Domenic Forte · Swarup Bhunia
Mark M. Tehranipoor *Editors*

Hardware Protection through Obfuscation

Springer

Hardware Protection through Obfuscation

Domenic Forte · Swarup Bhunia
Mark M. Tehranipoor
Editors

Hardware Protection through Obfuscation

Springer

Editors
Domenic Forte
University of Florida
Gainseville, FL
USA

Mark M. Tehranipoor
University of Florida
Gainseville, FL
USA

Swarup Bhunia
University of Florida
Gainseville, FL
USA

ISBN 978-3-319-49018-2 ISBN 978-3-319-49019-9 (eBook)
DOI 10.1007/978-3-319-49019-9

Library of Congress Control Number: 2016955916

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Over the last two decades, the semiconductor industry has slowly moved toward the globalization of its supply chain. Due to increasing costs and complexity, it is no longer possible for a single corporation or entity to design, test, and fabricate today's integrated circuits (ICs) under one roof. With state-of-the-art semiconductor fabrication facilities or 'foundries' located across the world, the ability for IC design houses to monitor their own intellectual property (IP) has become limited. Similarly, system-on-chips or SoCs have also given rise to the concept of reusable IP-based design, whereby IP cores are sourced from several different vendors and integrated into one single SoC design. These trends have helped the industry to deal with the ever-growing complexity of ICs while keeping costs low and accelerating time-to-market.

Unfortunately, the benefits of globalization come at the inevitable cost of security. The convoluted supply chain introduces numerous opportunities for malicious parties to engage in IP piracy, counterfeiting and even introducing backdoors into a design. These malicious parties could be in the form of (i) untrusted foundries that fabricate the IC, (ii) third-party IP vendors, (iii) electronic design automation (EDA) tools, (iv) rogue insiders and disgruntled employees in a design house, (v) test or assembly facilities, and (vi) reverse engineers once the IC enters the supply chain. Thus, threats exist in each stage of the supply chain, and it is virtually impossible for one entity to 'trust' others with their IP in such a global landscape.

This book introduces the state of the art in hardware obfuscation which can be used to protect semiconductor IP at various levels of abstraction (e.g., register transfer, gate, or layout level). Hardware obfuscation techniques either conceal or lock the functionality and/or structure of the IC/IP so that it becomes difficult for a malicious or unauthorized party to engage in piracy or backdoor insertion. In contrast to watermarks or patents which are passive methods for IP protection, hardware obfuscation techniques are active; i.e., they deter reverse engineering and prevent piracy from ever happening in the first place.

While software obfuscation has received much attention over the years, the field of hardware obfuscation is relatively new. The past five years have seen an almost exponential growth in the amount of research work that has been done in the field.

Further, semiconductor companies and government entities have also shown an increased interest in developing viable options for hardware obfuscation, especially in light of numerous recent news on IP infringement cases, potential backdoors in chips manufactured offshore, IC reverse engineering techniques which were once thought to be impossible, and so on.

This book provides a comprehensive overview of various hardware obfuscation techniques that have been recently proposed by the research community. Although none of the individual techniques can provide a one-size-fits-all solution for all problems in hardware IP protection, they counter specific threats in the semiconductor supply chain, be it in the form of an untrusted foundry, reverse engineers in the supply chain, or a SoC designer willing to compromise a vendor's IP. The proposed techniques are also applicable to various levels of design abstraction, with some of them working to protect register transfer or gate-level IPs and others working to secure an IC layout.

A brief outline of the book is provided below.

1. **Hardware Obfuscation Preliminaries**: The first part of the book includes two introductory chapters on hardware obfuscation and background topics.
   - Chapter 1 describes the modern semiconductor supply chain, including each step of IC design and fabrication, the parties involved in these steps, and the resulting threats to security and trust. Recent advances in the field of hardware obfuscation, which are the subject of the remaining book chapters, are introduced as well. In addition, hardware obfuscation is differentiated from software obfuscation, cryptography, and other related work.
   - Chapter 2 provides background material on VLSI verification and testing. Topics include satisfiability, equivalence, fault modeling, controllability, observability, design-for-test, and other testing concepts that are often applied during hardware obfuscation methods and attacks. Popular hardware security primitives such as physical unclonable functions (PUFs) and true random number generators (TRNGs) that appear frequently throughout the book are also discussed.
2. **Logic-Based Hardware Obfuscation**: The second part of the book focuses on hardware obfuscation for combinational logic circuits, based on mechanisms such as key-based locking, permutation, and secure test infrastructure.
   - Chapter 3 introduces the concept of logic encryption, which involves 'locking' the functionality of a combinational circuit by inserting key-controlled gates. The chapter introduces basic logic encryption techniques, heuristics for inserting key gates, recent attacks on logic encryption (such as boolean satisfiability attacks and key propagation), and appropriate countermeasures.
   - Chapter 4 introduces the concept of circuit camouflaging, which involves configuring cells to perform different functionalities while maintaining an identical look to reverse engineers. A circuit partition-based attack and corresponding mitigation approach are proposed. The advantages of multiplexer-based circuit obfuscation cells are also discussed.

- Chapter 5 focuses on permutation-based obfuscation. The authors discuss the impact of permutation networks (such as Benes network) on resistance to brute-force attacks, discuss details of obfuscation on printed circuit boards (PCBs), and analyze the resiliency of permutation-based obfuscation to various physical attacks.
- Chapter 6 discusses data leakage vulnerabilities introduced by test infrastructures such as scan chains and JTAG. Obfuscation techniques that lock the scan chain, scramble test responses, etc., are discussed to protect against such attacks.

3. **Finite State Machine (FSM) Based Hardware Obfuscation**: The third part of the book deals with sequential circuit obfuscation by locking of the finite-state machine (FSM) description of the circuit.

- Chapter 7 introduces the concept and flow of active hardware metering where the finite-state machine (FSM) description of a design is modified with additional states and a PUF. The security of the proposed approach is evaluated against FSM reverse engineering and brute-force attacks to guess the state transitions needed to unlock the design.
- Chapter 8 introduces a hybrid scheme for FSM locking, in which modifications to the state transition graph of a circuit are combined with modifications to the original circuit in order to maximally deviate the circuit from its correct functionality. The benefits of this approach with respect to IP protection and targeted hardware Trojan insertion are also discussed.
- Chapter 9 introduces the concept of 'best possible obfuscation' for sequential circuits. Four unique structural transformation operations along with a key are employed to lock the IC which functions in a degraded mode unless it is initialized properly.

4. **Hardware Obfuscation Based on Emerging Integration Approaches**: The fourth part of the book looks at emerging integration technologies such as 2.5D/3D ICs and split manufacturing for obfuscation against untrusted foundries.

- Chapter 10 leverages split manufacturing techniques to securely conceal design information from an untrusted foundry. Heuristic algorithms and gate anonymity metrics are introduced for lifting wires to the trusted back-end-of-line or BEOL layers.
- Chapter 11 discusses the limitations of using split manufacturing and built-in self-authentication (BISA) independently against an untrusted foundry. A combined technique, called obfuscated BISA (OBISA), is introduced in order to combat both piracy and hardware Trojan threats. In OBISA, wire-lifting and filling of white spaces with fully testable functional filler cells are simultaneously performed to actively detect any tampering done by an untrusted foundry.
- Chapter 12 leverages 2.5D IC technology in order to protect the design against an untrusted foundry. In 2.5D integration, an interposer layer

connecting different die is kept secret. Algorithms that partition a gate-level netlist and place-and-route with security and performance in mind are described.

5. **Other Hardware Obfuscation Building Blocks**: The fifth and last part of the book looks at secure mechanisms for key exchange to enable obfuscation at various steps in the semiconductor supply chain.

   - Chapter 13 discusses the building blocks and cryptographic primitives needed to transfer and protect secret keys (used by obfuscation) in different application instances (3PIP vendor and SoC designer, SoC designer and foundry, etc.). The IEEE P1735 standard is combined with hardware obfuscation and digest mechanisms in order to protect from additional attacks such as IP piracy and tampering.

We hope that this book serves as an invaluable reference for students, researchers, and practitioners in the field of hardware IP protection.

Gainesville, FL, USA

Domenic Forte
Swarup Bhunia
Mark M. Tehranipoor

Contents

Part I Hardware Obfuscation Preliminaries

1 Introduction to Hardware Obfuscation: Motivation, Methods and Evaluation 3
Bicky Shakya, Mark M. Tehranipoor, Swarup Bhunia and Domenic Forte

2 VLSI Test and Hardware Security Background for Hardware Obfuscation 33
Fareena Saqib and Jim Plusquellic

Part II Logic-Based Hardware Obfuscation

3 Logic Encryption 71
Jeyavijayan (JV) Rajendran and Siddharth Garg

4 Gate Camouflaging-Based Obfuscation 89
Xueyan Wang, Mingze Gao, Qiang Zhou, Yici Cai and Gang Qu

5 Permutation-Based Obfuscation 103
Zimu Guo, Mark M. Tehranipoor and Domenic Forte

6 Protection of Assets from Scan Chain Vulnerabilities Through Obfuscation 135
Md Tauhidur Rahman, Domenic Forte and Mark M. Tehranipoor

Part III Finite State Machine (FSM) Based Hardware Obfuscation

7 Active Hardware Metering by Finite State Machine Obfuscation 161
Farinaz Koushanfar

8 State Space Obfuscation and Its Application in Hardware Intellectual Property Protection 189
Rajat Subhra Chakraborty and Swarup Bhunia

9 Structural Transformation-Based Obfuscation 221
Hai Zhou

Part IV Hardware Obfuscation Based on Emerging Integration Approaches

10 Split Manufacturing 243
Siddharth Garg and Jeyavijayan (JV) Rajendran

11 Obfuscated Built-In Self-authentication 263
Qihang Shi, Kan Xiao, Domenic Forte and Mark M. Tehranipoor

12 3D/2.5D IC-Based Obfuscation 291
Yang Xie, Chongxi Bao and Ankur Srivastava

Part V Other Hardware Obfuscation Building Blocks

13 Obfuscation and Encryption for Securing Semiconductor Supply Chain 317
Ujjwal Guin and Mark M. Tehranipoor

Index 347

Contributors

Chongxi Bao University of Maryland, College Park, MD, USA

Swarup Bhunia Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA

Yici Cai Tsinghua University, Beijing, People's Republic of China

Rajat Subhra Chakraborty Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, West Bengal, India

Mingze Gao Tsinghua University, Beijing, People's Republic of China; University of Maryland, College Park, MD, USA

Domenic Forte ECE Department, University of Florida, Gainesville, FL, USA

Siddharth Garg New York University, New York City, NY, USA; The University of Texas at Dallas, Richardson, TX, USA

Ujjwal Guin Auburn University, Auburn, AL, USA

Zimu Guo University of Florida, Gainesville, FL, USA

Farinaz Koushanfar University of California, San Diego, CA, USA

Jim Plusquellic University of New Mexico, Albuquerque, NM, USA

Gang Qu University of Maryland, College Park, MD, USA

Jeyavijayan (JV) Rajendran Department of Electrical Engineering, The University of Texas at Dallas, Richardson, TX, USA; New York University, New York City, NY, USA

Fareena Saqib Florida Institute of Technology, Melbourne, FL, USA

Bicky Shakya University of Florida, Gainesville, FL, USA

Qihang Shi ECE Department, University of Connecticut, Storrs, CT, USA

Ankur Srivastava University of Maryland, College Park, MD, USA

Md Tauhidur Rahman University of Florida, Gainesville, USA

Mark M. Tehranipoor University of Florida, Gainesville, FL, USA

Xueyan Wang University of Maryland, College Park, MD, USA; Tsinghua University, Beijing, People's Republic of China

Kan Xiao Intel Corporation, Santa Clara, CA, USA

Yang Xie University of Maryland, College Park, MD, USA

Hai Zhou Northwestern University, Evanston, IL, USA; Tsinghua University, Beijing, People's Republic of China

Qiang Zhou Tsinghua University, Beijing, People's Republic of China

Part I
Hardware Obfuscation Preliminaries

Chapter 1
Introduction to Hardware Obfuscation: Motivation, Methods and Evaluation

Bicky Shakya, Mark M. Tehranipoor, Swarup Bhunia and Domenic Forte

1.1 Introduction

While piracy of intellectual property (IP) relating to daily commodities such as clothing, medicine, and fashion items has had a long history, IP violation of technological assets, such as computer software and hardware, has become a recent albeit concerning problem. In April 2016, a Wisconsin grand jury slapped a $940 million penalty on Tata Consultancy Services for allegedly stealing Epic System Inc.'s healthcare database management software and incorporating it into its own products [1]. In 2013, two men, Jason Vuu and Glen Crissman, were indicted by the NY state Supreme Court for allegedly stealing source code from their former employer, flow trader, and a market-trading software provider [2]. In the same year, there was also a high-profile case involving Xilinx, a reputed and well-known FPGA manufacturer, and Flextronics International Ltd., a chip supplier. Xilinx alleged that Flextronics bought Xilinx's FPGA chips at a discounted rate (by lying about the intended end users), remarked the chips as higher grade and sold them for elevated prices, thereby violating Xilinx's IP through misrepresentation and exposing them to liabilities [3]. In 2015, Versara, a Texas-based software company, filed a lawsuit against automotive giant Ford [4]. The lawsuit alleged that Ford developed an in-house tool based on Versara's intellectual property, immediately after terminating its longtime contract with the company which provided Ford with its proprietary vehicle management software. The incidents highlighted above are just a modest sampling of the countless cases in which the electronic intellectual property of companies (and people) was violated, resulting in protracted litigation and massive loss of revenue/reputation.

B. Shakya (✉) · M.M. Tehranipoor · S. Bhunia · D. Forte
University of Florida, Gainesville, FL, USA
e-mail: bshakya@ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_1

1.1.1 Obfuscation for Intellectual Property Protection

In a world of tough competition, companies often spend a great deal of time and resources in reverse engineering and understanding their competitor's products. This routinely happens in industries ranging from automotive and computer software to electronics. While reverse engineering in itself is not a crime (in fact, it is protected by law), the information derived from reverse engineering could be used in a number of malicious ways. Consider a company that exploits their competitor's intellectual property by incorporating the IP into their own products, without providing any credit or compensation to the IP's rightful owner. Now, think of this scenario in the context of today's global economy where IP protection laws (and the degree of their enforcement) vastly vary from one part of the globe to another. Due to such realities of today's global economy, IP protection can no longer be limited to passive methods such as patents, copyrights, and watermarks. An active approach to intellectual property protection is required, of which obfuscation is a vital part.

1.1.1.1 Definitions

Obfuscation is defined as the technique of obscuring or hiding the true meaning of a message or the functionality of a product, in order to protect the intellectual property (IP) inherent in the product. A more formal definition of obfuscation, in the context of 'circuits' (boolean operators computing a logic function) or programs (that implement an algorithm or a function), has been provided in the field of cryptology. Formally, an obfuscator O is defined as a 'compiler' that transforms a program P into its obfuscated version $O(P)$ that has the same functionality as P yet is unintelligible to an adversary trying to recover P from $O(P)$. The obfuscator O has two key requirements: (i) $O(P)$ computes the same function as P (i.e., it is functionality-preserving) and (ii) anything that can be efficiently computed (in polynomial time) from $O(P)$ can also be computed given 'oracle' access to P (i.e., $O(P)$ can be used as a 'virtual black box') [5].

1.1.1.2 Encryption Versus Obfuscation

Encryption is the most effective way to achieve security and privacy of data and communication, but it cannot provide a full-fledged solution for IP piracy. The overall differences between encryption algorithms and obfuscation are summarized in Fig. 1.1. As is clear from the figure, algorithms, such as Advanced Encryption Standard (AES) and Rivest–Shamir–Adleman (RSA), are cryptographic primitives that transform plaintext data into mathematically random ciphertext given a key (encryption phase). They also perform the reverse operation (ciphertext to plaintext) on application of the same or different key (decryption phase). In contrast to encryption, obfuscation does not necessarily rely on key-based access control. It allows

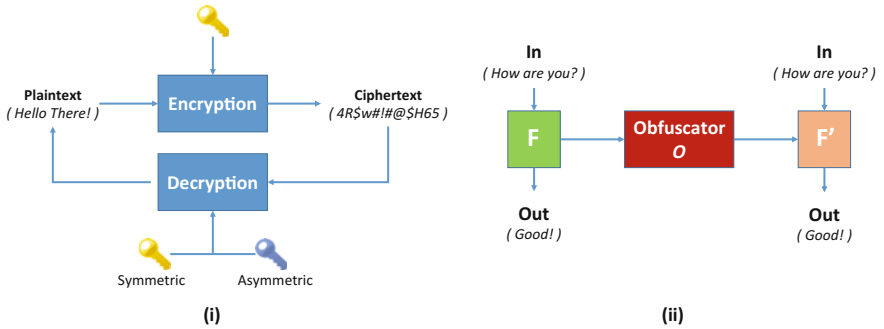


Fig. 1.1 **a** Encryption. **b** Obfuscation

an attacker to use the obfuscated program $O(P)$ as a virtual black box and is only concerned with the protection of the program P. Obfuscators are also commonly used in the context of obfuscating *programs* that implement an algorithm as opposed to general data (e.g., user information/authentication and passwords) that encryption can be used for.

Unfortunately, one major weakness that obfuscation has in relation to encryption is that the security of obfuscation techniques cannot be reduced to mathematically 'hard' problems such as integer factorization for RSA. In a landmark paper [5], it was shown that the general notion of virtual black-box obfuscation is not achievable for all programs. The authors argued that there exists a family of functions (represented as boolean circuits) that were inherently 'unobfuscatable.' In other words, given a program P' that computes the same input–output relationship as P, the attacker can feasibly reconstruct P or extract a secret s from P' about P with non-negligible probability. This means that unlike encryption, which does not prevent any set of data from being encrypted securely, obfuscation is not a universal operation.

1.1.1.3 Alternative Definitions

In spite of the above, later work has shown that although 'all functions' cannot be securely obfuscated (reduced to a 'hard' computational problem), some functions (such as a point function, which can be thought of as a password checking program) can be securely obfuscated [6]. Later, the authors in [7] showed that the virtual black-box property, which implies that the program should leak absolutely no other information than its input–output relationship, might be too strong to be achievable in practice. They proposed a relaxed definition termed as 'best possible obfuscation' which states that an obfuscated program $O(P)$ implementing a function F can leak as little information as any other program that computes the same function F. In other words, an adversary can learn no more information about the obfuscated program $O(P)$ than he or she can learn from any other program computing the same function.

While this does not guarantee what kind of information is securely hidden in $O(P)$, it assures that the obfuscation is literally the best possible [7].

In parallel, another definition of obfuscation, termed as 'indistinguishability obfuscation,' has also been proposed [5]. This definition implies that given two programs (or circuits) C_1 and C_2 which compute the same function F, an indistinguishability obfuscator O exists such that $O(C_1)$ and $O(C_2)$ are indistinguishable. This means that an attacker only has a random chance of being able to figure out whether he has $C1$ or $C2$ given possession of $O(C_1)$ or $O(C_2)$. Candidate constructions for such indistinguishability obfuscators have also been recently proposed using multilinear maps [8]. Note that the definition of indistinguishability obfuscation does not consider the strength of obfuscation of C_1 or C_2, i.e., it only says $O(C_1)$ and $O(C_2)$ are indistinguishable, not how strongly obfuscated they are on their own. It should also be noted that such obfuscators, although provably secure, are nowhere near practical at this point, as shown by a case study on a 16-bit point function (consisting of 15 *AND* gates) which blew up to a 31.1 GB file after running on a 32-core server for 9 h [9]. Nonetheless, the field of program obfuscation has received a great amount of attention in the past few years. More developments in optimization (as well as cryptanalysis) of these techniques can ensure practicality of provably secure obfuscation in the near future. It can also help us to gradually move away from ad hoc obfuscation practices such as code scrambling and white space removal for software obfuscation (see Sect. 1.3), which rely on the highly contested notion of 'security through obscurity.'

1.2 Hardware Obfuscation

Hardware obfuscation, as it relates to circuits (combinational, sequential, or system-on-chip), is concerned with protecting semiconductor intellectual property (IP). In the context of hardware, IP refers to a reusable unit of logic, cell, or chip layout design that is either licensed to another party or owned and used solely by a single party. IP protection has become a hot topic for research (and practical implementation), especially in light of today's globalized semiconductor production flow where trust between various entities in the supply chain is hard, if not impossible, to maintain. While techniques for achieving hardware obfuscation may be completely different than its software counterpart (see Sect. 1.3), the basic motivation remains the same in both cases: protecting intellectual property from adversaries capable of reverse engineering, piracy, and malicious alteration. Before discussing how hardware obfuscation can be realized, it is vital to understand the semiconductor supply chain and all the threats that are involved in the production of today's integrated circuits (ICs).

1.2.1 Integrated Circuit Supply Chain

Figure 1.2 shows the various steps involved in today's semiconductor supply chain. Each step is carried out in different parts of the globe by different entities (not necessarily under the same company) in order to reduce the insurmountable costs associated with producing state-of-the-art ICs and to reduce time-to-market.

- *IP Vendor*: Integrated circuits today are most commonly in the form of system-on-chips or SoCs. This means that a single silicon die contains intellectual property from several different vendors who could be scattered all across the globe. For example, the power management circuitry may come from an analog IP vendor in Texas, while the cryptographic IP core might come from a separate vendor in Europe. With increasing complexity of today's ICs and short turnaround times, it only makes sense for a design house (or SoC integrator) to buy IPs from several different vendors (usually at a much better rate) than to build the entire IC in-house from scratch. These IPs can generally be classified as (i) soft IP (RTL level, e.g., in the form of Verilog or VHDL), (ii) firm IP (gate-level IPs), and (iii) hard IPs (layout-level IPs, also known as hard macros, e.g., embedded SRAM).
- *Design House/SoC Integrator*: After collecting the necessary IP blocks, the design house puts these IPs in a single design and performs exhaustive simulations/verification to ensure that the overall design functions as intended. During this stage, electronic design automation (EDA) tools, which are commonly purchased from external EDA vendors, are heavily employed in order to perform synthesis, place and route (P&R), timing analysis, and verification. Note that most design houses today have gone 'fabless' meaning that they do not maintain their own silicon production facility (i.e., foundry).
- *Design-for-Test*: Design-for-Test (DFT) is a stage in the IC design process where infrastructures are integrated on-chip for post-manufacturing tests. A few decades ago, it was feasible to comprehensively test bare circuits after manufacturing so that defects generated during fabrication (e.g., short-circuit, damaged gates) could be detected quickly. However, with the sheer scale of ICs today, it is no longer possible to engage in exhaustive logic testing post-fabrication. As a result, the design house can choose to send its entire design (usually in gate-level form) to a separate DFT facility that specializes in inserting test infrastructures into the circuit. This can include specialized flip-flops (FFs) called scan FFs, which can allow a tester to observe and control internal parts of the design (which, otherwise, might not be accessible through the primary inputs of the circuit). Other advanced test structures for self-diagnosis such as built-in-self-test or BIST and test compression can also be incorporated at this stage to ensure good fault coverage of the circuit-under-test.
- *Foundry*: After performing comprehensive tests and design, the SoC integrator generates a final layout file (usually in the form of a GDSII file) and sends it to a semiconductor foundry. The foundry first generates a mask from the file and then etches the patterns from the mask onto an actual silicon wafer to produce the IC die (after dicing the wafer). The foundry may also test the individual die at this point, using manufacturing test patterns that are provided by the design

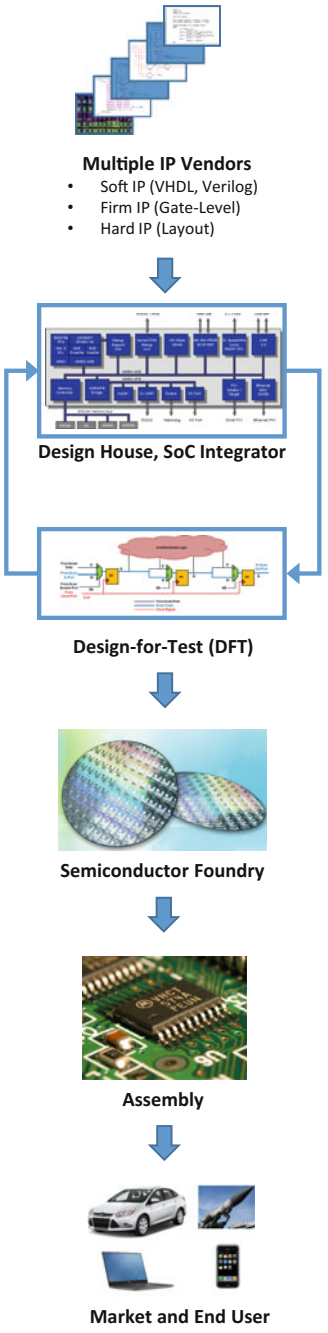


Fig. 1.2 Semiconductor supply chain

house/DFT facility. It is important to note that the foundry is usually the most cost-intensive stage in the flow, as advanced nodes (<22 nm) require state-of-the-art tools for atomic layer deposition (ALD), extreme ultraviolet lithography (EUV), and a large-scale clean room capable of high-volume production. As a result, most foundry facilities are located offshore where labor and operation costs can be kept to a minimum.

- *Assembly*: After a foundry manufactures the IC, they are sent to a separate facility that specializes in packaging the die into a complete chip. The die is first mounted on a substrate after which bond wires or solder bumps (for flip-chip packaging) are used to connect the I/O, clock, and power ports on the die to actual pins on a plastic packaging.[1] After this, the packaged IC is again tested and ready for incorporation within a larger electronic system.
- *Market, End User*: Once the ICs are integrated into a system, it usually reaches a distributor who is then responsible for delivering the product to the final vendors, after which it reaches the market and ultimately the consumer.

3D Integrated Circuits

While the flow described above applies to most ICs in production today, newer technologies such as 3D or 2.5D integrated circuits add a few additional steps to the pre-existing flow. Both 3D and 2.5D ICs allow the integration of multiple dies on the same package (vertically for 3D and horizontally for 2.5D) with high-bandwidth interconnects between them. They offer numerous advantages in terms of reduced footprint for increased computational power, shorter average wire length for reduced parasitics and power dissipation, possibility of heterogenous integration (e.g., MEMs stacking and dies at different technology nodes in the same package), and reduced latency (e.g., when stacking processor and memory). In through-silicon-via (TSV) enabled 3D IC manufacturing, multiple dies (or wafers) with different partitions of the design are fabricated separately. After fabrication, the die (or wafers) is aligned on top of each other and die-to-die (or wafer-to-wafer) high-bandwidth interconnects (called TSVs) are created to connect the partitions. In 2.5D IC, two (or more) dies are placed on top of a single interposer layer and microbumps on the individual die connect to wires in the interposer layer to create a final integrated die. Note that the die-to-die or wafer-to-wafer integration can take place in the same foundry or a separate foundry. More detailed discussions on these emerging integration technologies can be found in their respective chapters in this book.

FPGA Manufacturing

While the IC design flow we presented is geared for ASICs (application-specific integrated circuits), one could also consider the design flow for FPGAs (field-programmable gate arrays), which are widely used today. While FPGA manufacturing follows the same flow as ASICs, the design flow for a FPGA-based product is completely different. In the FPGA product flow, a design house simply buys FPGA chips from a vendor (who specializes in manufacturing FPGAs). The design house

[1] Note that the foundry might also have packaging capabilities.

then combines soft IPs (from multiple vendors as well as the design house itself) and then integrates them into one final wrapper, which is then converted to a bitstream file (using vendor EDA tools). This file configures the lookup tables (basic building blocks of FPGAs) and routing resources of the FPGA to realize the design in silicon. Note that this FPGA flow is much shorter (and simpler) than that of ASICs. However, the high monetary cost associated with FPGAs (per unit), difficulty of high-volume production, lower performance (in terms of clock speed), and higher-power consumption limit the scope of FPGAs despite their flexibility. Nonetheless, the protection of IP contained within the FPGA bitstream is an important area of research.

PCB Manufacturing

Printed circuit board manufacturing, while not as complex as IC manufacturing, also tends to have a distributed supply chain, which is described below.

- *Design House*
  - A design house who manufacturers electronic/embedded systems first creates the layout file of his or her PCB with the help of an electronic design automation tool. During this stage, ICs that are part of the PCB are placed onto a template and wires are created between components by a combination of manual and autorouting.
  - After placement and routing, extensive simulations are performed to check the integrity of signals as they pass through the board (e.g., in terms of cross talk and signal degradation in the case of long wires).
  - The final layout of the PCB (in the form of a Gerber file) is then produced with contains information about (i) the number of layers in the board (modern PCBs have as many as 8–10 layers stacked) and (ii) exact coordinates of vias, wires, and components.
- *PCB Manufacturer*
  - The (Gerber) design is then electronically transmitted to a PCB manufacturer who uses it to produce the final board. The production can involve photolithography, milling, silk-screen printing, or a combination of the three to create the copper wire traces and vias on FR4 layers (the insulating base of a PCB). The PCB manufacturer also usually offers services for mounting + soldering the desired ICs on the board as well. After manufacturing, the PCB can go back to the design house, but it is usually forwarded to an assembly or distributor, who integrates the PCB into a complete electronic system.

1.2.2 Threats in the Supply Chain

It is clear that while the distributed supply chain for IC production has helped to drive cost and time-to-market down, it has also created a number of security and

trust concerns among different entities. These threats, as they pertain to silicon IP security, are discussed below.

1.2.2.1 Reverse Engineering

The goal of reverse engineering is for the malicious party to recover the IP. After recovery, the malicious party can (i) use it to create a product that it can then sell to other parties; (ii) use the IP in its own product without compensating the rightful IP owner (breach of contract); (iii) find security vulnerabilities in the IP (e.g., weak random number generation) and exploit it later on; and (iv) insert a targeted backdoor in the IP after gaining a complete white box understanding of the IP. The threats could be present in different stages.

- *Design House*: The design house could potentially reverse engineer the IP core it receives from the IP vendor (firm and/or hard IP).
- *Foundry*: The design house provides the complete design in the form of a GDSII file to the foundry, along with manufacturing test patterns. The foundry could easily recover the netlist from the GDSII file by finding the connectivity information from the layout and coupling it with the standard cell library it had previously provided to the design house.
- *Market*: Once an adversary obtains a manufactured IC (either through the open market or through theft), he or she could try to reverse engineer the chip through destructive means to recover the complete design/IP [10]. Note that since we are concerned with IP protection here, side-channel/noninvasive/semi-invasive attacks that try to recover the secret key will not be considered. For destructive reverse engineering, the attack begins by decapsulating the chip from its packaging (either by mechanical abrasion or by corrosive acids), which exposes the bare die. Once the die is exposed, the attacker begins the slow process of imaging of all the layers of the IC (via scanning/transmission electron microscopy, high-resolution optical microscopy, focused ion beam, etc.). After imaging each layer, the attacker reaches for the next layer by using techniques such as chemical mechanical polishing (CMP) or plasma etching which helps to grind away a specific depth of the chip (and specific materials, depending on the etchant used). Once images for each layer are obtained, they are stitched together using automated image processing algorithms to obtain the full layout of the IC. If a standard cell library is available, the layout can then be converted to a gate-level netlist. While the process seems extensive, there are dedicated companies (e.g., TechInsights and Chipworks in Canada and Integrated Circuit Engineering Corp. in Arizona) that can perform the RE tasks at reasonable price/turnaround time and impressive accuracy. For a more detailed treatment of invasive reverse engineering, we refer the reader to [11].
  - *PCB Reverse Engineering*: Similar kinds of destructive delayering and imaging attacks can also be applied to PCBs in order to recover the design. Since PCB traces are not nanoscale (yet), noninvasive techniques such as X-ray computed tomography can be used to image each layer of the PCB and get the connectivity

information (which can be used to produce a Gerber file) in a matter of hours [12].
– *FPGA Reverse Engineering*: When it comes to an FPGA, the IP that an adversary would try to steal would be the FPGA bitstream, which is usually stored in an onboard/on-chip nonvolatile memory. If the bitstream is unencrypted, the attacker could read out the memory, by either probing or imaging. In case of an encrypted bitstream, side-channel attacks can first be used to recover the encryption key [13]. After recovering the bitstream, the attacker can use it illicitly on another FPGA device. He or she could also convert the bitstream to its corresponding netlist [14].

1.2.2.2 IP Piracy

Apart from reverse engineering, an adversary in the supply chain could also use the IP illicitly as is. With respect to the supply chain, the following threats are involved.

- *Foundry*: A foundry may only be contracted to produce a certain number of the design house's IP. However, since the foundry has the complete working design, they may produce excess copies of the design and sell them in the market (as is or relabeled/remarketed as a 'cloned' product). This effectively allows them to forgo any research and development (R&D) costs and make a profit by illicitly using someone else's IP. This practice is referred to as 'overproduction'.
- *Design House*: A design house may only be licensed to use a vendor's IP core on a limited number of chips (and paying royalties depending on how many chips were manufactured) or for a specified period of time. Unfortunately, if the IP vendor does not have a means to 'meter' the number of chips produced or actively track the IP's license, the design house could engage in IP piracy by 'overusing' the IP. This is in addition to the threat of the design house modifying the IP and selling it under a new name to other unauthorized parties.
- *Design-for-Test*: An offshore untrusted DFT facility may also pirate the IP (by producing a cloned version of the IP and selling it to unauthorized third parties), as it has the complete gate-level design in its possession.

1.2.2.3 Tampering

An adversary in the supply chain could also tamper with the design and introduce vulnerabilities or backdoors (i.e., hardware Trojans) into it. Two types of attacks might be possible in this scenario: targeted and untargeted. In a targeted attack, an adversary, such as an untrusted foundry, design house, DFT facility, or even an untrusted EDA tool, could gain a partial or complete understanding of the IP-under-attack and insert Trojans that bring about a specific malicious effects. For example, a foundry could decrease the entropy produced by a random number generator in a fabricated crypto-core [15]. It could also severely thin down the interconnect on a

critical path on the design, such that it fails prematurely. An untrusted DFT facility could increase the observability of an internal node in the design, such that it reveals a critical internal asset directly through its primary outputs.

In an untargeted attack, an adversary's goal is sabotage or denial-or-service attack, by exploiting some critical portions of the design (e.g., power supply net and clock pin) and without gaining a complete understanding of the underlying design (through reverse engineering). For example, a foundry, if it finds unused space in the layout of the design, could implement a Trojan that, once triggered, resets a number of flip-flops or dramatically decreases the clock speed in the design. Similar kind of attacks could also be implemented by an untrusted DFT facility (without any space constraints, since the design is at gate level). It should also be noted that such targeted and untargeted Trojan attacks can also be implemented by an IP vendor (who perhaps has cloned another vendor's design introduced a Trojan into it and then sold it to a design house). For a comprehensive review of hardware Trojan attacks, we refer the readers to [16].

1.2.3 Why Isn't Encryption a Solution?

Due to the convoluted nature of the supply chain, numerous attacks which could compromise an entity's IP rights are possible. On quick thought, one might naively think that the solution to most, if not all of these issues, is to encrypt the IP. However, unlike software, encryption is not a viable option for ICs. While encryption implies a certain storage or speed overhead in software, hardware encryption implies actual gates. Since these gates are cast in silicon and encryption/decryption cores are not exactly area-efficient, encryption becomes unreasonable. Moreover, during most stages in the supply chain (such as foundry, DFT, and design house), the design is a complete white box to the adversary, i.e., information relating to all the gates and their interconnections is available. Since most encryption/decryption cores have repeating structures of arithmetic operations (e.g., AES-128 has 10 identical rounds of permutation/substitution), they are easily identifiable under a white-box attack model. This means that the adversary could simply go in and remove the core, thereby nullifying the security provided by the crypto-core. Lastly, encryption, as we discussed in Sect. 1.1.1.2, requires keys. This implies that a key management infrastructure must be available throughout each stage of the supply chain, and the design would also need to be periodically 'unlocked' in order to perform tests (especially in the case of encrypted soft IP cores—see Sect. 1.2.4.1). This creates further logistical issues to an already complex supply chain. Thus, alternatives are required in order to protect semiconductor intellectual property which could potentially cost billions to develop.

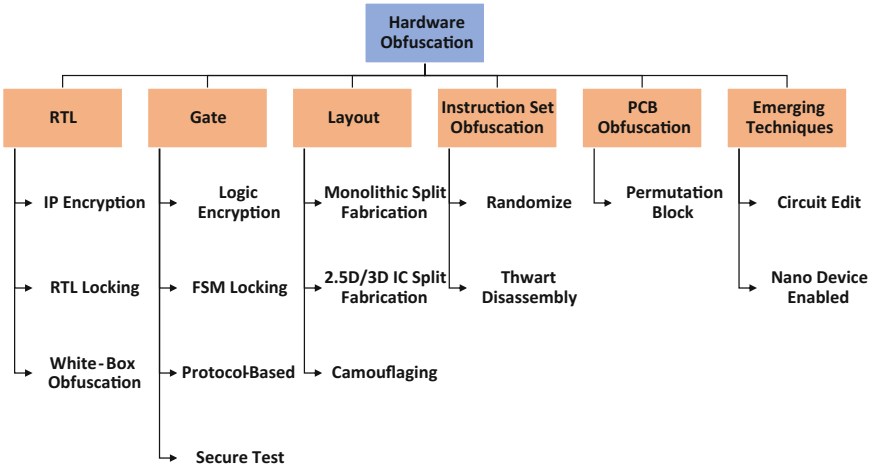


Fig. 1.3 A taxonomy of hardware obfuscation techniques

1.2.4 Techniques for Hardware Obfuscation

With the semiconductor supply chain and its inherent threats in mind, researchers have developed numerous hardware obfuscation techniques over the past decade. A taxonomy of these techniques is presented in Fig. 1.3 and is briefly introduced below.

1.2.4.1 RTL Level

Register-transfer level intellectual properties (IPs), also known as soft IPs, are commonly in the form of Verilog or VHDL code. Soft IP obfuscation can be achieved through the following methods.

- *IP Encryption*: The entire soft IP can be encrypted by common encryption techniques such as AES or RSA. In this setting, key management is usually handled by the EDA tools (which are assumed to be trusted[2]) and the IP buyer simply uses the encrypted IP as a black box. Unfortunately, the technique is limited to flexibility as the buyer/customer might be limited to a particular EDA tool. Recently introduced standards such as the IEEE P1735 encryption standard [17] have attempted to ease the interoperability of encrypted IPs across various EDA tools.
- *RTL-level Locking*: The authors in [18, 19] have proposed separate key-based locking approaches for RTL-based IPs. In these approaches, RTL code is first represented as a data flow [18] or state transition graph [19]. The graph is then

[2] A trusted party is committed to ensuring a proper IC design/fabrication flow (i.e., does not insert Trojans and protects IP confidentiality).

modified with key states, i.e., additional states in the FSM representation of the code that must be traversed with the help of a key sequence [18] or code word [19]. The IP comes into functional mode only when the correct keys are applied; otherwise, the IP is stuck in a non-functional, obfuscated mode. After obfuscating the graph and embedding locking features, the RTL code is regenerated from the graph, resulting in the final obfuscated IP.

- *White-box Obfuscation*: Soft IPs can also be obfuscated in terms of intelligibility and readability. The authors in [20] have utilized techniques such as loop transformations and reordering of statements in order to make a VHDL source code unintelligible yet functionally akin to the original code. The work in [21] explores control flow flattening, where a function or loop is broken into blocks and delegated to 'switch' statements, due to which the control flow of the program becomes much less obvious to an attack (as opposed to a simple 'for' loop where the execution order is obvious). Both of these techniques are more in line with traditional software obfuscation, which will be explored in Sect. 1.3. Unfortunately, such white-box obfuscation does not lock or obfuscate the functionality of the IP, leaving it vulnerable to IP piracy and overuse.

1.2.4.2 Gate Level

Gate level IPs, commonly referred to as firm IPs, are expressed in the form of netlists. In a netlist, the IP is expressed in the form of nets (connections) and a collection of standard logic cells. In order to protect gate-level IP, traditional encryption can be applied. However, this usually comes at the cost of significant hardware overhead and possible 'removal' attacks, as we previously discussed in Sect. 1.2.3. With these challenges in mind, several novel obfuscation techniques at the gate level have been recently proposed, of which the notable ones include the following.

- *Logic Encryption*: In this technique, extra gates such as XOR, XNOR, and MUX are inserted into the netlist of a design [22, 23]. These gates (and their logical output) are controlled by key bits which can be stored in a tamper-resistant nonvolatile

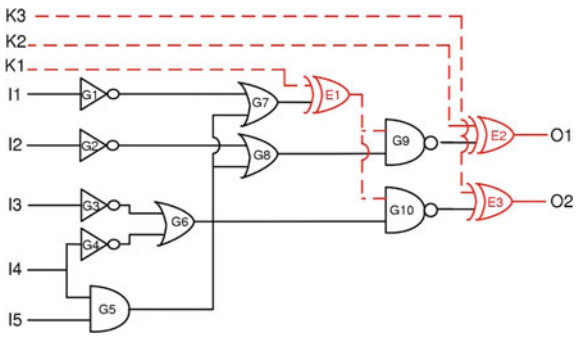


Fig. 1.4 A logic-locked circuit with three key gates [23]

memory or be derived from PUFs (see Fig. 1.4). The security of this approach lies in the fact that only the trusted design house knows and can apply all the correct key bits. Without the correct key bits, incorrect logical values are generated in the internal circuit nodes which eventually lead to faulty outputs. This effectively obfuscates the circuit to a third party who does not possess the correct key. Unfortunately, such locking techniques are vulnerable to Boolean satisfiability (SAT)-based attacks [24] as well as attacks that directly propagate the key bits to the circuit outputs [23] (details regarding both these attacks can be found in Sect. 1.2.5.2). Also, such techniques have only been studied on small-sized benchmark circuits, and their scalability is yet to be assessed.

- *FSM-based locking*: Several finite-state machine (FSM)-based locking techniques that are geared specifically toward sequential circuits have also been proposed. Among the most notable approaches, the authors in [25] have proposed the embedding of an authenticating FSM into a gate-level design. This authenticating FSM has to be traversed by an authorized user through a series of specific state transitions which are triggered by applying a series of input patterns only known to the user. If the chip is not unlocked via such a traversal, faulty values are generated by an additional modification kernel function and injected into the gate-level design in order to obfuscate the functionality of the locked chip. The security of the approach lies in the fact that the whole circuit is resynthesized after embedding the authentication and obfuscation features into the design, leaving an attacker with an insurmountable challenge of identifying (and removing) the implemented obfuscation. Although the authors propose utilizing pre-existing unreachable states in the FSM to incorporate the locking mechanism, the technique remains high in overhead (in terms of area, power, and delay).
- *Protocol-level*: The authors in [22, 26] have also utilized such locking techniques at a protocol level (with key exchange), in order to prevent an untrusted foundry from engaging in IC overproduction and IP piracy. These techniques are also commonly referred to as hardware metering. In [26], the authors utilize the aforementioned FSM-based locking technique, with the addition of a PUF to generate a unique start-up state for each IC. Upon manufacturing, a foundry relays the generated challenge–response pair from the PUFs so that the trusted design house can compute a unique unlocking FSM sequence for each chip. Additionally, the concept of black-hole states are introduced, which are irreversible state transitions from which the FSM cannot be reset. These black-hole states help in tamper detection in case a foundry attempts to randomly traverse the locked FSM. Although a cryptographically secure construction of the locked FSM is provided in [26] (via reduction to multi-point functions), such metering techniques are costly in overhead and do not take into account the testing procedure for ICs, a concern which has been more adequately addressed in [27].
- *Secure Test*: Recent work has shown that crypto-cores (AES, RSA, etc.) are particularly vulnerable to scan-based attacks, i.e., attacks that exploit scan flip-flops (FFs) embedded in a design as part of DFT, in order to reveal internal circuit values (including the secret key itself) [28]. These attacks are all possible because scan FFs are just normal registers in a design that can either be configured as scan or

normal FFs, depending on a 'scan enable bit' that is loaded from a multiplexer driving the *D* pin of the FF. Attackers who gain access to the scan chain can load multiple plaintexts into the cryptographic core (e.g., DES and AES) and switch from normal to scan mode. After doing this, they can flush out the corresponding intermediate values in internal registers and use hamming distance-based analysis to extract the secret key (or at least the individual round keys). Further, if the registers storing the round keys are part of the scan chain, the key could be directly read out without any analysis [29]. In order to prevent the attacker from accessing the scan chain, various obfuscation techniques have been employed. Simple solutions include the use of test compression structures that compress the value of several scan flip-flops into a single output, thereby making the observation of individual FF values unfeasible. Unfortunately, such compression-based obfuscation technique comes at the cost of high area overhead. Locking techniques have also been proposed that allow the designer to scramble the responses of the scan chain (chain of scan FFs) unless a secret key is applied. The lock-and-key technique proposed in [30] breaks up the scan chain into sub-chains. These sub-chains are configured properly and can be fed with the patterns sequentially only when the correct key is applied. If the wrong key is applied, the sub-chains are configured incorrectly by an LFSR, resulting in a 'lock' of the scan chains, which then results in wrong scan-out values. The work in [27] uses a similar concept to combat IC piracy. In the proposed technique, a 'scan locking block' (composed of a scrambling block and an XOR network) is utilized in order to perturb the scan chain responses. These perturbed responses can only be verified as correct by the design house, who can then appropriately chose whether to 'pass' or 'fail' a chip, thereby preventing the foundry from engaging in overproduction and allowing the design house to meter the number of chips produced. This approach also prevents out-of-spec/defective ICs from entering the market by giving the IP owner remote access to test responses. A more detailed treatment of scan-based attack and defenses can be found in the chapter on scan chain security in this book.

1.2.4.3 Layout Level

Layout-level IPs, also known as hard IPs, come in the form of geometrical and spatial information about the design which can be directly fabricated by a foundry. In order to protect the layout from piracy and possible Trojan insertion by an untrusted foundry, several split-manufacturing techniques have been proposed [31–33] (see Fig. 1.5).

- *Monolithic Integration*: In a traditional split-manufacturing flow, an untrusted foundry only fabricates the front-end-of-line (FEOL) layers, which include the expensive and state-of-the-art transistor/active layers. After FEOL fabrication, the design is sent back at the wafer level to the design house, who then uses a trusted foundry in order to complete the less costly back-end-of-line (BEOL) metal layers (see Fig. 1.6). While such techniques hide connectivity information from the foundry, attacks have been mounted on naive split manufacturing, which utilize

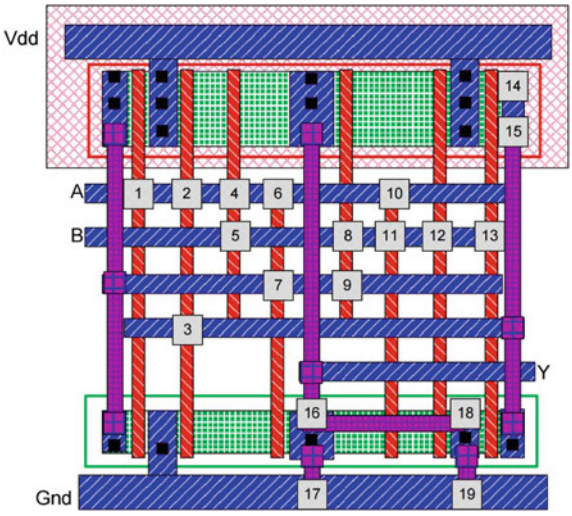


Fig. 1.5 A camouflaged standard cell that can function as a *NAND*, *NOR*, or *XOR*, depending on contact configuration [35]

proximity information from the assumptions that EDA tools use (e.g., gate distance to minimize wire length) [34]. Moreover, the biggest hurdle to split manufacturing is that the design house is still required to maintain a foundry to complete the BEOL, whose cost might be prohibitively expensive depending on the split layer. Further, foundry compatibility and wafer alignment with such monolithic split-fabrication techniques may also hurt IC yield.

