Thus, to identify a Trojan one has to increase the switching activity within the Trojan circuit. This way a defender can activate the Trojan and observe its malicious behavior or can increase its power consumption, resulting in its detection [32].

1) Threat Model: As shown in Fig. 9, a designer synthesizes the design and generates the layout. He then characterizes its power and delay characteristics. An attacker in the foundry can insert Trojans into the design. The manufactured ICs with Trojans are then sent to the designer. The designer measures the power and delay characteristics of the manufactured IC by applying input patterns and

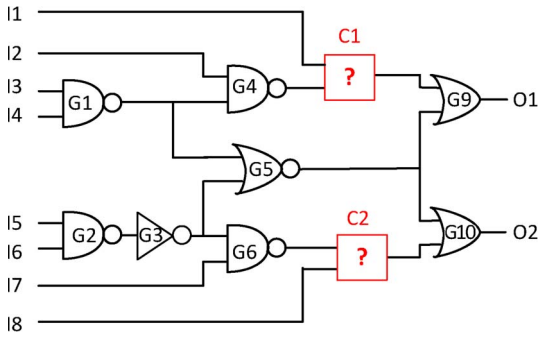

Fig. 8. *Problem with IC camouflaging. C1 and C2 are camouflaged gates that could implement one of {XOR, NAND, or NOR}. An attacker can identify the functionality of C1 as XOR by applying 010XXXX at the inputs and observing a 0 at O1. If he/she observes a 1, 110XXXX can be applied from the inputs. If O1 is 0, the functionality of C1 is resolved as NOR. Otherwise, the functionality of C1 is resolved as NAND. Similarly, the functionality of C2 can be resolved [41].*

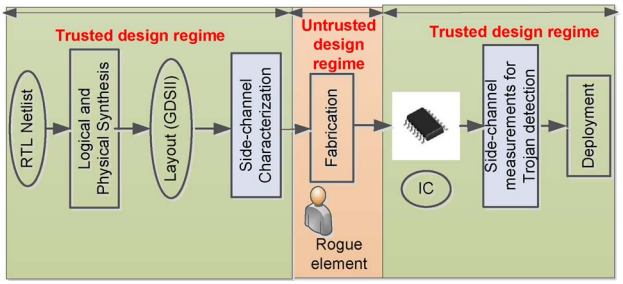

Fig. 9. *Threat model for hardware Trojans. An attacker in the foundry can insert Trojans. A defender can characterize the side channel of the designs and compare it against the measurements made on the manufactured IC [4], [32]–[34].*

compares them against the characterization. Any anomaly is considered as Trojan. The designer can use statistical and machine learning techniques to eliminate the effect of process variations [34], [35].

2) Related Work: Trojans are detected by characterizing the side channels such as power and delay on the golden model of the IC and comparing it against the measurements made on the manufactured IC. Based on the assumption that Trojans consume additional power, measurement of IC power dissipation can be used to detect Trojans [36]. Circuit delay characteristics have been used to detect Trojans [33], [35], [37]. While Jin and Makris [35] and Rajendran *et al.* [37] measure the delay of paths in the design and generate a delay fingerprint, Wang *et al.* [33] propose a method that measures the delays of all the paths using shadow latches. Potkonjak *et al.* [34] and Wei *et al.* [38] present nondestructive techniques to detect hardware Trojans in the presence of process variations where each IC component (e.g., a gate or an interconnect) has unique parameters. Unlike Salmani *et al.* [32] who use switching power, the techniques proposed in [34] and [38] use leakage power to detect Trojans. These techniques combine algebraic, numerical, and statistical methods with power and delay measurements to detect hardware Trojans. Furthermore, on-chip sensors are being incorporated to aid Trojan detection techniques [39].

Criterion 6 (Sufficient Switching Activity by the Trojan): To detect Trojans by activation or by power side-channel analysis, a defender has to cause sufficient switching activity (for example, $> 10^{-1}$) at the inputs of a gate to detect a Trojan.

Problem 6: One cannot always guarantee sufficient switching activity in all the gates. This is because certain gates have a low probability of transition.[5] Thus, in order to maintain the stealthy nature, an attacker can connect

[5]Probability of transition ($P_{\text{Transition}}$) of a gate is the product of probability of obtaining a 1 and probability of obtaining a 0 at its output.

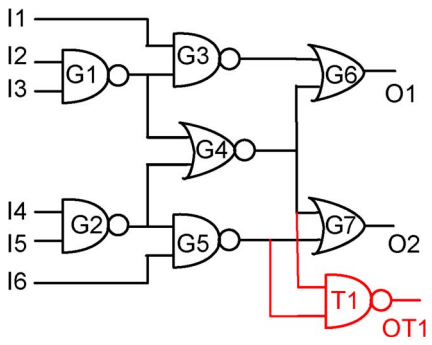

Fig. 10. ***Problem with Trojan detection by activation. The inputs of a Trojan are connected to gates with low switching probability. Thus, one cannot cause sufficient switching activity within the Trojan to detect it.***

the inputs of a Trojan to the outputs of gates which have a low probability of transition.

Example 7: Consider the design shown in Fig. 10. *T*1 is the Trojan gate. The transition probability of the gates in the design is listed in Table 2. In this design, the inputs of the Trojan gate are connected to the output of the gates *G*4 and *G*5. This causes $P_{\text{Transition}}$ to be 615/16384, which is less than the required value ($> 10^{-1}$). In this way, the Trojan remains stealthy and can circumvent the Trojan detection technique.

IV. STRENGTHENING DfTr TECHNIQUES THROUGH VLSI TESTING PRINCIPLES

In this section, we present the application of VLSI test principles in the context of hardware security. First, we provide a background on a set of test principles that can be leveraged for hardware security. Then, we show how the DfTr techniques presented in Section III can be strengthened by leveraging these test principles.