- *2.5D/3D IC*: Attempts have also been made to utilize pre-existing 3D/2.5D IC technology to perform split manufacturing, as opposed to the monolithic integration technique proposed in [31]. In [32], wire-lifting is performed on a layout so that the lifted wires can be fabricated as separate layer at a trusted facility and the complete IC can be assembled by through-silicon-via (TSV) bond points in a normal 3D IC design flow. The authors also introduce the notion of k-security, by which every gate in the design in the FEOL layers is structurally akin to at least k other gates in the same design (as the BEOL information of the upper tier is missing).

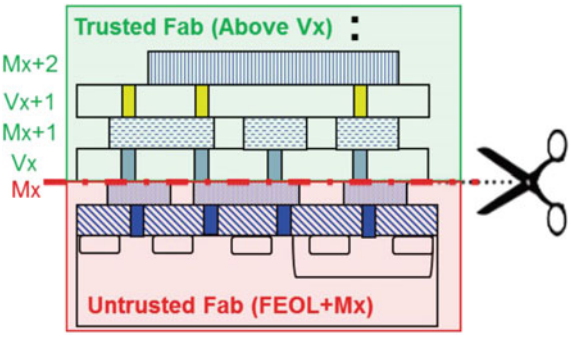


Fig. 1.6 Split manufacturing leveraging monolithic integration [37]

This makes it infeasible for the attacker (untrusted foundry) to identify the gates and thus launch a targeted hardware Trojan attack. In [33], 2.5D IC technology is leveraged in order to securely partition a gate-level design so that two or more partitions of the design can be fabricated at an untrusted foundry and the interposer layer connecting these partitions can be fabricated at a trusted facility. They introduce the concept of 'secure' partitioning, in which gates are iteratively moved from one partition to another until the global objective of a set wire-length penalty and 50% hamming distance (see Sect. 1.3.1) is met. Unfortunately, split manufacturing based on 3D/2.5D IC technology has the same drawbacks of requiring a separate fabrication facility. Further, these techniques require significant amounts of gate-swapping and wire-rerouting operations for obfuscation, leading to large area and delay overheads.

- *Camouflaging*: In order to protect against chip-level reverse engineering, the authors in [35] have proposed the use of special camouflaged standard cells. These cells have a layout that makes them appear the same to an invasive reverse engineer, whether they implement a *NAND*, *NOR*, or *XOR* functions. This is achievable through dummy contacts in the dielectric layer of the gate: Some contacts in the gate go all the way through the dielectric layer and into the metal layers above, while others are cut off. However, to an adversary, the contact looks the same from the top regardless of whether the contact is cut off or joined, giving rise to ambiguity while trying to identify the gate. This prevents the attacker from obtaining the complete netlist of the design. On the downside, the camouflaged gates themselves have a non-negligible power, area, and delay overhead, and a large number of these gates might have to be used in a design to achieve strong security for industrial designs. Further, recent attacks have shown that a design, even with an unrealistic number of camouflaged gates, can be effectively 'decamouflaged.' These attacks leverage SAT solvers and discriminating input patterns to resolve the hidden functionality of the camouflaged gates in negligible time [36].

1.2.4.4 Instruction Set Obfuscation

Every computer system has an underlying instruction set architecture (ISA) associated with it, which dictates the type of commands, data types, address space, and operation codes (opcodes) it can handle. The ISA serves as an intermediary between the software and hardware of the computer and is usually public knowledge. Unfortunately, this also means that for any attacker trying to remotely attack a system (or even invasively, via compromise of the memory unit holding the instructions), the ISA is well defined too, and thus, he or she can plant the attack on all computers using the same ISA. This serves as a basis for attacks such as buffer overflow. To combat the predictability of the ISA, the authors in [38] proposed a technique to scramble each byte of code (using pseudorandom numbers) and reversing the scrambling only when the code is executed in machine. This means that any unauthorized program, which was never scrambled, will be descrambled to random bits, thereby preventing any targeted malicious behavior. A similar approach which XOR's the instructions with

a secret key as they are transmitted between the processor and the main memory (as implemented on a simulated x86 environment), was proposed in [39]. An attacker without key access can only inject malicious code which is incorrectly decoded, thereby raising a flag or causing a detectable error.

Another technique for instruction set obfuscation focuses on disrupting the disassembly phase of reverse engineering (i.e., conversion of machine code, in hexadecimal or binary form, to assembly code in human-readable form) [40]. This is achieved by carefully inserting 'junk bytes' in the instruction stream of the code. These junk bytes cause an automatic disassembler to either misinterpret the instructions or the control flow of the program but do not affect the program's functionality (i.e., semantics) as they are unreachable instructions during run-time.

1.2.4.5 PCB Obfuscation

In order to prevent a PCB design from piracy, the authors in [41] propose the inclusion of a permutation block on the board. The permutation block, implemented with a complex programmable logic device (CPLD) or an FPGA, takes in suitable pins (e.g., general-purpose I/O) from programmable component in the design (e.g., microcontrollers) and permutes its pin connections before they reach their destination. The permutation is resolved to the correct configuration only when the correct key is applied to the CPLD or the FPGA (the permutation block).

1.2.4.6 Emerging Techniques

There have also been a variety of novel obfuscation techniques that have been proposed recently. A few of the notable ones are highlighted below.

- *Chip Editor*: The basic idea of circuit edit (CE) is to utilize technologies such as focused ion beam (FIB) in order to edit an integrated circuit after fabrication on a one-by-one basis. CE has been widely used in the semiconductor industry in order to perform failure analysis and circuit/mask repair without undergoing the humongous costs of remaking the IC mask. Recent work has focused on leveraging circuit edit for logic obfuscation and trusted fabrication [42]. The proposed technique called Chip Editor involves the inclusion of extra gates or wires into the IC design which is then fabricated and tested by an untrusted foundry. After fabrication, the IC is returned to the design house (or other trusted party), who then uses circuit edit techniques such as FIB in order to remove the added gates or wires and revert the circuit to its intended functionality. The security of the approach lies in the fact that the foundry is unable to determine the gates or wires that are added in or modified by the design house, even when the added gates have been accommodated with design-for-circuit-edit features such as widened pads. Figure 1.7 shows one example of a gate insertion-based obfuscation enabled by circuit edit, where an *AND* gate is inserted and is edited to a buffer post-fabrication,

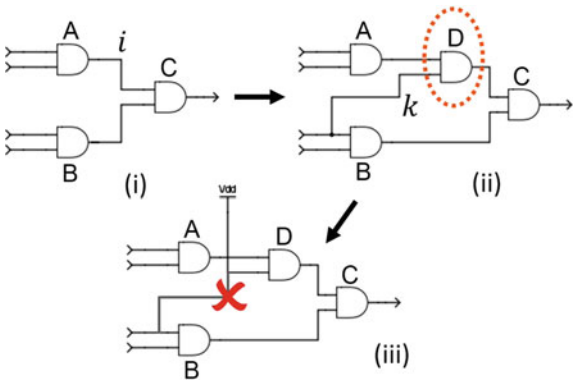


Fig. 1.7 Gate insertion enabled by circuit edit. **a** Original design with selected net *i*. **b** Gate D and wire *k* added. **c** Wire *k* edited to make gate D act as a buffer [42]

by tying one of its inputs to *VDD*. The authors in [42] also outline several other techniques for obfuscation, such as wire-swapping and insertion with other gate types. Compared to split manufacturing and 2.5D/3D IC obfuscation, the circuit edit technique removes the need for the design house to maintain a costly foundry to complete the BEOL or interposer layers and only requires a moderate cost FIB machine. The need for key management and secure key storage (e.g., in logic encryption) is also alleviated. However, the drawback of the approach is that it can only be utilized for low-volume IC production (since FIB operations take time and can only be done on a one-by-one basis). Similar to split manufacturing and 2.5D/3D IC obfuscation, it also does not prevent the IC from being reverse engineered once it enters the open market, thus limiting its scope to tightly controlled supply chains (e.g., military and aerospace).

- *Nanodevice enabled*: Several other techniques leveraging emerging nanoscale devices for circuit obfuscation have also been proposed. The authors in [43] propose the use of polarity-controllable silicon nanowire FETs (SiNW FETs) in order to make low-overhead IC camouflaging gates. As opposed to traditional camouflaged gates which require as much as 12 transistors, the authors show that it is possible to use only 4 SiNW to achieve a NAND, NOR, XNOR, and buffer functionality from the same gate, resulting in significant area and power savings. They also utilize SiNW FETs to make polymorphic gates, which can be configured to be either a NAND or a NOR gate depending on how the VDD and GND terminals are configured. Unfortunately, the drawback of using these novel devices is that they suffer from high leakage current and fabrication issues (imprecise device characteristics and limited scaling from top-down or traditional fabrication and yield issues from bottom-up or gate-first fabrication) [44].

1.2.5 Key-Based Classification

Hardware obfuscation techniques can also be classified on the basis of whether they employ a key-based locking mechanism or not.

1.2.5.1 Keyless

Keyless hardware obfuscation techniques ensure security by stripping away specific information regarding the design from the adversary. This includes techniques such as (i) white-box obfuscation for RTL (control flow graph obfuscation, removing comments, code compression, etc., to prevent reverse engineering); (ii) split fabrication (take away BEOL connectivity information from foundry so that they do not have the complete design); (iii) 2.5D/3D IC obfuscation (take away partition of a design, in the form of a separate die, from the foundry); (iv) instruction set obfuscation (make reverse engineering of code difficult); (v) IC camouflaging (make a gate incomprehensible for reverse engineers); and (vi) circuit edit (foundry is unable to find the added/modified gates or wires), all of which were discussed in detail in Sect. 1.2.4. While these techniques do not require key management, they usually have a more restrictive attack model (e.g., tightly controlled supply chain so that an attacker does not gain information about the design).

1.2.5.2 Key-Based

Key-based hardware obfuscation techniques include (i) IP encryption (via AES); (ii) RTL/FSM locking (correct sequence of state transitions required to unlock IP/IC); (iii) logic encryption (through embedding of locking key gates into design); and (iv) permutation block PCB locking. These techniques rely on the fact that the key used to unlock the design stays secret. Without the secret key, the design remains nonfunctional to an adversary. The idealistic assumption is that the adversary is reduced to using brute force in order to guess the correct secret key for the locked design. Assuming a large enough key length, such attacks become quickly unfeasible for most key-based obfuscation techniques (e.g., 2^{128} guesses for 128-bit key). However, several recent attacks have shown that an attacker can do much better than brute force in order to extract the secret key from an unlocked IC (and thereby use it to unlock other ICs) or check the correct key in far fewer tries than brute force. Some of these attacks are discussed below.

- *Boolean satisfiability (SAT) solvers*: Logic encryption techniques have recently been subjected to Boolean satisfiability (SAT) solver-based attacks [24]. In this attack, it is assumed that an adversary gains an unlocked version of the IC from the open market and uses this unlocked IC to query the correct input–output (IO) functionality of the design. Using this correct IO information and a separate locked IC implementing the same functionality, the attacker iteratively tries to rule out multiple incorrect key values. This is done by inputting the circuit in conjunctive normal form (CNF) to a SAT solver. Using the solver, the attacker identifies distinguishing input pairs (DIPs) that can help him or her to rule out multiple wrong keys in a single guess. This drastically reduces the key search space, enabling the attacker to deduce the right keys in a very reasonable amount of time.

- *Key propagation attacks*: The authors in [45] propose an automatic test pattern generation (ATPG)-based attack on logic encryption. In this approach, test patterns are generated, which can selectively 'mute' the effect of other gates in a circuit (e.g., by applying non-controlling values, such as logic 0 to an OR gate) and cause the key value on the key gates to propagate all the way to the primary outputs. Thus, through a topological evaluation of the circuit and generation of appropriate test patterns, the attacker can use to obtain the secret key and use it to unlock other ICs.

More details regarding both of these attacks can be found in the following chapter on logic encryption. The chapter also highlights appropriate countermeasures to these attacks, such as interference graph-based logic encryption (allowing key gate insertion at locations which do not allow propagation of the key bits to the output) and SARLock (utilizing the output of one-way random functions such as AES to set the key bits; this prevents an attacker from correlating the AES key to the circuit outputs, making SAT attacks provably unfeasible). However, note that both of these countermeasures are ad hoc in nature, thereby requiring large area overhead.

Besides these attacks based on intelligently stealing the key, key-based HW obfuscation techniques also suffer from an array of issues arising due to key management. Most of the obfuscation techniques rely on the key being stored in some form of non-volatile memory, which themselves are prone to imaging and probing attacks [46]. Further, key management (i.e., exchange of keys through a network allowing remote activation and authentication) is a non-trivial requirement, requiring area overhead for key exchange circuitry (e.g., RSA) and a secure network resistant to man-in-the-middle and replay attacks. An attempt at secure key exchange for remote authentication/activation of chips is presented in [47]. In the proposed FORTIS approach, RSA is utilized for end-point authentication between the chip at the foundry and the design house, and one-time pad (OTP) symmetric encryption is used for protecting the keys during exchange. A detailed treatment of the entire protocol can be found in the chapter on FORTIS in this book.

1.3 Software Obfuscation

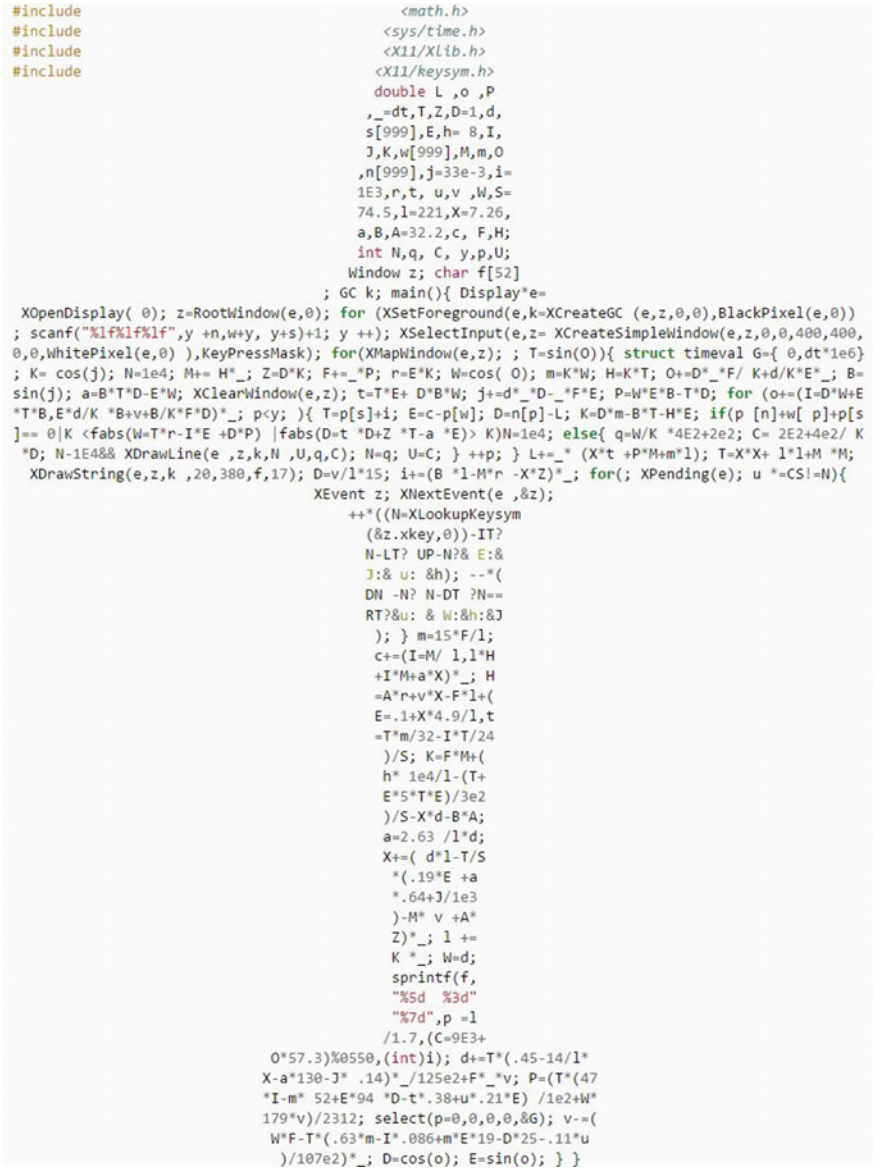
Computer software is the field that the term 'obfuscation' is most commonly associated with. While software obfuscation is beyond the scope of this book, we briefly introduce the field and highlight some reasons for utilizing it below.

Informally, software obfuscation relates to the practice of programmers concealing the implementation of their algorithm in code. This can include techniques ranging from simple comment or white space removal to more elaborate techniques such as loop unrolling (from the control flow graph or CFG representation of the program) [48]. While the overarching goal of obfuscation is IP protection (through prevention of reverse engineering), more specific reasons for considering obfuscation include the following.

- *Protection against malicious intent*: Computer vulnerabilities such as malware, virus, and Trojan horses often require the adversary to have a complete white box understanding of the software system they are targeting. For example, a buffer overflow attack requires the attacker to exactly know the instruction set architecture (ISA) of the victim's host system. Obfuscation techniques such as address space randomization [49] can be effective in preventing such attacks. Also, a software company, on discovering a bug in their software, could obfuscate their program with a patch to the bug and release it. This could be done so that an adversary is unable to recover the original bug from the patched program, so that he/she is prevented from exploiting customers of the software without the patch. Conversely, a malware could also employ obfuscation on itself in order to prevent detection by antivirus or intrusion detection packages [50].
- *Protection against IP theft and misuse*: One of the strongest reasons for obfuscation, in terms of IP protection, is to prevent IP theft and misuse. The software piracy cases that we discussed in Sect. 1.1 show the importance of strong obfuscation and why measures need to be taken to actively protect software IP from competitors and even potential customers.
- *Code Minification*: Apart from IP protection, obfuscation is also routinely used in the software industry in order to perform code compaction. Popular tools such as ProGuard [51] work with specific programming languages such as Java in order to simultaneously obfuscate and minimize code by removing unnecessary classes, shortening variable names, etc. This helps to produce executables that are compact in size and ease memory/storage constraints. Note that such 'minification' (and software obfuscation, in general) does not affect the functionality of the original code.
- *Recreational Obfuscation*: On a lighter note, competitions such as the international obfuscated C code contest (IOCCC) encourage participants to reproduce a code or algorithm in the most artistic or esoteric way possible [52] (Fig. 1.8).

1.3.1 Metrics for Hardware Obfuscation

Although we discussed the various techniques used for hardware obfuscation at different levels of abstraction, it is also important to note the metrics that go into (i) implementing and (ii) evaluating these techniques. *Implementation* metrics, which can be used for performing 'good' obfuscation, have two key requirements. First, they should be reasonably fast to compute (e.g., linear complexity) so that they can still be practical when applied to large circuits. Second, they should ideally be able to incorporate overhead (area, power, delay, etc.) and security constraints as the obfuscation technique is iteratively applied to the circuit. In this way, the designer does not have to wait till the entire obfuscation procedure is complete in order to evaluate the resulting security and overheads.



```
#include                                    <math.h>
#include                                   <sys/time.h>
#include                                   <X11/Xlib.h>
#include                                  <X11/keysym.h>
                                          double L ,o ,P
                                         ,_=dt,T,Z,D=1,d,
                                         s[999],E,h= 8,I,
                                         J,K,w[999],M,m,O
                                        ,n[999],j=33e-3,i=
                                       1E3,r,t, u,v ,W,S=
                                       74.5,l=221,X=7.26,
                                       a,B,A=32.2,c, F,H;
                                       int N,q, C, y,p,U;
                                     Window z; char f[52]
                                  ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                  XEvent z; XNextEvent(e ,&z);
                                      ++*((N=XLookupKeysym
                                        (&z.xkey,0))-IT?
                                        N-LT? UP-N?& E:&
                                        J:& u: &h); --*(
                                        DN -N? N-DT ?N==
                                        RT?&u: & W:&h:&J
                                         ); } m=15*F/l;
                                         c+=(I=M/ l,l*H
                                         +I*M+a*X)*_; H
                                         =A*r+v*X-F*l+(
                                         E=.1+X*4.9/l,t
                                         =T*m/32-I*T/24
                                          )/S; K=F*M+(
                                          h* 1e4/l-(T+
                                          E*5*T*E)/3e2
                                          )/S-X*d-B*A;
                                          a=2.63 /l*d;
                                          X+=( d*l-T/S
                                           *(.19*E +a
                                           *.64+J/1e3
                                           )-M* v +A*
                                           Z)*_; l +=
                                           K *_; W=d;
                                           sprintf(f,
                                           "%5d  %3d"
                                           "%7d",p =l
                                          /1.7,(C=9E3+
                                  O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                                 X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                                 *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                                 179*v)/2312; select(p=0,0,0,0,&G); v-=(
                                  W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                                   )/107e2)*_; D=cos(o); E=sin(o); } }
```

Fig. 1.8 IOCCC Flight Simulator: Winning entry of the 1998 International Obfuscated C Code Contest [53]

Evaluation metrics can be grouped into two categories: preobfuscation and post-obfuscation. *Preobfuscation* evaluation metrics can be used to judge the difficulty in obfuscating a circuit before the actual technique is applied to it. These kinds of metrics help the designer to gauge the effort required (e.g., overheads that might

need to be committed and computation time) in order to obfuscate the circuit before performing the actual obfuscation. They could also help in deciding from an array of candidate obfuscation techniques. *Post-obfuscation* evaluation metrics are applied to a circuit after the obfuscation is complete. In order to use these metrics, a golden circuit (i.e., the unobfuscated circuit) is required which is then compared to the obfuscated version in order to evaluate the resultant security and overheads.

Some of the metrics frequently which recur across different obfuscation techniques and can be used for implementation and/or evaluation are briefly discussed below.

- Implementation Metrics
  - *Fault Impact*: The authors in [23] have used fault impact in order to judge the appropriate locations to insert key gates into a circuit. The fault impact for a gate can be expressed as the product of the total number of test patterns that detect a stuck-at-0 fault at the gate's output and the number of outputs that get affected by the stuck-at-0 fault at the gate's output (the total fault impact is the sum of stuck-at-0 fault impact and stuck-at-1 fault impact). This metric roughly tells how likely it is that a fault (which, in this case, is the bit-flip induced by applying a wrong key at the key gate) can propagate to the outputs and cause an output error. The shortcoming of this approach is that fault detection is a computationally hard problem, and generating the patterns for detecting the faults would require exponential run-time as the circuit size scales. The authors in [33] have utilized a similar metric, which they term as mean observe and control values (MOV/MCV). As opposed to a stuck-at fault detection approach which was used by fault impact, MOV/MCV tracks the number of bit-flips in a wire and the corresponding number of bit-flips of the outputs in output cone (for MOV) or inputs in the input cone (for MCV) of the same wire. MOV/MCV has the same linear computational complexity as logic simulation.
  - *k-security*: The authors in [32] introduce the notion of k-security for indistinguishability of a gate in a circuit fabricated by split manufacturing. Given that an attacker has the complete netlist of the design and a partial netlist recovered from the FEOL layer, the authors claim that a design is k-secure if for every gate in the FEOL design, there exists at least k subgraph isomorphisms, i.e., k distinct gates in the complete netlist that the gate can be mapped to. For example, if a gate in the FEOL netlist has $k = 2$, it implies that the attacker has to randomly guess between two gates in the original netlist in order to identify the gate in the FEOL netlist. Unfortunately, the authors showed that determining whether a circuit is $k - secure$ is NP-complete. Therefore, they employed a SAT solver to gradually lift wires from the FEOL to the BEOL and heuristically check graph isomorphism after each lifting operation to construct a k-secure circuit. This also unfortunately leads to impractical area overheads.

- Evaluation Metrics
  - *Hamming Distance*: Hamming distance (HD) is a metric used to evaluate the difference between two given bitstreams b_1 and b_2. It performs a bit-by-bit comparison of two bitstreams and uses a percentage figure to describe how many bits are different between b_1 and b_2. For circuits, the output bits of a combinational circuit (or a sequential circuit at the same time instance) can be thought of as a bitstream. An average HD of 50% between b_1, the output of a circuit and b_2, the output of its obfuscated or locked counterpart, tells us that the responses between the two circuits can be the same only with a probability that is as good as random chance. In other words, the obfuscated circuit's response is completely different than that of the original circuit. This metric has been widely used for evaluating gate-level obfuscation techniques such as logic locking [23] and also camouflaging [35]. It is either calculated once post-obfuscation or recursively calculated after each change (e.g., after one key or camouflaged gate insertion). The drawback of this metric is that it is based on outputs and requires logic simulation which does not scale well with the number of inputs and size of the circuit. Most techniques employing HD estimate the figure by a random sampling of the input space (e.g., 1000 randomly selected input vectors).
  - *Verification Failure*: The authors in [25, 42] have used percentage verification failure as a metric for evaluating the performance of their obfuscation techniques. To calculate this metric, formal verification tools such as Synopsys Formality are used to perform logical equivalence checking between the obfuscated design and its original counterpart. The equivalence checking involves the use of proprietary static analysis techniques on the logic cones of the two designs to compare their output ports and flip-flop outputs (pseudo-output ports). The final verification failure figure is expressed as a percentage of failing comparison points (ports that failed equivalence checking) to the total number of comparison points (total no. of ports). The advantage of this metric over simulation-based Hamming distance is that it is much faster and scalable and does not suffer from the inaccuracies or coverage issues that arise due to simulation with a limited set of vectors/patterns. However, for purely combinational circuits, the metric might not be applicable due to the increased size of the logic cones (extending from the primary inputs all the way to the primary outputs, without any flip-flops in between). Note that verification failure can only be used as a post-obfuscation evaluation metric, as it requires the complete obfuscated circuit and the original circuit.
  - *Entropy*: Entropy refers to the amount of information contained in a system. In terms of obfuscation, entropy is used to determine the extent of information that can be non-trivially attained by an adversary by observing the obfuscated version of the circuit. The authors in [54] have used entropy as a measure of how easy it is for the attacker to gain information about the functionality of the circuit from the distribution of gate types. For example, a circuit synthesized with only two types of gates will have a very high entropy compared to a circuit synthesized with 30 different types of gates, from which the attacker might deduce clues

(e.g., a collection of XOR gates might hint to the addRoundKey stage of AES). Along the same lines, the authors also proposed a complimentary metric they term as 'standard cell composition bias.' This metric analyzes the proportion of standard cells (such as XOR, flip-flops, or arithmetic gates) in the design. A design with high bias (e.g., with a lot of XORs and few FFs/arithmetic gates) might indicate a cryptographic core, while a design with significantly more FFs might indicate a state machine logic. The goal is to synthesize the design with low bias, i.e., with equal proportion of different types of standard cells so that the attacker cannot make a generalized guess about the high-level functionality or purpose of the circuit. Both entropy and composition bias can be used as preobfuscation and post-obfuscation evaluation metrics.

– *Neighbor connectedness*: The authors in [54] introduce neighbor connectedness, which gives us an idea of how connected a cell in a design (in layout form, with respect to split manufacturing) is. If a cell is connected to a lot of other cells in its neighborhood (e.g., a 4×4 grid within a small radius R), its functionality/purpose could be deduced from the connectivity information. On the flip side, if connected cells are more 'spread out' (i.e., R is increased), an attacker without BEOL information could wrongly assume that a cell is connected to another functionally unrelated cell. Therefore, a design with low neighbor connectedness (i.e., where connected cells are far apart) would increase the reverse engineering required by the adversary (as he or she would keep making wrong connections based on distance (mis)information). Note that low neighbor connectedness also unfortunately implies an increase in wire length. This metric can be used for both preobfuscation and post-obfuscation evaluation.

Thus, there are an array of metrics that have been proposed in order to implement as well as evaluate hardware obfuscation techniques. The key issue with most of the proposed metrics is that they are impractical, either in terms of computing the metric itself or implementing the design guided by the metric. For example, evaluation metrics such as HD require logic simulation, which are unscalable on large designs, have significant errors when estimated with a small set of random vectors, and cannot be calculated as is by partitioning the design (as we discussed above). On the other hand, metrics such as k-security end up being too strong and an unfeasible notion of security, as is evident by the unacceptable area overhead that arises in trying to meet the metric. Thus, there exists a delicate trade-off between the computational complexity involved in metric computation and the overhead that results from adopting a particular metric to guide obfuscation.

1.3.1.1 Software Obfuscation Metrics

Although an in-depth treatment of software obfuscation is beyond the scope of this chapter/book, a few relevant metrics for software obfuscation, which indicate the level of complexity in reverse engineering or comprehending a program, are highlighted below.

- *Cyclomatic complexity* relates to the control flow graph (CFG) representation of a program. A program without any control statements (e.g., *IF*) would be assigned a complexity of 1, whereas a program with one *IF* statement and one evaluation condition would be assigned a complexity of 2 (one part evaluating to *TRUE* and another evaluating to *FALSE*, for a total of two linearly independent paths in the CFG). The complexity increases as more control flow statements are introduced in the program, indicating an increase in the test cases required to comprehend the program functionality.
- *Halstead complexity metric* defines a suite of measures such as program length, difficulty, and effort, which are all based on the number of distinct operators and operands that are utilized in the program [55]. Since this metric solely relies on the operators + operands, it is language-independent and has no notion of control flow. Nonetheless, the metric directly correlates to program execution time and amount of time a reverse engineer has to spend to evaluate the program.

1.3.2 Hardware Obfuscation Benchmarks

In order to show the efficacy of their obfuscation techniques, researchers frequently utilize 'benchmark circuits' on which they apply the technique and present results on the incurred area/delay/power overhead and security metric utilized. For gate-level techniques, popular examples of used benchmarks include the ISCAS '85 [56], '89 [57], and ITC '99 [58] benchmark sets (which are widely available in synthesized netlist form). Researchers have also utilized the placed-and-routed layout of these benchmarks to explore split-manufacturing obfuscation. These benchmarks were initially created as example designs (to be used as functional black boxes) for researchers to explore test pattern generation, scan chain insertion, fault coverage, and other VLSI test-related topics. As a result, their use in HW obfuscation has been more or less ad hoc. Moreover, most of these benchmarks date back decades and are only a few thousands in gate count. Due to these reasons, it becomes hard to argue about the scalability of HW obfuscation techniques implemented on these benchmarks to the million gate designs that are commonplace today.

1.4 Conclusion

In this chapter, we presented a general overview of hardware as well as software obfuscation. Software obfuscation focuses on developing general-purpose obfuscating compilers that can work for all programs. However, hardware obfuscation can vary in technique and scope, depending on the threat model and level of abstraction. Due to this, there cannot be a one-size-fits-all solution for all hardware obfuscation problems.

We also explored the various threats involved in each stage of the integrated circuit supply chain and presented a brief review of obfuscation techniques that have been developed to counter these threats. Lastly, we reviewed relevant metrics for HW obfuscation that the reader will encounter throughout the rest of this book.

References

1. Weber J (2016) Epic systems wins $940 mln US jury verdict in Tata trade secret case, reuters. http://www.reuters.com/article/us-tata-epic-verdict-idUSKCN0XD135. Accessed April 2016
2. Kirk J (2013) Three indicted in alleged source code theft from trading house, PC world. http://www.pcworld.com/article/2053020/three-indicted-in-alleged-source-code-theft-from-trading-house.html. Accessed Oct 2013
3. Rosenblatt J (2013) Xilinx sues Flextronics alleging fradulent chip resale, bloomberg technology. http://www.bloomberg.com/news/articles/2013-12-11/xilinx-sues-flextronics-alleging-fraudulent-chip-resale. Accessed Dec 2013
4. Bunkley N (2015) Ford accused by software maker of intellectual property theft, automotive news. http://www.autonews.com/article/20150604/OEM06/150609919/ford-accused-by-software-maker-of-intellectual-property-theft. Accessed June 2015
5. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K (2001) On the (im) possibility of obfuscating programs. Annual international cryptology conference. Springer, Heidelberg, pp 1–18
6. Canetti R, Dakdouk RR (2008) Obfuscating point functions with multibit output. Annual international conference on the theory and applications of cryptographic techniques. Springer, Heidelberg, pp 489–508
7. Goldwasser S, Rothblum GN (2007) On best-possible obfuscation. Theory of cryptography conference. Springer, Heidelberg, pp 194–213
8. Garg S, Gentry C, Halevi S, Raykova M, Sahai A, Waters B (2013) Candidate indistinguishability obfuscation and functional encryption for all circuits. In: IEEE 54th annual symposium on foundations of computer science (FOCS). IEEE, pp 40–49
9. Apon D, Huang Y, Katz J, Malozemoff AJ (2014) Implementing cryptographic program obfuscation. IACR Cryptol ePrint Arch 2014:779
10. Torrance R, James D (2009) The state-of-the-art in ic reverse engineering. In: Cryptographic Hardware and Embedded Systems-CHES. Springer, pp 363–381
11. Quadir SE, Chen J, Forte D, Asadizanjani N, Shahbazmohamadi S, Wang L, Chandy J, Tehranipoor M (2016) A survey on chip to system reverse engineering. ACM J Emerg Technol Comput Syst (JETC) 13(1):6
12. Asadizanjani N, Shahbazmohamadi S, Tehranipoor M, Forte D (2015) Non-destructive PCB reverse engineering using x-ray micro computed tomography. In: 41st International symposium for testing and failure analysis, ASM, 1–5 November 2015
13. Moradi A, Barenghi A, Kasper T, Paar C (2011) On the vulnerability of fpga bitstream encryption against power analysis attacks: extracting keys from xilinx virtex-ii fpgas. In: Proceedings of the 18th ACM conference on Computer and communications security, pp. 111–124. ACM, 2011
14. Note JB, Rannaud E (2008) From the bitstream to the netlist. In: Proceedings of the 16th international ACM/SIGDA symposium on field programmable gate arrays, series FPGA 2008, New York, USA. ACM, pp 264–264. http://doi.acm.org/10.1145/1344671.1344729
15. Becker GT, Regazzoni F, Paar C, Burleson WP (2013) Stealthy dopant-level hardware trojans. In: International workshop on cryptographic hardware and embedded systems. Springer, pp 197–214
16. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25

17. IEEE computer society, IEEE recommended practice for encryption and management of electronic design intellectual property. https://standards.ieee.org/findstds/standard/1735-2014.html. Accessed December 2014
18. Chakraborty RS, Bhunia S (2010) RTL hardware IP protection using key-based control and data flow obfuscation. In: 2010 23rd international conference on VLSI design. IEEE, pp 405–410
19. Desai AR, Hsiao MS, Wang C, Nazhandali L, Hall S (2013) Interlocking obfuscation for anti-tamper hardware. In: Proceedings of the eighth annual cyber security and information intelligence research workshop. ACM, p 8
20. Brzozowski, M, Yarmolik VN (2007) Obfuscation as intellectual rights protection in VHDL language. In: 6th International conference on computer information systems and industrial management applications, CISIM 2007. IEEE, pp 337–340
21. Kainth M, Krishnan L, Narayana C, Virupaksha SG, Tessier R (2015) Hardware-assisted code obfuscation for FPGA soft microprocessors. In: Proceedings of the 2015 design, automation and test in Europe conference and exhibition. EDA Consortium, pp 127–132
22. Roy JA, Koushanfar F, Markov IL (2008) Epic: ending piracy of integrated circuits. In: Proceedings of the conference on design, automation and test in Europe. ACM, pp 1069–1074
23. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Logic encryption: a fault analysis perspective. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, pp 953–958
24. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: IEEE international symposium on hardware oriented security and trust (HOST) (2015). IEEE, pp 137–143
25. Chakraborty RS, Bhunia S (2009) Harpoon: an obfuscation-based soc design methodology for hardware protection. IEEE Trans Comput Aided Des Integr Circuits Syst 28(10):1493–1502
26. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans Inf Forensics Secur 7(1):51–63
27. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFTS). IEEE, pp 196–203
28. Yang B, Wu K, Karri R (2004) Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: Proceedings of the ITC international test conference on 2004. IEEE, pp 339–344
29. Nahiyan A, Xiao K, Yang K, Jin Y, Forte D, Tehranipoor M (2016) AVFSM: a framework for identifying and mitigating vulnerabilities in FSMS. In: Proceedings of the 53rd annual design automation conference. ACM, p 89
30. Lee J, Tehranipoor M, Patel C, Plusquellic J (2007) Securing designs against scan-based side-channel attacks. IEEE Trans Dependable Secure Comput 4(4):325–336
31. Vaidyanathan K, Liu R, Sumbul E, Zhu Q, Franchetti F, Pileggi L (2014) Efficient and secure intellectual property (IP) design with split fabrication. In: IEEE international symposium on hardware-oriented security and trust (HOST) 2014. IEEE, pp 13–18
32. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In: Presented as part of the 22nd USENIX security symposium (USENIX security 2013), pp 495–510
33. Xie Y, Bao C, Srivastava A (2015) Security-aware design flow for 2.5D IC technology. In: Proceedings of the 5th international workshop on trustworthy embedded devices. ACM, pp 31–38
34. Rajendran JJ, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, pp 1259–1264
35. Rajendran J, Sam M, Sinanoglu O, Karri R (2013) Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security. ACM, pp 709–720
36. El Massad M, Garg S, Tripunitara MV (2015) Integrated circuit (IC) decamouflaging: reverse engineering camouflaged ICS within minutes. In: NDSS

37. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ICS using split fabrication. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 1–6, May 2014
38. Barrantes EG, Ackley DH, Palmer TS, Stefanovic D, Zovi DD (2003) Randomized instruction set emulation to disrupt binary code injection attacks. In: Proceedings of the 10th ACM conference on Computer and communications security. ACM, pp 281–289
39. Kc GS, Keromytis AD, Prevelakis V (2003) Countering code-injection attacks with instruction-set randomization. In: Proceedings of the 10th ACM conference on computer and communications security. ACM, pp 272–280
40. Linn C, Debray S (2003) Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the 10th ACM conference on computer and communications security. ACM, pp 290–299
41. Guo Z, Tehranipoor M, Forte D, Di J (2015) Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In: Proceedings of the 52nd annual design automation conference, series DAC 2015, New York, NY, USA. ACM, pp 114:1–114:6. http://doi.acm.org/10.1145/2744769.2744862
42. Shakya B, Asadizanjani N, Forte D, Tehranipoor M (2016) Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In: IEEE/ACM international conference on computer-aided design (ICCAD)
43. Bi Y, Shamsi K, Yuan J-S, Gaillardon P-E, Micheli GD, Yin X, Hu XS, Niemier M, Jin Y (2016) Emerging technology-based design of primitives for hardware security. ACM J Emerg Technol Comput Syst (JETC) 13(1):3
44. Cui Y, Zhong Z, Wang D, Wang WU, Lieber CM (2003) High performance silicon nanowire field effect transistors. Nano Lett 3(2):149–152
45. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64(2):410–424
46. Skorobogatov SP (2005) Semi-invasive attacks: a new approach to hardware security analysis, Ph.D. dissertation, Citeseer
47. Guin U, Shi Q, Forte D, Tehranipoor MM (2016) Fortis: a comprehensive solution for establishing forward trust for protecting IPS and ICS. ACM Trans. Des. Autom. Electron. Syst., 21(4):63:1–63:20. http://doi.acm.org/10.1145/2893183
48. Collberg C, Thomborson C, Low D (1997) A taxonomy of obfuscating transformations. The University of Auckland, New Zealand, Technical report, Department of Computer Science
49. Bhatkar S, DuVarney DC, Sekar R (2003) Address obfuscation: an efficient approach to combat a broad range of memory error exploits. Usenix Secur 3:105–120
50. You I, Yim K (2010) Malware obfuscation techniques: a brief survey. In: 2010 international conference on broadband, wireless computing, communication and applications (BWCCA), pp 297–300
51. Lafortune E et al. (2004) Proguard. http://proguard.sourceforge.net
52. Noll LC, Cooper S, Seebach P, Leonid AB (2005) The international obfuscated C code contest
53. IOCCC, IOCCC flight simulator. In: International obfuscated C code contest (1998). http://www.ioccc.org/1998/banks.c
54. Jagasivamani M, Gadfort P, Sika M, Bajura M, Fritze M (2014) Split-fabrication obfuscation: metrics and techniques. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 7–12
55. Halstead MH Elements of software science, vol 7
56. Hansen MC, Yalcin H, Hayes JP (1999) Unveiling the iscas-85 benchmarks: a case study in reverse engineering. IEEE Desi Test 16(3):72–80
57. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In: IEEE international symposium on circuits and systems. IEEE, pp 1929–1934
58. Corno F, Reorda MS, Squillero G (2000) RT-level ITC 1999 benchmarks and first ATPG results. Ieee Des Test Comput 17(3):44–53

Chapter 2
VLSI Test and Hardware Security Background for Hardware Obfuscation

Fareena Saqib and Jim Plusquellic

2.1 Introduction

Hardware obfuscation is a technique to conceal the design from malicious insider and outsider adversaries. Obfuscation techniques transform the original design such that the obfuscated version is functionally equivalent to the original design, but it does not reveal the design details and is much harder to reverse-engineer [1]. As discussed earlier in Chap. 1, the business model of distributed and outsourced design, integration, manufacturing, packaging, and distribution channels creates challenges such as intellectual property (IP) piracy, reverse engineering of the netlist from GDSII, integrated circuit (IC) cloning, and counterfeiting opportunities.

Nanometer-sized integrated circuit feature sizes and increased gate density per wafer have been made possible with the advancements in photolithography techniques. However, this has driven the cost and maintenance of fabrication facilities into the billions of dollars, making the business model difficult to justify and sustain. Consequently, many major companies have become fabless and instead outsource their designs to offshore foundries as a cost-effective alternative to owning and operating their own fabs. Unfortunately, the horizontal dissemination of the design process to companies all over the world decreases the trustworthiness and increases the security risks of the design process [2, 3].

This chapter overviews the traditional design flow of integrated circuits and assesses processes in terms of how much information is revealed to aid in reverse engineering the design. We survey the proposed schemes that are designed to enhance the security properties of traditional verification and testing mechanisms to make designs resilient to attacks. This chapter also investigates IP protection schemes that

F. Saqib (✉)
Florida Institute of Technology, Melbourne, FL, USA
e-mail: fsaqib@fit.edu

J. Plusquellic
University of New Mexico, Albuquerque, NM, USA

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_2

are designed to prevent illegal modifications and piracy for system-on-chip (SoC) IP reuse-based design flows. This problem is challenging because IP can be distributed as soft (RTL level), firm (netlist level), or hard (GDSII level) and is usually transparent at system design level, in manufacturing facility, and in the distribution chain, making it susceptible to security and privacy attacks. The objective of hardware obfuscation is to make it difficult for an adversary to reverse-engineer the functionality at any level of abstraction throughout the design process.

Threat Models:

IC piracy, cloning, counterfeiting, and sabotage have become major security concerns under the current business model of IP reuse and offshore manufacturing. The following provides a partial list of attack vectors open to an adversary:

(1) Reverse engineering: GDSII-to-netlist reverse engineering enables the adversary to steal and reproduce the IP.
(2) Clones: An attacker in the system design flow can steal the IP or IC and make exact clones or, with a few modifications, claim the ownership and make illegal copies.
(3) Overbuilding: Building more copies of the IC than requested by the customer is referred as overbuilding. Overbuilt ICs can be sold on the black market. Without specialized metering techniques, preventing overbuilding is a challenge.
(4) Counterfeit chips: Counterfeit chips are intended to deceptively represent an authentic component and can be created from recycled chips or from cloning [4].
(5) Trojan detection insertion: After reverse engineering the design, the adversary can insert hardware Trojans in a set of counterfeit clones. Hardware Trojans are hidden malicious circuits that can be designed to allow activation through backdoors during the fielded operation of the chip. Activation can involve leaking sensitive information or causing the chip and system to fail catastrophically.

This chapter is organized as follows: Sect. 2.2 introduces the VLSI verification and test concepts and discusses the vulnerabilities, attacks, and countermeasures. Section 2.3 describes the obfuscation techniques that can be integrated into the design flow to make the design more resilient to reverse engineering and summarizes the evaluation metrics for these techniques. Section 2.4 covers the review of nonvolatile memory and emerging technologies and discusses the associated vulnerabilities of key management on nonvolatile memories (NVMs). Section 2.5 introduces the hardware-based cryptographic primitives, physical unclonable functions (PUFs), and true random number generator (TRNG) and their use in hardware obfuscation techniques to improve the resilience against reverse engineering.

2.2 VLSI Verification and VLSI Test Concepts

Very large-scale integration (VLSI) verification is a presilicon procedure to verify the design before fabrication. Random test vectors and formal verification techniques are used to verify design behavior and coverage of generated test vectors. Satisfiability

(SAT) solvers are used in formal verification to find design issues presilicon. Several SAT solver algorithms have been integrated into the electronic design automation (EDA) tools.

In contrast, VLSI testing is applied post-silicon to ensure high quality and reliability in the shipped IC products and to find design problems that affect yield early. The increasing level of complexity and smaller geometries used in modern IC fabrication introduces new failure mechanisms that act to reduce the yield and the quality of shipped products. VLSI testing is critically important to screening defective products, with the ultimate goal of reaching zero defects. VLSI testing also provides important feedback for accelerating product yield ramps and has a direct impact on profitability.

2.2.1 Satisfiability (SAT) Problem

Satisfiability is defined as a condition of boolean expression evaluating to be true for a set of logical values of the variables. Outputs of combinational logic can be expressed as boolean expressions, constituting conjunctions (and) of disjunctive (or) clauses and variables in the form of conjunctive normal form (CNF).

An example of function in the CNF form is as follows:

$$F = (a \text{ v } b \text{ v } c) \wedge (a' \text{ v } c \text{ v } d) \wedge (b' \text{ v } d') \tag{2.1}$$

where a, b, c, and d are the variables that can be '1' or '0'.

The decision problem for SAT, to find a satisfying assignment that makes the function true, is a nondeterministic polynomial (NP) problem. For example, for n variables, 2^n boolean combinations of input variables are examined. Each SAT formula has a polynomial time verifier that takes an input string, a zero, or one assignment for all the variables and outputs a true or false evaluation for the provided inputs.

The SAT problem has exponential complexity in the worst case, but given the importance of the algorithm in CAD, researchers have developed many types of efficient heuristically SAT solvers that provide near-optimal solutions. These SAT solvers have many applications in the electronic design automation (EDA) in verification as well as in the synthesis. The SAT solver algorithms are categorized as conflict-driven clause learning and stochastic local search algorithms. These algorithms have been developed to automatically solve the instances/combinations with large number of variables and clauses. Recent work in the development of efficient SAT solvers includes GRASP [5], Satz [6], and Chaff [7]. These algorithms use SAT solvers in formal or semiformal verification methods.