Table 2 Probability of Obtaining a 0 (P_0), Probability of Obtaining a 1 (P_1), Transition Probability ($P_{\text{Transition}}$), and Average Number of Input Patterns Required to Obtain a Transition at the Output of the Gates Shown in Fig. 10. The Probabilities Are Calculated Based on the Assumption That Probabilities of Obtaining 0 and 1 (P_0, P_1) at a Primary Input Are 0.5 and 0.5, Respectively [32]

Gate	P_0	P_1	$P_{\text{Transition}}$	Avg. # of input patterns
G1	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{3}{16}$	4.3
G2	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{3}{16}$	4.3
G3	$\frac{3}{8}$	$\frac{5}{8}$	$\frac{15}{64}$	3.3
G4	$\frac{15}{16}$	$\frac{1}{16}$	$\frac{15}{256}$	16.0
G5	$\frac{3}{8}$	$\frac{5}{8}$	$\frac{15}{64}$	3.3
G6	$\frac{45}{128}$	$\frac{83}{128}$	$\frac{3735}{16384}$	3.4
G7	$\frac{45}{128}$	$\frac{83}{128}$	$\frac{3735}{16384}$	3.4
T1	$\frac{5}{128}$	$\frac{123}{128}$	$\frac{615}{16384}$	25.6

As listed in Table 3, these principles can be utilized in order for the following to occur.

- Control the corruption at the outputs in three DfTr techniques: 1) in case of logic encryption, an incorrect key will result in incorrect outputs; 2) in case of split manufacturing, an incorrect BEOL connection will result in a design that produces incorrect outputs; and 3) in case of IC camouflaging, an incorrect assignment of a function to a camouflaged gate will result in incorrect outputs. The corresponding DfTr technique can then judiciously 1) insert the logic gates at proper locations; 2) swap block-level pins in split manufacturing; and 3) select the gates to be camouflaged. This requires the modeling of the injected corruption (corresponding to incorrect key, incorrect BEOL connections pins, or ambiguity of camouflaged gates) as faults, and making the DfTr decisions so as to favor the activation and propagation of these "faults."
- Make sure that the DfTr technique is difficult to break. This enhancement applies to logic encryption, where encryption key needs to be protected, and to IC camouflaging, where the ambiguity regarding the one-of-many functionalities of camouflaged gates in the reverse-engineered netlist needs to be maintained.
- Convert low-activity regions to high-activity regions. In case of Trojan activation, the increased switching activity in the high-activity region exposes the Trojan.

A. Solution to Problem 1: Fault-Analysis Driven Logic Encryption

Logic encryption should be performed by inserting the key gates at carefully selected locations in the design so that 50% of the output bits are corrupted when an incorrect key is applied. The following observations relate logic encryption and fault analysis in IC testing and can be leveraged to guide the insertion of XOR/XNOR key gates for this purpose.

Connection to Test Principle 1: Application of a wrong key is analogous to excitation of a fault. For a wrong key, either a stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1) fault will get excited, when XOR/XNOR gates are used for encryption. This is illustrated for the *C*17 circuit encrypted with one XOR gate (*E*1) as shown in Fig. 11(b). If a wrong key ($K1 = 1$) is applied to the circuit, the value of net *B* is the negated value of net *A*. This is the same as exciting an s-a-0 (when $A = 1$) or s-a-1 (when $A = 0$) fault at the output of *G*7, as shown in Fig. 11(a).

Connection to Test Principle 3: Corruption of an output due to a wrong key is analogous to the propagation of an excited fault to this output. This is illustrated for the circuit shown in Fig. 11(b). Let a wrong key ($K1 = 1$) be

Table 3 Enhancing DfTr Techniques Using Test Principles. The Number Within the Parentheses Indicates the Criterion/Test-Principle Number

DfTr technique	Criterion	Test principle(s)	Effect
Logic encryption	Output corruption (2)	Fault excitation (1)	Application of a wrong key-bit
		Fault propagation (3)	Effect of a wrong key-bit propagating to an output
		Fault masking (4)	A pair of wrong key-bits masking each others effect
	Difficult to break (3)	Sensitization (2)	Leaking of key-bits
Split manufacturing	Resiliency to proximity attack (4)	Fault excitation (1)	Wrong connection
		Fault propagation (3)	Effect of a wrong connection propagating to an output
		Fault masking (4)	A pair of wrong connections masking each other's effect
IC camouflaging	Output corruption (5)	Fault excitation (1)	Assigning a wrong functionality to a camouflaged cell
		Fault propagation (3)	Effect of a wrong function assignment propagating to an output
		Fault masking (4)	A pair of camouflage cells with wrongly assigned functionality masking each others effect
	Difficult to break (3)	Sensitization (2)	Leaking of functionality of camouflage cells
Trojan activation	Increased switching activity (6)	Controllability (5)	Increased power consumption in Trojan circuits

applied to the circuit. For the input pattern 00000, an s-a-0 fault gets excited at the output of $E1$ and propagated to both outputs. The value at the output of the gate $E1$ is 0 instead of 1, and the output is 11 instead of the correct output 00. For the input pattern 01110, even though the s-a-0 fault gets excited at the output of $E1$, the output is 00, which is the same as the functional output, as the fault effects have been blocked.

Connection to Test Principle 4: Cancellation of the effect of multiple wrong key bits is analogous to the masking of multiple excited faults in a faulty design. Consider the encrypted circuit in Fig. 11(c). When key bits ($K1, K2$, and $K3$) are 000, the correct functional output is 00 for the input pattern 00000. However, if the key bits are 111 (wrong key), the effect introduced by the XOR key gate $E1$ is masked by the XOR key gates $E2$ and $E3$, resulting in the undesired correct output 00.

Meeting Criterion 1: Insert XOR/XNOR gates such that a wrong key will affect 50% of the outputs. In terms of fault simulation, this goal can be stated as finding a set of faults, which together will affect 50% of the outputs when excited. To insert an XOR/XNOR gate, we need to determine the location in the circuit where, if a fault occurs, it can affect most of the outputs for most of the input patterns. To determine this location, we use the concept of fault impact defined by (1). From a set of test patterns, we compute the number of patterns that detect the s-a-0 fault (*number of test patterns*$_{\mathrm{s-a-0}}$) at the output of a gate Gx, and the cumulative number of output bits[6] that get affected by that s-a-0 fault (*number of O/Ps*$_{\mathrm{s-a-0}}$). Similarly, we compute *number of test patterns*$_{\mathrm{s-a-1}}$ and *number of O/Ps*$_{\mathrm{s-a-1}}$

$$\begin{aligned}\text{FaultImpact} &= (\#\text{ of test patterns}_{\mathrm{s-a-0}} \times \#\text{ of O/Ps}_{\mathrm{s-a-0}}) \\ &\quad + (\#\text{ of test patterns}_{\mathrm{s-a-1}} \times \#\text{ of O/Ps}_{\mathrm{s-a-1}}). \qquad (2)\end{aligned}$$

Upon inserting an XOR/XNOR gate for encryption at the location with the highest fault impact, an invalid key will likely have the most impact on the outputs (i.e., the wrong outputs appear). Upon inserting a sufficient number of

[6]In sequential circuits, one should calculate the fault impact assuming that all the flip-flops are scan flip-flops and attacker has access to scan chains. Hence, each scan flip-flop is considered as a pseudo primary input and pseudo primary output.