SAT solvers can also be used by malicious attackers to circumvent logic encryption-based hardware obfuscation by applying SAT-based algorithms to derive the keys [8]. The technique utilizes the approach of iteratively applying input pat-

terns on a set of selected inputs and identifying distinguishing inputs, for which the functions become unsatisfiable. This approach of testing key combinations has proven to weaken the security of hardware obfuscation, and research has focused on countermeasures designed to instill worst-case (exponential) behavior in SAT algorithms.

2.2.2 Equivalence of Circuits

Equivalence checking is one approach to functional verification. Equivalence checking is performed at different stages of design flow to verify the functional equivalence of combinational and sequential logic. Equivalence checking takes two descriptions of the design that are structurally different and verifies whether their behavior is functionally equivalent. The designs are compared using formal methods and simulation techniques. Formal methods such as binary decision diagram (BDD) and SAT-based are applied to the compare points in both the reference design and the implemented design, and functional equivalence is verified using simulations. Traditional equivalence checking utilizes the logic cones by analyzing the compare points. Logic cones are generic attributes of digital circuits consisting of reconvergent segments that fan-in to a common output. The input and outputs of cones are connected to the primary inputs, registers, or primary outputs that are also referred as compare points.

The implemented design is verified to prove or disprove the functional equivalence once all the compare points are verified. Several commercial tools such as Cadence Conformal equivalence checker and Synopsys Formality are equivalence checking tools. These tools use the netlist generated from Genus (Cadence) or Design Compiler (Synopsys) synthesis generated netlist to compare with the RTL design description using mathematical models.

Additional research is needed that investigates the equivalence checking in the design flows that implement obfuscation techniques. Equivalence checking of the reference design with the obfuscated netlists shown in Fig. 2.1 is further discussed in another chapter.

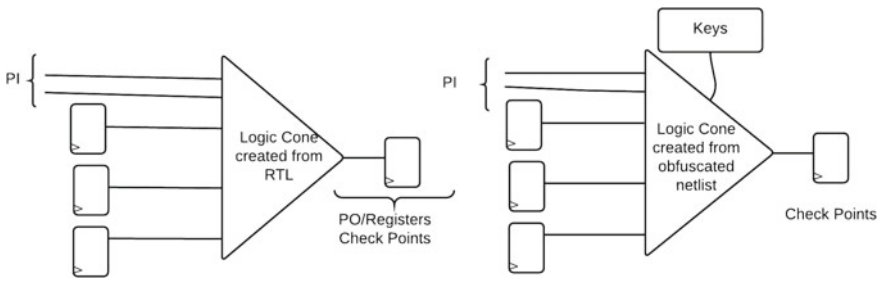


Fig. 2.1 Logic equivalence of reference and obfuscated design

2.2.3 Types of Testing: Functional Testing and Structural Testing

2.2.3.1 Functional Testing

Functional test verifies that the chip performs the correct operations, i.e., can the chip run the Windows operating system or carry out a matrix inverse software operation. ATPG can alternatively and/or additionally be used to generate functional test vectors. For example, vectors can be generated to test the 'critical paths,' which are the longest paths in the chip, and test the chip under worst-case power conditions. Therefore, the roles of functional testing also include timing and power verification.

2.2.3.2 Structural Testing

Structural testing refers to techniques that are based on fault models as discussed above. The goal is to check the integrity of the structural characteristics of the chip, i.e., its individual wires and logic gate functions. SSF tests verify that circuit nodes are not shorted to VDD or VSS, while transition and path delay tests verify that the logic gates and selected paths are able to propagate transitions to capture points (flip-flops and POs) using the functional ('at-speed') clock frequency. As indicated, ATPG is used to derive the test vectors and automatic test equipment (ATE) is used to apply them. Note that both functional testing and structural testing are constrained by the economics of testing, i.e., a great deal of effort is made to determine the smallest set of test vectors that meets the coverage requirements. This is true because manufacturing tests are applied to every chip, and therefore, to be economical, the test time per chip must be as small as possible.

2.2.4 Fault Modeling

Physical defects can occur in the IC during the manufacturing process, such as interconnect defects or packaging defects, gate–oxide shorts, metal trace bridges, open vias, and shorts to power or ground. Fault modeling is a mechanism to abstract and simplify all the possible ways defects can cause a malfunction in a chip. The most common fault models are single stuck-at fault (SSF), and transition and path delay fault models. The SSF models have also been proposed as a mechanism to strengthen hardware obfuscation techniques, as discussed below. The SSF model represents each defect as a single gate-level pin or net shorted to VDD or VSS. The terms 'stuck-at-1 (SA1)' and 'stuck-at-0 (SA0)' are used to represent these conditions. The SSF model assumes that only one fault (or no fault) exists in each chip. Figure 2.2a shows 6 gate stuck-at faults for a two-input NAND gate, and Fig. 2.2b shows one instance of SA0 fault in combinational logic.

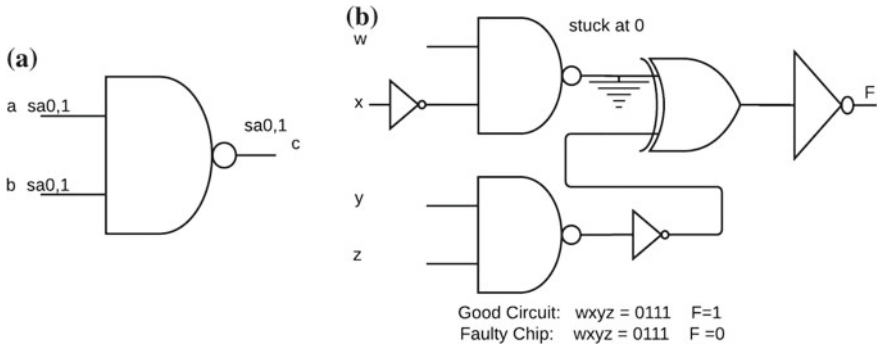


Fig. 2.2 **a** NAND gate inputs and output stuck-at model. **b** Combinational stuck-at fault

A combinational SSF is detected by determining the primary inputs (PIs) input assignments that introduce the appropriate state on the target gate inputs while simultaneously ensuring that the target gate output is observable on one or more primary outputs (POs). A stuck-at fault test determines whether the target node is SA0 or SA1 but also implicitly tests all gate inputs along the path for fault conditions.

The SSF model verifies the structural integrity and truth table description of combinational logic but does not verify whether the chip meets its timing specification. Separate fault models and sets of test vector sequences are required to verify timing. The transition and delay fault models target timing-related defects that cause logic transitions to take longer than expected to propagate through gates and along paths in the chip, i.e., conditions that cause the chip to violate timing constraints. Figure 2.3 shows the examples of delay faults. Defects that affect the drive strength of the gate, transistor doping levels, metal capacitive loading, open vias, and/or resistive gate–oxide shorts can introduce transition and delay faults in the chip.

The delay fault can be represented as a single gate fault, interconnect fault, or path delay fault. A single gate delay fault models the defects that affect gate strength, transistor doping, etc., i.e., anything that causes the timing of the input value at a pin to be slow-to-rise or slow-to-fall. Interconnect delay faults model defects that introduce variations in wire width or cause signal degradation because of resistive shorts to other nodes. Path delay faults model distributed defects, i.e., defects that effect the delay of the entire path. Delay tests are timed two-vector sequences (unlike SSF tests) that are applied at a constant rate using the clock. Such tests are critical for ensuring quality in modern nanometer-sized technologies.

2.2.5 Fault Coverage

Testing methods are evaluated in terms of fault coverage, where it is represented as follows:

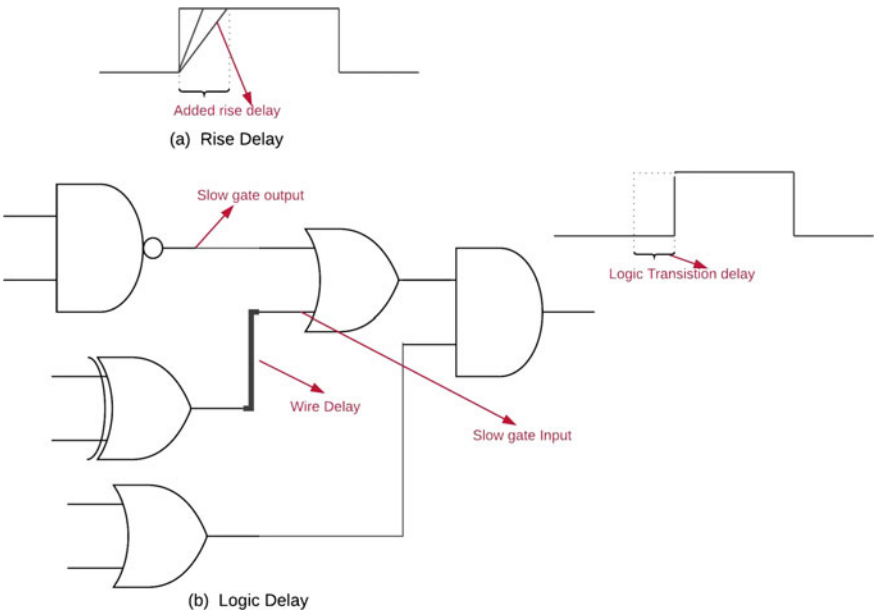


Fig. 2.3 Delay fault model

$$\text{Fault coverage} = \frac{\text{Total detected faults}}{\text{Total fault population}} \tag{2.2}$$

Fault coverage refers to fraction of faults under a given fault model that are covered by the test patterns. Fault coverage is typically computed and reported by automatic test pattern generation (ATPG) tools as these tools derive test patterns to test the faults. Ideally, the coverage should be 100%. Unfortunately, deriving test patterns is an NP-complete problem, and therefore, ATPG algorithms employ heuristics which, in many cases, are not able to find tests for all of the faults. Despite this limitation, test pattern generation using the SSF fault model is able to achieve high levels of coverage, typically 95–99%. It should be noted that fault coverage can be reported differently by different ATPG tools. For example, some tools eliminate the untestable faults from the fault population before applying the equation given above, while others do not. Fault coverage can also be used to identify difficult-to-test nodes and can therefore serve as a basis to guide design-for-testability (DFT) strategies.

2.2.6 Automatic Test Pattern Generation (ATPG)

As indicated above, ATPG is CAD software tool that automatically derives a set of test patterns for a specified list of faults using heuristic algorithms. The fault

model defines the nodes and/or paths in the chip that are the targets of ATPG. ATPG algorithms automatically derive a fault list from the netlist and fault model given as inputs. With the fault list available, a long, incremental process is started in which tests are derived that detect the faults. The faults detected by a test pattern are checked off in the fault list, and fault simulation is typically run to determine other faults that are 'accidentally' detected by the test pattern. Commercial vendors provide a variety of different runtime options and support for a fixed set of fault models, including SSF and transition and path delay fault models. ATPG and fault models can be leveraged in hardware obfuscation algorithms to produce strong keys, as discussed in Sect. 2.3. Figure 2.4 shows a typical ATPG flow.

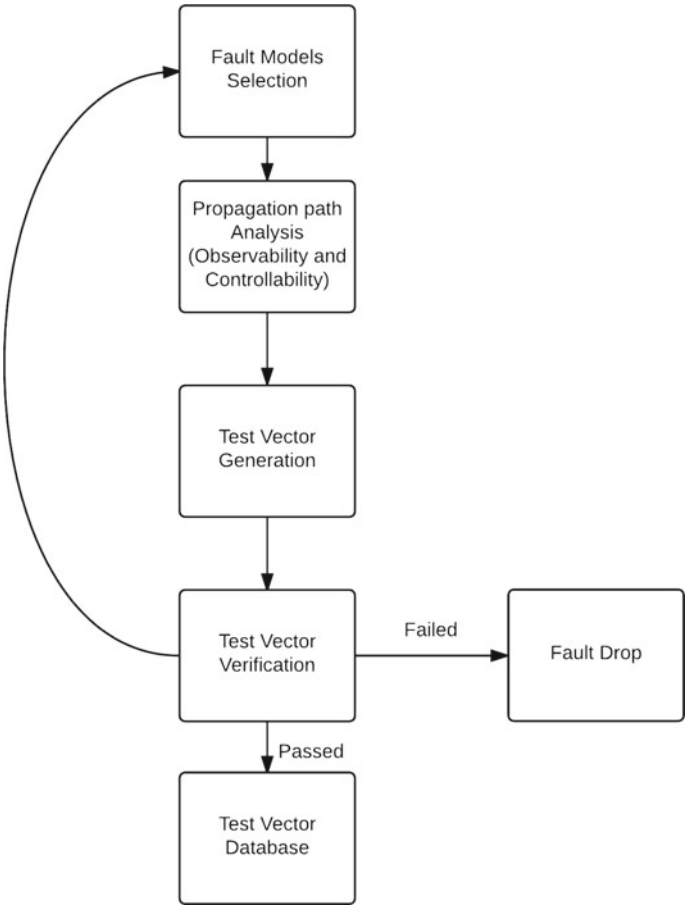


Fig. 2.4 ATPG flow

2.2.7 Testing Metrics: Controllability and Observability

As indicated above, test pattern generation is an NP-complete problem. As is true of many NP-complete problems, the task of generating a test pattern is doable in polynomial time for most of the faults in the fault list. Unfortunately, there are typically a small set of faults that elicit worst-case (exponential) time behavior in the ATPG algorithm. In response to this issue, the manufacturing test community developed a new set of algorithms that compute metrics for each of the faults in advance of ATPG that reflect the likely level of difficulty in generating test patterns for the faults. The metrics are probabilistic measures called controllability and observability. It should be noted that these algorithms also address an NP-complete problem and, like ATPG algorithms, employ heuristics. Unlike ATPG which can fail to find a test pattern for hard-to-test nodes, the heuristics used in algorithms that compute controllability and observability may produce inaccurate information for cases in which the task falls into a worst-case scenario.

Algorithms that compute controllability and observability (first coined by Rutman in 1972) produce numerical estimates regarding the difficulty of setting an internal node to a specific logic value and making an internal node observable on an output of the circuit. Several approaches to computing these testability measures have been proposed including SCOAP [9, 10], CAMELOT [11], TMEAS [12], COP [13], and PREDICT [14]. Testability analysis involves circuit topological analysis. For example, SCOAP traces through the design description and assigns controllability and observability weights to the nodes designated using the following six labels:

1. Combinational 0-controllability CC0(sig),
2. Combinational 1-controllability CC1(sig),
3. Combinational observability CO(sig),
4. Sequential 0-controllability SC0(sig),
5. Sequential 1-controllability SC1(sig), and
6. Sequential observability SO(sig).

The primary inputs (PI-sig) are all set to 1 for both combinational and sequential ‘0’ and ‘1’ controllabilities, i.e., CC0(PI-sig) = 1, CC1(PI-sig) = 1, SC0(PI-sig) = 1, and SC1(PI-sig) = 1. Combinational 0 and 1 controllabilities of internal nodes are calculated as the minimum number of combinational node assignments needed to justify a ‘0’ or ‘1’ on the output of a gate driving the node. Sequential 0 and 1 controllabilities on the other hand estimate the minimum number of sequential nodes that must be specified to set the internal node to ‘0’ or ‘1’. Starting from the primary inputs to primary outputs, node weights are computed such that the circuit depth of the node is factored into the combinational controllability equations. The following rules are used to compute the output combinational controllability

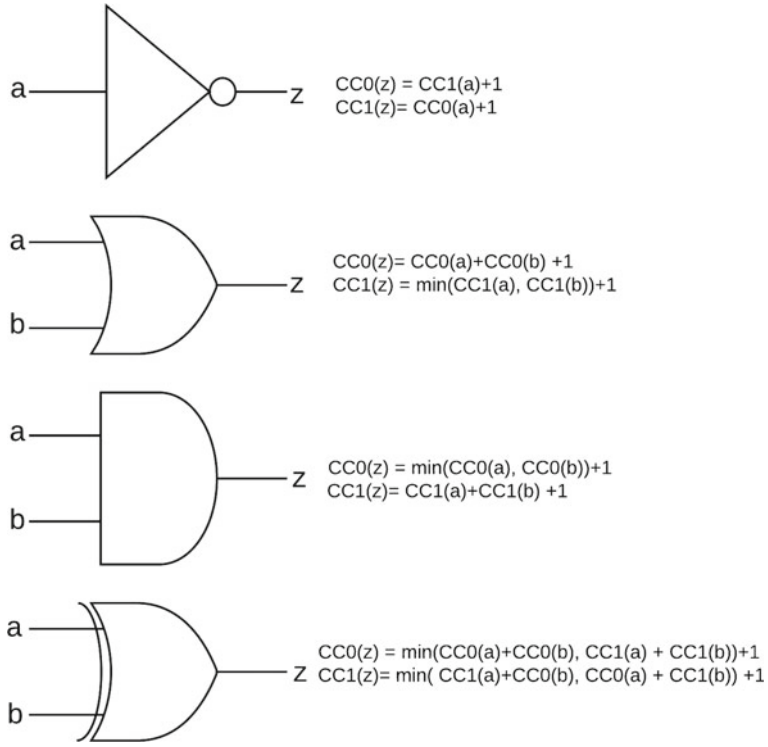


Fig. 2.5 SCOAP controllability calculation

$$Output controllability = \begin{cases} min(input\ controllability) + 1\ ,\ if\ one\ input\ sets\ gate\ output \\ sum(input\ controllability) + 1\ ,\ if\ all\ inputs\ sets\ gate\ output \\ min\ (controllabilities\ of\ input\ sets),\ if\ output\ is\ determined \\ by\ multiple\ input\ sets,\ e.g.,\ XOR \end{cases} \tag{2.3}$$

Figure 2.5 describes the output controllability calculation for a set of standard cells. In contrast, for observability, all the primary outputs (PO-sig) are set to 0. A PO to PI traversal adds 1 to internal nodes as their depth, measured to a PO, is increased. For example, the observability of a gate input is computed using the gate's output observability and the controllabilities on its inputs as follows:

$$Input\ observability = output\ observability + sum\ (controllabilities\ of\ all\ other\ input\ pins\ to\ noncontrollable\ value) + 1 \tag{2.4}$$

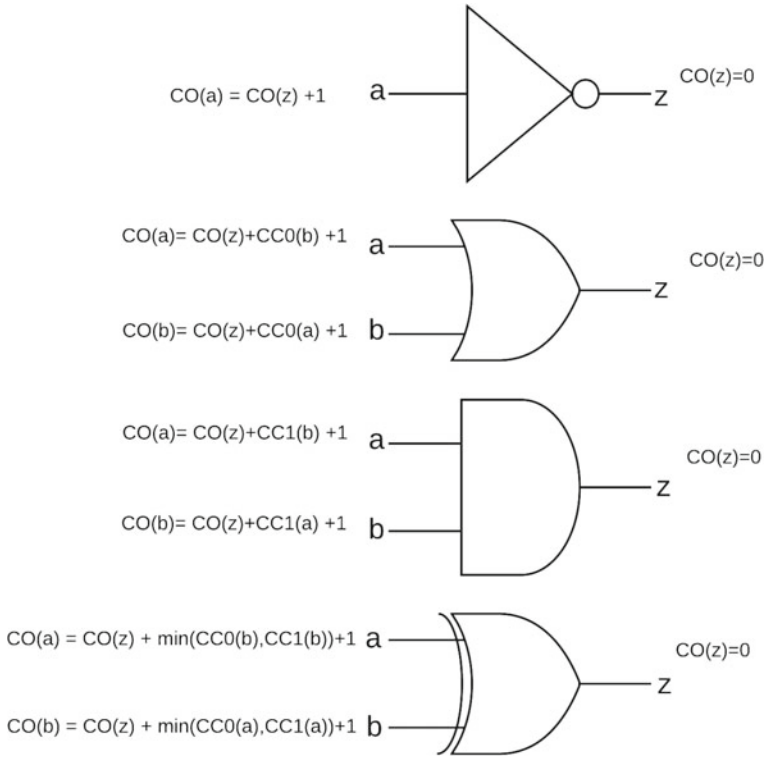


Fig. 2.6 SCOAP observability calculation

Figure 2.6 gives the equations for a common set of logic gates, including NOT, AND, OR, and XOR. The SC0 and SC1 calculations carried out for the sequential gates, e.g., the D-FF, take into account how many times the FF must be clocked to reach a particular output state of '0' or '1'.

The heuristics used in computing testability metrics provide the algorithms with linear runtime complexity. The analysis aids in the design process and can be used to provide guidance to ATPG algorithms as to which nodes are difficult to test. Designs for testability (DFT) techniques can be used in the design flow to add additional nodes as a means of improving the overall controllability and observability of circuit nodes. DFT methods are categorized as ad hoc methods and structured methods. In ad hoc methods, test structures are inserted into designs to target-specific problem areas on a case-by-case basis. Structured DFT methods, on the other hand, include standardized test structures such as scan and built-in self-test (BIST). DFT is typically carried out as an integral component of the design flow to ensure testing requirements can be met. For example, DFT can improve fault coverage and reduce the test generation time.

2.2.8 Testing and Security

The goal of manufacturing test is to detect defects that occur during fabrication or packaging before chips are shipped to customers or enter the supply chain. For security and trust, the goal is to provide a high assurance, trusted product. Unfortunately, detecting security and trust problem is much more difficult than providing high-quality, defect-free chips. This is true because the random nature of manufacturing defects makes it possible to find nearly all of them with the test vectors that provide high levels of fault coverage. The adversary for security and trust, on the other hand, will apply sophisticated techniques to break security systems and add malicious components (hardware Trojans) that are nearly impossible to activate and discover using current manufacturing test techniques.

Secondly, traditional approaches that are based on 'security-by-obscurity,' where internal design components are manipulated to impair reverse engineering attacks, are in fact partially defeated by DFT structures that assist with manufacturing test. Scan and other ad hoc DFT approaches that increase controllability and observability make it easier for adversaries to obtain internal design details in reverse engineering attacks. In subsequent sections, we discuss the security vulnerabilities introduced by DFT and the proposed countermeasures to allow test engineers to leverage them for finding defects, but simultaneously prevent adversaries from using them for reverse engineering attacks and as a 'backdoor' to break security mechanisms. DFT strategies that enable attack vectors include the following:

(1) Scan,
(2) Boundary scan, and
(3) Built-in self-test (BIST)

2.2.8.1 Scan-Based

Scan Cells

Most DFT strategies, including scan insertion, are implemented during synthesis. Scan insertion replaces the flip-flops (FFs) in the design with a special scan-based FFs. Scan FFs add a special 'test mode' of operation to the FF that allows all or a portion of the FFs to be linked together into a scan chain. The scan input of the scan chain is connected to a primary input, and the scan-chain output is connected to a primary output. This enables the test engineer to set and observe the internal state of the FFs directly and therefore significantly simplifies the task of testing internal combinational blocks for defects.

Figure 2.7 shows the modifications that are made to conventional DFF to convert them into scan FFs. Two styles of scan insertion are shown, called MUXD-SFF and LSSD-SFF. The MUXD-SFF cell now includes a multiplexed input and a control signal that allows functional mode (with normal D input selected) and scan mode

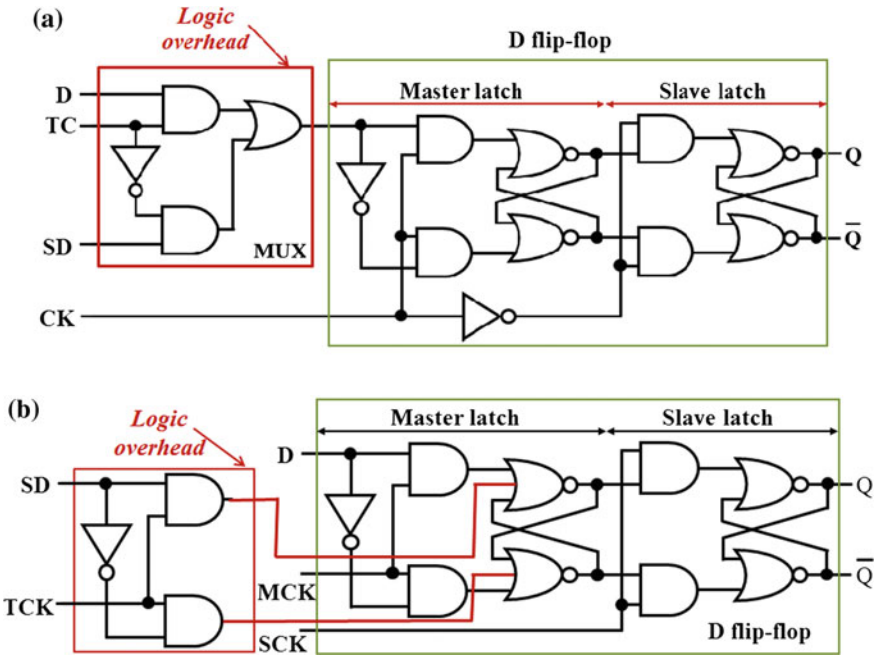


Fig. 2.7 Scan flip-flop design. **a** MUXD-SFF. **b** LSSD-SFF

(with scan input selected). During manufacturing test, the test mode signal called scan enable is asserted to enable the scan operation. The operation of scan-based testing (Fig. 2.8) is a three-step process:

(a) Configuration of the scan cells with a test vector,
(b) Application of the system clock to capture the results, and
(c) Readout of the scan data for analysis.

Built-in self-test (BIST) also adds DFT components to the chip as a mechanism to enable a self-testing mode of operation. BIST can significantly reduce the test costs by making it possible for the chip to self-detect problems, which reduces the dependency and time required on expensive automatic test equipment (ATE).

Scan-Based Attacks on Obfuscation

Scan-based testing is a significant and important tool for reducing cost and improving coverage of manufacturing test, but it can also be used to support noninvasive attacks designed to steal important information such as keys or to bypass security mechanisms and aid adversaries in reverse engineering attacks. Scan chains are easily exploitable by an adversary who has access to the chip and can use it as a 'side channel' for malicious activities such as cryptanalysis [15, 16]. Scan-based attacks are categorized broadly into two categories: scan-based observability attacks and scan-based controllability/observability attacks. A scan chain provides the adversary with

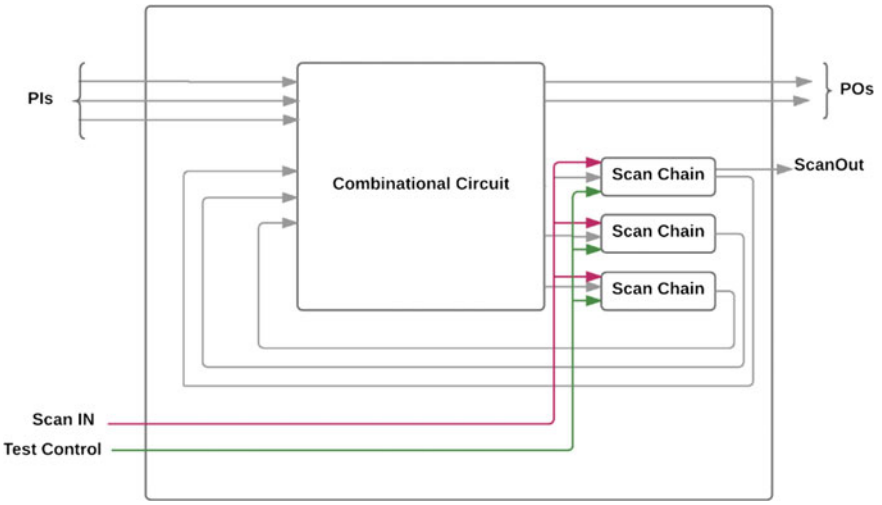


Fig. 2.8 Scan-based testing

the ability to take snapshots of the chip in different states to help reverse-engineer the design. Alternatively, the adversary can set registers to specific values while operating the chip in test mode and hence can access internal secrets, such as key registers, and change key operational modes as a means of bypassing any inserted security mechanisms.

Countermeasures for Scan-Based Attacks

Several techniques are proposed to secure the scan chain, such as disabling scan chain after manufacturing test and scrambling scan chain to make it harder for the adversary to carry out reverse engineering attacks. The scan infrastructure can be secured by blowing fuses to disable scan chain after manufacturing test [17]. In this approach, the protected registers can be made uncontrollable and unobservable by eliminating physical access to them. The disadvantage of this approach includes the fuse-blowing post-processing step and the vulnerability of fuses to focused ion beam (FIB) attacks [18]. FIB tools have been developed to enable 'circuit edit,' i.e., the adding and removal of metal at specific regions in the chip. The adversary can use FIB to repair the blow fuses and re-enable access to the secret keys and design details.

Scan-chain scrambling techniques obfuscate the register-to-scan-chain mapping to make it harder to interpret scan data [19]. The technique requires a key to establish the correct assignment of register-to-scan-chain mapping. Incorrect keys randomly map the registers to scan-chain elements, effectively scrambling the data. This technique protects embedded secret information as well as details of the internal design, making it difficult to reverse-engineer the chip.

An alternative is to implement a key separation method that disables access to the secret key register in test mode [16]. The proposed method introduces a mirror key register (MKR) which is muxed-in when scan mode is enabled and which prevents

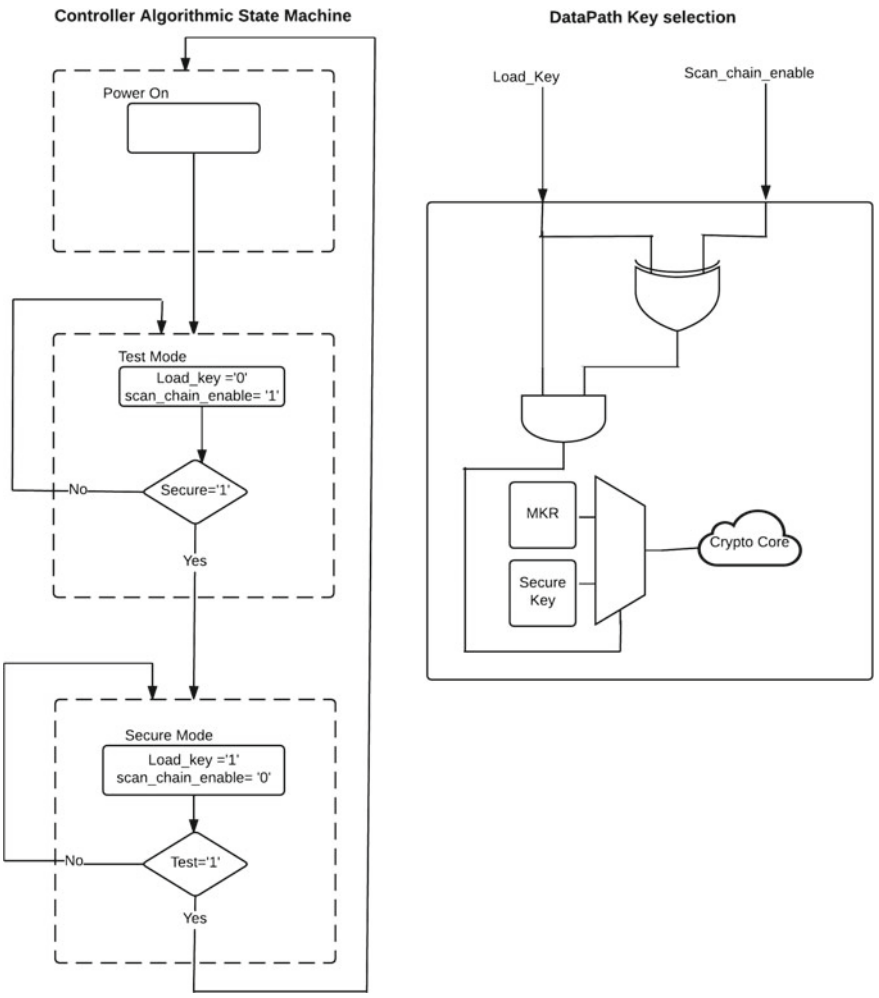


Fig. 2.9 Algorithmic state machine and data path for mirror key

access to the cryptographic key register. This approach enables the cryptographic unit to be tested for manufacturing defects but prevents an adversary from using scan to steal the secret key during functional mode. The control signal to the MKR is the scan-enable signal, so the switch to the MKR is performed automatically when scan is enabled. A block diagram of the proposed method is shown in Fig. 2.9 using the algorithmic state machine and data path. This technique protects the secret key, but still allows the probing of internal design details and therefore does not prevent reverse engineering attacks.

A low-cost secure scan (LCSS) technique is proposed to overcome this deficiency by introducing dummy flip-flops in the scan chain [20]. The proposed architecture is

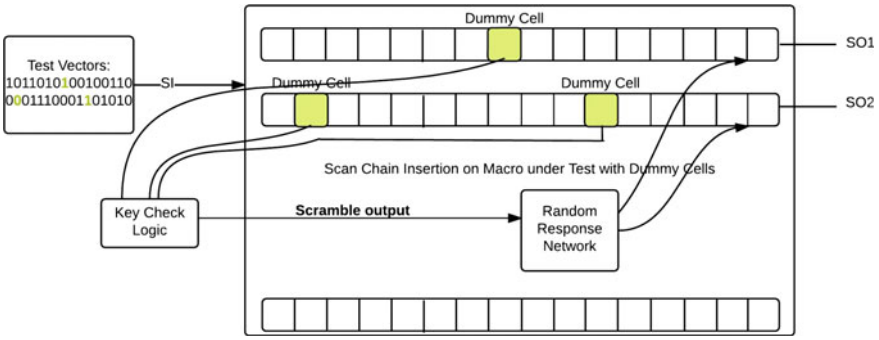


Fig. 2.10 Low-cost secure scan (LCSS)

shown in Fig. 2.10. LCSS requires only small changes to be made in the design flow to accommodate the insertion of additional scan cells and can be used to protect secret embedded keys and the chip's intellectual property. All the dummy cells are checked with key checking logic (KCL) to determine whether the dummy cells have been programmed with the correct code. Incorrect codes disable access to the scan-chain data and instead enable a q-bit LFSR which generates random data on the scan-chain output.

2.2.8.2 Boundary Scan

Boundary scan is a DFT mechanism for printed circuit board (PCB)-level testing, that is similar to the scan DFT technique used inside the chip. Boundary scan creates a shift register out of the I/O pads of chip and allows the chip solder connections and interconnect on the PCB to be tested for manufacturing defects. JTAG is an IEEE standard 1149.1 developed by a working group called the Joint Test Access Group, along with other scan architectures including IEEE Std. 1500 and IEEE P1687 (IJTAG) for reconfigurable scan networks.

JTAG

JTAG provides a single test interface across heterogeneous components/devices on a printed circuit board (PCB) and hence facilitates testing. Figure 2.11 shows the interface signals of the test access port (TAP).

The test signals for the JTAG interface are defined as follows:

TCK: Test clock—all the boundary scan cells are shifted with the event on TCK.
TMS: Test mode select determines the next state. There are 16 states in JTAG.
TDI: Test data in—test vectors are provided through this signal. Additional JTAG instructions are also provided by TDI.
TDO: Test data out—scan out the responses.

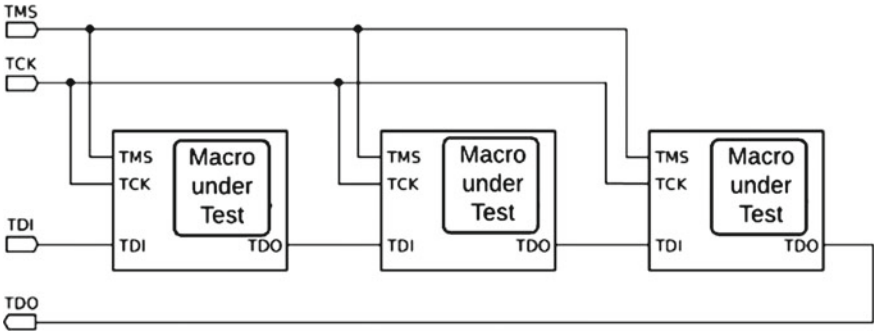


Fig. 2.11 JTAG

JTAG-Based Attacks on Obfuscation

JTAG makes the chips and entire PCB vulnerable to attacks because it does not implement any type of device authentication in its daisy chain topology. Several attacks have been reported that exploit the JTAG interface as a means of stealing secret keys, of carrying out piracy of intellectual property, and to circumvent standard policies. The adversary can also replace genuine chips with counterfeit clones without fear of being detected. Therefore, JTAG has the same type of vulnerabilities as scan-chain design and additionally is vulnerable to the insertion of malicious devices. One such attack model is discussed in [21] explaining that the adversary can hijack the shared resources, such as bus, and launch a denial-of-service attack or spoof information.

Countermeasures Against JTAG Attacks

One countermeasure proposed in [22] is to implement JTAG interface using fuses and electronically destroy it after the completion of manufacturing test, thus eliminating security risks. A better solution is to add a security mechanism to JTAG that is designed to limit the access to authorized users. An authentication mechanism can also be added to allow a controller or centralized trusted server to authorize chips to perform certain tests [23]. An alternative is to allow the controller chip to abort test traffic by introducing security policies [24]. Compact cryptographic modules can be further included to encrypt test data and carry out key-based authentication of chip under test [21]. Keys can be programmed on chip in tamper-evident nonvolatile memory or they can be generated on the fly using physical unclonable functions [25].

2.2.8.3 Built-In Self-Test (BIST)

Built-in self-test is a testing technique that can generate and apply random test vectors on chip and then validate that the results are fault-free, thus eliminating ATE at the cost of additional area overhead on the chip. BIST is commonly used to test embedded memories that do not provide external pins for direct access. An example showing

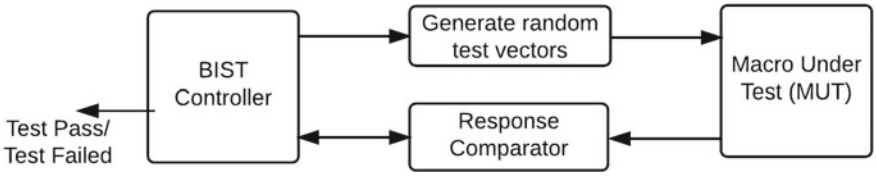


Fig. 2.12 Built-in self-test process

how BIST can be implemented is shown in Fig. 2.12. The controller applies random test vectors to the macro under test (MUT) and verifies the responses on chip. The interface control signals only convey whether the circuit is passed or failed and does not transfer the responses. BIST can be implemented with chips and boards that also include JTAG. Since BIST does not reveal the data or state of the system, it naturally provides obfuscation.

2.3 Hardware-Based Obfuscation Design Primitives

To better understand hardware primitives that are currently used in several obfuscation techniques, we first discuss a classification scheme for obfuscation techniques.

2.3.1 Types of Hardware Obfuscation

Hardware-based obfuscation is broadly categorized as passive hardware obfuscation, active hardware obfuscation, and reconfigurable logic-based obfuscation. Recently proposed method includes active key-based hardware obfuscation schemes that can be further classified as combinational logic obfuscation and finite state machine (FSM)-based obfuscation.

2.3.1.1 Passive Hardware Obfuscation

In keyless or passive hardware obfuscation techniques, the design description is obfuscated and/or encrypted using cryptographic primitives. A register transfer-level (RTL) design obfuscation technique is discussed in [26], which renames signals and reorganizes the code to obscure its meaning to adversaries. Research reported in [27–29] encrypts the hardware description language (HDL) before distributing to untrusted entities in the supply chain. The IP designer provides key to legal customers to decrypt the design for integration or for fabrication.

In passive or keyless hardware obfuscation, the functionality is not modified and only the design file or netlist is obfuscated. Passive hardware obfuscation techniques

do not stop the adversary from using the design as a black box or from distributing or overbuilding the design. Also, passive techniques cannot prevent the customers from distributing the decrypted copy.

2.3.1.2 Active Hardware Obfuscation

Active hardware obfuscation or key-based techniques, on the other hand, modify the functionality as a mechanism to harden the design against reverse engineering. Logic-based obfuscation involves embedding the key in the functional unit itself and requires the user to provide the correct keys along with the functional inputs to get the correct results. Key integration is accomplished by the insertion of key-based logic in the combinational logic paths and/or finite state machine (FSM) of the design. For example, most proposed techniques add states to the FSM and XOR and XNOR gates to the data path.

Combinational Logic Obfuscation

Logic obfuscation modifies the design by incorporating additional gates such as XOR and XNOR to the data path, which have one or more of their inputs driven by registers that store the key. Timing analysis is typically performed in advance to select insertion points that do not impact the timing characteristics of the design. The insertion points can be selected randomly [30, 31] among the noncritical paths available, or more sophisticated techniques can be employed, such as those based on graph theory [32] or fault model analysis [33]. These techniques are covered in detail in Chaps. 5 and 6.

FSM-Based Logic Obfuscation

FSM-based obfuscation, also referred as IC metering, modifies the circuit design and locks each chip using a unique state transition path that can only be unlocked when the chip receives the correct key from a key management authority or design house. The key ensures the chip follows an unlocking sequence of state transitions when powered up to run in functional mode [34]. These techniques can be designed to require a unique key for each chip, that is either stored in a NVM such as an EEPROM or fuses or be generated on the fly using a physical unclonable function (PUF). The key is paired with an augmented FSM in such a way that only the design house can unlock the chip. Sections 2.4 and 2.5 discuss the key management using NVMs, PUFs, and TRNGs.

Reconfigurable Logic-Based Obfuscation

Reconfigurable logic-based obfuscation technique suggests to make a small component of the design reconfigurable in the chip. This approach hides the functional details of the obfuscation method during the manufacturing process which hinders the untrusted fabrication facility from reverse engineering the design. The technique proposed in [35] utilizes a fingerprinting technique by altering the implementation

slightly as a mechanism to detect clones or overbuilding. The use of embedded reconfigurable logic against code injection attacks on an open source SPARC processor is discussed in [36, 37].

2.3.2 Metrics of Hardware Obfuscation

Active hardware-based obfuscation hardens the design against reverse engineering, but such a scheme is vulnerable to side-channel attacks on keys and simulation-based attacks designed to decode key–gate values. It is assumed that if the malicious user is given enough time and resources, the obfuscation will fail. The authors of [38] propose the following objectives and metrics for hardware obfuscation:

(I) The size of the input space must be large enough to make brute force attacks on FSM and combinational logic obfuscation infeasible.
(II) The obfuscation method should attempt to maximize the impact of wrong key guesses, such as the hamming distance between the correct outputs and obfuscated outputs is 50%.

2.4 Volatile and Nonvolatile Memories

Active hardware obfuscation techniques require a key storage mechanism to produce the correct results, and for the case of programmable logic, netlist configuration information must be available at power-on to reconfigure the field programmable gate components. A variety of technologies exist to permanently store the keys including volatile memory and nonvolatile memory (NVM).

2.4.1 Volatile Memory

Key-related information stored in a volatile memory, such as dynamic random-access memory (DRAM) or static RAM (SRAM), is lost over power cycles of the chip unless it is powered from a battery, which represents a cost overhead for the system and reduces its reliability and availability. Other disadvantages of using volatile memory for key storage are that it is vulnerable to ‘cold boot attacks’ and requires the key communication process to be secure.

2.4.2 Nonvolatile Memory

NVM retains data across power cycles and can be categorized according to the writing mechanism that they employ. ROM, EPROM, EEPROM, and FLASH are common NVMs that are read-only, read mostly, and rewritable, respectively.

2.4.2.1 Read-only Memory (ROM)

ROM is a read-only memory that is programmed during the manufacturing process and cannot be changed in the field. ROM is a high-speed, high-density, and low-cost memory, thus making it an attractive medium for low-cost and embedded devices. Keys are permanently stored, are immutable, and are usually the same for all the manufactured devices. Furthermore, ROMs are vulnerable to attacks whereby adversaries can use specialized tools to read out their contents.

2.4.2.2 Reprogrammable Memory

EPROM and EEPROM are reprogrammable memories that can be programmed after manufacturing. These memories utilize floating gate-type technology which allows the data storage transistor to be reprogrammed by changing the trapped charge on the gate input. The floating gate retains the trapped charge across power cycles, and therefore, it does not require a battery. However, specialized hardware is required to add and remove the trapped charge. A benefit of floating gate technologies is that keys can be programmed after manufacturing, thereby preventing the manufacturer from engaging in reverse engineering attacks.

2.4.2.3 One-Time Programmable Memory

Antifuse, e-fuse, and laser fuses are one-time programmable memories and therefore represent a class of fused-based technologies. Fused-based storage is more vulnerable to invasive attacks which probe the layout of the chip as a means of stealing the secret information and bypassing the security mechanisms.

2.4.2.4 Emerging Technologies: RRAM or ReRAM, PCM, and STT-MRAM

Resistive random-access memory (RRAM) or ReRAM is a NVM that stores ‘0’ and ‘1’ by changing the resistance of memristor devices. PCM is similar to ReRAM, which stores information using resistance levels. STT-RAM stores information on ferromagnetic layers using magnetic polarization and has the access speeds close to

the caches. These memories are nonvolatile and therefore do not require an energy source to maintain their contents.

2.4.3 Limitations of Current Key Storage Mechanisms

The advantage of storing information in RAM is that the secret information is lost after a power cycle. On the other hand, the battery-backed RAM introduces reliability issues because data is lost if the battery fails. Conventional key management systems utilize NVMs to store master keys or session keys, but as pointed out earlier, NVMs are vulnerable to invasive and noninvasive attacks.

Invasive attacks such as microprobing and laser-cutting attacks allow the adversary to learn the secrets by decapping the chip and reading the memory cells. To mitigate physical attacks, variants of tamper-resistant NVM include sensors to detect physical access to the device. The sensors require a battery to remain active while power is turned off. Therefore, NVM is not attractive for use in embedded and resource-constraint devices.

Noninvasive attacks that target key extraction from NVM include glitch attacks, fault injection (timing, voltage, temperature, radiation), and power analysis. Mitigation techniques include randomized design flows or design techniques that equalize power consumption. The drawback of these approaches is that it does not follow traditional design flows and increase area overhead to the design. Furthermore, emerging NVM technologies are promising but have not been validated as to whether they provide enhanced security properties over conventional NVMs.

2.5 Design Obfuscation: PUF and TRNG

2.5.1 Physical Unclonable Functions (PUFs)

Physical unclonable function (PUF) is an emerging physical layer cryptographic primitive used in hardware security and privacy protocols. They are embedded structures that utilize inherent manufacturing process variations to extract unique but reproducible secrets. The concept was first introduced as physical one-way functions [39] and later as physical unclonable functions [40]. PUFs measure variations in propagation delays, wire resistances, and other analog circuit parameters to produce digital bitstrings that are random and unique across instances of the chip population. PUFs are unclonable because the random information source on which they are based (the source of entropy) cannot be replicated, i.e., manufacturing process control cannot reduce the physical and electrical variations that occur across and within chips to zero tolerance levels. PUF bitstrings are generated on the fly as needed and therefore eliminate the need for NVM, e.g., EEPROM and e-fuses. This new key

generation mechanism eliminates the probing attack vulnerabilities discussed earlier in reference to NVMs because PUFs do not store digital versions of the bitstrings and the analog nature of the entropy source makes it tamper-evident whereby physical probing changes and/or destroys the ability to regenerate the same bitstring.