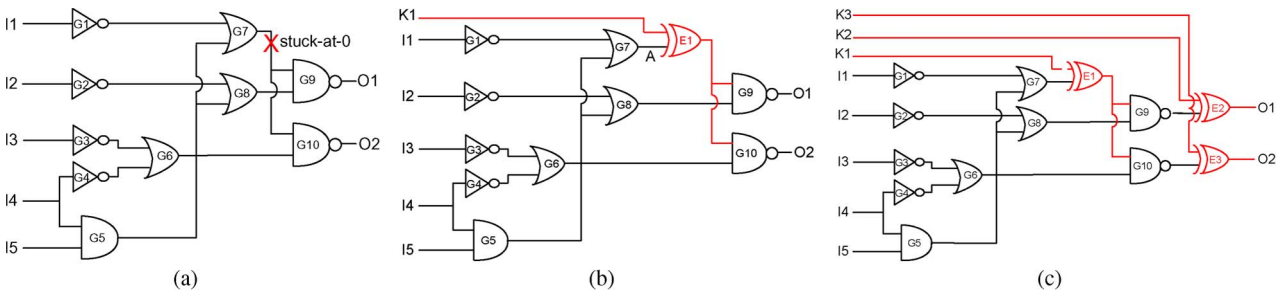

Fig. 11. *Solution to problem 1 [20]: By relating logic encryption and fault analysis via testing principles—fault excitation, propagation, and masking—a designer can find optimal places in a design to insert the key gates to satisfy criterion 1. (a) A faulty circuit. (b) An encrypted circuit with a wrong key ($K1 = 1$) equivalent to the faulty circuit. (c) A circuit encrypted with three XOR gates (E1, E2, and E3).*

XOR/XNOR gates, the 50% output corruption will be satisfied via a heuristic approach [18], [20].

B. Solution to Problem 2: Difficult-to-Break Logic Encryption

To prevent attacks that aim at leaking the logic encryption key, key gates have to be inserted judiciously such that the attacker should be forced into decrypting the key bits using brute force. This way, even with full access to the netlist the attacker will not be able to circumvent the defense.

Connection to Test Principle 2: A key bit can be sensitized to an output only if the side input of every gate in the path to the output can be justified to a noncontrolling value. However, if a side input is not justifiable to a noncontrolling value due to an interference from another key bit, the target key bit cannot be sensitized to an output. Consequently, an attacker will not be able to decipher its value without knowing the value of the other key bit.

Consider the example circuit in Fig. 12 which is functionally identical to circuit in Fig. 2(b) but with the two key gates $K1$ and $K2$ inserted at different locations. The attacker cannot sensitize $K1$ to an output as the side input of $G4$ cannot be set to 0 (noncontrolling value of an OR gate). This is because an attacker neither knows the value of $K2$ nor can control $K2$. Similarly, $K2$ cannot be sensitized to an output due to $K1$. The attacker then has to decrypt these key bits together rather than individually.

Meeting Criterion 2: Logic encryption can be strengthened by inserting key gates with complex interferences among them. An interference exists between two key gates if the value of one key bit cannot be propagated or sensitized without controlling or knowing the value of the other key bit. By inserting the key gates such that they block each other's path, and/or they converge in some other gate, a difficult-to-break logic encryption that forces the attacker to perform brute force attempts can be implemented [13].

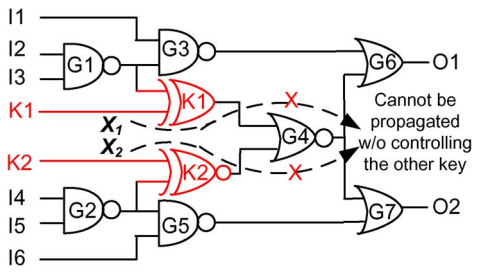

Fig. 12. *Solution to problem 2 [13]: An attacker cannot sensitize the effect of key bits K1 and K2 individually to the outputs. Hence, the attacker has to determine the values of K1 and K2 by brute force effort.*

This will prevent linear complexity attacks that target individual key gates and decipher individual key bits one at a time.

C. Solution to Problem 3: Fault Analysis Driven Split Manufacturing

Proximity attack can be overcome by rearranging the pins such that they are no longer closest to the pins that they are supposed to connect. A proximity attacker will thereby be deceived into making the wrong BEOL connections. The objective of a designer then is to swap a sufficient number of pins such that the functionality of the deceiving netlist[7] differs from that of the original netlist. This objective can be quantified using the Hamming distance metric.

In the context of split manufacturing, the Hamming distance is defined as the number of output bits that differ on applying an input to the original design versus to the design with a pair of FEOL pins swapped. Ideally, 50% of the output bits should differ on applying any input for any swapped pin pair. This metric is formally defined as follows. For a circuit C which produces an output y for an input x on applying the valid key, split manufacturing of C should satisfy

$$\frac{\sum_{x\in X}\sum_{m\in M;m'\in M;m\neq m'} HD(y_{m,m',x}, y_{m',m,x})}{\left(2^{|M|^2}-1\right)\times|y|}\times 100\% = 50\% \quad (3)$$

where X is the set of all inputs, M is the set of FEOL pins, $y(m,m',x)$ is the output vector on applying the input x where the FEOL pins m and m' are connected correctly, and $y(m',m,x)$ is the output vector on applying the input x where the FEOL pins m and m' are swapped. $|x|$ and $|y|$ denote the size of the input and output in bits, respectively. $|M|$ is the number of FEOL pins.

To find a swapping pin for a target pin, similar to an attacker, the defender can build the list of candidate pins for that target pin. Then, he/she can randomly select the swapping pin from that list. Unfortunately, such random selections might not guarantee that the attacker will get a wrong output on making a wrong connection. Hence, one can use VLSI test principles to select the swapping pin for a target pin in order to achieve the 50% output corruption criterion, as illustrated in Fig. 13.

Connection to Test Principle 1: The fact that different values on the swapping and the target pin introduce an error is analogous to the excitation of a fault. Ideally, such pins must be swapped to introduce errors. Otherwise, the resulting design, even with the wrong connections, will still produce mostly correct outputs.

[7]A deceiving netlist is created by the defender by swapping the pins.

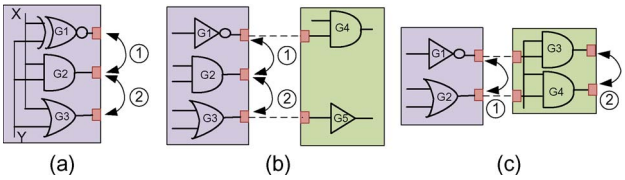

Fig. 13. *Solution to problem 3 [23]: Applying IC testing principles to split manufacturing. (a) Pin that has a logical value opposite to that of the swapped pin is preferred (fault excitation). Values at $P_{G1.A,\text{out}}$ and $P_{G2.A,\text{out}}$ differ only when $X = Y = 0$; Values at $P_{G2.A,\text{out}}$ and $_{G3.A,\text{out}}$ differ in two cases: $X = 1, Y = 0$ and $X = 0, Y = 1$. Thus, $P_{G3.A,\text{out}}$ is selected as the swapping pin for $P_{G2.A,\text{out}}$. (b) If $P_{G1.A,\text{out}}$ is selected as the swapping pin for $P_{G2.A,\text{out}}$, the wrong value will propagate only when the other input of G4 is 1. However, if $P_{G3.A,\text{out}}$ is selected as the swapping pin, the buffer G5 will always propagate the wrong value (fault propagation). (c) The logical error introduced by swapping $P_{G1.A,\text{out}}$ and $P_{G2.A,\text{out}}$ is cancelled by swapping $P_{G3.B,\text{out}}$ and $P_{G4.B,\text{out}}$ (fault masking).*

Connection to Test Principle 3: Corruption of an output due to two swapped pins is analogous to the propagation of an excited fault. Ideally, swapping should be done so that many outputs are corrupted (fault propagates to many outputs).

Connection to Test Principle 4: Cancellation of the effect of multiple swapped pin pairs is analogous to the masking of multiple excited faults. Sometimes, logical values corrupted by swapping pins in partition A can be restored to their original value because of swapping pins in partition B.

Meeting Criterion 3: Instead of randomly selecting the swapping pin, the pin that affects most of the outputs for most of the input patterns on swapping is selected. This accounts for fault activation, propagation, and masking scenarios. We define the fault impact metric, to select a swapping pin Y for a target pin X

$$\text{FaultImpact}_{X,Y} = \sum_{i=1}^{\#\text{ of test patterns}} \#\text{ of corrupted outputs}. \quad (4)$$

One can swap the target pin X with the swapping pin Y in the netlist and identify the cumulative sum of the corrupted output bits over a set of random test patterns. Fault impact quantifies the effect of swapping on the outputs of the design. Fault impact metric is used to select the swapping and target pins. The selected pins are then swapped and the netlist is updated. The above steps are repeated until all the partition pins and input ports are swapped or the Hamming distance value reaches 50%. For the ISCAS-85 benchmark designs, a designer achieves the 50% Hamming distance metric by swapping only a small set of pins (< 20 for most designs) via a heuristic approach [23]. Apart from fault-analysis-based pin swapping, one can make proximity attack difficult by manufacturing FEOL layers to metal layer 1 and below in the untrusted FEOL foundry [40]. However, this will tremendously increase the cost of trusted BEOL foundry, as it has to support more metal layers.

D. Solution to Problem 4: Fault Analysis Driven IC Camouflaging

IC camouflaging should be performed by carefully selecting the logic gates to be camouflaged with the ultimate goal of meeting the output corruption criterion. The following observations relate IC camouflaging and fault analysis in IC testing and can be leveraged to guide the selection of gates to camouflage for this purpose.

Connection to Test Principle 1: Attempting the wrong functionality of a camouflaged gate is analogous to excitation of a fault. For the wrong functionality, either an s-a-0 or s-a-1 fault may get excited. This is illustrated in Fig. 14 for camouflaged gate $C1$ that could implement one of many functionalities of XOR, NAND, or NOR. The example pattern justifies the inputs of $C1$ to 01; if the actual functionality of $C1$ is XOR/NAND and the attempted (by the reverse engineer) functionality is NOR, an error is introduced. The same error is introduced when an s-a-0 at the output of $C1$ is excited. On the other hand, if the actual functionality is XOR and the attempted functionality is NAND, this pattern fails to introduce any corruption; a different pattern is needed in that case.

Connection to Test Principle 3: Corruption of an output due to attempting a wrong functionality of a camouflaged gate is analogous to the propagation of an excited fault. This is illustrated for the circuit shown in Fig. 7, where the corruption is propagated from the output of $C1$ to $O1$.

Meeting Criterion 4: Select gates to be camouflaged such that attempting wrong functionalities will affect 50% of the outputs. In terms of fault simulation, this goal can be stated as finding a set of faults, which together will affect

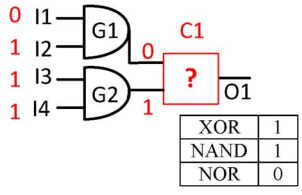

Fig. 14. *Solution to problem 4 [41]: Camouflaged gate C1 can implement one of {XOR, NAND, or NOR}. Incorrect values are always propagated to the output.*

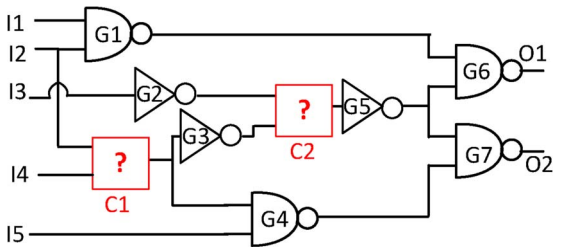

Fig. 15. ***Solution to problem 5 [41]: An attacker cannot identify the functionalities of the camouflaged gates C1 and C2 without brute force effort.***

50% of the outputs when excited. For this purpose, we define the output corruption metric

$$\text{OutputCorruptibility} = \sum_{i=1}^{\#\text{ of test patterns}} \#\ \text{O/Ps},$$
$$\text{with ambiguity activated and propagated.} \quad (5)$$

Camouflaging the gates with the highest output corruptibility values is expected to reduce the usefulness of the reverse engineered netlist [41].