2.5.1.1 PUF Operation

(a) Apply Challenges

PUFs are based on a challenge–response pair (CRP) mechanism. The challenge for a PUF is defined as a digital input, usually in the form of a bitstring of '0's and '1's. The output of a PUF is also digital, but for most PUFs, this requires an on-chip mechanism to convert the small analog variations leveraged by the PUF to be digitized. The digitization process occurs automatically for some PUFs, such as the SRAM PUF, where the bitstring is produced immediately after power-up. The challenge to the PUF typically selects a unique set of elements from the entropy source or, as is more common, selects a set of elements that are combined in a unique fashion. The randomness of the entropy source ensures that the CRPs are unique across chips, i.e., the response bitstring produced by the PUF is different for each chip even when using the same challenge. In the best case, 50% of the bits in the response bitstring are uniquely defined by each PUF instance.

(b) Enrollment

PUF-based security applications require an enrollment process. Enrollment is carried out in a secure environment where CRPs are measured and stored by a trusted authority in a secure database. Enrollment can also be done while the chip is in the field as long as the PUF's existing secrets can be used to securely transmit new CRPs to the trusted authority.

(c) Regeneration

Regeneration is a process that is carried out by a fielded chip usually in response to a request issued by an application that requires a key for encryption or a unique bitstring for authentication. When exact replication of the bitstring is required, e.g., key generation for encryption, PUFs require some type of helper data as a means of fixing or avoiding bit flip errors that occur when the CRPs are reapplied. Helper data is typically stored by the trusted authority during the enrollment process and is transmitted to the fielded device in-the-clear when needed. Therefore, helper data does not leak any, or leaks very little information, about the secret bitstring. In other applications such as authentication, exact reproduction of the bitstring may not be required, and instead, a close match is sufficient to confirm the identity of the chip.

2.5.2 PUF Evaluation Measures and Parameters:

2.5.2.1 Effect of Environmental Variations

Reproducing the bitstring exactly without helper data is challenging for PUFs because of changes that occur to the entropy source when the environment changes, i.e., the outside temperature is high or a low battery causes the supply voltage to drop on the chip. Changes in the environment introduce bit flip errors in the response bitstring, making it difficult to achieve high reliability. Error-tolerant mechanisms can be designed into the PUF architecture to help mitigate these types of adverse effects, e.g., [41]. However, error tolerance is not sufficient in many applications, and helper data must also be used as described above to meet the reliability requirements.

2.5.2.2 Evaluation Metrics of PUF

A survey on the PUF evaluation metrics is reported in [42] and includes techniques designed to measure and quantify the quality of PUF response bitstrings. The most important of these are summarized as follows:

1. Uniqueness

Uniqueness is a quality metric that ensures the responses generated by any two chips for a given challenge should be substantially different. Interchip hamming distance (HD) is used to quantize the difference, and in the ideal case, it is 50%.

$$uniqueness = \frac{2}{k(k-1)} \sum_{i=1}^{n-1} \cdot \sum_{j=i+1}^{n} \left(\frac{HD(R_i, R_j)}{n} \right) \tag{2.5}$$

where 'R' is the response, 'k' is the number of chips, and 'n' is the length of the response bitstring. Interchip (HD) quantifies the number of differences that occur across a set of response bitstrings. Ideally, each bit position of the response bitstring has the equal probability of being a '0' or '1'. If some bits are biased to one value or the other, then these bits are 'more predictable' from an adversarial point of view. Bit biasing is measured across each bit position in the response bitstrings from a set of chip using hamming weight as given by Eq. 2.3, where 'i' is the bit position and 'm' is the number of chips.

$$Bit\ aliasing[i] = \frac{1}{k} \sum_{j=1}^{k} \left(R_j \right) \tag{2.6}$$

2. Reproducibility

Response bitstring reliability or reproducibility is measured using intrachip HD. The data used in the analysis is the response bitstring measured under different

environmental conditions from the same chip using the same challenge. The data from other chips is typically factored in by computing the average intrachip HD from the individual analyses. The ideal value is 0%, i.e., no chip has any bit flip errors under any environmental conditions. As indicated above, this ideal result is not possible using the 'raw' response bitstrings directly, and instead, helper data is required to achieve error-free regeneration. PUFs that are able to achieve relatively low interchip HDs without helper data can be used in some authentication applications that just require most of the bits in enrollment and regeneration bitstrings to match.

$$Reproducibility = \frac{1}{m}\sum_{i'=1}^{m}\left(\frac{HD\left(R_i, R_{i'}\right)}{n}\right) \tag{2.7}$$

Equation 2.4 gives the formula for computing intrachip HD. It counts the number of bits that are different in the bitstrings collected over 'm' different environmental conditions, called temperature–voltage corner conditions.

2. Randomness

The PUF architecture, including the circuit structure used as the source of entropy, and measurement technique can induce bias and reduce the randomness in the sequence of bits generated by each chip. Bitstring randomness is measured using statistical tests; for example, NIST has developed statistical testing software for evaluating randomness in bitstrings generated by pseudorandom number generators [43]. The test includes uniformity and frequency tests that count the number of '0's and '1's in a bitstring. The frequency test requires the balance of '0's and '1's in any given bitstring to fall within a tolerance; otherwise, the bitstring fails the test. The test suite includes other tests that look for patterns in a set of bitstring subsequences that occur more often than expected from bitstrings drawn from a truly random source. The NIST software tool suite applies a set of up to 15 different tests to each of the input bitstrings and reports the number of tests that each bitstring passes. Several other randomness evaluation tools have also been developed including DIEHARD [44] and AIS.31 [45].

2.5.3 Classification of PUFs

PUFs are classified into weak and strong PUFs, based primarily on two criteria: the size of their CRP space and the level of resilience they have against model-building attacks. A third related criteria is the size of the entropy source, i.e., how many independent random varying components are used to generate the bitstrings. A second PUF classification uses the terms 'intrinsic' and 'nonintrinsic,' which relates to whether the PUF is self-contained on the chip (intrinsic) or requires external instrumentation to support it. Yet a third classification uses the terms 'nonelectronic' and 'electronic' to refer to the underlying structure of the entropy source, e.g., silicon versus a material that exhibits random properties.

2.5.3.1 Weak and Strong PUFs

Strong PUFs can produce a very large, unique set of bits per device and have a very large CRP space to support this characteristic. The very large CRP space makes it impractical for an adversary, who has possession of the PUF chip, to apply them all as a means of building a database, i.e., a 'digital' clone of the PUF. Many consider a second property, i.e., model-building resistance, to be equally important to the very large CRP space. Model-building refers to an attack mechanism whereby the adversary uses the machine learning algorithms to derive a system capable of predicting the response of the PUF after being trained on only a small portion of the CRPs. Resistance to model-building attacks is best realized using an entropy source with a large set of randomly varying components, but many proposed PUF architectures, e.g., the Arbiter PUF, only have a small set and instead use cryptographic primitives such as secure hash functions and XOR networks to obscure the CRP interface. The HELP PUF is an example of a PUF which is based on a large entropy source, i.e., the best-case scenario [25].

Weak PUFs on the other hand have fewer CRPs and, in some cases, only one response pair which is the case for the SRAM PUF. Weak PUFs are usually limited to key generation where model-building attacks do not apply because the secret does not leave the chip. Weak PUFs can also serve applications that require only a unique ID from the PUF. Weak PUFs have capabilities similar to those provided by an NVM, but provide a tamper-evident property which enhances their security over NVM. Examples of weak PUFs include physically obfuscated keys (POKs) [41], SRAM PUF [46], and Butterfly PUF [47].

2.5.3.2 Intrinsic and Nonintrinsic PUFs

As indicated above, intrinsic PUFs are completely self-contained architectures on the chip, capable of making measurements, and carry out bitstring generation, whereas nonintrinsic PUFs require benchtop instrumentation, e.g., photonic-based sensors, to implement the measurement components. Intrinsic PUFs are far more popular, and the number of proposed architectures continues to grow. Manufacturing process variations on the chip manifest in many forms on the chip including within-transistor threshold voltages and metal resistance characteristics. For example, PUFs based on variations in delay include the Arbiter [48, 49], Ring Oscillator [41], and HELP [25] PUFs. Delay is popular because there are many well-defined on-chip delay measurement techniques that are available.

2.5.3.3 Sources of Entropy

Examples of nonelectronic PUFs include coating PUFs, optical PUFs, and CD Player PUFs. Nonelectronic PUFs, to date, have not been considered as support primitives for implementing hardware obfuscation functions. Electronic intrinsic PUFs are the

most common class of PUFs proposed for this purpose and include those based on variations in transistor threshold voltages [50, 51], propagation delay (as indicated above, the Arbiter, Ring Oscillator [41], and HELP [25] PUF are examples), and power-up patterns in memory, e.g., the SRAM PUF [46]. The list of electronic intrinsic PUFs keeps growing and includes the ROM PUF [52], leakage current PUF [53], the metal resistance PUF [54, 55], the transistor transconductance PUF [56], and other path delay-based PUFs [57, 58].

2.5.4 PUFs: Candidates for Hardware Obfuscation

In this section, we discuss the following PUFs, both weak and strong, that are considered good candidates for hardware obfuscation applications:

(1) Memory-based intrinsic SRAM PUF.
(2) Delay-based intrinsic PUFs including the Arbiter, Ring Oscillator, and HELP PUFs.

(1) SRAM PUF

The SRAM PUF is classified as a weak intrinsic PUF that uses the randomness in the power-up bit patterns of SRAM as source of entropy [46]. The SRAM cell is implemented as a pair of cross-coupled invertors whose geometries are identical (Fig. 2.13). Manufacturing process variations cause mismatches in the transconductance parameters of the inverters resulting in the random power-up states that remain constant for the majority of the cells. The power-up pattern varies from one chip to the next, enabling the SRAM PUF to serve in chip identification roles and in PUF-based hardware obfuscation protocols to map reconfigurable logic to a specific function.

SRAM PUF behavior is affected by the systematic variations, where the number of '1's and '0's can be biased, thus degrading its randomness statistical metric, making it vulnerable to model-building attacks (SRAM PUFs cannot be model-built). SRAM

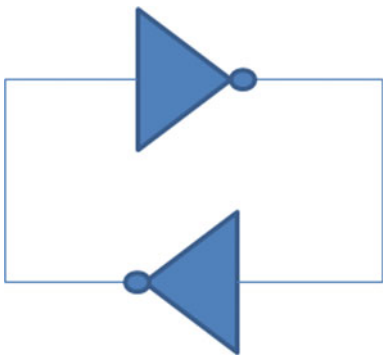


Fig. 2.13 Cross-oupled NOT gate SRAM cell

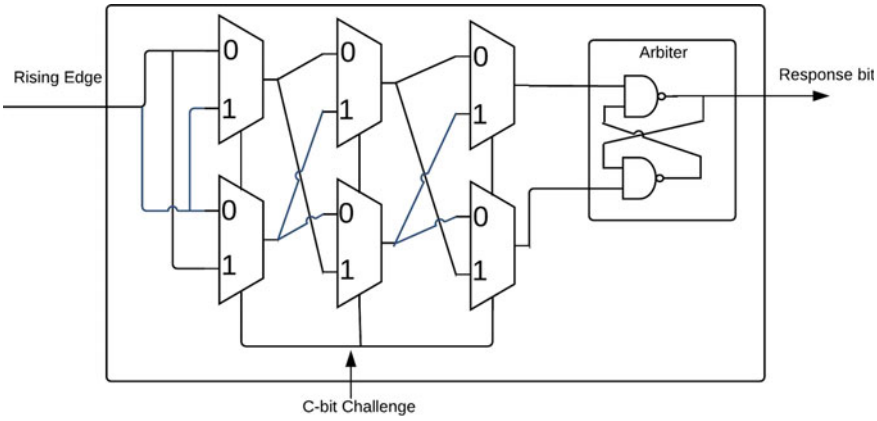


Fig. 2.14 Arbiter PUF

PUFs typically have poor reproducibility, reported as high as 20% or more in some cases.

(2) Arbiter PUF

The Arbiter PUF is a delay-based PUF defined using a sequence of multiplexers and an arbiter, e.g., a cross-coupled NAND latch as a mechanism to provide an unbiased evaluation mechanism as shown in Fig. 2.14 [48, 49]. The PUF leverages the delay variation between two identical paths to generate a bit. Challenge bits select the configuration of the switches that in turn determines the specific configuration of the paths. The pairs of multiplexers serve as switch boxes, either routing the two paths straight through the switches or flipping their connections. For a given challenge, the Arbiter PUF measures the delay of two identical length paths. A rising signal is given input to leftmost pair of multiplexers, as shown in Fig. 2.14. The input signal races along the two delay lines, and the arbiter at the end assigns a '0' or '1' based on which path is faster. The connection of the path endpoints to the D and Clk inputs allows the arbiter gate to automatically compute the result of the race.

The arbiter PUF is vulnerable to model-building attacks because of its linear structure and small number of components. A precise timing model can be constructed to learn the parameters from a relatively small set of CRPs. To reduce the effectiveness of model-building attacks, the authors of [41, 59] propose a parallel Arbiter architecture which includes an XOR obfuscation network on the outputs.

(3) Ring Oscillator PUF

A Ring Oscillator PUF (RO PUF) is a weak PUF composed of identical delay loops and counters [41]. RO PUFs measure path delay variations as differences in the 'ring' frequency of the delay loops (Fig. 2.15). Challenges select a pair of identical oscillators and compare the number of oscillations produced by each oscillator of the pair. Frequency is measured by connecting the output of each RO to a separate

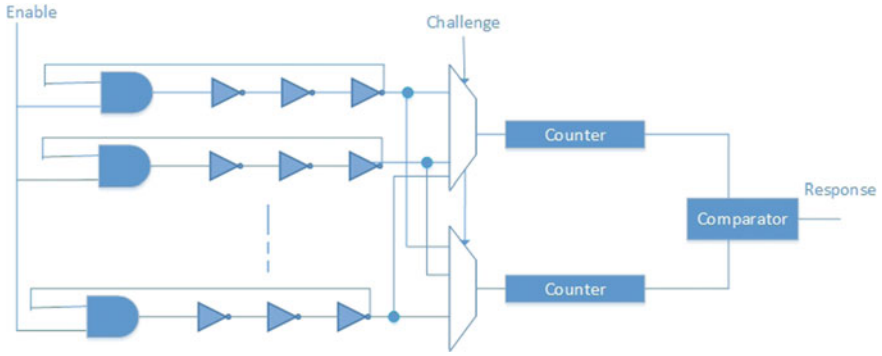


Fig. 2.15 Ring oscillator PUF

counter. The result of the comparison generates a single '0' or '1' bit in the bitstring. Other pairings are used to construct the additional components of the bitstring.

RO PUFs are also subject to model-building attacks in common usage scenarios in which the same RO is used in multiple different pairings. Machine learning algorithms attempt to determine the relative frequencies of all ROs, which, once known, make it possible to predict the response bitstring to any sequence of challenges used to build the bitstring [60].

(4) HELP PUF

A hardware-embedded delay PUF (HELP PUF) proposed in [25, 57] is a strong PUF. It leverages delay variations in existing design functional units and does not require identical structures, unlike other existing delay-based PUFs. HELP also implicitly provides tamper protection of the existing functional unit(s), i.e., any change in the structural characteristics of the functional unit will change the measured path delays.

Figure 2.16 shows the architecture of HELP with the functional unit representing the entropy source. The inputs and outputs of the functional unit are connected to a set of launch row and capture row flip-flops (FFs), respectively. A series of launch–capture clocking events are applied to the functional unit using two clocks, Clk_1 and Clk_2 as shown on the left side of Fig. 2.16. The phase shift between Clk_1 and Clk_2 is adjusted dynamically across the sequence of launch–capture tests, where the digitally selected value of the fine phase shift between the two clocks is referred as the launch–capture interval (LCI). The smallest LCI interval that allows the propagating edge along a path to be captured in the capture FF is used as the digitized timing value for the path.

PUF response bits are computed from delay differences between nonidentical path delays. A modulus technique is proposed as a means of removing the bias in the path delays of the nonidentical paths used in the difference operation while fully preserving the smaller within-die delay variations.

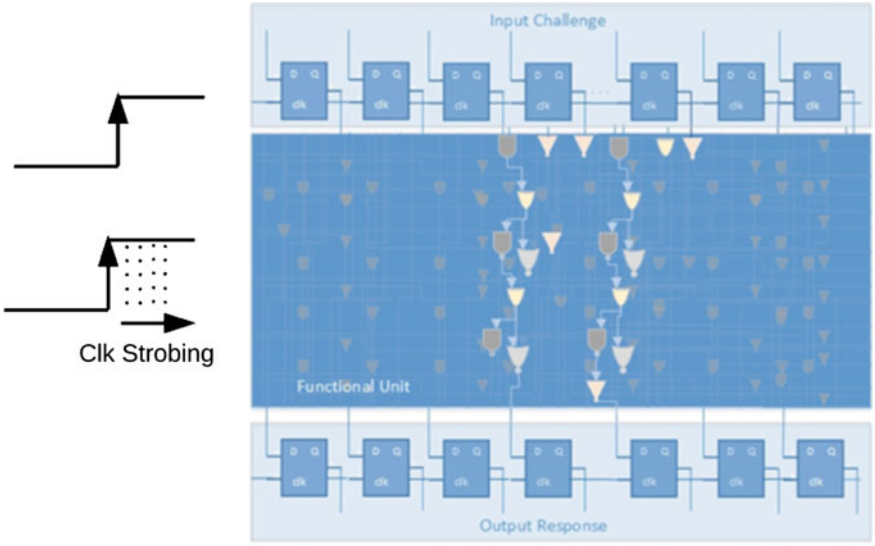


Fig. 2.16 HELP PUF

2.5.5 True Random Number Generator (TRNG) Use in Hardware Obfuscation

True random number generators are hardware primitives that are used in many hardware-based security techniques, including hardware obfuscation. A true random number generator (TRNG) uses randomness and noise to generate secrets that are not reproducible. The randomness or noise should have uniform distribution to avoid bias. The TRNG is an important primitive for cryptographic applications, which is used for generating nonces for authentication protocols, for generating one-time pads and for providing a selection mechanism for primes, as a unique key per device, etc. Hardware obfuscation and hardware metering using a TRNG are proposed in [31] to define randomized chip IDs upon power-up that are then stored in tamper-resistant NVM.

A TRNG can be implemented using on-chip variations [46, 48]. Examples of such TRNG are arbiter-based TRNGs, ring oscillator-based TRNGs, and technology-independent TI-TRNGs [61]. TRNGs are evaluated with respect to randomness and the uniformity of their distribution. Environmental variations such as supply voltage or temperature variation can adversely affect the noise distribution and introduce bias, making the output from the TRNG more predictable.

TRNGs are used to generate unique keys for input to key gates in combinational logic and in obfuscated state machines. The obfuscated data path and control path produce the correct output when the correct key is applied. TRNG-based key generation requires storage of the generated key in a battery-backed RAM or NVM.

The disadvantages of using TRNGs for producing keys are that the stored keys in battery-backed memory or NVM can be stolen and cloned, allowing designs to be reverse-engineered and security features completely eliminated from the design. Additionally, the overhead of manufacturing of NVM requires additional masks and manufacturing steps, thus increasing the costs of the chip. Thus, other alternatives such as physical unclonable functions are better suited for the generation of reproducible secret keys, as long as high reliability to bit flip errors can be ensured.

2.5.6 *Applications of PUFs and TRNG in Hardware-Based Obfuscation Techniques*

PUFs and TRNGs can be incorporated into logic obfuscation for the chip authentication [31, 34] or for obfuscation of logic [62]. PUFs and TRNGs can use nonelectrical properties such as heat, atmospheric noise, and fiber optics as a source of entropy; however, focus has been on the silicon process variations that can be more easily measured and digitized. PUF-based obfuscation and activation schemes can be used to improve security by allowing each chip to be assigned its own unique challenge-response pairs, thereby allowing each chip to exclusively modify and hide the design and authenticate to allow correct functionality, respectively.

A finite state machine (FSM)-based metering technique described in [34] hides the functionality with an augmented FSM structure known as black hole finite state machine (BFSM). The PUF response directs the state transition from obfuscated states to the valid state, and only valid transitions can bring the chip to a properly functioning operational state. The PUF is used to generate a unique key for the finite state-based activation and hides actual functionality from the adversary, thereby preventing illegitimate overbuilt chips. This augmented FSM can be implemented using reconfigurable logic, where each chip has a unique key based on the chip identifier. A PUF-based BFSM technique is shown in Fig. 2.17.

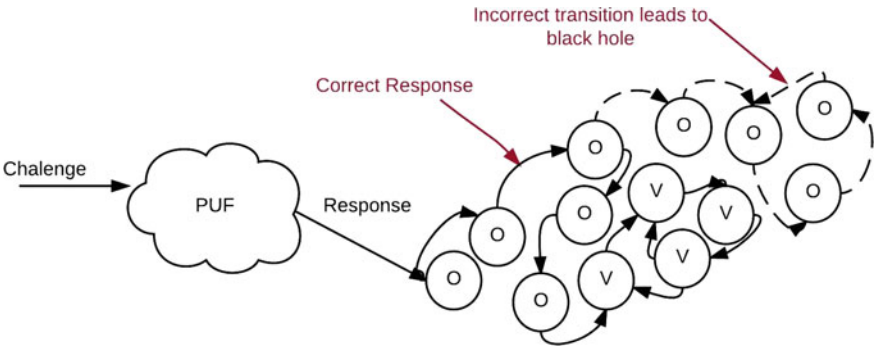


Fig. 2.17 PUF-based BFSM technique

This technique was subsequently modified by [63] using a smaller number of obfuscated states for remote activation of resource-constraint devices. Some valid states are replicated, and the transition through the replicated states is only possible with the correct key.

An FSM-based hardware obfuscation and metering technique using TRNG is described in [31]. As explained earlier, a TRNG can be used to define randomized and unique identification (ID) upon power-up that is burnt into the electrically programmable fuses, such as an electronic fuse unit (EFU).

A PUF is proposed in [62] to implement hardware obfuscation for logic and interconnect obfuscation. The scheme is shown in Fig. 2.18, where each instance of the obfuscated integrated circuit is different, thus making it resilient to reverse engineering. The adversary not only is required to guess the gates but also needs to characterize the PUF responses or use a brute force method to explore all possibilities. Interconnect obfuscation is achieved by using switching gates such as multiplexers to create wire swapping. As shown in Fig. 2.19, only the correct key or PUF response will establish correct connections.

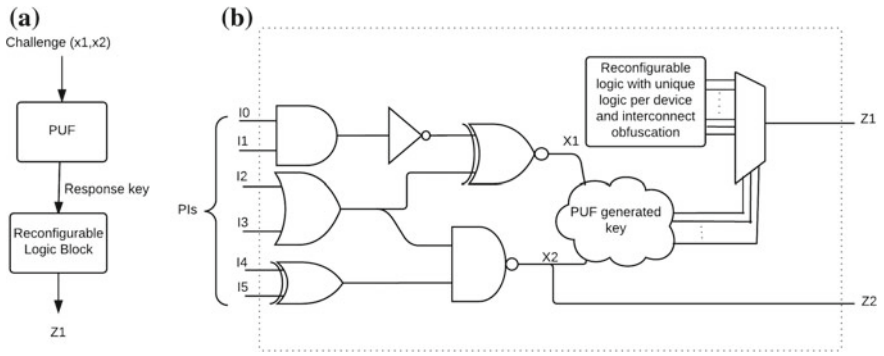


Fig. 2.18 **a** PUF-based random logic obfuscation. **b** Integration in design flow

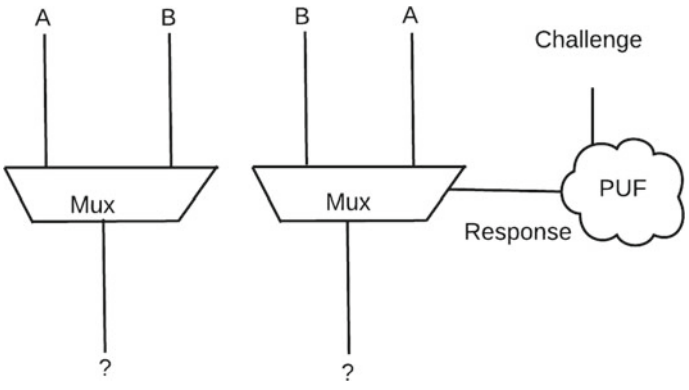


Fig. 2.19 Signal path obfuscation

By choosing PUF-based logic that affects multiple outputs, placement of obfuscated logic with uncontrollable flip-flops can further improve the security of these types of obfuscation techniques. Additionally, the selection of wire pairs to implement swaps between highly correlated pairs can increase the difficulty in reverse engineering. Therefore, hardware obfuscation schemes based on PUFs and TRNGs can effectively mitigate piracy attacks.

Summary

This chapter covered preliminary concepts and techniques of VLSI verification and testing. We describe a set of related vulnerabilities associated with VLSI verification techniques and testing structures that can expose the design details and help reverse-engineer the functionality to compromise the security through obscurity. Proposed changes to existing techniques are discussed that are designed to provide countermeasures against such attacks. The taxonomy of hardware obfuscation techniques is also presented, as well as a set of hardware primitives and related concepts. Hardware obfuscation techniques are motivated because of growing trend of offshoring the fabrication process, where the foundry has the complete knowledge of the design details in the form of GDSII. Obfuscation techniques modify the design and require correct keys as input in order to make the designs functional. Section 2.4 discusses different key storage schemes, such as nonvolatile memory and their vulnerabilities and overhead. Section 2.5 covers hardware-based cryptographic functions, physical unclonable functions (PUFs), and true random number generator (TRNG) as building blocks that further enhance the IC design obfuscation resilience against reverse engineering and mitigate IC piracy attacks. Subsequent chapters further discuss their applications to mitigate IC piracy, cloning, overbuilding, and use of counterfeit chips.

References

1. Zhuang X, Hsien-Hsin TZ, Lee S, Pande S (2004) Hardware assisted control flow obfuscation for embedded processors. In: Proceedings of international conferences on compilers, architecture, and synthesis for embedded system, pp 292–302
2. Rajendran J, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Proceedings of the IEEE design, automation and test in Europe conference and exhibition (DATE), Grenoble, France, 18–22 March 2013, pp 1259–1264
3. Tehranipoor M, Wang C (eds) (2011) Introduction to hardware security and trust. Springer, New York, p 427
4. Guin U, DiMase D, Tehranipoor M (2014) Counterfeit integrated circuits: detection, avoidance, and the challenges ahead. J Electr Test Theory Appl (JETTA) 30:9–23
5. Marques-Silva JAP, Sakallah KA (1996) GRASPVA new search algorithm for satisfiability. In: Proceedings of the ICCAD, pp 220–227
6. Li CM, Anbulagan (1997) Heuristics based on unit propagation for satisfiability problems. In: Proceedings of IJCAI, pp 366–371
7. Malik S, Zhao Y, Madigan CF, Zhang L, Moskewicz MW (2001) Chaff: engineering an efficient SAT solver. In Proceedings of the DAC, pp 530–535. ([62] Marques-Silva JAP, Sakallah KA (1996) GRASPVA new search algorithm for satisfiability. In: Proceedings of the ICCAD, pp 220–227)

8. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms, HOST
9. Goldstein LH (1979) Controllability/observability analysis of digital circuits. IEEE Trans Circuits and Syst (CAS) 26(9):685–693
10. Goldstein L, Thigpen E (1980) SCOAP sandia controlability/observability analysis program. In: Proceedings of the 1980 design automation conference, pp 190–196
11. Bennetts R (1984) Design of testable logic circuits. Addison-Wesley, Reading
12. http://www.eng.auburn.edu/~agrawvd/COURSE/E7250_05/REPORTS_TERM/Kantipudi_Tmeas.pdf
13. Brglez F, Pownall P, Hum R (1984, October) Applications of testability analysis: from ATPG to critical delay path tracing. In Proceedings of the 1984 international test conference on the three faces of test: design, characterization, production (ITC'84). IEEE Computer Society, Washington, DC, USA, pp 705–712
14. Seth SC, Pan L, Agrawal VD (1985, June) PREDICT-probabilistic estimation of digital circuit testability. In: Proceedings of the fault tolerant computing symposium, pp 220–225
15. Yang B, Wu K, Karri R (2004) Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: Proceeding of the IEEE international test conference 2004 (ITC 2004), 26–28 October 2004, pp 339–344
16. Yang B, Wu K, Karri R (2005) Secure scan: a design-for-test architecture for crypto chips. IEEE Trans Comput Aided Des Integr Circuits Syst 25(10):2287–2293
17. Ebrard E, Allard B, Candelier P, Waltz P (2009) Review of fuse and antifuse solutions for advanced standard CMOS technologies. Elsevier Microelectr J 40(12):1755–1765
18. Young R, Carlson P (2004) (Dual-beam FIB/SEM): a tool for advanced failure analysis. In: Evaluation engineering, online magazine September 2004. http://www.evaluationengineering.com/
19. Hely D, Flottes ML, Bancel F, Rouzeyre B, Berard N, Renovell M (2004) Scan design and secure chip (secure IC testing). In: Proceedings of the 10th IEEE international on-line testing symposium, pp 219–224
20. Lee J, Tehranipoor M, Patel C, Plusquellic J (2007) Securing designs against scan-based side-channel attacks. IEEE Trans Dependable Secure Comput 4(4):325–336
21. Rosenfeld K, Karri R (2010) Attacks and defenses for JTAG. IEEE Des Test Comput 27(1):36–47
22. Sourgen L (1993) Security locks for integrated circuit. US Patent # 5264742
23. Busky RF, Frosik BB (2006) Protected JTAG. Proceeding of the IEEE 2006 international conference on parallel processing workshops. Columbus, OH, USA, pp 407–414
24. Clark CJ, Riccihetti M (2004) A code-less BIST processor for embedded test and in-system configuration of boards and systems. IEEE test conference 2004:857–866
25. Saqib F, Areno M, Aarestad J, Plusquellic J (2014) An ASIC implementation of a hardware-embedded physical unclonable function. In: IET Comput Dig Tech 8(6):288–299 (Patent Pending)
26. Thicket family of source code obfuscators. http://www.semdesigns.com
27. Batra T, Methodology for protection and licensing of HDL IP. http://www.us.design-reuse.com/news/?id=12745&print=yes
28. Goering R, Synplicity initiative eases IP evaluation for FPGAs. http://www.scdsource.com/article.php?id=170
29. Xilinx IP evaluation. http://www.xilinx.com/ipcenter/ipevaluation/index.htm
30. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD) 28(10):1493–1502
31. Roy J, Koushanfar F, Markov I, EPIC: ending piracy of integrated circuits. In: Proceedings of the design automation and test in Europe (DATE), pp 1069–1074
32. Rajendran J, Pino Y, Sinanoghu O, Karri R (2012) Security analysis of logic obfuscation. ACM/IEEE49th design automation conference (DAC), 3–7 June 2012. CA, USA, San Francisco, pp 83–89

33. Ranjendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoghu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64(2):410–424
34. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. Proceedings of 16 USENIX security symposium, (2007) USENIX Association. Berkley, CA, USA, pp 291–306
35. Koushanfar F, Qu G (2001) Hardware metering. In: Proceedings of the IEEE design automation conference 2001 (DAC 2001), pp 490–493
36. Liu B, Wang B (2014) Reconfiguration-based VLSI design for security. IEEE J Emerg Selected Top Circuits Syst 2014 (JETCAS 2014), 5(1):98-=108
37. Baumgarten A, Tyag A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. IEEE Des Test Comput 27(1):66–75
38. Rostami M, Koushanfar F, Rajendran J, Karri R (2013) Hardware security: threat models and metrics. In: Proceedings of the 2013 IEEE/ACM international conference on computer-aided design (ICCAD 2013). San Jose, CA, USA, 18–21 November 2013, pp 819–823
39. Pappu R (2001) Physical one-way functions, PhD thesis, Massachusetts Institute of Technology
40. Gassend B, Clarke D, Van Dijk M, Devadas S (2002) Silicon physical random functions. In: Proceedings of the 9th ACM conference on computer and communication security, 2002, pp 148–160
41. Suh GE, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th ACM/IEEE design automation conference (DAC '07). San Diego, CA, USA, 4–8 June 2007, pp 9–14
42. Maiti A, Gunreddy V, Schaumont P (2011) A systematic method to evaluate and compare the performance of physical unclonable functions. J Int Assoc Cryptogr Res (IACR) ePrint, 657:22
43. NIST: computer security division, statistical tests. http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html
44. Marsaglia G (1995) Diehard battery of tests of randomness. http://www.stat.fsu.edu/pub/diehard/
45. Killmann W, Schindler W (2011) A proposal for: functionality classes for random number generators. In: AIS, September 2011, p 133
46. Su Y, Holleman J, Otis B (2007) A 1.6pJ/bit 96 percant stable chip ID generating circuit using process variations. In: Proceedings of the 2007 IEEE international solid-state circuits conferences (ISSCC), pp 200–201
47. Kumar SS, Guajardo J, Maes R, Schrijen GJ, Tuyls P (2008) Extended abstract: the butterfly PUF protecting IP on every FPGA. In: Proceedings of the IEEE international workshop on hardware-oriented security and Trust, 2008 (HOST 2008). Anaheim, CA, USA, June 2008, pp 67–70
48. Gassend B, Lim D, Clarke D, Van Dijk M, Devadas S (2004) Identification and authentication of integrated circuits. Concurrency Comput Pract Exper 16(11):1077–1098
49. Lee JW, Lim D, Gassend B, Suh GE, Dijk MV, Devadas S (2004) A technique to build a secret key in integrated circuits for identification and authentication applications. In: Digest of Technical Papers, IEEE 2004 VLSI Circuits Symposium, 17–19 June 2004, pp 176–179
50. Lofstrom K, Daasch WR, Taylor D (2000) IC identification circuits using device mismatch. In: IEEE digest of technical papers, (2000) international solid state circuits conference. IEEE, San Francisco, CA, USA. February, 2000, pp 372–373
51. Puntin D, Stanzione S, Iannaccone G (2008) CMOS unclonable system for secure authentication based on device variability. Conference on solid-state circuits 2008:130–133
52. Ruhrmair U, Jaeger C, Bator M, Stutzmann M, Lugli P, Csaba G (2011) Applications of high-capacity crossbar memories in cryptography. IEEE Trans Nanotech 10(3):489–498
53. Ganta D, Vivekraja V, Priya K, Nazhandali L (2011) A highly stable leakage-based silicon physical unclonable functions. IEEE 2011 24th international conference on VLSI design. Chennai, India, 2–7 January 2011, pp 135–140
54. Helinski R, Acharyya D, Plusquellic J (2009) Physical unclonable function defined using power distribution system equivalent resistance variations. In: 46th ACM/IEEE design automation conference. San Francisco, CA, USA 26–31 July 2009, pp 676–681

55. Ismari D, Plusquellic J (2014) IP-level implementation of a resistance-based physical unclonable function. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST, 2014). Arlington, VA, USA, 6–7 May 2014, pp 64–69
56. Chakraborty R, Lamech C, Acharyya D, Plusquellic J (2013) A transmission gate physical unclonable function and on-chip voltage-to-digital conversion technique. In: IEEE 2013 50th ACM/EDAC/IEEE design automation conference (DAC, 2013). Austin, TX, USA, 29 May–7 June 2013, pp 1–10
57. Che W, Saqib F, Plusquellic J (2015) PUF-based authentication, invited paper, international conference on computer aided design, November 2015, pp 337–344
58. Zheng Y, Krishna AR, Bhunia S (2013) ScanPUF: robust ultralow-overhead PUF using scan chain. In: IEEE 2013 18th Asia and South Pacific design automation conference (ASP-DAC, 2013). Yokohama, Japan, 22–25 January 2013, pp 626–631
59. Rahman T, Forte D, Fahrny J, Tehranipoor M (2014) ARO-PUF: An aging-resistant ring-oscillator PUF design. In: IEEE design, automation, and test in Europe conference, 2014 (DATE, 2014). Dresden, Germany 24–28 March 2014, pp 1–6
60. Rührmair U, Sehnke F, Sölter J, Dror G, Devadas S, Schmidhuber J (2010) Modeling attacks on physical unclonable functions. In: Proceedings of the 17th ACM conference computer and communications security 2010 (CCS '10), pp.237–249
61. Rahman MT, Xiao K, Forte D, Zhang X, Shi J, Tehranipoor M (2014) TI-TRNG: technology independent true random number generator. In: 2014 51st ACM/EDAC/IEEE design automation conference (DAC 2014). San Francisco, CA, USA, June 2014, pp 1–6
62. Wendt JB, Potkonjak M (2014) Hardware obfuscation using PUF-based logic. In: 2014 IEEE/ACM international conference on computer-aided design (ICCAD 2014). San Jose, CA, USA, 2–6 November 2014, pp 270–271
63. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: Proceedings of the IEEE/ATM international conference on computer-aided design (CAD), 2007. San Jose, CA, USA, 4–8 November 2007, pp 674–677
64. Eichelberger EB, Williams TW (1977) A logic design structure for LSI testability. In: Proceedings of the design automatic conference (DAC), pp 462–468

Part II
Logic-Based Hardware Obfuscation

Chapter 3
Logic Encryption

Jeyavijayan (JV) Rajendran and Siddharth Garg

3.1 Introduction

Logic encryption[1] hides the functionality and the implementation of a design by inserting additional gates into the original design [4–6]. In order for the design to exhibit its correct functionality (i.e., produce correct outputs), a valid key has to be applied to the encrypted design. The gates inserted for encryption are the *key-gates*. Upon applying a wrong key, the encrypted design will exhibit a wrong functionality (i.e., produce wrong outputs).

Example. Consider the circuit shown in Fig. 3.1 which is encrypted using key-gates *K1* and *K2*. The inputs *I1–I6* are the functional inputs, and *K1* and *K2* are the *key-inputs* connected to the key-gates. On applying the correct values of the keys (K1 = 0 and K2 = 1), the design will produce a correct output; otherwise, it will produce a wrong output.

EPIC [1] incorporates logic encryption into the IC design flow, as shown in Fig. 3.2. In the untrusted design regime, the IC is encrypted, and its functionality is not revealed. Post-fabrication, the IP vendor activates the encrypted design by applying the valid key. The key is stored in a tamper-evident memory inside the design to prevent access to an attacker.

Logic encryption prevents attacks such as piracy and hardware Trojans. Since the design is encrypted by the designer, the foundry cannot use any copies or overproduce

[1] Researchers have previously used the terms "logic obfuscation" [1, 2] and "logic locking" [3] for this purpose.

J. Rajendran (✉)
Department of Electrical Engineering, The University of Texas at Dallas,
800 W Campbell Road, Richardson, TX 75080, USA
e-mail: jv.ee@utdallas.edu

S. Garg
New York University, New York, NY, USA
e-mail: siddharth.garg@nyu.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_3

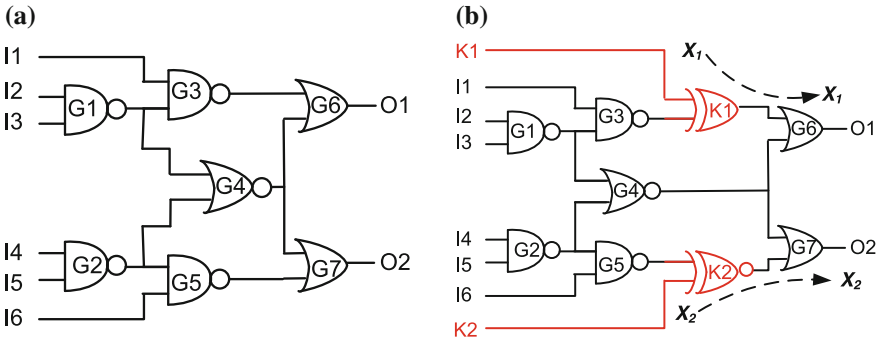


Fig. 3.1 **a** Original circuit. **b** A circuit encrypted using two key-gates K1 and K2 based on the technique proposed in [1]. By applying the input pattern 100000, an attacker can sensitize key-bits K1 and K2 to the outputs O1 and O2

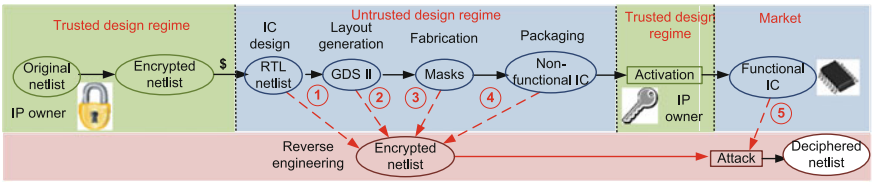


Fig. 3.2 The top *blue* box represents the EPIC design flow [1]. The design is in the encrypted form in the untrusted design regime. In the untrusted regime, an attacker can obtain the encrypted netlist from (*1*) the IC design or by reverse engineering the (*2*) layout, (*3*) mask, or (*4*) a fabricated IC, and (*5*) the functional IC from the market. Using this attack, the attacker can get a deciphered netlist and make pirated copies

ICs without the secret keys. Furthermore, it prevents an attacker from analyzing the structural behavior of the design, thereby hindering Trojan insertion.

Outline of the chapter. Section 3.2 explains the protocol on how logic encryption can be used in an IC supply chain, and Sect. 3.3 defines the threat model for logic encryption, listing out the capabilities of the attackers and their limitations. Section 3.4 lists the different security properties and metrics for logic encryption. Since its introduction in 2008, several attacks have been introduced against logic encryption. Section 3.5 details a set of attacks that enables an attacker to learn the correct outputs of the design, even when the design is subjected to logic encryption. In another set of attacks explained in Sect. 3.6, an attacker can learn the correct key used for logic encryption, by observing input–output pairs. Additionally, this section also explains the set of countermeasures against this class of attacks. Recently, new vulnerabilities due to an untrusted test facility have undermined the security of logic encryption. Unlike the previous attacks, this class of attacks relies only on the test patterns and responses. Section 3.7 details this class of attacks. Finally, there are several

techniques that rely on the security offered by logic encryption, which are explained in Sect. 3.8. Section 3.9 concludes this chapter by comparing different attacks and their countermeasures.

3.2 Protocol

The protocol for logic encryption is as follows [1].
Step 1: The designer encrypts the design with a common key, *CK*. This is the key shown in Fig. 3.2. On applying CK, the encrypted design produces correct outputs. The target IC is then designed along with the encrypted design, a public key cryptographic algorithm (e.g., RSA) and an on-chip random number generator. The IC also has a master public key (Master-Pub). Master-Pub's private pair is master private key (Master-Pri), which is not stored on-chip.
Step 2: The designer sends this design to the untrusted foundry, where the chip is manufactured. The manufactured chip is then sent to the designer.
Step 3: The designer activates the on-chip random number generator to generate a public–private key pair, RCK-Pub and RCK-Pri, respectively. RCK-Pub is known to everyone. The designer encrypts CK with MK-Pri and RCK-Pub. The resultant ciphertext is called input key (IK).
Step 4: IK is given to the user of this chip. The user applies IK to the chip to activate it. The public key cryptographic module within the chip decrypts IK with MK-Pub and RCK-Pri to obtain CK. The on-chip infrastructure applies CK to the encrypted design and makes the IC functional. This process is called activation. Since IK is encrypted with public–private key pairs, it does not reveal CK. Thus, the user may not be able to unlock (i.e., reverse engineer) the design, even if he is able to unlock his chip (i.e., make the IC functional).

Identifying MK-Pri and RCK-Pub will help an attacker to overproduce the ICs. An attacker does not need design to a new mask. He can reuse the existing mask and unlock them by determining their IKs using MK-Pri and RCK-Pub. Identifying CK will help an attacker to pirate the design and identify "safe" places in a design to insert Trojans[2]. An attacker can extract the protected design (without the public key cryptographic algorithm and on-chip random number generator), unlock it using CK, create a new mask, and manufacture the pirated ICs. For the rest of this chapter, we will consider how an attacker can extract CK. We refer to CK as the "key."

3.3 Threat Model

The attacker can be either in the foundry or be the end user. The objective of the attacker is to determine the secret keys used for logic encryption. By determining the keys, he/she can decipher the functional netlist, make pirated copies, and sell

[2]Safe places in a design in the context of hardware Trojans refer to circuit nodes with low observability, low controllability, minimal impact on power and delay [7, 8]. places to insert Trojans.

them illegally, thereby defeating the purpose of logic encryption. Furthermore, with the knowledge of the keys, he/she can analyze the structural behavior of the design, thereby inserting Trojans at "safe places."

The attacker needs the encrypted netlist and a functional IC. He/she can obtain the encrypted netlist from (1) the IC design, or by reverse engineering the (2) layout, (3) mask, or (4) a manufactured IC as shown in Fig. 3.2. The functional IC, (5) in Fig. 3.2, is bought in the open market.

3.4 Security Properties and Metrics

In the rest of this chapter, we focus on security of logic encryption assuming that the common key (CK) is the most critical security asset. Indeed, if CK is compromised, the attacker learns the IC's intended functionality, thus compromising the designer's IP and opening the door to hardware Trojan insertion. Furthermore, we note that even though the EPIC protocol does protect against overbuilding even if CK is compromised (each chip is activated only after the designer supplies the chip-specific IK), a determined attacker can still overbuild ICs using a new mask in which CK is hardwired to its compromised value. Such an attack is easily within the range of capabilities of a foundry attacker.

To thwart these attacks, a logic encryption technique has to satisfy the following properties:

1. **Correctness.** A logic encryption technique should produce a correct output upon applying the correct key. If it produces an incorrect output upon applying the correct key, the encrypted IC/design will violate the design specification, and the design will be considered "defective."
2. **Resilient against output-guessing attacks.** A logic encryption technique should prevent an attacker from guessing the correct outputs from previously observed input–output pairs. To thwart such attacks, the output entropy upon applying the wrong key should be maximized. In other words, the Hamming distance between the outputs of the design upon applying the incorrect key should be 50%, as this value maximizes the entropy [4].
3. **Entanglement.** One should not be able to remove the key-gates from the protected circuit. Otherwise, an attacker can remove the key-gates, analyze the unprotected components, and obtain the original design.
4. **Resilient against key-guessing attacks.** A logic encryption technique should prevent an attacker from guessing the correct key value from previously observed input–output pairs. To thwart such attacks, key-gates should be inserted such that the number of input–output pairs required to obtain the key value is exponential to the size of the key.
5. **Overhead.** The logic encryption technique should aim to minimize area, delay, and power overheads, but not at the expense of the security objectives listed above.

3.5 Thwarting Output-Guessing Attacks

In logic encryption, different combinational logic elements are inserted in a circuit to conceal the functionality of a design. These elements can be XOR/XNOR gates [1, 4, 5, 9], AND/OR gates [6], MUXes [3, 4], or a combination of these elements [10]. One of the inputs to these gates serves as a key-input, which is a newly added signal driven by a tamper-evident on-chip memory. Unless the correct key is loaded onto the on-chip memory, a design will not work correctly. The activation of a encrypted IC can be conducted either prior to or after the manufacturing test. Secure communication infrastructure is needed if the keys are to be loaded remotely onto the chip [9, 11].