E. Solution to Problem 5: Difficult-to-Break IC Camouflaging

Similar to implementing a difficult-to-break logic encryption, the attacker should be forced into using brute force in his/her attempts to identify the functionality of the camouflaged gates. Thus, the gates to be camouflaged should be selected judiciously; interference between the camouflaged gates increases the brute force effort for the attacker, forcing him/her to target camouflaged gates in large groups rather than individually.

Connection to Test Principle 2: An attacker can determine the functionality of a camouflaged gate by sensitizing its output to a primary output of the design. A gate's output can be sensitized to an output only if the side input of every gate in the path to the output can be justified to a noncontrolling value. However, if a side input is not justifiable to a noncontrolling value due to the ambiguity stemming from another camouflaged gate, the target gate's output cannot be sensitized to an output. Consequently, an attacker will not be able to resolve the functionality of the target camouflaged gate without knowing the functionality of the other camouflaged gate.

Consider the example circuit shown in Fig. 15 with two camouflaged gates *C*1 and *C*2. The attacker cannot sensitize *C*1's output to a primary output as the side input of *G*7 cannot be set to 1 (noncontrolling value of a NAND gate). This is because an attacker neither knows the functionality of *C*2 nor can control *C*2. Similarly, *C*2's output cannot be sensitized to a primary output without knowing the functionality of *C*1. Thus, an attacker cannot identify the functionalities of *C*1 and *C*2 without brute force effort.

Meeting Criterion 5: IC camouflaging can be strengthened by creating complex interferences among the camouflaged gates. By selecting the camouflaged gates such that they block each other's path, and/or they converge in some other gate, a difficult-to-break IC camouflaging that forces the attacker into brute force can be implemented. This prevents linear complexity attacks that target individual camouflaged gates and identify individual gate functionalities one at a time [41], [42].

F. Solution to Problem 6: Increased Switching Activity

To increase the transition probability ($P_{\text{Transition}}$) of a gate, one can introduce dSFFs [32] at its outputs. Through the dSFFs, one can easily control the input of the gate, increased $P_{\text{Transition}}$. This results in increased switching activity stemming from the Trojan circuit, causing the Trojan to consume more power. A defender can easily identify this additional power consumption during side-channel measurements and detect the Trojan. The dSFFs are used only when the designer needs to measure the side channels. During normal mode, they are bypassed and hence do not alter the functionality of the design. For this purpose, dSFFs are accompanied by a bypass gate (usually AND or OR).

Connection to Test Principle 5: Since the dSFFs are directly accessible to the designer through the scan chains, a designer can easily control them, injecting transitions. Hence, the $P_{\text{Transition}}$ value of dSFF is 0.25, which is the same as that of the primary input. Adding the dSFF to a wire increases its $P_{\text{Transition}}$ value as well as the that of the following wires. This results in increased switching activity in the design, thereby enabling Trojan detection.

Consider the circuit shown in Fig. 16. Here *D*1 is the dSFF gate and *G*8 is the bypassing gate. During normal

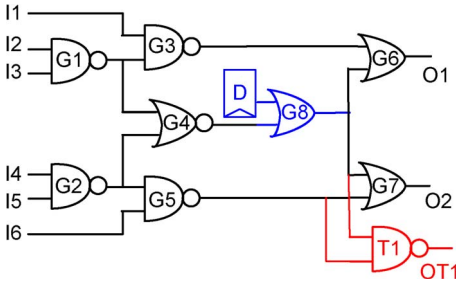

Fig. 16. ***Solution to problem 6 [32]. Inserting dSFF (D1) increases the transition probabilities of G4 and G5. This results in increased switching activity, enabling Trojan detection. G8 is the bypassing gate.***

Table 4 The $P_{\text{Transition}}$ Values of the Gates in the Design Shown in Fig. 16 Before and After dSFF Insertion. Gates With Unchanged $P_{\text{Transition}}$ Values Are not Listed

	Before dSFF insertion			After dSFF insertion		
Gate	P_0	P_1	$P_{\text{Transition}}$	P_0	P_1	$P_{\text{Transition}}$
G8	–	–	–	$\frac{15}{32}$	$\frac{17}{32}$	$\frac{255}{1024}$
G6	$\frac{45}{128}$	$\frac{83}{128}$	$\frac{3735}{16384}$	$\frac{45}{256}$	$\frac{211}{256}$	$\frac{9495}{65536}$
G7	$\frac{45}{128}$	$\frac{83}{128}$	$\frac{3735}{16384}$	$\frac{45}{128}$	$\frac{83}{128}$	$\frac{9495}{65536}$
T1	$\frac{5}{128}$	$\frac{123}{128}$	$\frac{615}{16384}$	$\frac{85}{256}$	$\frac{171}{128}$	$\frac{14535}{65536}$

mode, $D1$ is set to zero, retaining the original functionality of the circuit. When the designer wants to test for Trojans, he/she can set the desired input patterns at $D1$. Table 4 shows the $P_{\text{Transition}}$ values of the gates before and after dSFF insertion. It can be seen that the $P_{\text{Transition}}$ value of all the gates is greater than 0.1 after dSFF at their outputs.

Meeting Criterion 6: A designer can increase the $P_{\text{Transition}}$ value of a gate by inserting dSFF at its output along with the bypassing gate. On inserting a sufficient number of dSFFs, one can guarantee that all the gates in the design will have $P_{\text{Transition}}$ value greater than the desired value (for example, 0.1). An algorithm to insert dSFFs at optimal locations to have increased $P_{\text{Transition}}$ with minimal power, area, and delay overhead is given in [32]. While dSFFs are used to increase the sensitivity of Trojans to switching power, one can also use them to deliver input patterns that maximize the sensitivity of Trojans to leakage power [43].

V. DISCUSSION

A. Computational Complexity of DfTr Techniques

DfTr techniques are computationally intensive because most of them use automatic test pattern generation (ATPG), which is an NP-complete problem [14]. However, efficient heuristics developed for practical circuits reduce this complexity to polynomial in the number of gates in the circuit [44]. Table 5 lists the complexity of DfTr techniques. In the case of logic encryption and camouflaging, the complexity is polynomial in the number of key gates or the number of gates to be camouflaged. However, the number of key gates or the number of gates to be camouflaged is relatively small (for instance, 128), making these techniques practical. In the case of proximity attack, the complexity is quadratic in the number of FEOL pins. Even though there may be thousands of FEOL pins, if not millions, it does not deter an attacker from applying the proximity attack. In the case of dSFFs, the complexity is linear in the number of gates in the design.

Furthermore, one does not need to apply logic encryption and camouflaging techniques to the entire chip. For example, in the case of processors, one can apply these techniques to only the controller units. Without the controller unit, an attacker cannot compute [45].

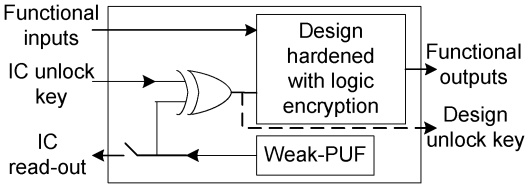

Fig. 17. ***Generating unique keys per chip using weak PUFs for logic encryption.***

B. Unique Unlock Keys Per Chip for Logic Encryption

In case of logic encryption, a designer should ensure that each chip has its own unlock key. Otherwise, a malicious user can use the key of one chip to unlock its pirated copies. In order to generate unique unlock keys per chip, one can leverage physical unclonable functions (PUFs). These are specialized circuits that leverage process variations to generate different outputs for the same set of inputs. One class of PUFs is called weak PUFs, which produce a unique signature per chip [46].

Table 5 Computational Complexity of DfTr Techniques

DfTr technique	Criterion	Worst-case complexity	Complexity of efficient heuristics	Notation
Logic encryption	Output corruption	$\mathcal{O}(2^{I\times K})$	$\mathcal{O}(N\times K)$	I- No. of inputs K - No. of key-gates N - No. of gates
	Difficult to break	$\mathcal{O}(2^{I\times K})$	$\mathcal{O}(N^3\times K^2)$	
Split manufacturing	Resiliency to proximity attack	$\mathcal{O}(2^{I\times M^2})$	$\mathcal{O}(N\times M)$	I - No. of inputs M - No. of FEOL pins N - No. of gates
IC camouflaging	Output corruption	$\mathcal{O}(2^{I\times K\times M})$	$\mathcal{O}(N\times K\times M)$	I- No. of inputs K - No. of camouflaged gates M- No. of functions implemented by a camouflaged gate N - No. of gates
	Difficult to break	$\mathcal{O}(2^I\times K^M)$	$\mathcal{O}(N^3\times K^M)$	
Trojan activation	Increased switching activity	$\Omega(2^I)$	$\Omega(N)$	I - No.of inputs N - No. of gates

Logic encryption can be coupled with weak PUFs to generate unique unlock keys per chip, as shown in Fig. 17. A designer encrypts a design using logic encryption. The key to unlock this design is called "design unlock key." The designer instantiates a weak PUF circuit in the IC along with the encrypted design, and sends the IC for fabrication. From the manufactured IC, the designer reads out the "chip key" produced by the weak PUF circuit. After this, the readout circuit is disabled. Because of the weak PUF circuit, each chip has its unique chip key. The designer XORs the chip-key with the design unlock key to compute the "chip unlock key." The chip unlock key is given to the user who applies it to his/her chip to make it functional.

A malicious user cannot use his/her chip unlock key on a different chip, as that chip will have a different chip key because of the weak PUF circuit. In addition, the user cannot read out the chip key, as the readout circuit is disabled by the designer. Thus, by utilizing weak PUFs, a designer can produce unique unlock keys per chip.

C. Which DfTr Techniques Should an IC Designer Use?

An IC designer can select any of the DfTr techniques depending on the trusted/untrusted entities in the IC design flow and the possible attacks that need to be thwarted (i.e., the threat model). An IC designer can use Table 6 as a guideline for selecting the appropriate DfTr technique(s).

Logic encryption and split manufacturing techniques provide an indirect protection against hardware trojans. Without the key or BEOL connections, a rogue element in the foundry will be unable to perform structural analysis to accurately identify safe places in the design to insert Trojans. Logic encryption protects the design IP against piracy and reverse engineering, and overbuilt ICs because, without the key, the chip will be nonfunctional, and thus, useless. Split manufacturing assumes the untrusted foundry but trusted end-user model; missing BEOL connections at the untrusted foundry will provide protection against IP piracy and IC overbuilding, but reverse engineering by end users will expose these missing connections.

IC camouflaging assumes the trusted foundry but untrusted end-user model; functionality of camouflaged gates, which is unknown to the end user, will provide protection against IP piracy and reverse engineering, but the dummy/real contact information that is available at the foundry may be used to circumvent the camouflaging technique, if the foundry is untrusted. Trojan activation only targets Trojans inserted at the foundry by increasing the switching activity within the circuit. Designers have to choose a set of techniques that suit their business model. Using more than one defense techniques will provide layers of defense against a variety of attacks.

Table 6 DfTr Techniques, Their Threat Models, and Security Guarantees

DfTr technique	Defender	Trusted entity	Attacker	Attacks thwarted
Logic encryption	Designer	–	Foundry	Trojan insertion
				IP Piracy
				IC overproduction
			Malicious user	Reverse engineering
				IP Piracy
Split manufacturing	Designer	BEOL foundry User	FEOL foundry	Trojan insertion
				IP piracy
				IC overproduction
IC camouflaging	Designer	Foundry	Malicious user	Reverse engineering
				IP Piracy
Trojan activation	Designer	User	Foundry	Trojan insertion

VI. SUMMARY AND CONCLUSION

In this paper, we elaborated on four DfTr schemes: logic encryption, split manufacturing, IC camouflaging, and Trojan activation. We outlined various different threat models, ranging from untrusted foundry to untrusted end-user, as well as DfTr techniques as a defense against such threats. Designers have to choose a set of techniques that suits their business model. Though using more than one defense technique will provide layers of defense against a variety of attacks, a DfTr technique suitable for one threat model may not be suitable for another one.