EPIC [9] is a logic encryption framework which inserts XOR/XNOR gates, referred as key-gates, such that these gates have minimal impact on circuit delay. One can configure these gates as buffers or inverters using these key-inputs. The insertion of the gates is done after logic synthesis and before physical synthesis. The design can then be resynthesized. If the key-gates were left as such without any other modifications to the circuit, the key-bits could be extracted by inspecting if a key-gate is XOR or XNOR. To eradicate such a simple deduction analysis of the key-gate types and the key values, the netlist can be synthesized such that the XOR/XNOR key-gates are replaced with other gates like AND/OR/NAND, or the inverters in the design can be moved around to change the polarity of the key-gates.

Since EPIC inserts the key-gates based on only delay overhead as a constraint, there is no guarantee that an incorrect key produces incorrect output for all the input patterns. Rajendran et al. [4, 5] developed a method, based on the principles of VLSI testing, that inserts XOR/XNOR gates or MUXes to achieve controllable corruption of the output bits. This technique maximizes the Hamming distance between the correct output and the incorrect outputs on applying a random incorrect key.

Testing principle-based insertion of key-gates Key-gates should be inserted in such a way that any wrong key causes a wrong output. This is similar to the situation where a circuit produces a wrong output when it has a fault that has been excited and propagated to the outputs. The following observations relate logic encryption and fault analysis in IC testing. These observations are used to insert XOR/XNOR gates.

Fault excitation: Application of a wrong key can be associated with the activation of a fault. For a wrong key, either a stuck-at-0 (s-a-0) or a stuck-at-1 (s-a-1) fault will get excited when key-gates are used for encryption.

Consider the C17 circuit (from the ISCAS'85 benchmark set) encrypted with one XOR gate (E1) as shown in Fig. 3.3(b). Here, E1 is the key-gate. 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 an s-a-1 (when A = 0) fault at the output of G7 as shown in Fig. 3.3(a). Please note that s-a-0 (s-a-1) fault activation can be attributed to the case where the net in question is supposed to yield a value of 1 (0) during the functional mode of operation.

Fault propagation: Not all wrong keys can corrupt the output as the effects of a wrong key may be blocked for some of the input patterns. This is similar to

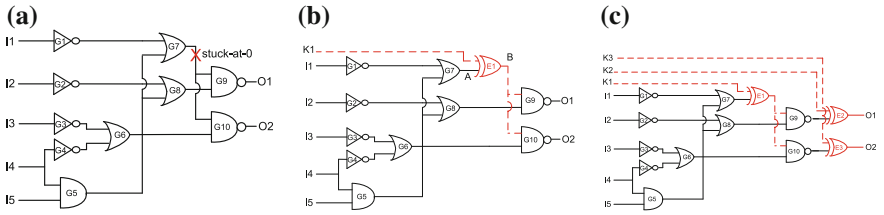


Fig. 3.3 Relation between logic encryption and IC testing: **a** fault excitation, **b** propagation, and **c** masking

the scenario where not all input patterns can propagate the effect of a fault to the output [12].

Consider the circuit shown in Fig. 3.3(b). Let a wrong key (K1 = 1) be applied to the circuit. For the input pattern 00000, an s-a-0 fault gets excited at the output of E1 and propagates to both outputs. The value at the output of E1 is 0 instead of 1, and the output is 11 instead of 00. For the input pattern 01110, even though the s-a-0 fault gets excited at the output of E1, the output is 11, which is the correct output, as the fault effects have been blocked.

To propagate the effect of an excited fault, in our case the wrong key, non-controlling values should be applied to the other inputs of the gates that are on the propagation path of the fault. Since not all input patterns guarantee the non-controlling values on the fault propagation path, a wrong key will not always corrupt the output.

Fault masking: Inserting a single key-gate and applying a wrong key are equivalent to exciting a single stuck-at fault. Likewise, inserting multiple key-gates and applying a wrong key are equivalent to simultaneously exciting multiple stuck-at faults. However, when multiple faults are excited, they might mask one another. Therefore, in logic encryption, when multiple key-gates are inserted, the effect of one key-gate might mask the effect of other key-gates.

Consider the encrypted circuit shown in Fig. 3.3(c). When the key-bits 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 gate, E1, is masked by the XOR gates E2 and E3. Consequently, the design produces the correct output, 00. Similar to fault masking in IC testing, the effect of one XOR gate is masked by the effect of the other two XOR gates.

Fault impact. To insert an XOR/XNOR as a key-gate, one needs 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, one uses fault impact defined by Eq. 3.1. From a set of test patterns, one can compute the number of patterns that detect the s-a-0 fault (NoP_0) at the output of a gate Gx and the total number of output bits that get affected by that s-a-0 fault (NoO_0). Similarly, NoP_1 and NoO_1 for s-a-1 faults are computed.

$$\text{Fault impact} = (NoP_0 \times NoO_0) + (NoP_1 \times NoO_1) \tag{3.1}$$

By inserting an XOR/XNOR key-gate 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), indirectly enabling the logic encryption technique to reach the 50% Hamming distance metric.

Improving fault analysis-based insertion. XOR/XNOR key-gates are combined with MUX key-gates to achieve a Hamming distance closer to 50% [10]. Dupuis et al. [6] propose a technique that inserts AND/OR key-gates to minimize the number of low controllability locations in a circuit, making it difficult to insert hardware Trojans in the circuit.

3.6 Key-Guessing Attacks

Multiple attacks have been presented against existing logic encryption techniques. The objective of an attacker is to figure out the key used for encryption of the circuit [3, 10, 13]. These attacks assume that the attacker has access to an encrypted netlist and a functional IC, on which one can apply inputs and observe outputs. There are two main types of attacks — key propagation and SAT attacks — that are described below:

3.6.1 Key Propagation Attacks [13]

The value of an *key-bit* can be determined if it can be sensitized[3] without being masked/corrupted by the other key-bits and/or inputs. By observing the output, the value of sensitized key-bit can be determined, given that other key-bits (similar to unknown X-sources[4]) do not interfere with the sensitized path.

Once an attacker determines an input pattern that sensitizes the key-bit to an output without any interference, it is applied to the functional IC, i.e., the IC with the correct keys. Now, this pattern will sensitize the correct value of the key-bit to an output. An attacker can observe this output and resolve the value of the key.

Example: Consider the key-input K1 in Fig. 3.1. It will be sensitized to output O1 if the value at the other input of gate G6 is 0 (non-controlling value for an OR gate). This can be achieved by setting I1 = 1, I2 = 0, and I3 = 0. As the attacker has access to the functional IC, he/she can apply this pattern and determine the value of

[3] 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 non-controllable values of the gates) which surjectively maps l to O and thus renders any change on l observable on O.

[4] X-sources: Uninitialized memory units, bus contentions, or multicycle paths are the source of unknown response bits, i.e., unknown-Xs in testing. They are non-controllable.

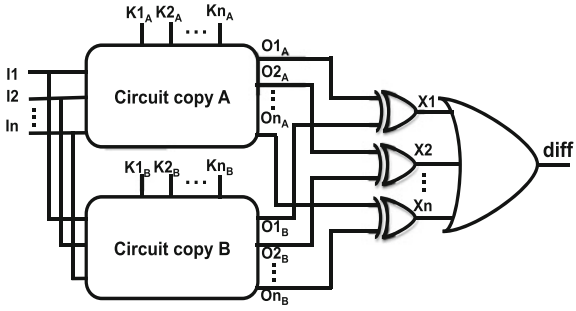


Fig. 3.4 Miter-like circuit to determine DIPs [13]

K1 on O1. For example, if the value of O1 is 0 for that input pattern, then K1 = 0; otherwise, K1 = 1.

3.6.2 Boolean Satisfiability (SAT) Attacks [13, 14]

The SAT attack iteratively rules out incorrect key values using distinguishing input patterns (DIPs). A distinguishing input pattern X_d is an input value for which at least two different key values, $k1$ and $k2$, produce differing outputs, $o1$ and $o2$, respectively. Since $o1$ and $o2$ are different, at least one of the key values or both of them are incorrect. It is possible for a single DIP to rule out multiple incorrect key values.

The DIPs are found by constructing a miter-like circuit as illustrated in Fig. 3.4. The primary inputs are common to the two copies of the encrypted circuit, while the key-inputs are left independent. The corresponding outputs of the two circuits are XORed and then ORed to generate *diff* signal. The conjunctive normal form (CNF) of the resultant circuit is generated and passed to a SAT solver. The SAT solver finds a DIP X_d for which *diff* = *1*, i.e., the outputs of the two circuits are different. X_d is applied to the functional IC, and correct output I_d is obtained. The input–output pair (X_d, I_d) is used to identify incorrect key values.

A single pattern may not rule out all incorrect keys. Hence, an iterative process is used in the SAT attack, as shown in Fig. 3.5. A new pair (X_d, I_d) is added to the SAT formula in each iteration, and the SAT formula is updated. The generated DIP is applied to the functional IC, and the set of keys that results in an incorrect output is eliminated. The attack is successful when no further DIP is found, which implies that all incorrect key values have been pruned. **Example.** Let us consider the application of the SAT attack on the encrypted example circuit in Fig. 3.6. Figure 3.7 presents the output of the original circuit in column Y and the output of the encrypted circuit for different key values in the following columns. For three key-inputs, there are eight possible key values, which are represented as k0, k1,..., k7. When the SAT attack is launched on the encrypted circuit, it takes four DIPs to identify the correct key [13]. In iteration 1, the DIP 011 is used. For this DIP, the key value k4 alone

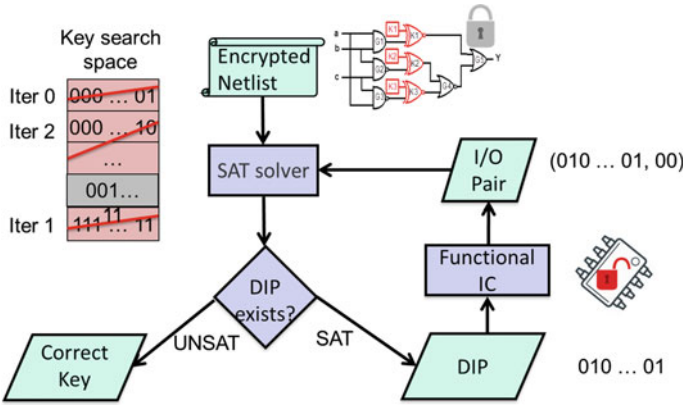


Fig. 3.5 SAT attack on logic encryption [13]

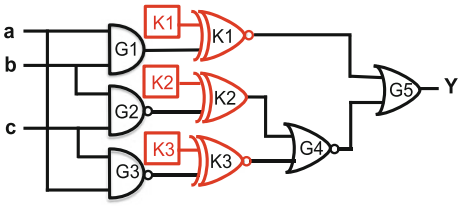


Fig. 3.6 Logic encryption using XOR/XNOR gates [1]. The correct key value is 110. This technique is vulnerable to SAT attack [13]

					Output Y for different key values								
No.	**a**	**b**	**c**	**Y**	**k0**	**k1**	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**	**Pruned key values**
0	0	0	0	0	1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	1	1	1	1	0	1	
2	0	1	0	0	1	1	1	1	1	1	0	1	
3	0	1	1	1	1	1	1	1	0	1	1	1	iter 1: k4
4	1	0	0	0	1	1	1	1	1	1	0	1	iter 4: all incorrect
5	1	0	1	1	1	1	1	1	1	1	1	0	iter 3: k7
6	1	1	0	1	1	1	0	1	1	1	1	1	
7	1	1	1	1	1	0	1	1	1	1	1	1	iter 2: k1

Fig. 3.7 Analysis of the SAT attack against logic encryption [13]. Columns k0–k7 show the encrypted circuit's output for different key values. *Red* entries in each row denote an incorrect output. The correct key is k6

produces a wrong output as highlighted in red. Thus, only one incorrect key is ruled out in the first iteration. In the second and third iterations, key values k1 and k7 are ruled out, using the patterns 111 and 101, respectively. The pattern 100, used in the

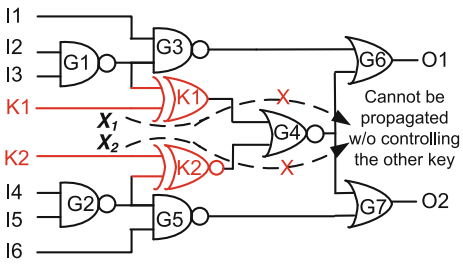


Fig. 3.8 A circuit encrypted using two key-gates K1 and K2 based on the technique proposed in [15]. This prevents key propagation attacks

fourth iteration, eliminates all incorrect keys and the attack successfully identifies the correct key as k6.

The attack could have succeeded in the first iteration with a single DIP 100, if this input pattern was tried first. Thus, the execution time of the attack depends on the order in which the input patterns are applied for the SAT attack. The SAT attack, however, chooses the DIPs arbitrarily [13]. The larger the number of incorrect key values ruled out per DIP, the fewer the patterns needed for the attack, which implies a smaller execution time of the attack algorithm.

3.6.3 Countermeasures to Attacks

Countermeasures to Key Propagation Attacks

In order to thwart key propagation attacks, key-gates are inserted such that an attacker cannot propagate the output of a single key-gate [15]. This way, the observed output value is a function of multiple key-gates. Key-gates are inserted such that their sensitization path is blocked by each other. Such key-gates form a "clique." As the size of the clique increases, the attacker's effort increases.

Example. Consider the circuit shown in Fig. 3.8 which is the same functional circuit shown in Fig. 3.1, but the two key-gates *K1* and *K2* are at different locations. Here, if the attacker has to propagate the effect of either of the keys, then one has to force a "0" (non-controlling value of NOR gates) on the other input of G4. In order to force this value, one has to control the key-inputs, which are inaccessible. Thus, one cannot propagate the effect of a key to an output, failing to determine the values of the key.

To break this scheme, the attack proposed in [10] targets the logic cones with the smallest number of key-inputs and recovers the secret key by employing brute force. The process is then repeated for the remaining logic cones in the circuit, sorted in an increasing order by the logic cone size. To increase the complexity, the number of MUX key-gates is increased to increase the size of the logic cone.

Countermeasure to SAT-Based Attacks [16]

To thwart this attack, clique-based insertion is used along with a cryptographic primitive called *one-way random functions (ORFs)* [17]. ORFs, such as AES with a fixed

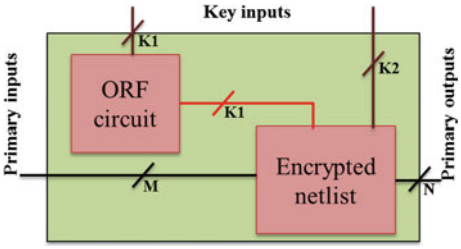


Fig. 3.9 ORF-based countermeasure against SAT-based attacks. K1 out of K key-inputs in the encrypted netlist is connected to the ORF circuit

secret key, prevent an attacker from determining the inputs from the output [18]. First, the designer synthesizes an AES design with a fixed secret key (unknown to the attacker). The resultant design implements a random function. Then, he applies a randomly selected input to the AES with the fixed secret key, which serves as the key for logic encryption. The output of the AES (with fixed secret key) is connected to a subset of XOR/XNOR key-gates added for logic encryption. The designer knows the fixed secret key to the AES and the input applied to AES, and he can configure the key-gates as XOR/XNORs accordingly. This technique is illustrated in Fig. 3.9. The original netlist is encrypted with $K = K1 + K2$ number of key-bits. K1 key-inputs of the encrypted netlist are connected to the output of the AES (with fixed secret key) circuit, and the remaining K2 key-inputs are connected to the on-chip memory.

This modified scheme will now withstand the SAT attack. A property of the AES is that it is computationally infeasible to determine the inputs of AES from its outputs when the key is unknown [18]. Thus, one cannot backtrace from the outputs of the design and determine the inputs to the AES. In other words, the input to the AES is the secret key for logic encryption.

The limitation of this scheme is it assumes the function-to-be-protected is an unknown logic. Thus, it cannot protect against known functions, because an attacker can "carve out" the logic implemented by the known function. It also assumes that the AES with a fixed key implements a random function. When this random function is cosynthesized with the target unknown function, an attacker cannot classify whether a given gate in the resultant design is part of the random function or the function-to-be-protected.

3.7 Impact of Testing on Logic Encryption

3.7.1 Motivation

Each fabricated chip goes through a manufacturing test that screens out the defective chips. Design for testability (DfT) engineers target generating test patterns that maximize the fault coverage, minimize test pattern count, and reduce test power consumption [19]. The state-of-the-art logic encryption frameworks pursue two different

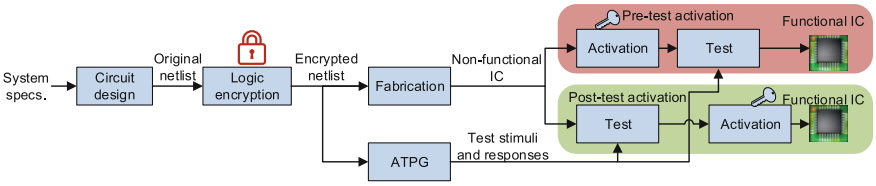


Fig. 3.10 Logic encryption and IC activation in the IC design flow. Pretest activation and post-test activation models

activation models, pretest and post-test activations, that differ in the time of activation of an IC with respect to the manufacturing test. The two models are illustrated in Fig. 3.10 and highlighted in red and green colors, respectively.

Pretest activation. The ICs are activated prior to the manufacturing test, typically conducted in the foundry or outsourced to an OSAT. Since the IP owner does not want to reveal the secret key to the untrusted foundry, on-chip public key cryptographic infrastructure [1] is used to load the secret key securely on the chip. On passing the manufacturing test, the ICs are shipped for assembly/sales directly from the foundry, which is useful in meeting time-to-market constraints.

Post-test activation. The ICs are activated after performing the manufacturing test. Either *remote activation* [20] or *in-house activation* [21] can be employed. In-house activation requires shipping of the encrypted IC from the foundry to the trusted facility, eliminating the need for on-chip cryptography.

Evolving business and threat models. Fabless and fab-lite are the evolving business models for semiconductor companies [22]. Fabless companies, such as Apple Inc., outsource IC fabrication (to Samsung and TSMC [23]), testing, and assembly services. Fab-lite companies such as TI [22] may outsource IC fabrication (to SMIC [24]) and testing, but conduct packaging and assembly in-house. Given the above business models, Apple may activate the ICs remotely using pretest activation, and TI may activate the ICs in-house using post-test activation. However, the test and security implications of these scenarios have never been studied.

3.7.2 Pretest Activation

In pretest activation, the secret key is loaded onto the IC prior to the manufacturing test. The manufacturing test can be conducted in the foundry or a separate test facility (OSAT [25]). Since an IP owner wants to protect the secret key from being exposed to either the foundry or the OSAT, he can load the secret key securely on the chip using public key cryptography infrastructure. Such infrastructure can incur significant area overhead [1].

As the test is to be conducted with the key in place, the secret key values are applied as constraints on the key-inputs during the test generation phase, which can impact the test quality and costs, as well as the security of logic encryption.

3.7.2.1 Threat Model

The attacker is a person in the foundry or test facility with access to the following:

1. An encrypted netlist E_K, which can be obtained by reverse engineering [26] or IP piracy [27].
2. Test stimuli T and responses Γ.

Impact on Security

To highlight the security vulnerabilities of pretest activation, we develop a *test data mining attack* that can reveal the secret key used in pretest activation of logic encryption.

Attack methodology. During the test pattern generation phase, a DfT engineer will apply the correct key K_{corr} as a constraint and obtain a set of test patterns that maximize fault coverage.

An attacker can therefore apply the test stimuli as input constraints and the test responses as output constraints and search for the potentially correct key K_P that maximizes the fault coverage under the specified constraints. The attack is an optimization problem: The objective is to maximize the fault coverage FC under the test stimulus T and test response Γ constraints, as follows:

$$\begin{aligned} &\text{maximize} \quad FC \\ &\text{subject to} \quad \underset{1 \le i \le N}{\forall} E_K(K_P, T_i) = \Gamma_i \\ &\text{solve for} \quad K_P \end{aligned} \tag{3.2}$$

The rationale for the attack to return the correct key is that the test patterns have been generated with the objective of maximizing the fault coverage in the presence of the correct key as a constraint. When the same set of test patterns are used as constraints, the key that maximizes the fault coverage will be the one that is used to generate the patterns.

Equation 3.2 formulates a system of Boolean equations which can be solved using techniques such as Boolean satisfiability (SAT) or integer linear programming (ILP) [28]. Test generation (ATPG) algorithms are capable of solving a system of Boolean equations while maximizing the fault coverage at the same time; ATPG is, therefore, a natural candidate for solving the optimization problem in Eq. 3.2. The complexity of the attack is NP-hard [29].

Let us consider the netlist shown in Fig. 3.8. When the correct key value $K_{corr} = 00$ is used as a constraint, eight test patterns are generated by the ATPG tool as listed in Table 3.1. An attacker will launch the attack described in Eq. 3.2 by applying the test stimuli and responses as constraints and search for the potential key K_P that maximizes the fault coverage. The only key K_P that maximizes the fault coverage and satisfies these test pattern constraints is 00, and the corresponding fault coverage is 82.43%. None of the other key values satisfies the test pattern constraints.

Table 3.1 Test patterns (pretest activation) for the netlist in Fig. 3.8. The correct key K_{corr} is used as a constraint during ATPG

Key(K_{corr})	Stimulus (T)	Response (Γ)
00	011001	10
00	101010	01
00	101111	01
00	011101	10
00	111010	11
00	000111	11
00	110001	00
00	001011	10

3.7.3 Post-test Activation

In post-test activation, the manufacturing test is conducted on an encrypted IC with *the rationale that manufacturing test is a "structural" test and that the chip need not be functional during the test*. The IC can be activated post-test in one of the following ways:

1. After manufacturing test, defect-free ICs are shipped to a trusted facility, activated by the IP owner, and shipped out for sale.
2. Tested ICs can also be activated remotely, similar to the case of pretest activation, via public key cryptography infrastructure [30].

Impact on security. In post-test activation, both test pattern generation and manufacturing test are conducted in the absence of the secret logic encryption key. Any analysis performed by the attacker will only reveal these arbitrary key values and not the secret key value. Therefore, post-test activation has no detrimental impact on the security of logic encryption.

3.7.4 Hill Climbing Attack

The hill climbing search-based attack [21] uses test data information to guess the secret key for pretest activated ICs. The attack tries to achieve zero Hamming distance HD_O between the test response and the encrypted circuit, for multiple random key guesses. The individual bits in the initial key guess are flipped if the flip minimizes the Hamming distance HD_O.

Example. Consider the circuit shown in Fig. 3.8 and the test patterns shown in Table 3.1. In iteration 1, the hill climbing attack starts with a random key value, say 00. With this key value, the test patterns are applied and the responses are collected. The cumulative Hamming distance between the collected responses and the responses from the test set is calculated. In this case, it is 12. Now, one of the key-bits will be flipped, say the new key is 01. The cumulative Hamming distance between the

collected responses and the responses from the test set is now reduced to 6. Since there is a reduction in the cumulative Hamming distance, the second bit is retained as 1. Now, the remaining bit is flipped (key = 00) and the cumulative Hamming distance is 0. Thus, the attacker identifies that 00 is the correct key.

3.8 Other Techniques Based on Logic Encryption

Logic encryption is used not only in the context of IP protection, but also in other applications as well. Two of them are described below:

3.8.1 Secure Split Test (SST) [20, 30]

An untrusted test facility can mark a defective IC, which failed the test, as a good quality one. Consequently, a designer unknowingly sells low-quality ICs, spoiling his reputation. Further, a test facility can also label "good" ICs as "faulty" ICs and sell them in the black market. Such an attack is possible only when the attacker (malicious tester) can identify whether an IC is defective or not. In order to prevent this attack, a designer should hide the test responses. For this purpose, the test infrastructure should be protected. Logic encryption aids in protecting the test infrastructure, thereby hiding the correct responses.

3.8.2 Securing Processor Architectures [31]

Modern processors are equipped with several security modules to aid detection and prevention of attacks. These modules usually inform the processor pipeline about the attack through a signal, which is carried by a wire. Unfortunately, such wires are susceptible to Trojan attacks, enabling an attacker to modify the target signal value at will. Detecting a Trojan that modifies a single wire is difficult. Consequently, he can launch traditional software attacks and still go undetected.

To prevent such attacks, logic encryption encrypts a processor module and stores the key within a security module. The security module, instead of sending the signal, sends the correct key to unlock a processor pipeline, when it does not detect an attack. When it detects an attack, it sends a wrong key, resulting in an incorrect computation. Since the size of Trojan required to mask/modify a multibit key is bigger than the one required to modify a single wire, such Trojans can be easily detected.

Table 3.2 A comparison of the attacks against logic encryption

Study	Attacker's location	Attacker's capabilities	Attack method	Defense
Rajendran et al. [15]	Foundry and end user	Encrypted netlist and an activated IC	Sensitization of key-bits to outputs	Clique-based insertion [15]
Subramanyan et al. [13]	Foundry and end user	Encrypted netlist and an activated IC	SAT-based algorithm to rule incorrect keys	Strong logic encryption OWF [16]
Plaza and Markov [21]	Foundry and test facility	Encrypted netlist and test set	Find the key that minimizes Hamming distance	Test-aware combinational logic encryption
Test data mining attack [32]	Foundry and test facility	Encrypted netlist and test set	Find the key that maximizes fault coverage	Post-test activation
SST [30]	Foundry	Activated ICs to be tested	Reduce yield	Encrypting test circuits

3.9 Conclusion

Logic encryption is an emerging technique to thwart IP piracy, reverse engineering, and hardware Trojans. Initially, most of the techniques proposed in the literature are based on VLSI testing principles. However, recent attacks have broken these techniques, even though the complexity of those techniques is NP-hard. Researchers are now trying to adopt concepts from cryptography and apply them to logic encryption.

Table 3.2 summarizes the different attacks and their countermeasures on logic encryption. Clique-based insertion of key-gates can prevent sensitization attacks but is susceptible to SAT attacks. Strong logic encryption can prevent both SAT and sensitization attacks, but it is applicable only for random unknown logic. Test data mining attack is applicable only when the attacker has access to test patterns and responses, which are generated for the correct key; they can be prevented by post-test activation.

References

1. Roy J, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In: Proceedings of the IEEE/ACM design, automation and test in Europe, pp 1069–1074
2. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(10):1493–1502
3. Plaza SM, Markov IL (2015) Solving the third-shift problem in IC piracy with test-aware logic locking. IEEE Trans Comput-Aided Des Integr Circuits Syst 34(6):961–971

4. Rajendran J, Zhang H, Zhang C, Rose G, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64(2):410–424
5. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Logic encryption: a fault analysis perspective. In: Proceedings of the IEEE/ACM design, automation and test in Europe, pp 953–958
6. Dupuis S, Ba P, Natale GD, Flottes M, Rouzeyre B (2014) A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: Proceedings of the IEEE international on-line testing symposium, pp 49–54
7. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: identifying and classifying hardware trojans. IEEE Comput 43(10):39–46
8. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25
9. Roy JA, Koushanfar F, Markov IL (2010) Ending piracy of integrated circuits. Comput 43(10):30–38
10. Lee Y-W, Touba N (2015) Improving logic obfuscation via logic cone analysis. In: Proceedings of the Latin-American test symposium, pp 1–6
11. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems, pp 196–203
12. Bushnell ML, Agrawal VD (2000) Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits. Kluwer Academic Publishers, Boston
13. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: Proceedings of the IEEE international symposium on hardware oriented security and trust, pp 137–143
14. Massad ME, Garg S, Tripunitara MV (2015) Integrated circuit (IC) decamouflaging: reverse engineering camouflaged ICs within minutes. In: NDSS
15. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Proceedings of the IEEE/ACM design automation conference, pp 83–89
16. Yasin M, Rajendran J, Sinanoglu O, Karri R (2015) On improving the security of logic locking. IEEE Trans Comput-Aided Des Integr Circuits Syst 99:1–1
17. Matsuzaki N, Tatebayashi M (1994) Apparatus and method for data encryption with block selection keys and data encryption keys. US Patent 5,351,299
18. Goldreich O (2001) Foundations of cryptography: basic tools, vol 1. Cambridge University Press, Cambridge
19. Bushnell M, Agrawal VD (2000) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits, vol 17. Springer, New York
20. Contreras G, Rahman M, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems, pp 196–203
21. Plaza SM, Markov IL (2014) Protecting integrated circuits from piracy with test-aware logic locking. In: Proceedings of the IEEE/ACM international conference on computer-aided design, pp 262–269
22. McLellan P (2013) A brief history of the foundry industry, part 2, [Sep 1, 2015]. https://www.semiwiki.com/forum/content/2109-brief-history-foundry-industry-part-2-a.html
23. AppleInsider (2015) Samsung reportedly nabs 75% of Apple's next-gen 'A9' SoC orders, [Aug 10, 2015]. http://appleinsider.com/articles/15/01/26/samsung-to-reportedly-take-75-of-apples-nextgen-a9-soc-orders
24. Releases SP (2014) SMICs Beijing fab wins TI quality excellence award, [Aug 10, 2015]. http://www.smics.com/eng/press/press_releases_details.php?id=107870
25. Wire B (2014) Research and markets: outsourced semiconductor assembly and test market (OSAT) trends, [Aug 22, 2015]. http://www.businesswire.com/news/home/20140324005628/en/Research-Markets-Outsourced-Semiconductor-Assembly-Test-Market
26. Torrance R, James D (2011) The state-of-the-art in semiconductor reverse engineering. In: Proceedings of the IEEE/ACM design automation conference, pp 333–338

27. Rostami M, Koushanfar F, Karri R (2014) A primer on hardware security: models, methods, and metrics. Proc IEEE 102(8):1283–1295
28. Clarke E, Gupta A, Kukula J, Strichman O (2002) SAT based abstraction-refinement using ILP and machine learning techniques. In: Proceedings of the computer aided verification, Springer, pp 265–279
29. Krishnamurthy B, Akers SB (1984) On the complexity of estimating the size of a test set. IEEE Trans Comput 33(8):750–753
30. Rahman MT, Forte D, Shi Q, Contreras GK, Tehranipoor MM (2014) CSST: preventing distribution of unlicensed and rejected ICs by untrusted foundry and assembly. In: Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems, pp 46–51
31. Rajendran J, Kanuparthi AK, Zahran M, Addepalli SK, Ormazabal G, Karri R (2013) Securing processors against insider attacks: a circuit-microarchitecture co-design approach. IEEE Des Test 30(2):35–44
32. Yasin M, Saeed SM, Rajendran J, Sinanoglu O (2016) Activation of logic encrypted chips: pre-test or post-test?. In: Proceedings of the IEEE/ACM design, automation and test in Europe

Chapter 4
Gate Camouflaging-Based Obfuscation

Xueyan Wang, Mingze Gao, Qiang Zhou, Yici Cai and Gang Qu

4.1 Circuit Camouflaging with Configurable Gates

One of the greatest threats to VLSI design intellectual property (IP) is reverse engineering [1]. ***Reverse engineering*** (RE) is the process of extracting the IP and design information from the target product and reproducing the product [2, 3]. Motivations of RE vary from the paranoia of the Cold War, through commercial piracy, to competitive intelligence, and courts of patent law. The targets of RE include systems as large as an aircraft or as small as a microchip, programming codes, a pill of medical drug, or any sort of IPs [2]. In the semiconductor sector, RE has become a powerful tool for IP piracy where the attacker analyzes a design and reproduces it with no or much less investment in research and development. These low cost illegitimate products bring security vulnerabilities to critical commercial and military systems, or they can be sold at a much lower price, giving them an unfair competitive edge against the authenticated products.

The popular digital circuit watermarking and fingerprinting techniques [1] are passive IP protection schemes because they do not prevent RE from happening or make it more difficult. Watermark and fingerprint can be embedded into the IP to make each instance of the IP unique. When necessary, they can be revealed to show the authorship or ownership of the IP and identify the parties that misuse the IP. Although it is hard or impossible to completely remove the watermark and fingerprint, RE attackers can still extract valuable information from the IP and reproduce the IP sillegally. The existence of watermark and fingerprint in the IP can deter RE attacks, but will not increase the complexity of RE.

X. Wang · Q. Zhou · Y. Cai
Tsinghua University, Beijing, People's Republic of China

M. Gao · G. Qu (✉)
University of Maryland, College Park, MD, USA
e-mail: gangqu@umd.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_4

Table 4.1 The configurable CMOS cell in Fig. 4.1 can perform three different functions: NAND, NOR, or XOR, based on different true and dummy contact combinations [7]

Function	Contacts	
	TRUE	DUMMY
NAND	2, 4, 6, 8, 11, 12, 16, 17	1, 3, 5, 7, 9, 10, 13, 14, 15, 18, 19
NOR	2, 5, 6, 11, 12, 18, 19	1, 3, 4, 7, 8, 9, 10, 13, 14, 15, 16, 17
XOR	1, 3, 4, 7, 9, 10, 12, 13, 14, 15, 18, 19	2, 5, 6, 8, 11, 16, 17

Gate camouflaging techniques have emerged as an effective countermeasure for RE attacks [4–7]. These techniques rely on the general belief that RE technology is normally 2–3 generations behind the latest CMOS design technology. That is, certain CMOS design features cannot be completely reverse engineered until several years later. For example, some logic cells can be configured to perform different functionalities while maintaining an identical look to RE attackers. In circuit camouflaging, conventional logic gates are intentionally replaced by these configurable CMOS cells to thwart RE attacks.

4.1.1 Configurable CMOS Cells

One popular approach to construct configurable CMOS cells is to use the true and dummy contacts [4, 5]. A true contact spans the dielectric between two adjacent layers and represents an electrical connection, while a dummy contact has a gap in the middle thus fakes the connection between layers. Figure 4.1 shows an example where 19 contacts are utilized in the configurable CMOS cell [7]. As demonstrated in Table 4.1, with different combinations of true and dummy contacts, the camouflaging gate can perform three different logic functions: NAND, NOR, or XOR. For instance, when contacts 2, 4, 6, 8, 11, 12, 16, 17 are true and contacts 1, 3, 5, 7, 9, 10, 13, 14, 15, 18, 19 are dummy, the camouflaged CMOS cell performs the functionality of a NAND gate.

When an attacker performs the top-down image processing-based RE attack, he is unable to detect whether a contact is true or dummy, because from the top view of the chip, the true and dummy contacts appear identical even under most powerful optical and electrical microscopes. Therefore, this configurable CMOS cell will appear the same look to the attacker regardless of the functionality it implements. Without knowing the functionalities of the camouflaged gates, the attacker will fail to reverse engineering the IP. Of course, the attacker can guess and try all the different possible configurations, which will increase the complexity of the RE attack. This also implies that against such brute force attack, the more camouflaged gates we have in the IP and the more functionalities a camouflaged gate can achieve, the more difficult it will be for attackers to recover the IP.

4.1.2 Circuit Camouflaging Technique

The circuit camouflaging technique proposed in [7] obfuscates a circuit by disguising the functionality of selective XOR, NAND, or NOR gate behind the configurable CMOS cell illustrated in Fig. 4.1. That is, each camouflaged gate will have the same appearance but three possible functionalities to an RE attacker. In the circuit shown in Fig. 4.2, two logic gates have been replaced by the configurable CMOS cells C1 and C2. Even when an RE attacker has successfully recovered the layout of the circuit, he cannot rebuild the circuit unless he knows the types of the two camouflaged cells C1 and C2.

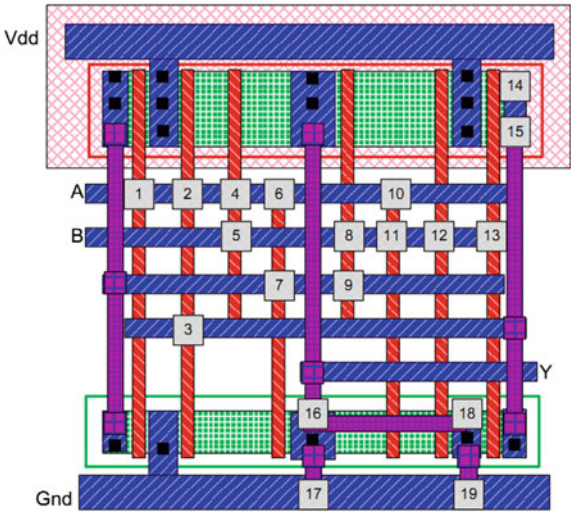


Fig. 4.1 A configurable CMOS cell with 19 contacts that can be configured to be either true or dummy [7]

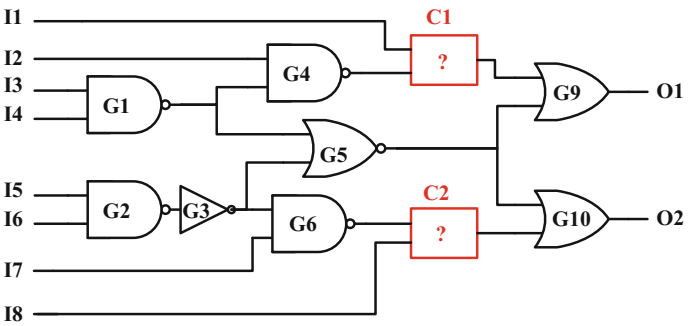


Fig. 4.2 A circuit with two camouflaged gates C1 and C2, which can be resolved individually by VLSI testing-based attack

Intuitively, the attacker has to guess $3^2 = 9$ possibly combinations, because each of the two camouflaged cells can be an XOR, NAND, or NOR gate. Moreover, since the attacker can only give input values to the circuit's primary inputs (PI) and observe the corresponding primary output (PO) values to verify whether a guess is correct or not. This seems to make attacker's job very challenging. Unfortunately, this is not the case.

In the above example, based on the fact that (i) the output of a camouflaged gate under input '00' can differentiate {XOR} from {NAND, NOR} (XOR outputs 0, while both NAND and NOR output 1), and (ii) the output under input '01' or '10' can differentiate {NAND} and {NOR} (NAND outputs 1, while NOR outputs 0), the attacker can apply the input pattern '010XXXXX' (X represents do not care values) at the PIs, this justifies the camouflaged gate C1's inputs as '00', and sensitize C1's output to PO O1. If O1 is 0, the functionality of C1 is resolved to be XOR. Otherwise, when O1 is 1, C1 will be either NAND or NOR. The attacker will then apply input pattern '110XXXXX' at PIs to justify C1's inputs as '10' and sensitize C1's output to O1. If O1 is 0, C1 is resolved to be NOR, otherwise C1 is resolved to be NAND.

This is known as ***VLSI testing-based attack***, where the attacker resolves a camouflaged gate's functionality by justifying the gate's inputs to certain values from the circuit's PIs then sensitizing the gate's corresponding output to PO to observe from a functional IC. In such attack, it is not necessary to obtain the entire truth table to resolve a camouflaged gate, a couple of selective input–output pairs will be sufficient. Thus, it is a very effective way to attack circuit camouflaging.

4.1.3 Enhanced Circuit Camouflaging

In the above example, we see that randomly selecting gates to camouflage is vulnerable to the VLSI testing-based attack. This can be fixed by judiciously selecting candidate gates to camouflage such that these gates will be interfered and cannot be revealed one by one [7].

Two camouflaged gates are ***interfered*** if one gate lies on a path between the other gate and an output, or the outputs of these two gates go into the same gate. Clearly, in Fig. 4.3, the three camouflaged gates C1, C2, and C3 are interfered. If gate G4 is also camouflaged, it will not interfere with C3, but it will interfere with C2 because their outputs meet at gate G6.

On the other hand, when camouflaged gates do not have any path in the circuit interfering with other camouflaged gates, they are called ***isolated***. Isolated camouflaged gates can be resolved independently by VLSI testing-based attack as we have seen in Fig. 4.2. However, this will not be the case when camouflaged gates are interfered.

For example, in Fig. 4.3, none of the camouflaged gates C1, C2, and C3 can be resolved by VLSI testing-based attack individually: C1's output cannot be observed from any of the POs without resolving C2 and C3 first; C3's inputs cannot be controlled before C1 and C2 are resolved; both the controllability of C2's inputs and the observability of its output rely on the functionalities of C1 and C3.

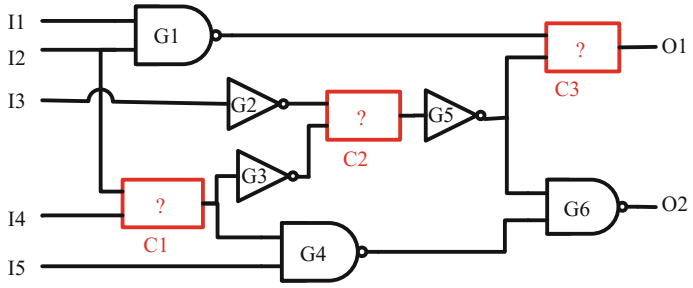


Fig. 4.3 A camouflaged circuit where the three camouflaged gates C1, C2, and C3 are interfered

In [7], it is argued that this will force the attackers to brute force search all the possible functionality combinations of these camouflaged gates. Specifically, for each possible combination, the attacker will simulate input patterns at PIs of the circuit to get the corresponding outputs at POs and compare them with an unpackaged/functional circuit. If they are not the same, the guess is incorrect and the attacker will check the next possible combination. Considering that each camouflaged gate has 3 possible functionalities, the needed brute force efforts will be 3^3. When the circuit has N selective camouflaged gates, the complexity will be 3^N. This exponential complexity leads to the claim that when there are sufficient number of interfered camouflaged gates, the circuit camouflaging is secure [7]. To end this section, we demonstrate by an example that this claim is not accurate, which motivates us to propose more accurate metrics to define the security of circuit camouflaging.

4.1.4 Defeating the Enhanced Circuit Camouflaging

Figure 4.4 is a sub-circuit of the secure camouflaged circuit in Fig. 4.3, where the two camouflaged gates C1 and C2 are clearly interfered with each other. We now show how both camouflaged gates can be revealed with no more than four input–output pairs.

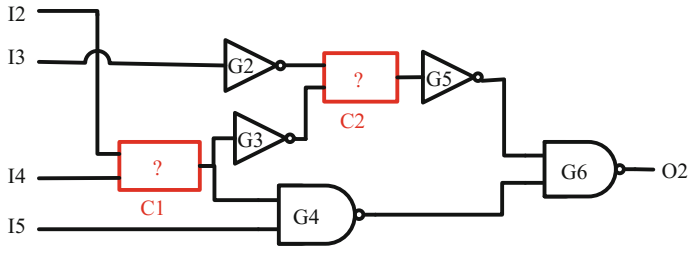


Fig. 4.4 A sub-circuit of the secure camouflaged circuit in Fig. 4.3

We consider the output O2 as a function of inputs I2, I3, I4, and I5 and denote it by O2(I2, I3, I4, I5). That is, when input values are I2 = 1, I3 = 1, I4 = 0, and I5 = 0, for example, the output value can be written as O2(1,1,0,0). Here is how an attack can reveal C1 and C2 easily:

(1) Apply (0, 1, 0, 1) as the input values for (I2, I3, I4, I5);
(2) If O2(0, 1, 0, 1) = 0 : apply (0, 1, 1, 0) as the input;
(3) if O2(0, 1, 1, 0) = 1 : apply (0, 0, 1, 0) as the input;
(4) If O2(0, 1, 0, 1) = 1 : apply (0, 1, 1, 1) as the input;
(5) apply (0, 1, 0, 0) as the input;
(6) if O2(0, 1, 0, 0) = 1 : apply (0, 0, 0, 0) as the input;

In step (1), we apply (0,1,0,1) as input to the circuit. If C1 is either NAND or NOR, C1(0,0) = 1; so G4(1,1) = 0 and we should be able to observe O2 = 1 regardless of the output of G5. Therefore, we conclude that if O2(0,1,0,1) =0, C1 must be XOR. Next, in order to reveal C2, we apply (0,1,1,0) in step (2). C1(1,1) = XOR (1,1) = 0, hence C2 will have both its input as 1 and it will output 0 only if C2 is XOR; otherwise, we need another input pattern to determine whether C2 is NOR or NAND, this can be done by for example using (0,0,1,0) as in step (3). Similar analysis can be done for steps (4)–(6) when C1 is either an NOR or an NAND.

Once we fully resolve the two camouflaged gates C1 and C2 in Fig. 4.4 (by applying no more than four input patterns), it will be trivial to resolve the last camouflaged gate C3 in Fig. 4.3 (with no more than two input patterns). Note that this takes us no more than six input–output pairs, much less than trying 3^3 possible combinations with multiple input–output pairs for each combination. We are able to do this because that the circuit can be partitioned to smaller sub-circuits and the set of {XOR, NAND, NOR} can be effectively distinguished. Next, we will elaborate an attack against circuit camouflaging based on circuit partitioning and then discuss several countermeasures to this attack.

4.2 Circuit Partition-Based Attack

It is believed that the enhanced IC camouflaging is secure because of the high brute force complexity that is exponential to the number of camouflaged gates [7]. However, this is just secure against the naïve brute force search. Like many other studies in security literature, a new type of attack would break a system that was previously proven secure.

As we have seen from the last example, an intelligent attacker does not need to resolve all the camouflaged gates together even though they are interfered. Instead, he may first partition the circuit to sub-circuits whose functions can be tested individu-

ally from a functional IC, and then perform a brute force search for the functionalities of the camouflaged gates in each sub-circuit individually. He can of course use some other smarter approaches that leverage the input–output difference of the potential gate types of the camouflaged gates.

The key idea of the circuit partition-based attack is to leverage the divide and conquer methodology to partition camouflaged gates into multiple sub-circuits, then target each sub-circuit individually. The benefit of circuit partition is breaking down the original large interference circles of camouflaged gates to multiple small interference circles in order to reduce the brute force complexity. Before elaborating circuit partition-based attack, we first give the definition:

Definition 1 The Maximum FanIn-Cone rooted at a primary output Z is defined as $MFIC_Z = \{G_i \mid \text{gate } G_i \text{ belongs to the circuit and there exists a path } G_i \rightarrow Z\}$, which is the set of all the gates whose outputs will directly or indirectly feed into the gate that generate output Z.

The word maximum in $MFIC_Z$ indicates that all the gates that will impact the value of output Z should be included. In another word, if a gate does not belong to $MFIC_Z$ for some output Z, then we will not be able to observe from Z any changes on that gate. Notice that in this chapter, unless it is specified otherwise, we will use $MFIC_Z$ for both the maximum fanin-cone rooted at the primary output Z and the corresponding sub-circuit that includes all these logic gates. For example, in the circuit shown in Fig. 4.3, we have $MFIC_{O1} = \{C1, C2, C3, G1, G2, G3, G5\}$, and $MFIC_{O2} = \{C1, C2, G2, G3, G4, G5, G6\}$, where the latter is the sub-circuit shown in Fig. 4.4.