We described not only the basic DfTr techniques, but also additional ways to further strengthen them via the use of a few basic VLSI test principles. By defining and quantifying security metrics and deciphering the relationship between the fundamentals of these DfTr techniques and VLSI testing principles, we show that these techniques can be enhanced through the use of VLSI testing tools. Furthermore, these techniques have to be reinforced with mathematical proofs.

In summary, this paper linked VLSI testing with hardware security, and thereby will 1) introduce hardware security issues to the test community and elicit how relevant the problems are to VLSI testing; 2) introduce the hardware security and trust community how VLSI testing tools [47], [48] and techniques can be leveraged for enforcing security; and 3) motivate computer-aided design tool developers to create dedicated tools to facilitate the manufacturing of trustworthy ICs. ■

REFERENCES

[1] International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: http://www.itrs.net/Links/2011ITRS/Home2011.htm

[2] DIGITIMES Research, "Trends in the global IC design service market." [Online]. Available: http://www.digitimes.com/news/a20120313RS400.html?chid=2

[3] Intelligence Advanced Research Projects Activity (IARPA), "Trusted Integrated Circuits Program," 2011. [Online]. Available: https://www.fbo.gov/utils/view?id=b8be3d2c5d5babbdffc6975c370247a6

[4] Defense Science Board (DSB), "High performance microchip supply," 2005. [Online]. Available: http://www.acq.osd.mil/dsb/reports/ADA435563.pdf

[5] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. IEEE/ACM Design Autom. Conf.*, 2011, pp. 333–338.

[6] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. IEEE/ACM Design Autom. Test Eur.*, 2008, pp. 1069–1074.

[7] R. Chakraborty and S. Bhunia, "HARPOON: An obfuscation-based SoC design methodology for hardware protection," *IEEE Trans. Comput.-Aided Design*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[8] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, and R. Karri, "Towards a comprehensive and systematic classification of hardware Trojans," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1871–1874.

[9] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan.–Feb. 2010.

[10] SEMI, "Innovation is at risk as semiconductor equipment and materials industry loses up to $4 billion annually due to IP infringement," 2008. [Online]. Available: www.semi.org/en/Press/P043775

[11] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2009, pp. 113–116.

[12] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security," in *Proc. USENIX Security*, 2007, pp. 291–306.

[13] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. IEEE/ACM Design Autom. Conf.*, 2012, pp. 83–89.

[14] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Boston, MA, USA: Kluwer, 2000.

[15] Maxim Integrated, "DS28CN01: 1 kbit I^2C/SMBus EEPROM with SHA-1 engine." [Online]. Available: http://www.maximintegrated.com/datasheet/index.mvp/id/5369

[16] B. Barak *et al.*, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 2, no. 6, pp. 1–48, 2012.

[17] W. Griffin, A. Raghunathan, and K. Roy, "CLIP: Circuit level IC protection through direct injection of process variations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 5, pp. 791–803, May 2012.

[18] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Trans. Comput.*, 2013, DOI: 10.1109/TC.2013.193.

[19] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 66–75, Jan./Feb. 2010.

[20] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Logic encryption: A fault analysis perspective," in *Proc. IEEE/ACM Design Autom. Test Eur.*, 2012, pp. 953–958.

[21] R. Jarvis and M. G. McIntyre, "Split manufacturing method for advanced semiconductor circuits," U.S. Patent 7 195 931, 2004.

[22] M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, and M. Fritze, "Split fabrication obfuscation: Metrics and techniques," in *Proc. IEEE Symp. Hardware Oriented Security Trust*, 2014.

[23] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *Proc. IEEE/ACM Conf. Design Autom. Test Eur.*, 2013, pp. 1259–1264.

[24] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. New York, NY, USA: Springer-Verlag, 2002.

[25] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara, "Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation," in *Proc. USENIX Security*, 2013, pp. 495–510.

[26] J. Valamehr *et al.*, "A 3-D split manufacturing approach to trustworthy system development," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 611–615, Apr. 2013.

[27] K. Vaidyanathan *et al.*, "Efficient and secure intellectual property (IP) design for split fabrication," in *Proc. IEEE Symp. Hardware Oriented Security Trust*, 2014.

[28] J. P. Baukus, L. W. Chow, and W. Clark, "Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide," U.S. Patent 0 096 776, 2002.

[29] J. P. Baukus, L. W. Chow, R. P. Cocchi, P. Ouyang, and B. J. Wang, "Camouflaging a standard cell based integrated circuit," U.S. Patent 8 151 235, 2012.

[30] J. P. Baukus, L. W. Chow, R. P. Cocchi, P. Ouyang, and B. J. Wang, "Building block for a secure CMOS logic cell library," U.S. Patent 8 111 089, 2012.

[31] J. P. Baukus, L.-W. Chow, J. W. M. Clark, and G. J. Harbison, "Conductive channel pseudo block process and circuit to inhibit reverse engineering," U.S. Patent 8 258 583, 2012.

[32] H. Salmani, M. Tehranipoor, and J. Plusquellic, "New design strategy for improving hardware Trojan detection and reducing Trojan activation time," in *Proc. IEEE Int. Workshop Hardware-Oriented Security Trust*, Jul. 2009, pp. 66–73.

[33] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Proc. IEEE Int. Workshop Hardware-Oriented Security Trust*, Jun. 2008, pp. 15–19.

[34] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey, "Hardware Trojan horse detection using gate-level characterization," in *Proc. IEEE/ACM Design Autom. Conf.*, 2009, pp. 688–693.

[35] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Proc. IEEE Int. Workshop Hardware-Oriented Security Trust*, Jun. 2008, pp. 51–57.

[36] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC fingerprinting," in *Proc. IEEE Symp. Security Privacy*, May 2007, pp. 296–310.

[37] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, "Design and analysis of ring oscillator based design-for-trust technique," in *Proc. IEEE VLSI Test Symp.*, 2011, pp. 105–110.

[38] S. Wei, S. Meguerdichian, and M. Potkonjak, "Gate-level characterization: Foundations and hardware security applications," in *Proc. IEEE/ACM Design Autom. Conf.*, 2010, pp. 222–227.