Accordingly, MFIC's function, as a sub-circuit, can be studied by directly feeding the PIs of the MFIC and observing the corresponding output. A camouflaged gate, like other logic gates, may belong to multiple MFICs. For instance, five logic gates, including C1 and C2, belong to both $MFIC_{O1}$ and $MFIC_{O2}$ in the above example. Thus, when an attacker applies brute force attack to resolve the camouflaged gates, he can start with the MFIC that has the fewest number of camouflaged gates. In the above example, after he resolves C1 and C2 from $MFIC_{O2}$, $MFIC_{O1}$ will have only one camouflaged gate C3 left to solve. This greedy approach is the basic idea behind the following smart circuit partition-based attack shown in Algorithm 1.

In line 1, we partition the circuit into MFICs, which can be done with standard algorithm. In the rest of the algorithm, we iteratively resolve the camouflaged gates in one MFIC at a time. We greedily choose MFIC with the minimum number of unresolved camouflaged gates to apply brute force attack (lines 2–3) (for example in Fig. 4.3, there are three camouflaged gates in $MFIC_{O1}$ and two camouflaged gates in $MFIC_{O2}$, so we will start with $MFIC_{O2}$). This selection ensures that brute force efforts in current iteration can be minimized (line 7). The obfuscated netlist will then be updated by replacing the resolved camouflaged gates with the corresponding logic gates they implement (line 8). When there are multiple eligible MFICs with the same minimum number of camouflaged gates, the algorithm will choose the one that minimizes the maximum number of camouflaged gate number in the remaining MFICs (lines 4–6). The while loop in lines 9–12 checks whether there exist relevant

unresolvable camouflaged gates that will become resolvable when the obfuscated netlist is updated (in Fig. 4.3, C3 will become resolvable after C1 and C2 are resolved in $MFIC_{O2}$). If there is any, we resolve it by the VLSI testing principles-based attack [7].

ALGORITHM 1. Smart Circuit Partition based Attack

Input: Camouflaged Netlist, Functional IC.
Output: Original Netlist.
1: partition the circuit to MFICs;
2: **while** *there exist unresolved camouflaged gates in the netlist*
3: find MFIC(s) with minimum unresolvable camouflaged gates;
4: **while** *there is more than one MFIC eligible*
5: select the one minimizes next maximum camouflaged gates number;
6: **end**
7: brute force search possible functionality combinations;
8: update netlist;
9: **while** *there are unresolvable camouflaged gates become resolvable*
10: resolve them;
11: update netlist;
12: **end**
13: **end**
14: **return** the resolved netlist.

Suppose that there are N camouflaged gates in the netlist and each camouflaged gate can implement any one of the three logic gates {XOR, NAND, or NOR}. When an RE attacker does not have any more information, there will be 3^N possible functionality combinations to enumerate. However, the circuit partition-based attack algorithm in Algorithm.1 clearly shows that the attacker can resolve all the camouflaged gates much more efficiently. Let n_i be the number of camouflaged gates in the MFIC we selected during the i-th iteration (line 3), the brute force search process will search 3^{n_i} cases in the worst case. Let r_i be the number of camouflaged gates that become resolvable (lines 9–12) during the same iteration after the i-th MFIC is resolved, we have $\Sigma_{i=1,2,\ldots}(n_i + r_i) = N$. Since it takes only two PI patterns to resolve each of the r_i camouflaged gates in line 10 [7], it becomes clear that the complexity of the algorithm in Algorithm 1 is dominated by $3^{n_{max}}$, where $n_{max} = \max\{n_i\}$, the largest number of camouflaged gates in the same MFIC that need to be resolved simultaneously. In real design, n_{max} normally is much smaller than N and does not increase with N. Our experiments on benchmark circuits indicate that n_{max} is usually small (less than 10) [8].

Theorem 1 *The security of a camouflaged circuit against the circuit partition-based attack is determined by n_{max}, the largest number of camouflaged gates in the same MFIC that need to be resolved simultaneously.*

4.3 Mitigating the Circuit Partition-Based Attack

From Theorem 1, we see that it is crucial to have a large n_{max}, which means that we want to keep the camouflaged gates together such that the attacker cannot partition them into multiple sub-circuits and resolve separately. A gate classification method can help us to select the to-be camouflaged gates for this purpose.

Definition 2 For a gate G, $MFICS_G$ is the set of $MFIC_{PO}$ that G belongs to. Formally, $MFICS_G = \{MFIC_{PO_i} \mid PO_i$ is a primary output and $G \in MFIC_{PO_i}\}$.

To compute $MFICS_G$, we can first compute $MFIC_{PO}$ for all the primary outputs, then construct each of the $MFICS_G$ by examining which $MFIC_{PO}$ gate G belongs to. For example, for the circuit in Fig. 4.5, we have

$MFIC_{O1} = \{G1, G2, G3, G5\}$,
$MFIC_{O2} = \{G2, G3, G4, G6\}$,
thus
$MFICS_{G1} = MFICS_{G5} = \{MFIC_{O1}\}$,
$MFICS_{G2} = MFICS_{G3} = \{MFIC_{O1}, MFIC_{O2}\}$,
$MFICS_{G4} = MFICS_{G6} = \{MFIC_{O2}\}$.

Theorem 2 *MFICS is an equivalent relation and thus we can partition the circuit by putting the gates with the same MFICS into the same equivalent class. That is, gates G1, G2, ..., Gn are partitioned to the same class if and only if* $MFICS_{G1} = MFICS_{G2} = \cdots = MFICS_{Gn}$.

In the above example, the circuit will be partitioned into three equivalent classes: {G1, G5}, {G2, G3}, and {G4, G6}. Notice that gates in the same equivalent class cannot be partitioned further. This is an important feature for the following practical gate selection method that mitigates circuit partition-based attack.

As a reverse engineering attacker can only assign values to the PIs and observe the corresponding POs from an unpackaged functional IC, $MFIC_{PO}$ will be the minimum sub-circuit whose function can be tested by the attacker. Therefore, if we select gates to obfuscate from the same equivalent class, for any $MFIC_{PO_i}$ of the circuit that the attacker can attack, either all of the camouflaged gates belong to it, or none of them belongs to it. Thus the attacker will not be able to partition the camouflaged gates

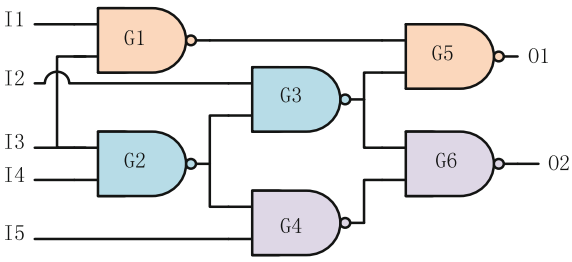


Fig. 4.5 A circuit example for gate classification. Gates in the same class, {G1, G5}, {G2, G3}, and {G4, G6}, are marked with the same color

into multiple sub-circuits to perform attacks individually. This criterion should be added to the enhanced circuit camouflaging method in [7] for better security.

4.4 Multiplexer-Based Circuit Obfuscation

Recall that the more functionalities a camouflaged gate can achieve, the more difficult it will be for attackers to resolve the camouflaged gate. As we have demonstrated above, when the configurable CMOS cell can only implement the functionality of XOR, NAND, or NOR gate, there will be two major security concerns. First, it restricts the selection of the gates to be camouflaged to only these three types of gates, limiting the value of n_{max}. Second, such camouflaged gate can be easily resolved by applying two distinct input patterns (as shown in the examples above). The multiplexer-based circuit obfuscation method [9, 10] solves this problem.

A 4×1 multiplexer (MUX) has four data lines {X1, X2, X3, X4}, two selection bits {A, B} and one output line S that comes from one of the data lines determined by the value of the selection bits A and B. Specifically, the output can be expressed as

$$S = A'B'X1 + A'BX2 + AB'X3 + ABX4$$

By assigning proper values to the data lines, a 4×1 MUX can implement any 2-input logic function (see Table 4.2). For example, when X1 = 0, X2 = 1, X3 = 1, and X4 = 0, the MUX becomes XOR.

In multiplexer-based circuit obfuscation, special designed MUX is utilized as the configurable logic unit to replace conventional gates. Programmable camouflage connector is used to configure the functionality of configurable logic unit. Similar to the contacts used in [7], the programmable camouflage connector can be programmed to be either a connection or an isolation, while appears to be physically identical under optical or electron microscopy.

More specifically, as shown in Fig. 4.6, the selection lines A and B and output line S of the multiplexer act as the inputs and output of the configurable logic unit, respectively. Each input line X_i (i = 1,2,3,4) is connected to Vdd and Vss by two

Table 4.2 A 4 × 1 MUX can implement all the 16 2-input Boolean functions

Function Number	X1	X2	X3	X4	Logic expression of S
1	0	0	0	1	AB
2	0	0	1	0	$A\overline{B}$
3	0	0	1	1	A + 0· B=A
4	0	1	0	0	$\bar{A}B$
5	0	1	0	1	0·A + B=B
6	0	1	1	0	A ⊕ B
7	0	1	1	1	A+B
8	1	0	0	0	$\bar{A} \cdot \bar{B} = \overline{A + B}$
9	1	0	0	1	A ⊙ B
10	1	0	1	0	$\bar{B}$
11	1	0	1	1	$A + \overline{B}$
12	1	1	0	0	$\bar{A}$
13	1	1	0	1	$\bar{A} + B$
14	1	1	1	0	$\overline{AB}$
15	1	1	1	1	const 1
16	0	0	0	0	const 0

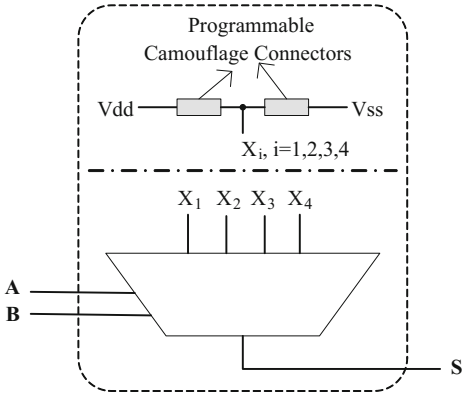


Fig. 4.6 Each input line of the MUX is connected to two programmable camouflage connectors, one is configured to be a connection and the other to be an isolation

camouflage connectors, but only one is programmed to be a connection, the other one is programmed to be an isolation. $X_i = 1$ when the camouflage connector that connected to Vss is configured as a connection, and $X_i = 0$ when the camouflage connector that connected to Vdd is configured as a connection.

Algorithm 2 shows a heuristic algorithm to perform circuit obfuscation with such MUXs. As analyzed in the previous section, we select gates from the same equivalent class so they cannot be resolved individually. The algorithm makes the gate that generates the PO, called outGates, of the MFICS of an equivalence class to appear as black boxes by obfuscating the outGates with MUXs, and blocking at least one input of the MUX (lines 4–9). Then desired number of gates are iteratively selected from the class to obfuscate, following the principle of minimizing performance overhead (lines 10–18). When there are more than one eligible gate class, the algorithm will get different versions of obfuscated circuits (line 19) and will return the one that results in minimum performance overhead (line 21).

ALGORITHM 2. Circuit Obfuscation with Multiplexers

Input: Original circuit *G*, to be obfuscated gate number *N*.
Output: Obfuscated circuit *G′*.
1: do gate classification by the *MFICS* gates belong to;
2: **for** *each class C with no less than M gates*
3: *currentG* ← *G*
4: **for** *each* $MFIC_{PO} \in$ *class C's MFICS*
5: obfuscate the *outGate* of the $MFIC_{PO}$;
6: **if** *none of the MUX's inputs is blocked*
7: obfuscate the MUX's one input gate whose *MFICS* is a subset of *C*'s *MFICS*;
8: **end**
9: **end**
10: **while** *obfuscatedGates* < *N*
11: select $gate_i$ with smallest *maxPathDelay* from unobfuscated gates in *C*;
12: **if** *there is an inverter* $gate_{INV}$ *before any input of* $gate_i$ *and* $gate_{INV} \in C$
13: merge $gate_{INV}$ and $gate_i$ as a logic block;
14: obfuscate the logic block with one MUX;
15: **else**
16: obfuscate $gate_i$;
17: **end**
18: **end**
19: save current obfuscated circuit *currentG*;
20: **end**
21: **return** obfuscated circuit with smallest performance overhead as *G′*.

4.5 Conclusions

In this chapter, we study the popular gate camouflaging-based obfuscation with focus on its security analysis. We use the circuit partition-based attack to demonstrate that the selection of camouflaged gates is crucial to increase the attack complexity of reverse engineering. We show that mitigation methods such as smart gate selection and the use of multiplexer can help to secure gate camouflaging against reverse engineering. It is our belief that both 'spear' and 'shield' need to be developed in the war against attackers.

To make this promising circuit camouflaging technique practical in thwarting reverse engineering attacks, there still exist many challenges. Perhaps the most significant one is that the overhead in applying CMOS camouflaging gates can be rather high in terms of circuit timing, power consumption, and area, especially when a high level of protection is needed. How to reduce the overhead incurred by circuit camouflaging would continue to be an urgent need. The second challenge is the development of countermeasures against the newly proposed and powerful de-obfuscation attacks based on SAT solver. Such attacks can effectively exclude incorrect functionality combinations of the camouflaged gates, successfully bypassing the exponential complexity of brute force. Although it cannot ensure to be effective in all circumstances, it has already posed a serious threat to many circuit camouflaging scenarios. Finally, it will be interesting to study intrinsic reconfigurable properties of emerging devices and how they can be utilized for circuit camouflaging.

Acknowledgements Mingze Gao and Gang Qu were supported in part by AFOSR MURI under award number FA9550-14-1-0351.

References

1. Qu G, Potkonjak M (2003) Intellectual property protection in VLSI designs: theory and practice. Kluwer Academic Publishers, Dordrecht. ISBN 1-4020-7320-8
2. Torrance R, James D (2011) The state-of-the-art in semiconductor reverse engineering. In: Proceedings of the ACM/IEEE design automation conference (DAC), pp 333–338
3. Quadir SE, Chen J, Forte D et al (2016) A survey on chip to system reverse engineering[J]. ACM J Emerg Technol Comput Syst (JETC) 13(1):6
4. Chow L, Baukus J, Clark W (2002) Integrated circuits protected against reverse engineering and method for fabricating the same using an apparent metal contact line terminating on field oxide. US Patent 20020096776
5. Chow L, Baukus J, Wang B, Cocchi R (2012) Camouflaging a standard cell based integrated circuit, US Patent 8151235
6. Cocchi RP, Baukus JP, Chow LW, Wang BJ (2014) Circuit camouflage integration for hardware ip protection. In: Proceedings of the 51st annual design automation conference (DAC 14), New York, NY, USA. ACM, pp 153:1–153:5
7. Rajendran J, Sam M, Sinanoglu O, Karri R (2013) Security analysis of integrated circuit camouflaging. In: Proceedings of the ACM conference on computer and communications security (CCS), pp 709–720

8. Wang X, Zhou Q, Cai Y et al (2016) Is the Secure IC camouflaging really secure. In: Proceedings of the IEEE international symposium on circuits and systems (ISCAS), pp 1710–1713
9. Liu B, Wang B (2014) Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In: Proceedings of the design automation and test in Europe (DATE). IEEE, pp 1–6
10. Wang X, Jia X, Zhou Q et al (2016) Secure and low-overhead circuit obfuscation technique with multiplexers. In: Proceedings of the ACM great lakes symposium on VLSI, pp 133–136

Chapter 5
Permutation-Based Obfuscation

Zimu Guo, Mark M. Tehranipoor and Domenic Forte

5.1 Introduction

As discussed in previous chapters, hardware obfuscation techniques can be categorized into chip level [1] and board level [2] according to the platform where these techniques are implemented. Chip-level obfuscation is performed on integrated circuits (ICs), while board-level obfuscation is performed on printed circuit boards (PCBs). The chip-level obfuscation techniques can be further classified into register-transfer (RT) level and gate level as per the design abstraction level. Several approaches can be exploited to accomplish design obfuscation. Based on the mechanism of these approaches, they can be classified as finite-state machine (FSM) obfuscation, logic encryption, and logic permutation. Figure 5.1 presents the relationship between these approaches and the design levels where they are applied. No single obfuscation approach can fit all the design levels. Additionally, some of them can be easily broken when implemented on a particular design level.

An obfuscation-protected system often consists of two operation modes: **functional mode** and **obfuscated mode** [3]. These two modes are controlled by one or more keys/configurations. The functional mode indicates that the system performs the designed functionalities, while obfuscated mode implies that the system presents no meaningful behavior.

Z. Guo (✉) · M.M. Tehranipoor · D. Forte
University of Florida, Gainesville, FL, USA
e-mail: zimuguo@ufl.edu

M.M. Tehranipoor
e-mail: tehranipoor@ece.ufl.edu

D. Forte
e-mail: dforte@ece.ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_5

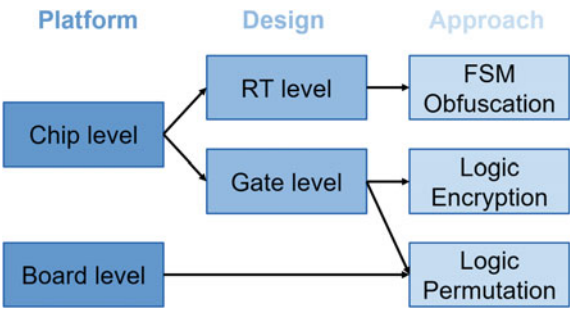


Fig. 5.1 Hardware obfuscation classification

During the rest of this section, a brief introduction about each of the obfuscation approaches is provided. Section 5.1.1 talks about chip-level obfuscation techniques, and Sect. 5.1.2 deals with board-level obfuscation. The rest of the chapter focuses on chip and board-level logic permutation.

5.1.1 Chip Level

For chip-level application, the following three obfuscation approaches can be applied.

- FSM obfuscation
- Logic encryption
- Logic permutation

As stated in Fig. 5.1, the **FSM obfuscation** can only be implemented by modifying the RTL design [4]. A general FSM obfuscation approach involves adding extra states into the original state transition graph (STG) of the design. These inserted states prevent the system from entering the functional mode without a correct key/configuration [4]. This key (which can either be fixed or generated based on the chip's ID) transforms the chip from the obfuscated mode to the functional mode. Some designs also contain another FSM called a black-hole FSM which permanently locks the chip if the applied key is incorrect [5]. The aforementioned FSM-based approach cannot be applied on gate-level nor board-level designs since neither of them provides the states which can be modified.

The **logic encryption** approach inserts locking blocks into the gate-level design [6]. These blocks can be as simple as a set of XOR gates which mask the internal signals with the keys or configuration bits. Since these encrypting units block the internal data/signal paths of the original design, they are also named as logic barriers [6]. These logic barriers can only be inserted in gate-level and board-level designs since there is no data/signal path abstraction at RTL level. However, logic encryption is much easier to be broken on board-level designs. A short answer for this is that it is much simpler to remove the encrypting blocks from board-level designs than chip-level ones. More detailed explanations will be given in Sect. 5.1.2.

The **logic permutation** approach permutes the data/signal paths instead of encrypting them. This method can be utilized to accomplish the obfuscation goal at both the gate level [7] and the board level [2]. By adding a *permutation block*, the correct orders of internal connections are concealed. Similar to other obfuscation approaches, a key/configuration is assigned to drive the system to the functional mode.

5.1.2 Board Level

Significantly, different constraints can be observed between chip and board-level obfuscation. These differences include the design modification opportunities, attack challenges, and design dimension differences. Limited by these differences, the number of obfuscation techniques which can be applied on PCBs is less than on ICs. The design modification feasibility indicates what levels of design can be manipulated to achieve obfuscation. For instance, the RTL and gate-level design can be obfuscated for a chip. For a board-level design, no RTL definition can be found. However, the chip-level intergate connections can be extended to board-level interchip connections. As a result, the gate-level obfuscation approaches, such as logic encryption and logic permutation, can also be applied at the board level. However, these obfuscation methods may be easily broken when they are applied directly on the board level under low-cost attacks. The designers should also be aware of the dimension difference between chips and boards. The onboard connections (e.g., between chips) are more straightforward to be discovered than the chip-level ones. The differences mentioned above enable the attackers to identify, bypass, or remove inserted obfuscation components from the board more quickly. An example is provided as follows to clarify how certain chip-level obfuscation techniques fail on the board level.

A simple bypass attack on board-level logic encryption is provided as follows. Shown in Fig. 5.2a, as logic barriers, XOR and XNOR gates are inserted in the middle of the in-chip paths [6]. These barriers are embedded within a fabricated chip and can only be identified and bypassed by applying costly techniques such as IC reverse engineering [8]. For board-level logic encryption, these logic barriers are implemented on a dedicated chip. The same attack on the boards can be simply accomplished by bypassing the logic barriers with jumper wires. As presented in Fig. 5.2b, the attacker only needs to connect corresponding inputs and outputs of the logic barrier chip. These jumper wires are presented in red dashed line in Fig. 5.2b. Even though these connections are not public, the attackers can always find the correct input/output pairs by matching waveforms. This matching process is named as the probing attack and will be elaborated on in Sect. 5.8.

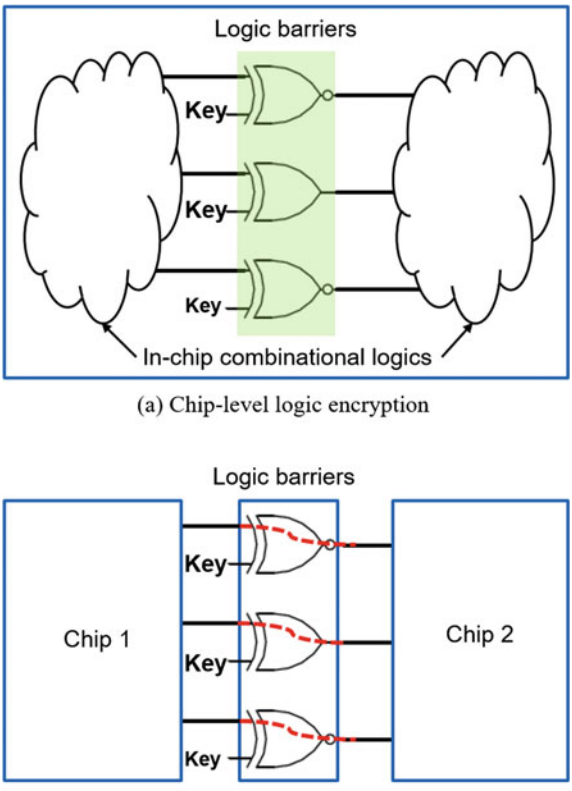


Fig. 5.2 Comparison of logic encryption on chip level and board level

5.1.3 Chapter Organization

In this chapter, the detailed framework for implementing logic permutation on both chip level and board level is provided. Section 5.2 presents the high-level permutation-based obfuscation framework. The goals of the attacker, as well as the designer, are discussed. In Sect. 5.3, the major differences between chip-level and board-level designs are presented. These differences should be carefully considered when obfuscating the design on different levels. Section 5.4 performs the analyses on implementing the permutation. These analyses help in determining which paths are good permutation candidates and why. This section provides a guideline for the designer to achieve the best obfuscation performance. Section 5.5 introduces the permutation networks which could be used for obfuscation. These permutation networks are studied by their capabilities and area utilizations. Capability indicates whether a permutation network can achieve the full permutation or not. Besides these analyses, two configuration scenarios of a popularly used permutation network are presented. According to these scenarios, the attacker's goal is reanalyzed. Section 5.6 discusses

the ways to generate and store the key/configurations used for obfuscation. The advantages and drawbacks of each key generation/storage approach are provided.

Section 5.7 provides a comprehensive flow for evaluating the performance of an obfuscated system against various attacks. This performance indicates how difficult an attacker can break the obfuscation. In Sect. 5.8, potential attacks and their corresponding countermeasures are provided. Both the permutation network and its key/configurations can be attacked. These attacks are discussed at the chip as well as board level. The attacks are organized into three classes based on the required information and resources to carry them out. Next, the countermeasures are grouped into three levels of security requirement. The coverage of the countermeasures is also discussed. Finally, the conclusions are summarized in Sect. 5.9.

5.2 Permutation-Based Obfuscation Overview

A general idea of hardware obfuscation is shown in Fig. 5.3. The solid-line rectangles imply the components which belong to the original design. For the chip level, these components can be either logic gates or registers. For the board level, these components can be chips on the board. In the original design, these components are connected directly by fixed wires or traces. Since all the intercomponent connections are established in the original designs, these products are also functional after being sold in the market. An attacker with the chip-level or board-level reverse engineering capabilities can obtain the design. The question mark in Fig. 5.3 denotes the obscurity introduced by the obfuscation which prevents simple reverse engineering. This mystery can be in the form of interconnection permutation, modification of signal values, or a combination of the two. In hardware obfuscation, typically a secret key is used to remove the obscurity and allow the design to behave as originally intended. In this chapter, we consider that the obscurity is provided by a permutation block. This permutation block consists of a key-controlled permutation network and permutes the selected intercomponent connections.

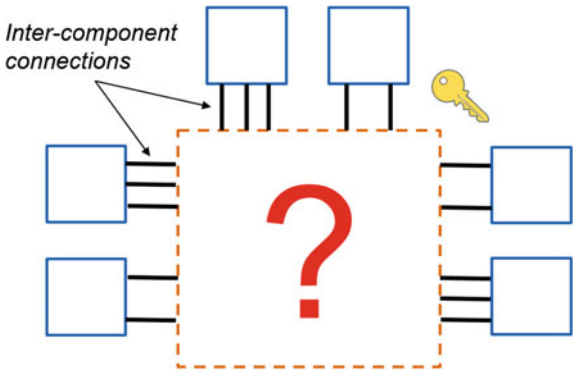


Fig. 5.3 General obfuscation scheme

Breaking the permutation-based obfuscation can be achieved by discovering the mystery above. The most straightforward way to accomplish this task is through brute force attacks. Two possible brute force objectives may be realized: **(i)** retrieving the original connections and **(ii)** key entries to the obfuscation chip. In the former, an attacker tests all the possible intercomponent connections to identify the one that results in the correct operation. This attack is usually achieved on board level since it is simple to examine the obfuscated connections by physically connecting them. On the other hand, the same procedure is nearly impossible to be accomplished at the chip level. The reason is that connecting the traces within the chip requires chip-level reverse engineering. Since such reverse engineering is usually destructive, the chip under test would no longer be functional. Compared with the former brute force objective, the latter one can be applied on both chip level and board level. Achieving this attacking goal, an attacker tests all possible key inputs through the permutation network's interface until the system functions correctly. Note that this objective differs from the first one because it depends entirely on the implementation of the permutation network (i.e., the relationship between keys and input/output combinations).

The term *breaking probability* denotes the likelihood of discovering the mystery through these two brute force methods.We use P_{com} to denote the breaking probability for examining the input/output combinations of the permutation network, and P_{key} denotes the breaking probability for reviewing the key entries. These breaking probabilities are used to evaluate the performance/strength of the permutation-based obfuscation. Besides the brute force attack, the feasibility analyses of other potential attacks are discussed in Sect. 5.8.

5.3 Obfuscation Considerations

Since significant design differences exist between ICs and PCBs, the obfuscation implemented on these platforms should be considered separately. In this section, these considerations are classified into two categories: **obfuscation-induced overheads** and the **cost of obtaining the design**.

The obfuscation-induced overheads indicate the amount of area, power, and cost overheads introduced by obfuscating the original design. Higher overheads make the obfuscation less practical. For the board-level obfuscation, the permutation block is usually implemented within an additional chip. The reason for adding another chip is that the functionalities of the chips in the original design are dedicated and cannot be used to permute the interconnections. However, this extra component induces overhead to the original design. Compared with obfuscating the board, accomplishing the same task within a chip presents fewer overheads. The area overhead is negligible when inserting hundreds of gates into millions of gates. For chip-level obfuscation, the obfuscation procedure should be achieved during the IC design phase. The insertion of the permutation block may require redesigning the entire IC.

Learning the layout or schematic of the obfuscated design benefits the attacker in breaking the obfuscation. Since this design information is usually not public, the attackers would exploit techniques such as hardware probing [9] and reverse engineering [10] to realize this goal. To learn the internal structure of an IC, an attacker can apply destructive [10] or nondestructive [11] reverse engineering. Destructive reverse engineering requires delayering of the chip and capturing high-resolution images for each layer. With these images, the layout of the chip can be recovered. However, reverse engineering a PCB is much easier. PCBs can be destructively reverse engineered by a similar process. However, automated nondestructive technique based on X-ray can be used to extract the PCB internal structure without delayering [11]. Many online resources provide low-cost PCB reverse engineering services. Besides reverse engineering, hardware probing aims to monitor the signals on a running board or chip. Similar to reverse engineering, probing the board is more straightforward than the chip. The interchip connections of a board are accessible simply by probing the traces exposed on the surface [12]. These low-cost board-level reverse engineering and hardware probing make it harder to hide secrets in a board than a chip.

5.4 Design Modification

As presented in Fig. 5.3, a mystery (i.e., permutation block) hides the actual intercomponent connections from the attackers. The designer needs to modify the design by inserting the permutation block and selecting the intercomponent connections to be obfuscated. For the board level, this permutation block can be implemented by a complex programmable logic device (CPLD), field-programmable gate array (FPGA), or application-specific integrated circuit (ASIC). For the chip level, this permutation block is inserted by modifying the gate-level design.

The intercomponent connections used for obfuscation should be carefully selected. A smart designer would not involve all the intercomponent connections into the permutation-based obfuscation. Even though performing this selection strategy increases the attack difficulty, it may cause significant timing, area, and power overheads on both chip and board levels. Further, many onboard chips may be commonly used parts, whose connections can be easily guessed by an attacker (e.g., clock signals and analog signal pins). This further constrains the type of interconnections that may be used for permutation-based obfuscation. Moreover, not all the types of signals can by permuted by the permutation block (e.g., the CPLD and FPGA can only take digital signals as input). In summary, three general requirements need to be met before involving an intercomponent connection into the obfuscation. These requirements are as follows:

(i) The obfuscated connection should not be easily guessed, matched, or bypassed;
(ii) The obfuscated connection should not be on a timing-critical path; and
(iii) The type of connection (digital or analog) should be manipulated by the permutation block.

Requirement (ii) should be considered when implementing both the chip-level and board-level obfuscations, while requirements (i) and (iii) are considered only during the board-level obfuscation. In this section, these design modification requirements are discussed on board level and chip level, respectively. Besides these three general rules, other more detailed design frameworks have been provided by researchers [2, 7, 13] to either minimize the overheads or improve the obfuscation performance.

5.4.1 Board Level

When implementing the board-level permutation-based obfuscation, all three requirements should be considered. First of all, it is crucial to obfuscate the connections which do not perform dedicated functionalities. The term functionality indicates what these ports can be utilized as (e.g., analog-to-digital converter and serial peripheral interface bus). This information is usually public and can be accessed by anyone including the attackers (e.g., by publicly available data sheets of the ICs).

As an example, a reference design from NXP is shown in Fig. 5.4. The reference design is based on Freescale ColdFire V1 MCF51MM256CLL microcontroller unit (MCU). This instrument is a noninvasive acquisition system incorporating a pedometer, ECG, food intake table, data storage, wireless communication, timer, and a chronometer. Biometric data can be saved and stored in the integrated Micro SD Card Reader. A touch-sensing interface allows the user to have control of the device through a capacitive touch-sensing film. As shown in Fig. 5.4, the digital accelerometer chip communicates with the MCU via interintegrated circuit (I^2C) bus. This chip only consists of 10 pins, and most of them are either reserved or required to be attached to the power/ground according to the datasheet. Thus, these reserved pins cannot be obfuscated. If the rest of the chip's pins are obfuscated, the attackers need to figure out the corresponding ports from MCU to break the obfuscation. Achieving this task, they may examine the MCU's ports which provide the I^2C function. According to the datasheet of this MCU, only two sets of port preform the I^2C function. As a result, the attacker could partially figure out the obfuscated connections. The breaking probability will be significantly increased if these I^2C pins are involved in the obfuscation.

The scenario above can be found in most of the modern embedded systems. Besides I^2C, the serial peripheral interface (*SPI*) is also widely utilized in interchip communications. These highly dedicated ports/connections are not good candidates for implementing obfuscation. On the other hand, examples of good obfuscation candidates can be the general-purpose input/output (GPIO) pins. They are generic pins on an integrated circuit whose behaviors are controllable by the user at run-time. Since GPIO pins have no predefined purpose, it is exceedingly difficult to discover the actual connections by their functionalities. Thus, the components which provide a significant number of these pins are ideal candidates for obfuscation. A system/PCB can be protected by the permutation-based obfuscation if it consists of one or more of these components. CPLDs, FPGAs, and MCUs are examples of the components mentioned

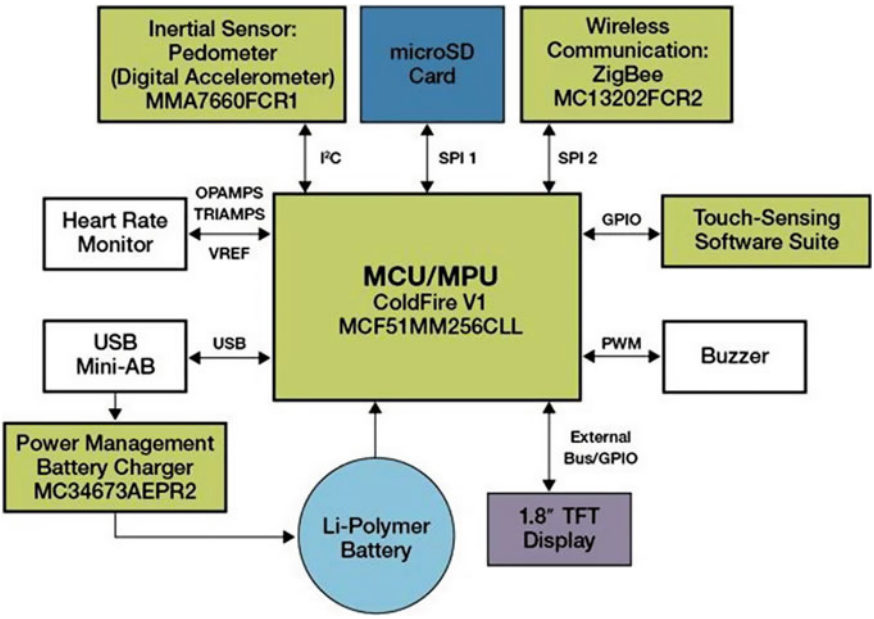


Fig. 5.4 Block diagram of NXPs activity monitor reference design

above. These components are named as **programmable components** since they can be programmed by the designer [2]. The components connected and controlled by the programmable components are named as **non-programmable components**. For instance, the heart rate monitor and display block in Fig. 5.4 are examples of the non-programmable components.

The second requirement is related to the timing constraints. Since the permutation block introduces additional propagation delays, the delay-sensitive interchip connections should be avoided in implementing the obfuscation. These connections consist of clock inputs, other control signals with strict timing requirements, etc. The last requirement specifies the types of the signal which can be permuted. Since the permutation block can only take digital signal as inputs, only the digital signals in the original design can be permuted.

Besides the connected ports in the original design, the system may not utilize all the ports of the programmable component. These unconnected ports can also be used in the obfuscation if they satisfy the requirements mentioned earlier. These unconnected ports are referred to as *dummy ports*, while the connected ones are referred to as *real ports* [2]. Involving dummy ports increases the total number of combinations which an attacker needs to examine. All the real ports and dummy ports are the inputs of the permutation block.

The schematic view of the board-level design modification is shown in Fig. 5.5. In this figure, the original connections are permuted by the permutation block. Moreover, only part of the permutation block's inputs are real inputs.

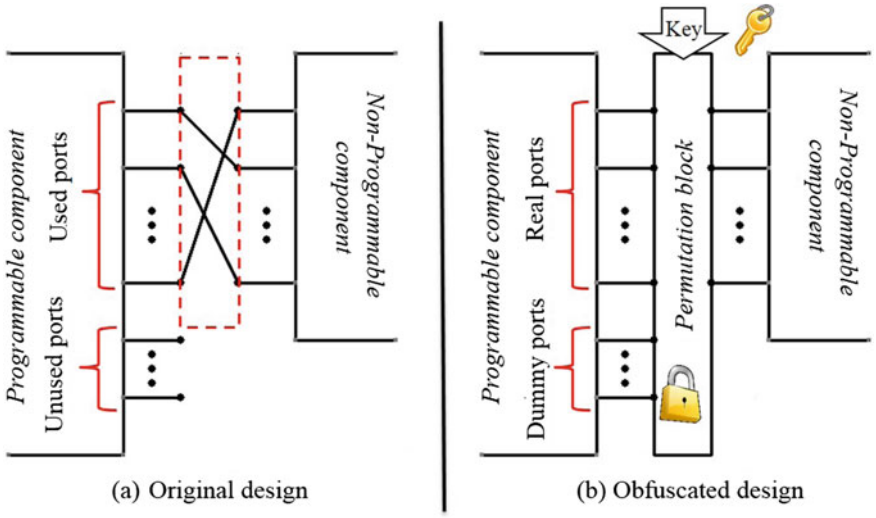


Fig. 5.5 Board-level design modification

5.4.2 Chip Level

For the chip-level application, the internal connections are the paths between gates. The functionalities of the intergate connections are difficult to be discovered without knowing the full design. Additionally, since the permutation network is merged into the gate-level design, all the intergate signals are digital. Thus, all the chip-level interconnections besides the timing-critical paths can be permuted since they are tough to be bypassed physically. As a result, the requirements (i) and (iii) need not be considered in the chip-level permutation-based obfuscation. However, the timing constraint, which is the requirement (ii), is crucial to be studied when inserting the permutation network.

Researches in [13] proposed a bus-based hardware IP protection scheme. This scheme is applicable to a broad category of electronic systems with a primary bus. Such designs include (1) numerous IP offerings for USB, PCI, PCI-E, AMBA, and other bus standards typically used in system-on-a-chip designs and computer peripherals, (2) SRAM-based FPGAs that are programmed through an input bus, (3) general-purpose and embedded microprocessors, including soft cores, (4) DSPs, (5) network processors, and (6) game consoles. This requirement is illustrated in Fig. 5.6 using a SoC architecture as an example.

The bus is equipped with additional bus-key inputs such that only a certain key combination activates the bus, while other combinations scramble it. In other words, a lock is merged with the bus and only the correct can remove the lock. According to Fig. 5.6, this lock can be allocated at the region where the bus is implemented. Several techniques can be utilized to accomplish this locking, such as bit-wise XOR, arithmetic transformations, and bit permutations.

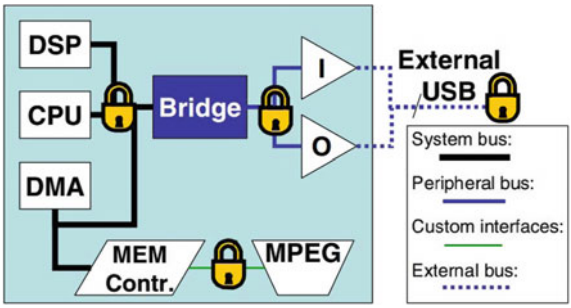


Fig. 5.6 Bus-based IP protection [13]

Comparing the requirements which need to be met in the chip-level and board-level obfuscation, obfuscating chip-level designs is more flexible than the board-level ones.

5.5 Permutation Network

Choosing a proper permutation network and implementing it in this permutation block play an essential role. The inputs of this inserted permutation block come from the selected components in the original design. The outputs are connected with the components in which these obfuscated connections are originally attached to.

In this section, the background of various permutation networks is introduced, and the best candidate for implementing the permutation-based obfuscation scheme is selected. As an example, according to the selected permutation network, this section provides the configuration approaches under different scenarios. Based on the capability of achieving full input/output permutations, permutation networks can be classified into *blocking* and *non-blocking* networks [14].

Blocking permutation network indicates that the network can only realize partial input/output combinations. Butterfly network [15] and basic Omega network [16] are two examples of blocking permutation network. These blocking networks are presented in Fig. 5.7. In this figure, the rectangles represent 2-to-2 switches. Each of these switches is controlled by a binary number. The key is formed by concatenating all these binary numbers. The total number of different keys is directly related to the number of switches. For instance, either of the permutation networks shown in Fig. 5.7 consists of 12 switches. The total number of different keys is $2^{12} = 4096$ for either butterfly network or Clos network. On the other hand, the total number of input/output combinations needed for any 8-input permutation network is $8! = 40320$. This number is greater than the total number of the keys. This observation indicates that a large percentage of input/output combinations cannot be accomplished, no matter how one configures the switches.

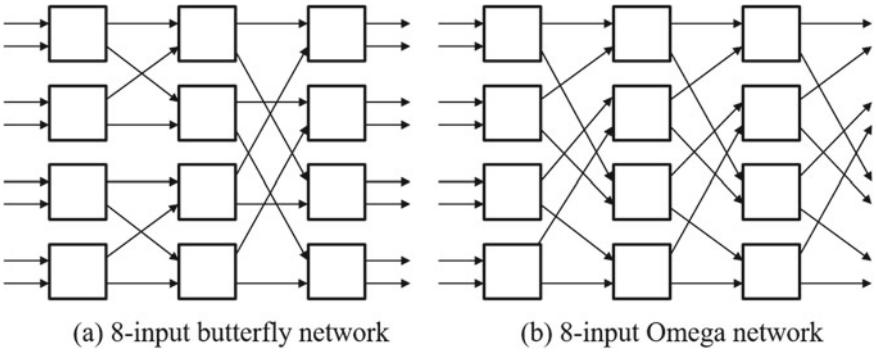


Fig. 5.7 Examples of blocking networks

Non-blocking permutation network indicates that the network can realize all input/output combinations with or without constraints. This permutation network category consists of three subcategories.

(i) *Strict-sense non-blocking network* [17] can construct a new path connecting unconnected inputs and outputs regardless of any pre-established paths. Designers can switch any two paths without adjusting the rest of the network settings. This path independence property makes strict-sense non-blocking network ideal for a telephone router. The multiplexer (MUX) array network and Clos network with high-order switches [18] are two examples of this type of network. A general design of Clos network is presented in Fig. 5.8. Clos networks have three stages: the ingress stage (Stage1), middle stage (Stage2), and the egress stage (Stage3). Each stage is made up of a number of crossbar switches, often just called crossbars. Clos networks are defined by three parameters n, m, and r. n represents the number of sources which feed into each of r ingress stage crossbar switches. Each ingress stage crossbar switch has m outlets, and there are m middle stage crossbar switches. These parameters should follow the relationship $m \geq 2n - 1$, in order to achieve the non-blocking property [19].

An example of 4-bit MUX array network is provided in [7]. This network is named as wire scrambling (WS) cells which shuffle the intergate connections. A generic WS-cell can be implemented by using multiplexers or pass transistors as shown in Fig. 5.9. The full shuffler (Fig. 5.9b) meets the definition of strict-sense non-blocking network, while the partial shuffler (Fig. 5.9c) does not. The permutations which can be achieved by this partial shuffler are a subset of the permutations that the full shuffler can achieve. For the MUX array network, to preserve the strict-sense non-blocking property, the number of MUXs should be equal to the number of the inputs of the permutation network. Additionally, each MUX should have the capability to select any input of the permutation network.

(ii) *Wide-sense non-blocking network* [20] does not provide the strict independence guarantee as a strict-sense non-blocking network does. It is still possible to connect any unused input to any unused output with certain algorithms.

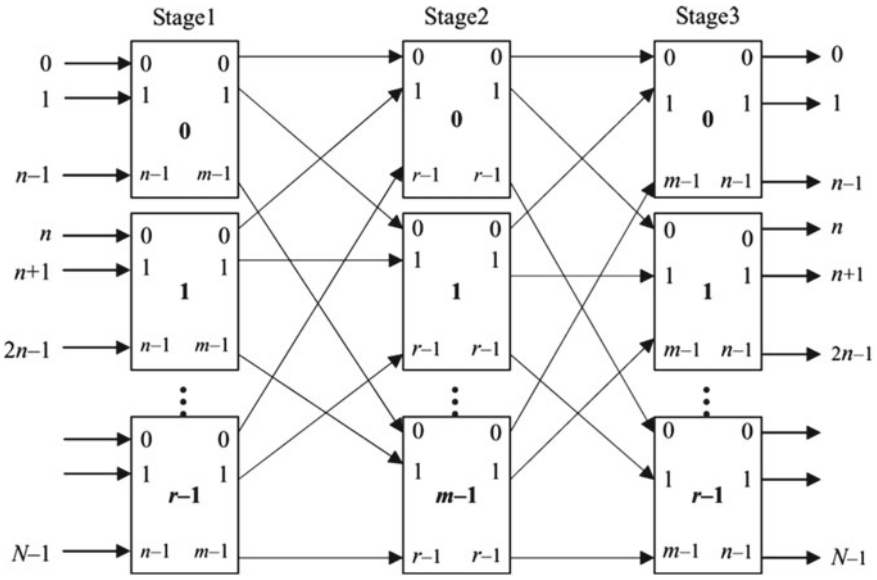


Fig. 5.8 $N \times N$ three-stage non-blocking Clos network [19]

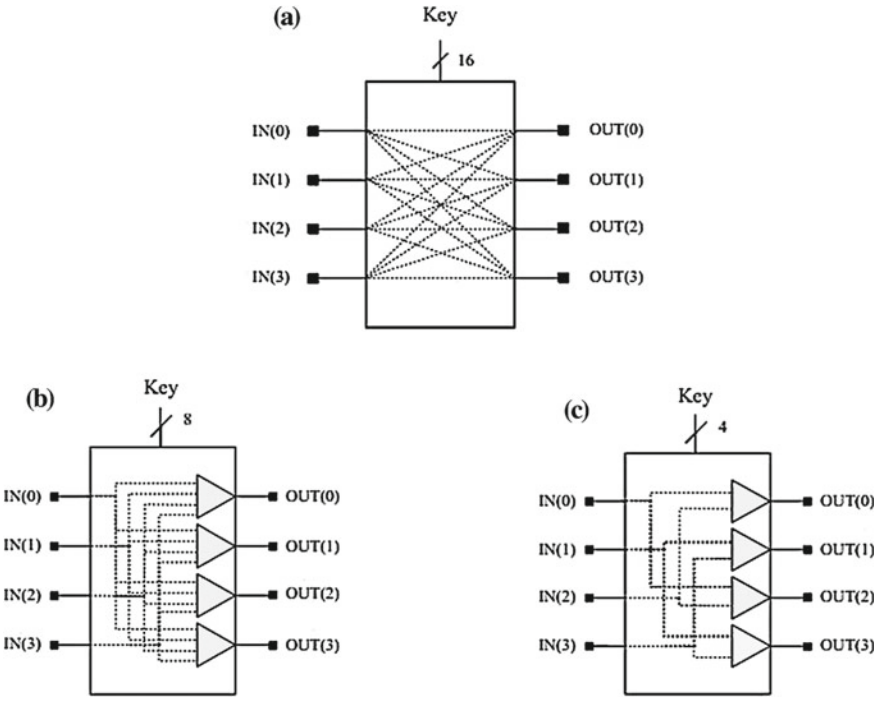


Fig. 5.9 **a** Structure of a generic WS-cell. **b** MUX base WS-cell structure, Full shuffler. **c** Partial shuffler. The multiplexers are shown as the triangles [7]

(iii) The weakest notion of non-blocking permutation network is *rearrangeable non-blocking network* [21]. This kind of network is not capable of fully realizing network configurations without the prior knowledge of inputs' and outputs' order. Benes network [22] is one example of this kind of network.