[39] X. Zhang and M. Tehranipoor, "RON: An on-chip ring oscillator network for hardware Trojan detection," in *Proc. IEEE/ACM design Autom. Test Eur. Conf. Exhibit.*, 2011, DOI: 10.1109/DATE.2011.5763260.

[40] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ICs using split fabrication," in *Proc. IEEE Symp. Hardware Oriented Security Trust*, 2014.

[41] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM Conf. Comput. Commun. Security*, 2013, pp. 709–720.

[42] J. Rajendran, O. Sinanoglu, and R. Karri, "VLSI testing based security metric for IC camouflaging," in *Proc. IEEE Int. Test Conf.*, 2013, DOI: 10.1109/TEST.2013.6651879.

[43] D. Duarte, Y.-F. Tsai, N. Vijaykrishnan, and M. Irwin, "Evaluating run-time techniques for leakage power reduction," in *Proc. IEEE Int. Conf. VLSI Design*, 2002, pp. 31–38.

[44] M. Prasad, P. Chong, and K. Keutzer, "Why is ATPG easy?" in Proc. IEEE/ACM Design Autom. Conf., 1999, pp. 22–28.

[45] J. Rajendran, A. Kanuparthi, M. Zahran, S. Addepalli, G. Ormazabal, and R. Karri, "Securing processors against insider attacks: A circuit-microarchitecture co-design approach," *IEEE Design Test*, vol. 30, no. 2, pp. 35–44, Apr. 2013.

[46] U. Ruhrmair, S. Devadas, and F. Koushanfar, "Security based on physical unclonability and disorder *Introduction to Hardware Security and Trust*. New York, NY, USA: Springer-Verlag, 2012, pp. 65–102.

[47] H. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 9, pp. 1048–1058, Sep. 1996.

[48] H. Lee and D. Ha, "An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation," in *Proc. IEEE Int. Test Conf.*, 1991, pp. 946–955.

ABOUT THE AUTHORS

Jeyavijayan Rajendran (Student Member, IEEE) received the M.S. degree from Polytechnic School of Engineering, New York University, Brooklyn, NY, USA, where he is currently working toward the Ph.D. degree at the Electrical and Computer Engineering Department.

His research interests include hardware security and emerging technologies.

Mr. Rajendran is the organizer of the annual Embedded Systems Security Challenge, a hardware security competition. His publications won the Best Student Paper Awards at the 2013 ACM Conference on Computer and Communications Security, the 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, and the 2011 IEEE VLSI Design Conference. He won the third place in the 2013 Grande Finals of the ACM Student Research Competition; the ACM Student Research Competitions for Design Automation at the 2012 Design Automation Conference and the 2013 International Conference on Computer-Aided Design; the ACM PhD Forum for Design Automation at the 2014 Design Automation Conference; the Myron M. Rosenthal award at the 2011 NYU-Poly; the Service Recognition Award from Intel; and the third place at the 2011 Kaspersky American Cup. He is a student member of the Association for Computing Machinery (ACM).

Ozgur Sinanoglu (Member, IEEE) received the B.S. degrees in electrical and electronics engineering and in computer engineering from Boğaziçi University, Istanbul, Turkey, in 1999 and the M.S. and Ph.D. degrees in computer science and engineering from the University of California at San Diego, La Jolla, CA, USA, in 2001 and 2004, respectively.

He is an Associate Professor of Electrical and Computer Engineering at the New York University–Abu Dhabi, Abu Dhabi, United Arab Emirates. He has industry experience from Qualcomm as a senior DfT engineer post-Ph.D. His research interests include design for test, design for security, and design for trust for VLSI circuits, where he has around 120 conference and journal papers, and 15 issued and pending U.S. patents.

Prof. Sinanoglu is the recipient of the best paper awards at the 2011 VLSI Test Symposium and the 2013 ACM Conference on Computer and Communication Security. During his Ph.D., he won the IBM Ph.D. fellowship award twice.

Ramesh Karri (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of California at San Diego, La Jolla, CA, USA, in 1993.

He is a Professor of Electrical and Computer Engineering at the Polytechnic School of Engineering, New York University, Brooklyn, NY, USA. His research interests include trustworthy integrated circuits (ICs) and processors, high assurance nanoscale IC architectures and systems, very large scale integration (VLSI) design and test, and interaction between security and reliability. He has over 150 journal and conference publications in these areas. He is the Area Director for Cyber Security of the NY State Center for Advanced Telecommunications Technologies at NYU-Poly, Hardware Security Lead of the Center for Research in Interdisciplinary Studies in Security and Privacy (CRISSP, http://crissp.poly.edu/), cofounder of the Trust-Hub (http://trust-hub.org/), and organizer of the annual red team blue team event at NYU, the Embedded Systems Security Challenge (http://esc.isis.poly.edu).

Prof. Karri was the recipient of the Humboldt Fellowship and the National Science Foundation CAREER Award, and Best Student Paper Awards at the 2013 ACM Conference on Computer and Communications Security, the 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, and the 2011 IEEE VLSI Design Conference. He cofounded and served as the Chair of the IEEE Computer Society Technical Committee on Nanoscale Architectures. He is a cofounder and steering committee member of the IEEE/ACM Symposium on Nanoscale Architectures (NANOARCH), Program Chair (2012) and General Chair (2013) of the IEEE Symposium on Hardware Oriented Security and Trust (HOST), Program Co-Chair (2012) and General Co-Chair (2013) of the IEEE Symposium on Defect and Fault Tolerant VLSI and Nanotechnology Systems, and the General Chair of the 2013 NANOARCH. He serves on several program committees, including the VLSI Test Symposium, the Design Automation Conference, the Hardware-Oriented Security and Trust Symposium, and the International Conference on Computer Design. He is the Associate Editor of the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, and the *ACM Journal of Emerging Technologies in Computing*. He is an IEEE Computer Society Distinguished Visitor and has organized invited tutorials on trustworthy hardware (including at the 2012 VLSI Test Symposium, the 2012 International Conference on Computer Design, the 2013 IEEE North Atlantic Test Workshop, the 2013 Design Automation and Test in Europe, the 2013 International Test Conference, and the 2014 IEEE/ACM Design Automation Conference).