5.5.1 Area Utilization

Comparisons of implementation area utilization are performed among three permutation networks: basic Omega network (blocking), MUX array-based network (strict-sense non-blocking), and Benes network (rearrangeable non-blocking). The Omega network and Benes network consist of 2-to-2 switches, while the multiplexer-based network is composed by n-to-1 multiplexers. The parameter n depends on the number of inputs of the permutation network.

These networks are synthesized using Quartus II software and implemented in Altera MAX V CPLD. Target CPLD consists of 570 logic elements (LEs), which is the basic logic unit in CPLD. The area utilizations are presented by the percentage of exploited LEs. The results are summarized in Table 5.1. It can be observed that the Benes network utilizes about half of the LEs (60%) compared with the multiplexer-based network (115%). The number of gates needed to achieve each network increases exponentially as the number of inputs/outputs increases.

Benes network is a good candidate in implementing the permutation block for the following two reasons: (i) It is apparent that blocking permutation networks are inappropriate since they produce limited inputs/outputs combinations. For our case, designers should choose a candidate from non-blocking networks. (ii) Benes network is much better than multiplexed based network considering the hardware area utilization. For this application, the strict-sense non-blocking networks have significant overhead and their benefits for the permutation-based obfuscation are not needed.

Even though the strict-sense non-blocking networks present much larger area utilizations, they have a unique advantage, which is presented in Fig. 5.10. According to this figure, this type of network can route any input to one or more outputs. This unique benefit provides the designer more choices in intercomponent connection selection. For certain applications, one signal may be routed to multiple chips/gates (e.g., the chip selection signal which controls multiple memory chips). In these applications, the above capability of the permutation network is required.

Table 5.1 Permutation network CPLD area utilization

I/O	4/4	8/8	16/16	32/32
Omega network (%)	2	6	20	23
Benes network (%)	2	7	22	60
Multiplex router (%)	2	10	31	115

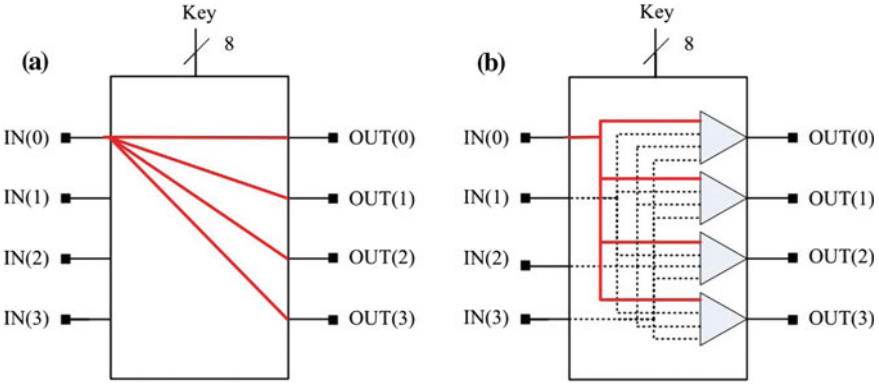


Fig. 5.10 The Multiterminal net connection in WS-cells. **a** IN(0) should be routed to all the outputs. **b** The MUX-based hardware implementation of the routing requirement shown in **a** [7]

5.5.2 Network Configuration

After selecting a permutation network, it should be configured properly. Since the Benes network presents several advantages mentioned above, its configuration situations are discussed in this section as an example. The most basic Benes network unit is a 1-bit controlled 2-to-2 switch as shown in Fig. 5.11a. The switches operate in two modes: straight mode when the control bit is 0 and exchange mode when the control bit is 1. Bits from the obfuscation *key* are used as the control bits for each switch. We define each column of switches as one stage. Two Benes network properties play a crucial role in the configuration process: (i) *Recursion* indicates that a Benes network can be split into two identical lower-dimension Benes networks. For example, in Fig. 5.11a, the 8-input Benes network can be split into two 4-input Benes networks (shown in red dashed boxes). This splitting can be recursively repeated on any lower-dimension Benes networks till reaching the basic unit (2-to-2 switch). (ii) *Symmetry* indicates that the Benes network possesses rotational symmetry across the center stage. According to this property, the stages on the left of center stage are named as *Forward stages* and the stages on the right as *Backward stages*. These stage categories are shown in Fig. 5.11b. A designer can take advantage of these properties for simplifying the configuration.

Network configurations can be classified into two situations based on the prior knowledge and the goals.

Situation 1: Designers aim to find the order of outputs with the knowledge of input order and a predefined key. Since Benes network possesses rotational symmetry, it always consists of an odd number of stages. The number of stages (S) can be found by $S = 2^* \log_2 N - 1$, where N is network dimension (i.e., the number of inputs to the Benes network). Each stage can be mathematically represented by a square permutation matrix (PM) derived from the key. A permutation matrix has exactly one '1' in each row/column and '0' elsewhere. Each permutation matrix represents the

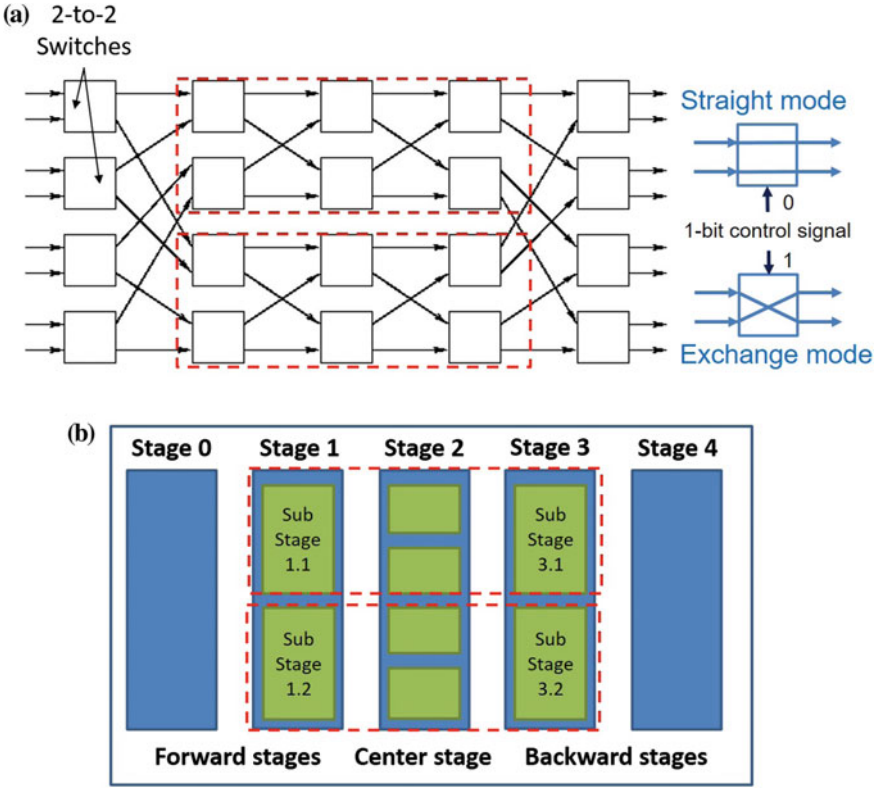


Fig. 5.11 **a** 8-inputs Benes network and **b** stage partition

permuted inputs passing the corresponding stage. Multiplying permutation matrices represents the inputs passing multiple stages coherently. The property of symmetry benefits designers by simplifying the permutation matrix calculation. With PMs computed and the input order (I), we can formalize the output order (O) as follows

$$O = I \times \prod_{i=1}^{\log_2 N - 1} PM_i \tag{5.1}$$

where N represents the number of inputs. The same procedure can be repeated for computing input order with the knowledge of output order and the key.

Situation 2: The designer's objective is to find one or more keys achieving a required input/output permutation. This procedure is also known as the network routing. Prior work such as [23] proposed an efficient routing algorithm based on searching *loops* (defined later) of *outer stages*. Outer stages (OS) are defined as mirrored stage pairs such as Stage 0 and Stage 4 in Fig. 5.11b. The number of outer stages K is defined according to the number of inputs N as follows,

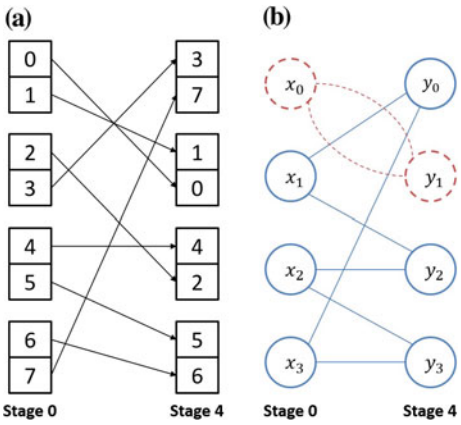


Fig. 5.12 Outer stage schematic [23] **a** switches connections and **b** groups illustration

$$K = \log_2 N - 1 \tag{5.2}$$

The *loop* structure is shown in Fig. 5.12. As an example, assume the network input as [0 1 2 3 4 5 6 7] and the output as [3 7 1 0 4 2 5 6]. To determine the loops, the proposed approach illustrates each switch as a node labeled x_0, y_0, etc., in Fig. 5.12b. The connections between nodes are constructed based on the input/output orders. A loop collects the interconnected nodes. For example, the dashed-line loop consists of nodes x_0 and y_1. Nodes within one loop should not connect the nodes outside this loop. Besides the outer stage formalized by Stage 0 and Stage 4, similar loop structures can be found in other outer stages such as Sub Stage 1.1 and Sub Stage 3.1 in Fig. 5.11b. A configuration can be accomplished by assigning '0' or '1' to each node (switch). As mentioned in [23], two equivalent key configurations can be assigned to each loop. These two configurations are complementary binary chains, such as '1011' and '0100'.

This multiple-configuration phenomenon can lead to more than one key achieving the same input/output combination. In this chapter, we refer to this phenomena as the *multiple-key effect*. This effect could diminish the protection strength by increasing the breaking probability. At the beginning of Sect. 5.2, the breaking probabilities are classified into two categories depending on what information the attacker attempts to obtain. This breaking probability refers to the probability of the attacker receiving the correct key by exhaustive search (P_{key}).

5.5.3 Multiple-Key Effect

For an 8-bit Benes network with a 20-bit key, the breaking probability is $P_{com} = 1/8! = 2.5e-5$ when the attacker examines the input/output pairs. If the attacker examines the keys instead of input/output pairs, the size of the key space to examine

is $P_{key} = 2^{20} = 1048576$. Besides the Benes network, this effect may be observed in other permutation networks where the key space is larger than the input/output combinations. Let the number of multiple keys be m for a given input/output combination. The following scenarios may happen: (i) $m/2^{20} < 1/8!$, multiple-key effect decreases the breaking probability. (ii) $m/2^{20} > 1/8!$, multiple-key effect increases the breaking probability. (iii) $m/2^{20} = 1/8!$, the breaking probability remains unchanged. The last situation only holds when a one-to-one correspondence between the key and the input/output combination holds. However, this situation does not hold when certain permutation networks, such as the Benes network, are implemented. Discovering minimal, maximal, mean, and medium values of m is invaluable for precisely evaluating obfuscation performance of the overall approach.

The breaking probability can be computed as Eq. (5.3) when the attacker chooses to examine the keys.

$$P_{key} = \frac{\text{Number of correct keys}}{\text{Total number of keys}} \tag{5.3}$$

Since the total number of keys is more than the total number of input/output combinations, the multiple-key effect may or may not increase the breaking probability.

The second configuration situation in Sect. 5.5.2 indicates that the switches within each loop can be configured in two ways, such as '1101' and '0010'. These switches' configurations are connected formalizing the key. In general, a relationship between the number of loops and multiple keys (*Mkey*) can be described in Eq. (5.4).

$$Mkey = \prod_{k=1}^{K} 2^{l_i} = 2^{\sum_{i=1}^{K} l_k} \tag{5.4}$$

where K is the number of outer stages (OS), and each of them consists of l_k loops. The number of loops (l_k) depends on the input/output combinations.

In order to break the proposed board-level obfuscation by brute force, the attacker would choose from the following two strategies: **Strategy (i)** examines the input/output order combinations and **Strategy (ii)** examines the keys. These two strategies correspond to the two types of breaking probabilities: P_{com} and P_{key}. Affected by multiple keys with the same permutated outcome, the attacker only needs to figure out one of the keys when choosing the strategy (ii).

Effects of multiple keys on obfuscation strength are studied by comparing the breaking probabilities of applying strategy (i) and strategy (ii). Applying **Strategy (i)**, the breaking probability is $1/32! = 3.8004E - 36$ for a 32-bit Benes network. For **Strategy (ii)**, the breaking probabilities can be computed through dividing the numbers of multiple keys by the total number of keys. The breaking probabilities under strategy (i) and strategy (ii) are summarized in Table 5.2. Row **# of multiple keys** represents the minimal, maximal, and mean number of multiple keys under strategy (ii). These values can be obtained from the number of loops distribution. Row **Probability** shows the corresponding breaking probabilities. The mean breaking probability is computed based on uniform distribution assumption of the Benes network input/output combinations.

Table 5.2 Effects of multiple keys

	Strategy (i)	Strategy (ii)		
		Min.	Max.	Mean
# Multiple keys	N/A	32768	1.8447E+19	8.1859e+07
Probability	3.8004E−36	1.4694E−39	8.2719E−25	3.6706e−36

Since breaking probabilities vary with different input/output combinations when applying strategy (ii), we compare the mean breaking probability with the breaking probability under strategy (i). According to Table 5.2, *exploiting the multiple-key effect of the Benes network decreases the expected (average) breaking probability. This means that it is even harder to execute strategy (ii) compared to strategy (i).*

5.6 Key Management

The permutation obfuscation is controlled by a key. This key configures the system (either a board or chip) and removes the mystery (the question mark in Fig. 5.3). Several options can be exploited to generate and store the key/configuration. Each of these options has advantages and drawbacks.

The key/configuration can be input into the permutation/obfuscated chip either internally or externally as shown in Fig. 5.13. The internal mode (Fig. 5.13a) implies that this key is permanently stored in the on-chip nonvolatile memory such as electrically erasable programmable read-only memory (EEPROM) or Flash. The booting unit automatically loads the saved configuration into the permutation network during each power-up. As discussed in Sect. 5.5.3, the stored key can be either the same or different among the various chips. Reference [13] proposed an IC activation protocol which utilizes the Diffie–Hellman (D-H) key exchange scheme. This protocol is presented in Fig. 5.14.

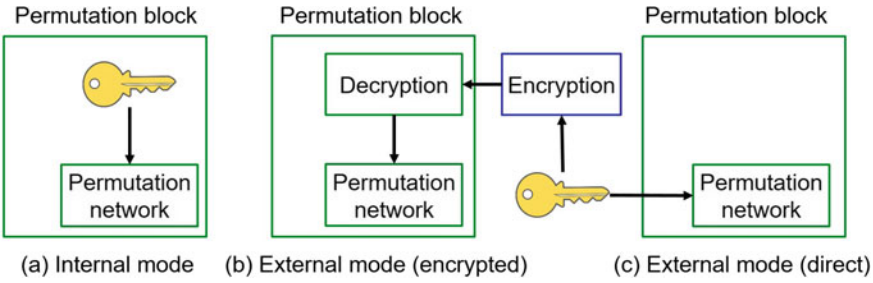


Fig. 5.13 Key loading modes

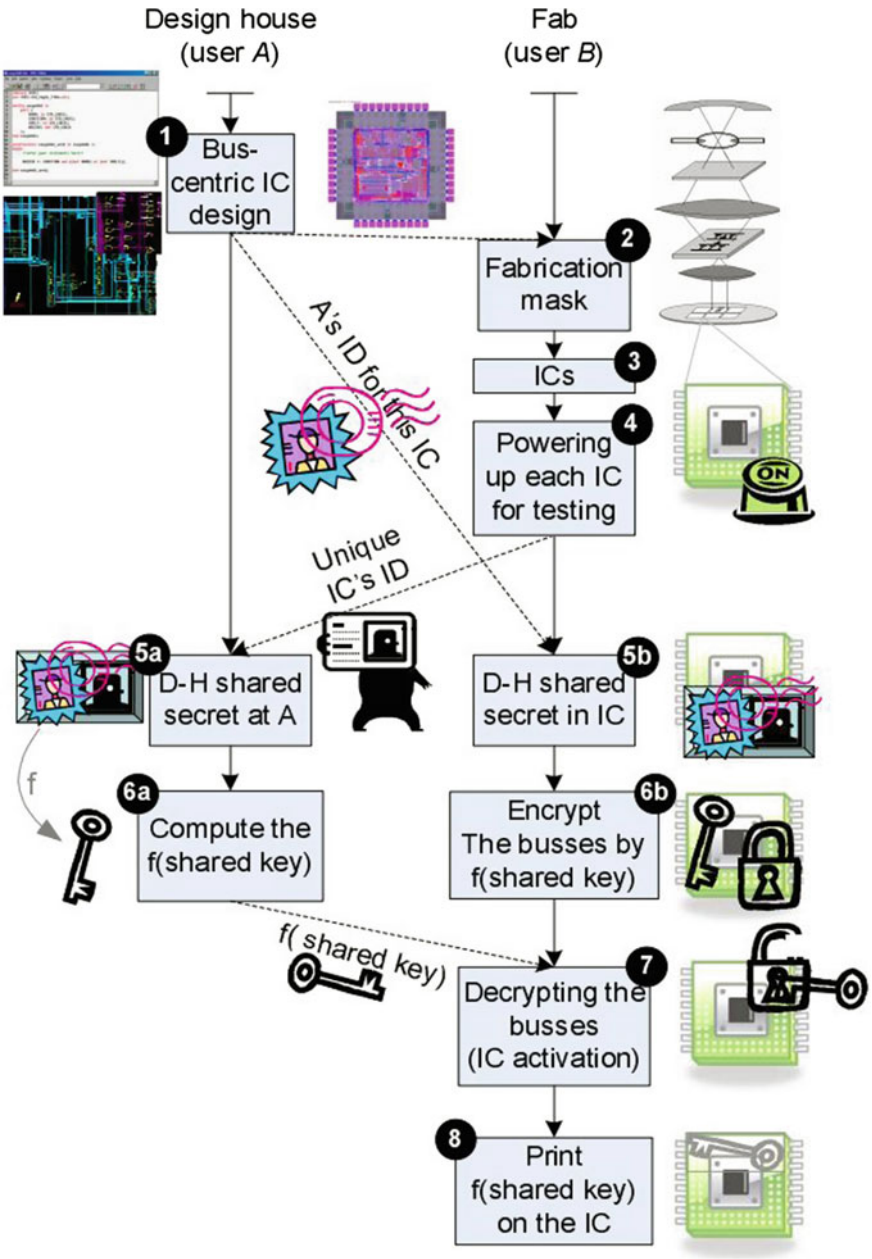


Fig. 5.14 IC activation protocol [13]

In this protocol, the obfuscated chip generates a unique identity number exploiting its embedded physical unclonable function (PUF) structure. The unique ID for the pertinent chip under test would be used as the D-H component b. The value $g^b \bmod p$ is then communicated to the design house (user A), who has also generated a unique ID a corresponding to the chip under test. In turn, A sends $g^a \bmod p$ to B. Now, the design house and the chip under test share the same secret. The key to unlock the chip can be computed by this shared secret by a secret one-way function f. This function is implemented in the hardware of the chip and only known by the designer. If the chip can be successfully activated, the key will be printed in this chip. The internal key storage mode is a convenient but unsafe scheme since these memories represent serious security concerns if they are compromised, and the keys are stolen [10, 24]. Besides the storage vulnerability, the D-H key exchange scheme encounters several known attacks. These vulnerabilities will be elaborated in Sect. 5.8.

Alternatively, the key can be loaded into the permutation network from outside the chip during powering up (Fig. 5.13b, c). The configuration information is stored in the embedded volatile memory and erased after the system loses power. As shown in Fig. 5.13b, the key is encrypted outside the chip and decrypted within the chip. Constrained by the power consumption, area overhead, etc., not all the permutation/obfuscated chips have this decryption capability. A more general case is provided in Fig. 5.13c. In this case, the key is loaded into the permutation block directly. For example, this could be done by the owner of the system through a USB drive or smart card.

5.6.1 Biometric-Based Key Generation

Besides the binary keys stored in the tokens, this key can be generated from a person's biometrics in the external key loading modes [25]. Biometrics have been investigated for identification, authentication, and key generation for many years [26]. Most of the biometric modalities, such iris and electrocardiogram (ECG), present strong capability against being duplicated by the attackers. A general ECG-based key generation flow is shown in Fig. 5.15. In this figure, the upper block illustrates the enrollment phase. During this phase, the user's biometric signal is collected and processed. The features of the processed signal are extracted and quantized to generate the enrolled key. The helper data shown in the enrollment phase consist of the information related to the key regeneration process. It is unnecessary to keep these helper data confidential since the enrolled key cannot be recovered from it. This enrolled key can be utilized to design the permutation network following the configuration situation 1 provided in Sect. 5.5.2. The key regeneration phase is presented in the lower block. This phase processes the following steps: preprocessing, feature extraction, and quantization with the support of helper data. These steps are the same as the enrollment phase and can be executed by the hardware integrated into the obfuscated system.

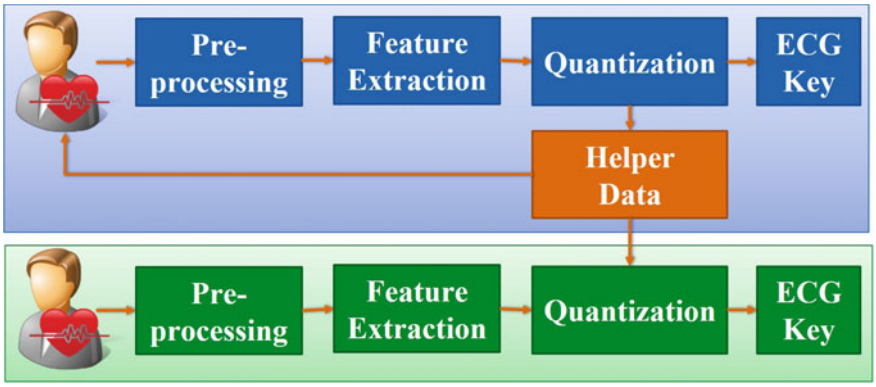


Fig. 5.15 ECG-based binary key generation

Bonding the biometrics and the obfuscated system provides a unique benefit. Combining the biometric-based key and the obfuscation protection enables a one-to-one relationship between the device and its operator. Based on the advantages of utilizing biometric-based keys, they are ideal candidates for some critical applications such as military devices.

5.7 Obfuscation Performance Evaluation

The obfuscation performance indicates how much work/time an attacker needs to devote in breaking the obfuscation. A comprehensive evaluation approach should be provided with the obfuscation framework. This evaluation can be split into two parts: (i) the feasibility of breaking the obfuscation by brute force and (ii) the robustness against other attacks.

The part (i) evaluation can be accomplished by providing expected time to break the obfuscation by brute force. This validation (T) can be calculated in Equation

$$T = \frac{t}{P} \tag{5.5}$$

where t refers to the time needed to verify one brute force guess. This guess can be either a key or an input/output combination of the permutation block. P refers the breaking probability defined at the beginning of Sect. 5.2. This breaking probability can be P_{com} or P_{key} depending which attacking strategy is applied. P_{com} and P_{key} are equal if and only if a one-to-one correspondence exists between the input/output combination and the key. As proved in Sect. 5.5.3, the expected P_{key} is smaller than P_{com} when the Benes network is attacked in a brute force manner. Thus, the breaking probability P_{com} evaluates the obfuscation performance better when the Benes

network is engaged. For other types of permutation networks, both of P_{key} and P_{com} should be computed, and the larger one indicates the obfuscation performance.

The calculation of P_{key} only depends on the total number of keys and the number of correct keys. For calculating the probability P_{com}, the result not only depends on the number of correct input/output combinations and the total combinations. For the board-level obfuscation, other knowledge from the chips' datasheets may help to increase P_{com}. An example [2] is provided in this section to show how this knowledge benefits in attacking the obfuscation. In the reference design presented in Fig. 5.4, the touch-sensing block will communicate with MCU via Rapid GPIO (RGPIO). Not all ports on the MCU have the capability to work as RGPIO (in fact, only 16 ports). In this sense, the attacker can shrink the scope by searching the correct connection for the touch-sensing device from 16 ports instead of all the obfuscated ports on MCU. In computing the breaking probability P_{com}, the functionalities of each obfuscated ports should be considered.

Part of the board-level obfuscation performance evaluations provided in [2] is presented in Table 5.3. The reference designs are listed below:

- Driving a Stepper Motor Reference Design with High-Performance MCU
- Ultralow Power Multi-sensor Data Logger with NFC Interface Reference Design
- Single-axis Motor Control Reference Design with Integrated Power Factor Correction
- SimpleLink Multi-Standard CC2650 SensorTag Reference Design

In the table, the column 'Programmable Component' provides the models of MCUs utilized in the reference. The column 'Break Probability' refers to the probability of breaking the obfuscation by examining the input/output combinations. The column 'Clock frequency' indicates the maximal operating clock frequencies of the programmable components. The parameter t in Eq. (5.5) is assumed as one clock cycle. Then, the column 'Validation Time' is computed by Eq. (5.5). The real validation time would be much longer than the value in Table 5.3 since it is extremely difficult for the attacker to validate each input/output combination in single clock cycle. In real world, he needs to observe the behavior of the board to tell whether this combination under test is correct. This process will be much slower than one clock cycle. Hence, these results should be considered as pessimistic. According to this table, besides the Design No.4, the validation times for the rest are longer than thousands of years. It is impossible to break the obfuscation by brute force in a reasonable time period. The validation time for Design No.4 is extremely short since this design only consists of a small programmable component with less than 32 pins. Among these pins, few of them can be obfuscated. Thus, in order to achieve a good board-level obfuscation performance, the original design is required to consist of a programmable component with more than 32 pins.

Besides the brute force attack feasibility analysis, the robustness against other attacks such as the hardware probing and reverse engineering should also be evaluated. The part (ii) evaluation studies the obfuscation's resistance of other types of attacks. This part of evaluation is presented in Sect. 5.8. In the same section, the corresponding countermeasures are also provided.

Table 5.3 Board-level obfuscation performance evaluation

Design no.	Programmable component	Break probability (P_{com})	Clock frequency (f) (Mhz)	Validation time (T)
1	TM4C123GH6PM	9.70E−41	60	5.4E+24 yrs
2	MSP430FR5969	1.34E−18	16	1.5E+03 yrs
3	TMS320F28050	1.23E−32	60	2.6E+24 yrs
4	CC2650	2.81E−15	48	85.8 h

5.8 Attack Analyses and Countermeasures

In the section, a list of potential attacks is provided. These attacks are grouped into three categories based on the attackers' capabilities. Against these attacks, the countermeasures are organized by the levels of security requirement.

5.8.1 Potential Attacks

To break the permutation-based obfuscation, the attacker needs to discover the following information: true intercomponent connections or the correct keys as discussed in Sect. 5.2. The following attacks can be carried out by the attacker to accomplish the attacking goal. These attacks cover the ones that can be either applied on chip level or board level. All the following attacks can be implemented on board level, while only a few of them are applicable on chips. Some of these attacks directly provide the attackers the information they desired, while some of them reinforce others. Since different attacks demand various equipment and knowledge, the capabilities of the attackers are studied.

Brute force attack is the most straightforward one and can be applied on both board level and chip level. As discussed earlier in Sect. 5.2, this attack entails attackers trying all the keys or the input/output permutations. The success of such attacks is highly dependent on the breaking probabilities (either P_{com} or P_{key}) and the time required to validate the system for correct behavior. It is extremely difficult for the attackers to break our obfuscation method by brute force as reported in [2].

Surface trace probing attack: The waveform between permutation block inputs and the corresponding outputs are the same for every system even though the key may be different. Attackers with a working system can probe all the I/Os of the permutation block with a multichannel logic analyzer. By simply matching the waveforms of the probed signals, attackers would find out the true connections between the programmable and non-programmable components. This attack can only be applied on board level since it applies to traces on the surface.

Storage compromise attack: The attacker may attempt to extract the keys/configurations if they are stored in the system. Several techniques can be utilized to retrieve the data from the nonvolatile memory [10, 24]. This attack is applicable on both board level and chip level when the internal key storage mode is applied. If the keys are the same for different chips/boards, this attack becomes powerful. Extracting one key from one chip enables the attacker to unlock other chips/boards.

Man-in-the-middle attack [13]: This is a well-known attack on the D-H key exchange scheme. Differing from the storage compromise attack, this attack provides an attacker the ability to compute the key for each chip/board. In this attack, the adversary intercepts designers public value and transmits its public value to the IC. When the IC sends its public value, the attacker substitutes it with it own and transmits it to the designer. Therefore, the adversary and designer agree on one shared key and IC and attacker agree on another shared key. Now, the attacker can simply decrypt messages transmitted by the designer and the IC and can read and potentially modify them before re-encrypting with the proper key and sending them to the other party. This vulnerability is possible because D-H key exchange protocol does not authenticate the users.

Permutation block reinstallation attack: This attack can only be applied on the board level. Removing the permutation block from the fabricated chip damages both the permutation block and the chip. Applying this attack, attackers unmount the permutation block (typically a CPLD) off the board and reinstall onto other platforms without damaging the package. Reinstalling the unmounted CPLD makes the surface trace probing applicable. This attack does not provide the attacker secret information directly. However, combining this attack makes surface trace probing attack more powerful.

Middle-layer probing attack: This is another attack which aims to empower the surface trace probing attack. Thus, this attack can be only applied on the board level. Attackers can discover full PCB layout by nondestructive reverse engineering approaches such as X-ray-based techniques [11, 27]. Guided by the layout information, attackers can mill holes for probing every CPLD port. These holes enable the attacker to access the CPLD ports without damaging other traces on the board. For the chip-level attack, microprobing could be used to read buses on the chip. This attack enables the attacker to sniff the data transmitted on the buses. However, this attack is much more expensive than the board-level probing.

IC Reverse engineering attack: The attacker can fully reverse engineer any chip on the board and learn completely secret information stored in these chips. These chips can be either the permutation block in board-level application or obfuscated chip in chip level. This secret information includes the firmware of the programmable component, internal or external memory content, and the permutation network structure. This attack can be applied on both chip level and board level.

In a well-known article from IBM [28], the authors suggest that the attackers can be grouped into three classes, depending on their expected abilities and attack strength:

- **Class I (clever outsiders)**:
  They are often very intelligent but may have insufficient knowledge of the system. They may have access to only moderately sophisticated equipment. They often try to take advantage of an existing weakness in the system, rather than try to create one.
- **Class II (knowledgeable insiders)**:
  They have solid specialized technical education and experience. They have varying degrees of understanding of parts of the system but potential access to most of it. They often have access to highly sophisticated tools and instruments for analysis.
- **Class III (funded organizations)**:
  They can assemble teams of specialists with related and complementary skills backed by excellent funding resources. They are capable of in-depth analysis of the system, designing sophisticated attacks, and using the most advanced analysis tools. They may use Class II adversaries as part of the attacking team.

The brute force attack, the surface probing attack, and the man-in-the-middle attack can be achieved by **Class I attackers**. Since these attackers only have insufficient knowledge of the system, brute force attack is the most straightforward one. Equipped with oscilloscopes, this class of attackers can accomplish the surface probing attacks. Compared with Class I attackers, **Class II attackers** can apply the storage compromise attack, permutation block reinstalling attack, and middle-layer probing attack with more sophisticated instruments. The tools required for these attacks should have the capabilities of uninstalling CPLDs without damaging the package, applying nondestructive reverse engineering, and drilling holes on a multilayer PCB. **Class III attackers** are the only ones that can apply IC reverse engineering attack since they have access to most advanced analysis tools and greatly funded.

5.8.2 Countermeasures and Attack Coverage

Considering the various classes of attackers and their capabilities, the countermeasures are organized into three security requirement levels.

Low-level security requirement: Devices satisfying this security requirement level directly store the same key in nonvolatile memories. To achieve low-level security requirement, we suggest that ball grid array (BGA) packages should be exploited for both the programmable component and permutation block and at least one middle layer in the PCB to route connections. The requirement of multiple layers is easy to meet due to the complexity of current PCBs. For the requirement of BGA package, a statistical result in Table 5.4 shows a significant portion of obfuscation candidates have BGA/VTLA (a package standard used by Microchip very similar with BGA package) package among the programmable components with equal to or more than 64 pins. As discussed in Sect. 5.7, a programmable component with more than 32 pins is required to guarantee a good obfuscation performance. Using BGA package introduces no additional area and power overhead. The cost is the same

Table 5.4 Percentage of obfuscation candidates with BGA/VTLA package

Manufacturer	⩾64 pins	BGA/VTLA	Percentage (%)
Microchip	241	107	44.40
Freescale	85	80	94.12
NXP	93	93	100
Total	419	280	66.82

between BGA package and other packages for the same model of the chip as well. When routing the PCB, the designer should identify all the connections between the programmable component and the permutation block and route them in the middle layers. The routing requirement is named as *middle-layer routing* for the remainder of the paper. This step is meant to reduce the attackers' chance of probing a working device to identify connections between chips. Low-cost tamper resistance techniques [29] should also be applied.

Medium-level security requirement: System rebooting is unavoidable for most industry systems and consumer electronics. Since reapplying keys after every reboot is often impracticable in this case, these systems need to store the keys in nonvolatile memory and reload them prior to rebooting automatically. This behavior provides attackers opportunities of breaking the obfuscation by certain techniques such as the permutation block reinstalling attack and storage compromise attack.

To obtain medium-level security requirements, the BGA packages and middle-layer routing need to be engaged. Moreover, instead of storing the same key for all the chips/boards, each of these systems should have its unique key. Since the internal key storage mode is applied, we can take advantage of the D-H key exchange scheme and the one-way function as provided in Sect. 5.6. Combining D-H key exchange scheme and a unique ID, each chip/board can be assigned a unique key. This unique ID can be generated by incorporating a chip-level or board-level PUF.

High-level security requirement: In this case, the key is not stored in system's nonvolatile memories and needs to be reapplied into the system after rebooting. Devices satisfying this security requirement should be capable of preventing all known attacks. Critical applications such as military installations and commercial devices storing sensitive data require this level of protection. The biometric-based keys (discussed in Sect. 5.6.1) can also contribute to this level of security requirement.

Since the key is not stored in nonvolatile memory, a D flip-flop chain is utilized to form a shift register in permutation block or obfuscated chip. One dedicated port on the chip is designed as the serial key input. When the system is powered off, D flip-flops lose their values, and the key is destroyed. The user needs to input the key to the system whenever it reboots. Additionally, BGA packages and middle-layer routing should also be involved in this level of security requirement.

Considering various attackers' capabilities with aforementioned security requirement levels, the protection coverage is analyzed in Fig. 5.16.

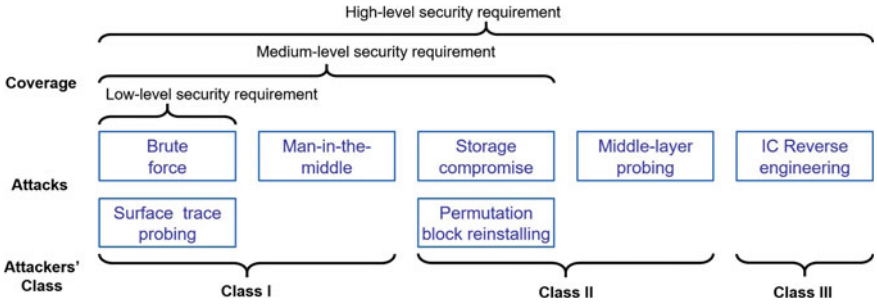


Fig. 5.16 Attacks coverage analysis

Devices satisfying low-level security requirement eliminate the brute force and surface trace probing attacks applied by Class I attackers. The surface trace probing attack is unavailable since the obfuscated connections are hidden in the middle layers. However, since the key is stored in the onboard nonvolatile memory, Class II attackers can unmount and reinstall the permutation block onto their platforms (permutation block reinstalling attack) for applying hardware probing attack. The techniques for extracting the content from the nonvolatile memory (storage compromise attack) also directly enable an attacker to learn the key. Since this key is the same among different boards, this compromised key can be used to active other boards.

The permutation block reinstalling attack, storage compromise attack, and man-in-the-middle attack can be eliminated if the device is compatible with the medium-level security requirement. Devices under this level of security requirement incorporate the obfuscation scheme with the following techniques: chip-level/board-level unique identifier; the D-H key exchange scheme; and the one-way function. This combination guarantees the following: (i) The permutation block can unlock the system using the prestored key only when it is attached to its original PCB; (ii) each chip/board can be only activated by a unique key; (iii) the man-in-the-middle attack is prevented. The permutation block reinstalling attack can be eliminated by the first guarantee, and the second guarantee prevents the storage compromise attack. Even if a man-in-the-middle attack establishes two different channels (one with the IC and one with the designer), he will not be able to compute the key specific to the IC or to use the key given for another chip.

Unfortunately, automatically loading the key after system rebooting again makes middle-layer probing attack available. Instead of reallocating the permutation block, this attack extracts the board layout nondestructively through techniques such as X-ray [11]. With this knowledge, Class II attackers can design and create holes to probe the permutation block pins. This attack can be accomplished during run-time without delayering the PCB. Even if the middle-layer probing attack cannot be achieved due to extremely complex middle layers, Class III attackers can always learn the key by reverse engineering the chips.

For completely preventing all known attacks, the key should be destroyed when the system loses power. High-level security requirement enables devices achieving this objective through storing the key in volatile memory (e.g., in the embedded D flip-flop chain). Since Class II and Class III attackers require a working device after rebooting, it is impossible for them to apply attacks on the devices which the keys are destroyed after powering off.

5.9 Conclusions

Among various obfuscation-based hardware protection approaches, the permutation-based technique presents certain advantages. For instance, this technique is the only one that can be exploited in protecting PCBs. Other obfuscation techniques such as logic encryption can be easily broken if applied at board level.

The permutation-based obfuscation permutes the intercomponent connections of either an IC or PCB. The designer needs to be aware of selecting these connections. Several requirements should be met when determining whether a connection is suitable for permutation. These requirements consist of the timing, functionalities, and signal types. Note that the requirements needed for the chip-level application are fewer than the ones for the board level. After appropriate connections are chosen, they will be permuted by a permutation network. The permutation network capability and its area utilization should be carefully balanced. For certain permutation networks (i.e., Benes network), the multiple-key effect can be observed. This effect causes more than one kind of the network's configuration result to end up with the same permutation outcome. This effect entails the designer to evaluate the brute force breaking probabilities in two ways (i.e., the probability when the attacker examines the keys or the input/output combinations). The larger breaking probability is used to determine the robustness against the brute force attack. The time required to break the obfuscation has a direct relationship with this breaking probability. As reported by the existing work, it may take longer than thousands of years to break a properly obfuscated system.

Besides the brute force attack, the designer should also evaluate the robustness against other attacks such as the hardware probing and reverse engineering. Since these attacks require various equipment and resources, they are classified into three levels based on the difficulty to execute them. Although some attacks are powerful in breaking the obfuscation, they can be mitigated by certain countermeasures. Similar to the classification of the attacks, the countermeasures are grouped into three levels based on the attacks they cover.

Since the permutation-based obfuscation is controlled by a key/configuration, various schemes can be exploited to manage the key either internally or externally. Each of these schemes has its advantages and drawbacks. The more convenient the key management scheme is, the less secure the system will be.

References

1. Roy JA, Koushanfar F, Markov IL (2008) Epic: Ending piracy of integrated circuits. In: Proceedings of the conference on Design, automation and test in Europe. ACM, pp 1069–1074
2. Guo Z, Tehranipoor M, Forte D, Di J (2015) Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In: Proceedings of the 52nd annual design automation conference. ACM, p 114
3. Chakraborty R, Bhunia S (2009) Harpoon: An obfuscation-based soc design methodology for hardware protection. IEEE Trans Comput-Aided Design Integr Circuits Syst 28:1493
4. Chakraborty R, Bhunia S (2010) Rtl hardware ip protection using key-based control and data flow obfuscation. In: 23rd international conference on VLSI design, VLSID'10. IEEE, pp 405–410
5. Koushanfar F (2012) Provably secure active ic metering techniques for piracy avoidance and digital rights management. IEEE Trans Inf Forensics Secur 7(1):51–63
6. Baumgarten AC (2009) Preventing integrated circuit piracy using reconfigurable logic barriers
7. Zamanzadeh S, Jahanian A (2016) Higher security of asic fabrication process against reverse engineering attack using automatic netlist encryption methodology. Microprocess Microsyst 42:1–9
8. Tehranipoor M, Wang C (2011) Introduction to hardware security and trust. Springer Science & Business Media, New York
9. Handschuh H, Paillier P, Stern J (1999) Probing attacks on tamper-resistant devices. Cryptographic hardware and embedded systems. Springer, Berlin, pp 303–315
10. Quadir SE, Chen J, Forte D, Asadizanjani N, Shahbazmohamadi S, Wang L, Chandy J, Tehranipoor M (2016) A survey on chip to system reverse engineering. ACM J Emerg Technol Comput Syst (JETC) 13(1):6
11. Asadizanjani N (2015) Non-destructive pcb reverse engineering using x-ray micro computed tomography. In: ISTFA
12. Zhang F, Hennessy A, Bhunia S (2015) Robust counterfeit pcb detection exploiting intrinsic trace impedance variations. In: IEEE 33rd VLSI test symposium (VTS). IEEE, pp 1–6
13. Roy JA, Koushanfar F, Markov IL (2008) Protecting bus-based hardware ip by secret sharing. In: Proceedings of the 45th annual design automation conference. ACM, pp 846–851
14. Waksman A (1968) A permutation network. In: JACM
15. Thamarakuzhi A, Chandy JA (2010) 2-dilated flattened butterfly: A nonblocking switching network. In: HPSR
16. Mitra D, Cieslak RA (1987) Randomized parallel communications on an extension of the omega network. In: JACM
17. Giacomazzi P, Trecordi V (1995) A study of non blocking multicast switching networks. IEEE Trans Commun 43:1163
18. Jajszczyk A (2003) Nonblocking, repackable, and rearrangeable clos networks: fifty years of the theory evolution. IEEE Commun Mag 41(10):28–33
19. Yang J, Yang J, Li X, Chang S, Su S, Ping X (2011) Optical implementation of polarization-independent, bidirectional, nonblocking clos network using polarization control technique in free space. In: Optic Eng 50(4):045 003–045 003
20. Feldman P, Friedman J, Pippenger N (1988) Wide-sense nonblocking networks. SIAM J Discrete Math 1(2):158–173
21. Pippenger N (1978) On rearrangeable and non-blocking switching networks. J Comput Syst Sci 17(2):145–162
22. Chang C, Melhem R (1997) Arbitrary size benes networks. Parallel Process Lett 7(3):279–284
23. Nassimi D, Sahni S (1982) Parallel algorithms to set up the benes permutation network. IEEE Trans Comput vC-31(2):148–154
24. Jeong H, Choi Y, Jeon W, Yang F, Lee Y, Kim S, Won D (2007) Vulnerability analysis of secure usb flash drives. In: IEEE international workshop on memory technology, design and testing, MTDT. IEEE, pp 61–64

25. Guo Z, Karimian N, Tehranipoor MM, Forte D (2016) Hardware security meets biometrics for the age of iot. In: IEEE International Symposium on Circuits and Systems (ISCAS)
26. Wayman J, Jain A, Maltoni D, Maio D (2005) An introduction to biometric authentication systems. Springer, London
27. Ahi K, Asadizanjani N, Shahbazmohamadi S, Tehranipoor M, Anwar M (2015) Terahertz characterization of electronic components and comparison of terahertz imaging with x-ray imaging techniques. In: SPIE sensing technology + applications
28. Abraham DG, Dolan GM, Double GP, Stevens JV (1991) Transaction security system. In: IBM Syst J
29. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: Identifying and classifying hardware trojans. IEEE Trans Comput 43(10):39–46

Chapter 6
Protection of Assets from Scan Chain Vulnerabilities Through Obfuscation

Md Tauhidur Rahman, Domenic Forte and Mark M. Tehranipoor

6.1 Background

Security and trust have become a critical aspect of modern IC design. Understanding the basic functionality of a system is essential for designing, analyzing, implementing, and assessing a system.

6.1.1 Cryptographic Algorithms and Security

Cryptographic primitives for data encryption and decryption, signature generation, and verification with the necessary cryptographic protocols are the core of all security and trust critical applications. System on Chips (SoCs) consist of a large number of sensitive assets that need to be protected from unauthorized access and malicious attacks or communications [1]. Multiple design blocks may be affected by the security policies because of the involvement of subtle interactions among hardware, firmware, OS kernel, and applications [1]. It requires end-to-end, layered security beginning at the device level to protect a device/system from all of these potential attacks. Cryptographic hardware unit is used to protect critical assets of a SoC from many forms of attack. It also assures that the code running in the board is authentic and/or trusted [1, 2]. In most of the cases, cryptographic primitives and schemes are critical blocks of a security solution. Confidentiality or privacy, integrity, authentication, identification, non-repudiation, and access control are the fundamental properties that need to be assured by the secure system. Confidentiality ensures that the information is secret from unauthorized parties. The integrity confirms that unauthorized party cannot change or modify any entity of a secure system. It also assures that the original

M. Tauhidur Rahman (✉) · D. Forte · M.M. Tehranipoor
University of Florida, Gainesville, USA
e-mail: rahman.tauhid@ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_6

data cannot be transmitted by an unauthorized party. Authentication is a service that identifies the origin of a message correctly. Non-repudiation prevents the message sender from denying any actions. Access control restricts the access to a resource for only authentic users [2–4].

Cryptographic algorithms are used to encrypt and decrypt confidential data. The secure communication and trusted executions rely on two important schemes: asymmetric or public-key cryptography and symmetric key cryptography.

A key encrypts a plaintext (confidential data in the form of a word, number, or phrase) and results in a ciphertext. The same key is used to decrypt the ciphertext in symmetric key algorithm where a different key is used to decrypt the ciphertext or encrypted message in public-key algorithm. The quality of both symmetric key algorithm and public-key algorithm depends on the secrecy of the keys being used because the algorithms are public. In symmetric key algorithm, the exchange of keys between two parties is critical since both encryption and decryption use the same key.

Block ciphers and stream ciphers are the two main divisions of symmetric key algorithm. Data encryption standard (DES) [3], advanced encryption standard (AES) [4], and triple DES [3] are the most popular block ciphers. However, AES is efficient for both hardware and software implementations. On the other hand, stream cipher encrypts or decrypts message bit by bit. OTP is the most popular example of stream cipher [5]. However, secure key exchange and storing those keys are the major challenges to the symmetric key cryptography. Another major problem is that the sender needs to store many keys for each receivers [6].

Public-key cryptography solves these problem by offering key pairs scheme: one for encrypting, the other for decrypting [6]. One key may be public and the other one is private so that only valid receiver can decrypt the message. Simplified key distribution, digital signature, and long-term encryption are the advantages of public-key algorithm over symmetric key algorithm. Digital signature ensures that a message is authenticated much like a signature or fingerprint on a paper document. RSA, ElGamal cryptosystem, and Elliptic curve cryptography (ECC) are the most popular examples of public-key cryptography [6–10].

In cryptography, hash functions are used for digital signatures, random number generator, one-way functions, message authentication codes (MAC), etc. [6]. Hash function takes an input of arbitrary or almost arbitrary length to output a fixed-length uniformly distributed numbers. SHA and MD families are the most popular hash function in cryptographic algorithm.

6.1.2 Fabless Supply Chain Vulnerabilities: Piracy and Prevention

In recent years, preventing IC theft, piracy, overproduction, out-of-spec/defective ICs by contract foundries or assembly is essential to both government and industries.

With the high costs associated with modern IC fabrication, most semiconductor companies have gone fabless, i.e., they outsource manufacturing of their designs to contract foundries. This horizontal business model has led to many well-documented issues associated with untrusted foundries including IC overproduction and shipping improperly or insufficiently tested chips.

Recycled, remarked, cloned, out-of-spec/defective, overproduced, cloned, forged documented, and tampered ICs are the types of counterfeit ICs that may be encountered in today's horizontal semiconductor supply chain. The following types are most relevant to this chapter.

- **Overproduced:** In horizontal business model, any untrusted foundry/assembly that has access to a designer's IP can exceed the agreed volume contact and sell the overproduced ICs on the gray/black market to gain high profit by avoiding the IP development cost [11–14].
- **Out-of-spec/defective:** There is no guarantee that untrusted fab or assembly will perform IC testing correctly, or even at all. Such defective parts may exhibit correct functionality for the most part and therefore be very difficult to spot in the supply chain. Untrusted foundry or untrusted third party can sell these rejected or out-of-spec components to open markets. These components can pose a serious threat to the quality and reliability of a system [11–13, 15].
- **Cloned:** A malicious manufacturer can obtain the design files illegally and clone the device in order to gain huge profit without developing IP. A cloned component is an unauthorized production without a legal IP and a very popular practice by the adversaries or untrusted parties. Cloning can be done through theft, espionage, or reverse engineering. Untrusted party can add changes and resue IP in its own products. Reverse engineering is usually used to obtain the whole design by taking their mask data. The mask data is then converted to functional level through transistor-level netlist and gate-level abstraction. Reverse engineering is also used to recover unavailable specifications of integrated circuits and to design new ICs using recovered data. Sometimes cloning can be done by copying the contents of a memory used in a tag for electronic chip ID, bitstream targeted to programmable gate arrays, etc. [11, 14, 15].

6.1.3 Overview: IC Testing, Scan Chain, and Industrial Compression Scheme

6.1.3.1 Scan-Based IC Testing

The main objective of a Design-for-Test (DfT) engineer is to maximize the test coverage while minimizing the costs and test time associated with testing. Scan chains are the most effective way to support testability of any ICs. Access to any nodes and observing them make scan chains based testing easier and popular. Scan chains based testing covers both functionality of an IC and stuck-at-faults caused by

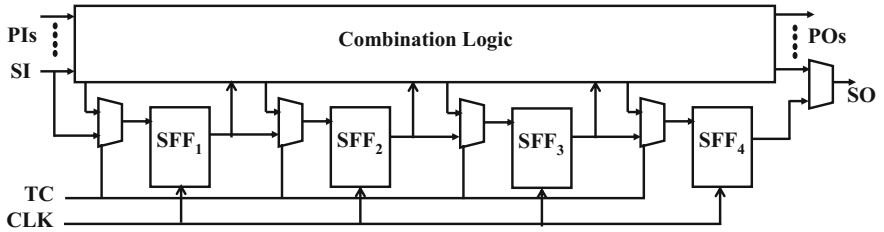


Fig. 6.1 Scan chain based testing [16]

manufacturing defects. Low overhead, effectiveness, and high fault coverage have made the scan chain the most popular and a standard method of testing digital chips. Setting and observing every flip-flop of an IC is the most prominent feature of scan chain based testing.

An example of scan chain is shown in Fig. 6.1. With scan chain, a synchronous sequential circuit works in two modes: normal mode and test mode. One additional pin, test control (*TC*), to the primary I/O is used in order to switch between test mode and normal mode. A MUX is placed at the input of flip-flop (*FF*) in such a way that all *FF*s can be connected in a shift register for one mux selection and to work in a normal mode in the other. The standard *FF*s with MUX are known as scan *FF*s (*SFF*s). Controllability and observability are the two main features that have made the scan chain testing popular. Controllability refers to the fact that the user can set the *FF*s to a desired state, while observability refers to the power to observe the content of the *FF*s. The *FF* registers make up the I/O to the combinational logic blocks in the chip, so test engineers are able to manipulate the values that are input (controllability) and view the output (observability) of each block. This is performed by multiplexing one primary input pin and one primary output pin as the scan-in (*SI*) pin and scan-out (*SO*) pin, respectively. Using the *SI* pin while the *TC* is enabled, a test pattern is scanned into the scan chain as dictated by the system clock [9, 10, 17]. When the entire pattern is scanned in, the *TC* is disabled, and the chip is run in normal mode for one cycle storing the responses back into the *SFF*s. *TC* is again enabled to scan out the response, while at the same time, scanning in a new test pattern to check for new faults that previous patterns were not able to detect. Using this method of test, sequential logic essentially becomes combinational logic during test. Creating test patterns that achieve high fault coverage is a much easier task for combinational logic than it is for sequential logic, significantly speeding up the test pattern generation process [9, 10, 17].

6.1.3.2 Advanced DfT Structures

Scan chain based testing offers better testability in terms of high fault coverage. Higher integration and short time to market have been major concerns because of the rapid growth of test volume of data. The testing time depends on the test data

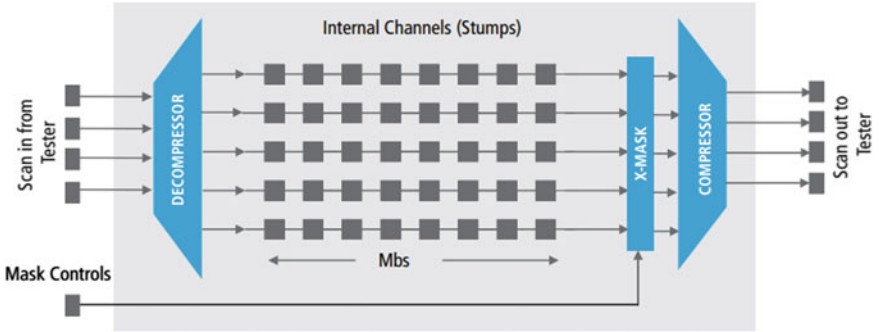


Fig. 6.2 Compression block diagram (*source* [19])

volume, core's test data bandwidth, and automatic test equipment (ATE) channel capacity. The production capacity, time to market, and test cost rely on testing time, and the goal of test engineers is to reduce the test time with a very high test coverage. Scan compression is the most commonly used DfT architecture to serve the above purposes. Figure 6.2 shows a typical compression structure which consists of three distinct blocks: a decompressor, a compressor, and an X-tolerance or X-mask. A long scan chain is split into multiple shorter scan chains in order to reduce the test cost and time. The decompression unit expands the input test patterns to fill several scan chains, and the compression unit compresses the scan outputs (i.e., response) to support limited number of output pins. XOR, multiple input signature register (MISR), serial MISR compression, on-product MISR (OPMISR), hybrid compression, etc. are commonly used compressors. On the other hand, broadcast and spreader are the most common decompessor techniques. Masking is used to filter out the unknown states so that they do not corrupt the responses. The unknown values (i.e., Xs) are due to uninitialized memories and flip-flops, or bus contention [18]. Test stimuli patterns, which are generated and stored in the tester, are applied to the decompressor. DFTMAX of Synopsys, Cadence encounter with OPMISR, VirtualScan of SynTest, and Tessent TestKompress of Mentor Graphics are the popular industrial test compression tools.

6.2 Assets, Threats, and Scan-Based Attack

Security ensures that anything in a circuit is safely stored within itself. The most common manner of providing security is to hide the information behind some form of recognition that would be able to tell a valid user from an attacker. Modern day security in all realms uses this method to protect vital belongings, whether it is a security code for a home, retinal scanner for a lab, or encryption key for information. Security relies on making information obscure and difficult to figure out. To build a

secure integrated circuit, a designer must decide what assets to protect, and which of the possible attacks to prevent. Further, IC designers must also understand who the players (attackers and defenders) are in the IC design supply chain and have the means for quickly evaluating the security vulnerabilities and the quality of the countermeasures against a set of well-defined rules and metrics.

6.2.1 Assets

As defined in [20], asset is a resource of value which is worth protecting with respect to the adversary. The main limit of a security solution of a system is that it cannot shield against all type of attacks. So an asset is an important design consideration for a particular system. An asset may be a tangible object, such as a signal in a circuit design, or may be an intangible asset, such as controllability of a signal. Driven by the increasing demand for secure computation and communication in the era of Internet of Things (IoT), more assets are created and need to be protected. Sample assets that must be protected in a SoC are listed below [15, 20, 21]:

- **On-device key:** Secret key for cryptographic applications.
- **Program content:** Change in algorithm (e.g., changing program to control mileage reading in automobile or changing program in banking).
- **Entitlements:** Accessing to sensitive information by right person.
- **Random numbers** [22]**:** Used in cryptographic operations.
- **Embedded firmware:** Low-level program instructions, proprietary firmware.
- **Design and user data:** Sensitive data such as personal information of clients and mileage reading of automobile.
- **Device configuration:** Configuration data of devices, initialization data for security processing, etc.

The secret key/random number used for different hardware obfuscation techniques is also an asset in need of protection.

6.2.2 Features and Assumptions to Perform Attack

To ensure a secure design, a threat model is created for a system that anticipates different kinds of threats and seeks to mitigate them. The security designer is also building in layers of protection, seeking to isolate a threat before it can affect secure operations. The main focus of this chapter is to focus on scan chain based security. As it has been said before that by designing for testability, a designer is essentially revealing all information about the chip through the use of scan test. If the aim of designing a chip is security, it is very difficult to justify the amount of controllability and observability that testability aims to provide because of these leaks. The following features and assumptions are key and resources of scan chain based attack [23].

- **Assets are secret:** Key for obfuscation, cryptographic key, random numbers, etc. are unknown.
- **Scan chain and assets:** Secret key is not connected to the scan chain. However, the flip-flops of locked IC form scan chain.
- **Control of *TC* pin:** The attacker has the ability to run the device under normal mode or test mode interchangeably using *TC* pin.
- **Known cryptographic algorithm:** The cryptographic algorithms might be known to attacker.
- **Architecture of scan chain is known:** Depending on the attackers, the architecture of scan chain might be known or unknown to the attackers.

6.2.3 Attack Model

In this section, we discuss about different types of potential attackers. There are many attackers in the world with many different motivations. They range from the noble, attempting to make their fellow developers aware of their pitfalls, the malicious, stealing information that does not rightfully belong to them, and simply the curious [17, 24]. Classification of attackers is vital for the chip designers to choose the right countermeasures.

Different attackers target different phases of IC supply chain depending on their capability, availability, and resources. The attackers can be classified in terms of attack at different phases of supply chain as follows [25]:

- **Authorized test engineer:** The attacker has full access to the test mode and he/she also knows the design but not the assets.
- **In-the-field attacker:** This type of attacker tries to activate the test features in the field.
- **IP provider:** A third party designer who designs hardware/software module for the system.
- **Foundry/assembly in fabless semiconductor model:** Foundry/assembly does not know about the functionality of an IC but may attempt to get the functionality in order to overproduce or sell the design to a third party.

6.2.4 Scan-Based Attack

6.2.4.1 Attack Principle

The main objective of a test engineer is to shift-in test input vectors and shift-out the corresponding responses. On the other hand, the attacker's main objective is to shift-in the corrupted/controlled data and shift-out the confidential data. The attackers can observe confidential data by controlling and observing the contents of *SFF*s. The

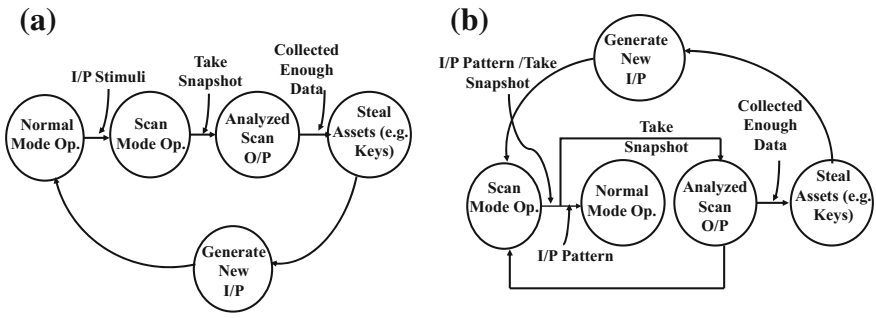


Fig. 6.3 **a** Scan-based observability attack and **b** Scan-based controllability/observability attack [17, 24]

scan-based attacks can be categorized into the following: *scan-based observability* and *scan-based controllability/observability* attacks. In both cases, an attacker has access to the test control (TC) pin. The type of attack depends on how a hacker decides to apply stimuli.

Scan based ob@Scan-based observability Attack: A *scan-based observability* attack relies on attackers' ability to use the scan chain to take snapshots of the system at any time, which is a result of the observability from scan-based testing. Figure 6.3a shows the necessary steps to perform a *scan-based observability* attack. The hacker begins this attack by observing the position of critical registers in the scan chain. First, a known vector is placed on the primary input (PI) of the chip and the chip is allowed to run in functional mode until the targeted register is supposed to have data in it. At this point, the chip is placed into test mode using *TC* and the response in the scan chain is scanned out. The chip is reset and a new vector that will cause a new response only in the targeted register is placed on PI. The chip again is run in functional mode for the specific number of cycles and then set into test mode. The new response is scanned out and analyzed with the previous response. This process continues until there are enough responses to analyze where in the scan chain the targeted register is positioned. Once the targeted register is determined, a similar process can be used to either determine a secret key in the case of cryptochips or determine design secrets for a particularly innovative chip.

Scan-based controllability/observability: *Scan-based controllability/observability* attacks take a different approach to applying stimuli to the CUT, which is shown in Fig. 6.3b. Scan-based controllability/observability attacks begin by applying the stimuli directly into the scan chain as opposed to the PI. In order to mount an effective attack, the hacker must first determine the position of any critical registers as was done for the *scan-based observability* attack. Once located, the hacker can load the registers with any desired data during test mode. Next, the chip can be switched to functional mode using the vector the hacker scanned-in, potentially bypassing any information security measures. Finally, the chip can be switched back to test mode to allow the hacker a level of observability the system primary output (PO) would not

provide otherwise. As opposed to using a known vector to scan into the chain, hackers also have the opportunity to choose a random vector to induce a fault in the system. Based off of the fault-injection side-channel attack [26, 27], by inducing a fault, the chip may malfunction potentially revealing critical data. The scan chain becomes an easily accessible entry point for inducing a fault and makes the attack easily repeatable. In order to protect from such side-channel attacks, additional hardware security measures must be included in the design.

6.2.4.2 Existing Scan-Based Attacks

As previously explained, attacker can know the information of an asset by controlling the state of *SFF* and observing the scan-out. However, the scan-based attacks have been more powerful when combined with side-channel attacks. Implementing encryption algorithms in hardware have revealed quite a few methods to discover the secret keys through side channels. These side-channel attacks include differential power analysis, timing analysis, fault injection and, most recently, scan chain hijacking, as demonstrated in [28]. It is also possible to reveal proprietary information through these side-channel attacks making these a particularly large concern to semiconductor industry. There have been different types of attacks that have been proposed and/or performed on cryptographic algorithms and DfT structures.

Differential scan-based attack is the most common and useful approach for retrieving secret keys. Figure 6.4 shows the basic concept of differential scan-based attack. In this method, an attacker applies a pair of plaintext $\{P1, P2\}$ in order to check the hamming distance between the values of intermediate registers, $\{SO1, SO2\}$, collected from scan output. The CUT is first reset and then the plaintext $P1$ is loaded through test data input (*TDI*). Then, after some clock cycles, the contents of intermediate registers, $SO1$, are read through scan output. A test mode select (*TMS*) is used to move from normal mode to scan mode. The same procedure is used for the other plaintext, i.e., $P2$, and corresponding contents of intermediate registers, i.e., $SO2$. The hamming distance between $SO1$ and $SO1$ helps to retrieve the secret information. Figure 6.5 shows different types of attack.

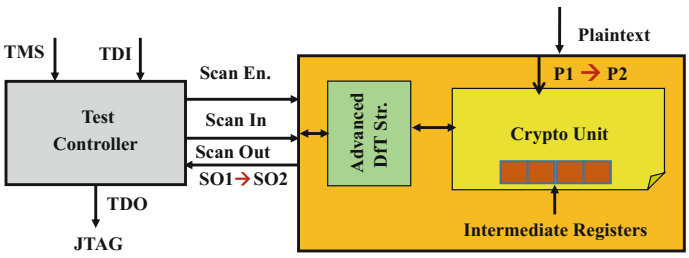


Fig. 6.4 Differential scan-based attack

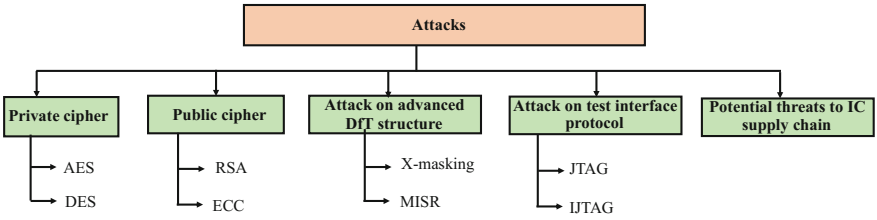


Fig. 6.5 Existing major scan-based attacks

- **Attacks on symmetric/private key encryption:** For symmetric key encryption, the intermediate state of a *SFF* depends on multiple secret bits. Yang et al. proposed the first scan-based attack on DES block cipher [28]. DES algorithm is a symmetric key encryption developed by IBM. In DES, a 56-bit user key and algorithm are applied to a 64-bit block of data simultaneously rather than one bit at a time. Each block is enciphered using the secret key into a 64-bit ciphertext by means of permutation and substitution. The process involves 16 rounds and can run in four different modes, encrypting blocks individually or making each cipher block dependent on all the previous blocks. A brute force attack would take a maximum of 2^{56}, or 72058 trillion attempts to find the correct key. The proposed scan-based attack is performed in two phases. The structure of the scan chain is determined in the first phase. This is done by switching the CUT between normal mode and test mode using *TC* (see Fig. 6.4). In this phase, at first, the CUT is run in normal mode for a clock cycle. Meanwhile, the result of XORing plaintext with the key and the first round is stored in the round register. Then, it is switched back to scan mode in order to collect the scan outputs. It is repeated another time for different plaintexts with a hamming distance of 1 byte. The locations of intermediate registers are found by observing the scan output patterns. In next phase, the attackers find the round key in a byte-by-byte manner by applying plaintext with a particular. At the same time, the corresponding word is observed on the round registers. One byte of the key is deduced by XORing corresponding ciphertext byte and plaintext byte. The process continues until all of the bytes of the key are deduced.
  The National Institute of Standards and Technology (NIST) announced an initiative to choose a successor to DES because the length of 56-bit key was not enough for strong encryption in modern computers. The advanced encryption standard (AES) replaces the DES to support strong encryption for modern computing. AES, also symmetric key encryption, offers three block varieties, AES-128, AES-192, and AES-256. Each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic 128-bit, 192-bit, and 256-bit keys, respectively. Initially, the main target of researchers was to design an encryption mechanism that is capable of protecting sensitive information well into the next century. However, an attack on AES is proposed in 2005 by Yang et al. [29]. The attack was based on differential scan attack with the fact that two particular inputs to the round function of a block

cipher can transform into output vectors with a unique Hamming distance after one round of encryption. In the proposed attack, the difference of scan contents is analyzed rather than the direct value itself. The proposed attack in [29] proves that the AES is vulnerable to scan-based attacks.

- **Attacks on public-key encryption:** Several scan-based attacks have been proposed to retrieve secret key of public-key cryptography [7, 30]. The Rivest–Shamir–Adleman (RSA) algorithm is the most popular public-key cryptographic algorithm. RSA is used in a wide range of key exchange protocols and digital signature schemes. Nara et al. [30] proposed scan-based attack on RSA using differential scan-based technique. The attack is focused on a 1-bit time sequence which is specific to some intermediate value called scan signature. Elliptic curve cryptography (ECC) is used to give the same level of security provided by an RSA. The attack on ECC was reported in [7]. Scan-based attack was proposed on stream ciphers in [31].
- **Attacks on advanced DfT infrastructure:** The advanced DfT structure is also susceptible to scan-based attack [32–34]. Liu et al. [35] showed that the advanced DfT structure (see Sect. 6.1.3.2) prevents some initially proposed scan-based attacks such as [28, 29]. They analyzed the complexity of side-channel attacks on designs with embedded decompression and compaction circuit [35]. Without compaction, the values stored in the *SFF*s are directly observable at the test output. In contrast, each bit of compaction output depends on contents of multiple *SFF*s. However, Rolt et al. [34] showed in signature-based attack that it is possible to retrieve the secret from compacted responses, even if the amount of information related to the secret key is extremely decreased by observing almost any *FF* containing information related to the secret data. But, the proposed signature-based attack focuses only on full-scan circuits with linear spatial response compactors. Rolt et al. expand the scan-based attack from linear spatial response compactors to common industrial techniques such as partial scan, X-masking, and MISR, all techniques for which only part of the circuit state can be observed on scan-out pins [33].
- **Attacks on test interface:** Secure test wrappers (protocol countermeasures) are inserted around the circuit under test (CUT) interface. Test wrappers provide both test access and test isolation during scan pattern application. The IEEE 1149.1 Boundary scan test, also known as the Joint Test Action Group (JTAG) standard, has been widely accepted and practiced in the testing community. The standard provides excellent testing features with low complexity. JTAG enables serial access to internal scan chains through external pins. Furthermore, instruction memory access is given to allow uploading and modification of the software/component. Access to these tools can allow an adversary to steal or change the intellectual property of an IC. Possible attacks and security for JTAG were presented in [36, 37].

  The IEEE P1687 (IJTAG) standard, designed to access embedded instruments, is an extension of 1149.1. [38]. In this standard, there is a scan network that allows access to embedded instruments by opening and closing Segment Insertion Bits

(SIBs). A SIB has to be open to access a new segment of the scan network. The SIB has to be closed in order to bypass the segment that makes the overall scan path shorter [38]. An attacker is able to find and open all SIBs by walking a logic 1 (or a logic 0) through the network with iterative DR scans [39].

- **Potential threats to IC supply chain:** Active metering, logic obfuscation, source code encryption, and bitstream encryption for FPGA are the major existing solutions to mitigate the reported attacks [12, 13, 15, 40–44]. Many of these schemes rely on encryption of combinational logic and/or finite-state machine (FSM) block via obfuscation and locking mechanisms. In the locking mechanism with obfuscation, only a valid specific input vector (i.e., a unique key) leads the IC to its correct functionality. Otherwise, the circuit/algorithm will function incorrectly because of logic obfuscation of the design. In logic obfuscation, extra logic blocks are inserted in the main design that only become transparent with a valid key. For example, a group of extra finite states are added in order to lock the FSM and only valid input sequence can bring the modified FSM to the correct initial state in normal working mode.

  Active metering allows the IP owner to lock and unlock each IC remotely. The locking mechanism is a function of the unique ID generated for each IC by a physically unclonable function (PUF, [45–50]). Only the IP owner/authentic user knows the transition table and can unlock the IC from this ID. In EPIC [43], each IC is obfuscated with randomly inserted XOR gates. The XOR gates will only become transparent with the application of valid key (effectively unlocking the IC). In this technique, a set of public/private keys needs to be generated by the IP owner, foundry, and each IC. The primary objective of these approaches is to give the IP owner control over the exact number of ICs that can enter the market by obfuscating the correct behavior. However, Maes et al. [51] presented a technique that finds the proper functionality and the placement of XORs. The above-mentioned techniques cannot prevent overproducing completely. Also, there is a chance that out-of-spec or defective ICs are sent to the market without letting the IP owners know.

  Contreras et al. [12] proposed a technique called secure split test (SST) that prevents overproduction and out-of-spec or defective ICs being in market along with other IC piracy. This technique locks both functionality and scan chain so that assembly and foundry do not know what are the responses of corresponding test vectors. This functional locking and scan locking mechanism helps IP owner to decide which chips should go to the market. The scan-locking mechanism has to be very strong to prevent IC piracy. A simple attack scenario is that an attacker will purchase an unlocked IC from the market and test that IC to get the correct response. However, a strong scan obfuscation with key is required to prevent this attack.

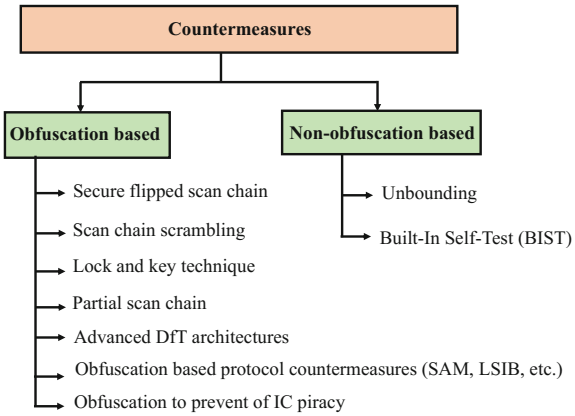


Fig. 6.6 Existing countermeasures against scan-based attacks

6.3 Countermeasures Against Scan-Based Attacks

Due to the side-channel attacks, a lot of attention has begun to be paid toward the inclusion of security during design. There have been several countermeasures against side-channel-based attacks. Traditional side-channel leaks have often been secured with the use of additional circuitry. Power analysis attacks can be prevented with noise inducing circuitry or applying additional circuitry to hide supply variations [52]. Timing attacks can be prevented by adding additional gates, so all operations are performed in the same amount of time or to add random delays to the processing time [53]. Finally, fault-injection attacks can be detected with additional logic that performs the inverse operation of the original logic to check if the result reproduces the input [54, 55]. There has not been much work done that is directly related to the security of scan chains. Each countermeasure technique suffers some limitations.

Two classes of countermeasures can be found in literature (Fig. 6.6): obfuscation-based and non-obfuscation-based.

6.3.1 Non-obfuscation-based Countermeasures

The main non-obfuscated countermeasure techniques and their limitations include the following:

- **Unbounding:** A traditional method, which has become popular in smart card security, has been to blow polysilicon fuses that interrupt interconnects to the test ports or directly in the scan paths. However, it has been shown that the fuses can be reconnected with minimally invasive techniques [56]. There is also the option to completely cut off the test interface with a wafer saw [57]. Either option eliminates any possibility for in-field testing. Most have gotten around the concern by using

BIST. Probing attack was proposed in [58]. There is also the option to completely cut off the test interface with a wafer saw.

- **Built-In Self-Test (BIST):** BIST provides inherent security since the test response patterns are not available externally. BIST is well known for its numerous advantages such as improved testability, at-speed testing, and reduced need for automatic test equipment (ATE). In BIST, a linear feedback shift register (LFSR) generates test patterns and an MISR compacts test responses [59]. In [60], Hafner et al. used BIST to test the entire cryptochip they designed. It provided high fault coverage for both the standard cells and memories but did not do fair nearly as well on the custom-designed portions of the chip. Both BIST and boundary scan were used in [61]. The fault coverage still was not nearly as high as what could have been achieved with automatic test pattern generation (ATPG) for scan-based design. Any security sensitive I/O were excluded from the boundary scan, and it was not specified how such I/O were tested.

6.3.2 Obfuscation-based Countermeasures

On the other hand, obfuscation-based countermeasures are less expensive and do not require secure test wrapper. In this technique, a secret function is inserted within the scan chains in order to obfuscate the contents of scan chains. Existing obfuscation-based countermeasures for scan-based attacks are presented in Fig. 6.6 and are briefly introduced below.

- **Secure flipped scan chain:** In this method, a certain number of inverters are inserted between randomly selected scan cells [63]. The working principle of secured flipped scan chain is similar to conventional scan chains and does not use any additional test key bits or clock cycles. One of the major advantages of this technique is that the random presence of inverters makes it very difficult for attackers to ascertain the structure of the scan chain. The randomly inserted inverters into the internal scan path change the values of scanned data. However, the positions of inverters can be determined by resetting all the flip-flops in the scan chain.
- **Scan chain scrambling:** Hely et al. [62] present a method to prevent invasive and semi-invasive attacks by modifying the scan chains to internally scramble the values if the test mode was not properly secured. In order to do so, Hely proposed splitting up the scan chain into segments that connect to some other segments in the scan chain. By using a random number generator, the segments would internally scramble the contents of the scan chain making the output difficult to decipher. In secure mode, the scan chain elements are ordered in predetermined manner, but the order is changed randomly in insecure mode (Fig. 6.7).
- **Lock and key technique:** Yang et al. [29] proposed a method that only prevents access to sensitive registers during test mode. With the use of a mirror key register (MKR), they were able to remove the encryption key of an AES hardware imple-

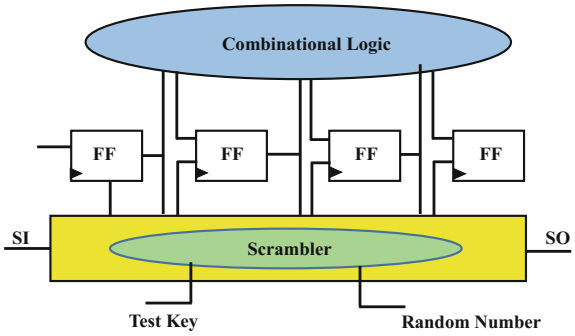


Fig. 6.7 Scan chain scrambling [62]

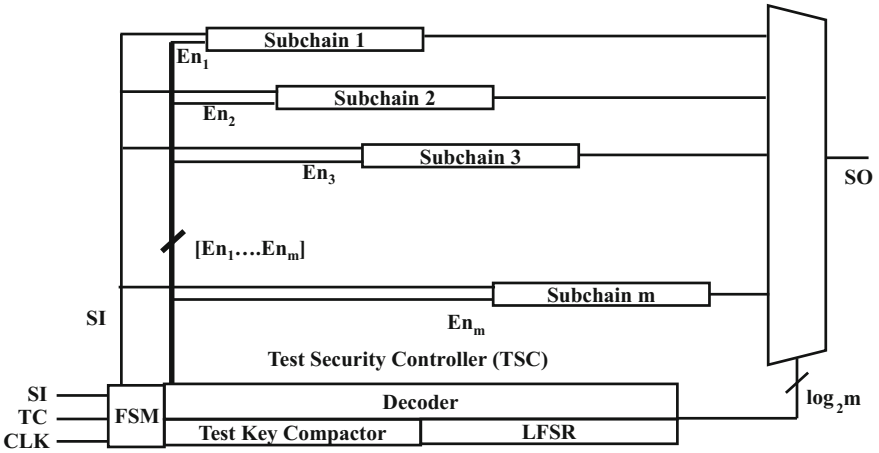


Fig. 6.8 Architecture of lock and key security measure [16]

mentation from the scan chain while the chip was set to insecure mode. Although this method works effectively to hide the secret key, it only provides security for special registers and not for the entire scan design. This method also requires a modification to the JTAG standard [64] in order to be effective.

Lee et al. proposed [16] a Lock and Key security measure that can be used to secure both single and multiple scan designs. For either case, the scan chain can be divided into smaller equal length subchains. Test vectors are not sequentially shifted into each subchain but rather a linear feedback shift register (LFSR) randomly selects a subchain to be filled. Figure 6.8 shows a general architecture for the proposed Lock and Key method for single scan design. The goal of this method is to prevent those who do not hold the test key from manipulating the scan chain and revealing vital information about the chip. This is ensured by the test security controller (*TSC*), which consists of a finite-state machine (FSM), test key comparator, LFSR, and decoder. The whole procedure can be summarized as follows:

- There are two states the *TSC* can be in: secure and insecure modes. When the CUT is initially reset, the FSM sets the *TSC* into insecure mode and will remain in this insecure state until *TC* is enabled.
- It is only after *TC* has been enabled for the first time and a test key has been entered that the *TSC* may exit the insecure state. When a test key is entered and a user has been ensured to be a trusted user, the FSM allows the *TSC* to enter secure mode.
- There are two steps the FSM must take before scanning in a test vector for the first time. When TC is initially enabled, the FSM will first need to check for a correct test key. It will feed the first k bits of the test pattern, which makes up the test key, to the test key comparator. The comparator will then return a pass or fail response to the FSM, which will then decide the next state. If the key passes, the FSM will switch the *TSC* to secure mode allowing predictable operation of the scan chains and will remain in this state until the CUT is reset. Otherwise, the *TSC* will remain in insecure mode and the behavior of the scan chain will no longer be predictable.
- Assuming the test key comparator returned a pass response to the FSM, the next $q = log_2(m + 1)$ bits will then be fed to the LFSR and used as an initial seed, where q is the size of the LFSR and m is the number of subchains being implemented. The seeded LFSR will then use a decoder as an interface to the subchains for a one-hot output, which individually enables each subchain. Assuming l is the length of each subchain, the LFSR/decoder generates the next one-hot output after l clock cycles. Finally, the FSM connects *SI* to the inputs of the subchains and the test pattern can be shifted into the scan chain.
- Once the scanning in process is finished for the first round, *TC* goes low and the CUT will function in normal mode capturing the response in the *SFF*s.
- Once the CUT returns to test mode, a new test vector is scanned into the subchains in the same or a new random order, depending on the design, as the previous vector was scanned in. The response is shifted out at the same time the new pattern is shifted in.
- If the entered key fails, the *TSC* remains in insecure mode and will seed the LFSR with an unpredictable random seed, essentially locking the scan chains from being used correctly. Since the choice of subchain is pseudorandom due to the LFSR, it is difficult to predict the response on *SO* if both the seed and the configuration of the LFSR are unknown. Even if the configuration of the LFSR taps is known, if the LFSR is large enough, it is difficult to know the subchain order without first realizing the initial seed. The need for a test key also compounds any attempt made by an unauthorized party to use the scan chain.

- **Partial scan chain**: Inoue et al. [65] propose partial scan method, based on partial scan and logic obfuscation, to guarantee high security and high testability simultaneously. This method is used to protect non-scan registers from scan-based attacks by employing a test controller that enables the test mode only [65, 66]. The proposed attack in [33] is quite effective against partial scan combined with

X-masking and X-tolerant logic [66]. Some *FF*s move to test mode from normal mode in VIm-Scan with the proper sequence of test keys [10].

- **Advanced DfT architectures:** Advanced DfT (i.e., compression) architectures are used to reduce the time overhead for large designs. It has been reported in [35, 67] that the advanced DfT architecture is resilient to scan-based attacks. Without compression, the contents of *SFF*s are directly observable. However, the value of a scan output pin depends on the contents of multiple *SFF*s, and thus, attacking advanced DfT structure is more complex compared to DfT structures without compression unit [35, 67, 68]. However, advanced DfT structures are not possible in resource limited IPs/designs. Recently, attacks on advanced DfT attacks have been proposed in [18, 32–34, 66, 68].
- **Obfuscation-based protocol countermeasures:** Pierce et al. proposed a [69] detailed hardware specifications for the enforcement of a multilevel access JTAG system. Their proposed challenge–response-based authentication protocol, they called it secure authentication module (SAM), is shown in Fig. 6.9. In the power-up, the JTAG interface is set to a locked state. This locked state prevents access to the JTAG instruction registers. There are also some dedicated instruction registers, accessible anytime, for SAM operation. The SAM instruction decoder only responds to an unlocking request instruction while in the locked state. The user is not able to change the selected registers through JTAG instruction registers without a complete authentication. At first, to complete the authentication process, the user sends a challenge request through the time delay integration port. SAM generates a challenge and loads back them into SAM instruction registers. The user then shifts out the challenge and reads it serially from the time delay observer port. The user passes this data to an authentication server along with his user ID and password. The authentication server sends back corresponding response and is able to access the internal assets.

  An unauthorized access to the JTAG ports must be denied in order to prevent any form of attacks. Chiu et al. [70] presented a similar approach targeting IEEE 1500 standard. IEEE 1500 is a standard architecture for enabling test reuse and integration for embedded cores and associated circuitry. This secure test wrapper always stays in lock mode to prohibit access to internal scan chains and primary inputs and outputs. Secure test wrapper is only unlocked when a secure test wrapper key is applied. Das et al. [71] proposed challenge–response-based test protocol using KATAN lightweight block cipher [72].

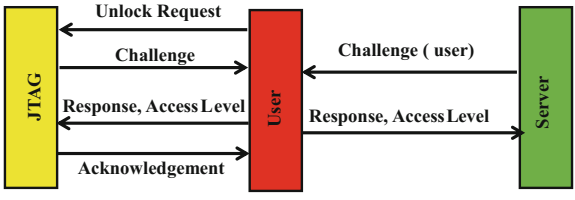


Fig. 6.9 Secure authentication module protocol [69]

An early solution to prevent attacks through JTAG is to disable JTAG access ports with anti-fuse [36, 73] with the cost of in-field debugging/updates. Controlling access to JTAG port by using secret keys is a solution to prevent attacks through JTAG [74]. In this method, a valid key allows to gain access to the CUT. An invalid/wrong key bypasses the inputs (to *TDI*) to the output pins (to *TDO*) (See Fig. 6.4). Key management and delay introduced by encryption/decryption are open issues of this technique's applicability in real-life JTAG (for debugging). However, a reliable physical unclonable function is able to solve this unique key management problem. Buskey et al. [75] proposed a three-entity protocol to protect the JTAG. In this method, an independent authentication server is used to manage authentication keys of each individual authenticated user/person. The server uses asymmetric encryption for secure communication. The advantage of this technique with the previous two-entity protocol (i.e., [74]) is that the key is unknown to the users/attackers. However, the high level of security provided by the three-entity protocol is achieved by the cost of area and time overhead compared to previous two-entity protocol. Park et al. [37] proposed a superior technique of [75].

On the other hand, the Segment Insertion Bit (SIB) in the IEEE P1687 standard is used to add or subtract scan path segments based upon the data scanned through the scan chain on a DR scan. In the method proposed by Dworak et al. [39], information related to key bits, LSIBs (locking SIBs), P1687 network structures, and secure instrument vectors are encapsulated in a license file. A secure communication is performed in order to make sure that there is no unauthorized access. In this method, each licensed copy of the software works only with a specific set of selected chip IDs (from physical instance). They also proposed a method to ensure communication between the chips on the board and the test access software tool. In this method, the software requests an encrypted chip ID from the board (Fig. 6.10a). The software also sends a random key (s-key) with the request. The board XORs s-key, real-time clock value, and chip ID and forms the encrypted chip ID (see Fig. 6.10b) and sends them to the software tool. The software allows access to the LSIB(s) if and only if the returned ID matches a permitted ID in the license file.

- **Obfuscation to prevention of IC piracy:** Secure split test (SST) [12, 13, 15] is a technique which prevents IC piracy. SST gives control over testing back to the IP owner. In SST, each chip and its scan chains are locked during test and only the IP owner can interpret the locked test results and unlock passing chips. The IP owner is the only entity who can interpret the locked test results and unlock passing chips. In this way, SST can prevent overproduced, defective, and cloned chips from reaching the supply chain. The quality of security depends on the quality of scan-locking mechanism. The SST uses scan chain scrambling technique to perturb the test responses.

  Scan-locking block is used to perturb the test response so that an outsider cannot determine the true test response from even an unlocked IC. Figure 6.11 shows the scan-locking block and its mechanism. The yellow blocks represent the scan chain of the design and the test structure commonly used in practice while the rest are required to implement SST. Test data compression is required to overcome the limitations of the automatic test equipment (ATE). The outputs of the

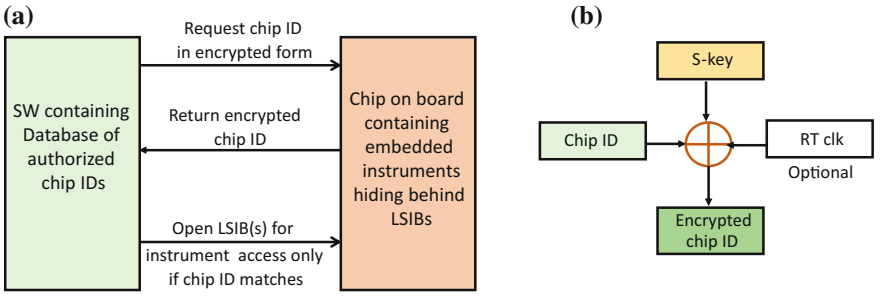


Fig. 6.10 **a** Authorized access to the chip/board under test and **b** Encrypting chip ID with data received from software tool (and real-time clock) [39]

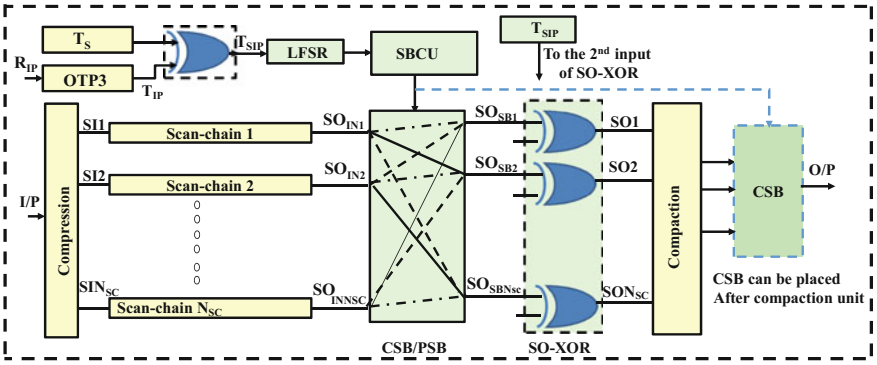


Fig. 6.11 Scan-locking block for preventing leakage information of assets

scan chains are scrambled through a scrambling block (SB) in order to perturb the functionally locked/unlocked response. The output of scrambling block (SB) is sent through SO-XOR blocks for further perturbation. Scrambling blocks are an essential component in telecommunication and microprocessor communication [76]. The scrambling block can be completely shuffled or partially shuffled. Complete scrambling block, CSB, is designed such that all inputs to the block can potentially go to any output pin received by a compaction circuit. On the other hand, a partial scrambling block (PSB) is designed in such a way that an input to the scrambling block is connected with only N_{SB} different outputs. Non-blocking crossbar switch [76] is a strong candidate for the scrambling block and can be designed with pass transistors or transmission gates.

The security strength depends on the type of shuffling block. A CSB will provide maximum security but higher cost. N_{SB} can be tuned with the consideration of area overhead and desired security strength. The scrambling block's controlling unit (SBCU) assures that all inputs to the scrambling block, either CSB or PSB, are seen at the output. The logic of SBCU depends on the PSB or CSB structure and number of scan chains (N_{SC}). The output of LFSR, which controls the SBCU, changes in

each clock cycle and depends on initial seed, $T_{SIP} = T_S \oplus T_{IP}$, which is known by the IP owner only. T_{IP} is the value stored in OTP3. In foundry, T_{IP} is set to all 0s or all 1s. But in assembly, the IP owner sends a random number, R_{IP}, independent to IC and TRN values. The initial seed of LFSR, T_{SIP}, is different for assembly and foundry for the same IC; hence, SBCU performs different scrambling (see more in Sect. 6.3). The output of CSB/PSB is sent through an SO-XOR block to add another layer of security. The scrambling block has to be ready before intermediate output of scan chain, SO_{IN}, is ready. In order to avoid timing failure, LFSR can be activated in the negative clock edge so that scrambling block is ready and SO_{IN} can pass through it.

The SO-XOR block is controlled by T_{SIP}. Depending on the second input of XOR in SO-XOR block, the output of scrambling block flips or goes transparent and is known by the IP owner only (as T_S is revealed to the IP owner). The scrambling block and SO-XOR block make it impossible for untrusted fab and untrusted assembly to determine the correct output responses.

The number of scannable flip-flops have been increasing significantly with the size of designs. Traditional full-scan test method, occupying between 5 and 20% of silicon area, is expensive in terms of test time and test volume because of long scan chains. Designs require on-chip test hardware to compress the time and memory of ATPG. The size of scan chains increases with gate counts and flip-flops, but ATE has only limited number of channels. Compaction circuitry is used to support the ATE. MISR is commonly used to compact the scan chain responses. CSB requires large area to support all scan chains in order to get high-quality perturbed response. The cost reduces with the size of CSB; for example, a 10×10 complete shuffle crossbar switch requires 100 transmission gates whereas a 4×4 requires 16. The CSB provides the best security level but requires large area. In order to reduce the cost, CSB can be placed after the compactor (we name it alternative place). An $r : 1$ compactor reduces r^2 times area for scrambling block.

6.4 Conclusion

In this chapter, we discussed the importance of cryptodevices in modern semiconductor industry because many IPs need to make available in public. We also reviewed the scan chain security to protect the assets of an IP/design. We focused the importance of scan chain security to prevent digital right management. We reviewed existing scan-based attack models and existing countermeasures. We also presented the importance of IC piracy and its prevention. We also discussed how obfuscation can protect an asset from scan-based attack. We also presented a scan chain locking mechanism and key exchange mechanism to prevent IC piracy.

References

1. Ray S, Jin Y, Raychowdhury A (2016) The changing computing paradigm with internet of things: a tutorial introduction. IEEE Des Test Comput 33(2):76–96
2. McGill K (2013) Trusted mobile devices: requirements for a mobile trusted platform module. http://www.jhuapl.edu/techdigest/TD/td3202/32_02-McGill.pdf
3. National Institute of Standards and Technology, FIPS-46-3: Data Encryption Standard (DES). October 1977, reaffirmed in October 1999
4. National Institute of Standards and Technology, FIPS197: Specification for the Advanced Encryption Standard (AES), 2001
5. Vernam GS (1926) Cipher printing telegraph systems for secret wire and radio telegraphic communications. J Am Inst Electr Eng 55:109–115
6. Wolf M, Weimerskirch A, Wollinger T (2007) State of the art: embedding security in vehicles. EURASIP J Embed Syst 2007:074706
7. Nara R, Togawa N, Yanagisawa M, Ohtsuki T (2010) Scan-based attack against elliptic curve cryptosystems. In: 15th IEEE Asia and South Pacific design automation conference, pp 407–412
8. Tehranipoor M, Wang C (2011) Introduction to hardware security and trust. Springer, New York
9. Ali SS, Sinanoglu O (2015) Scan attack on elliptic curve cryptosystem. In: IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFTS), IEEE
10. Paul S, Chakraborty R, Bhunia S (2007) VIm-scan: a low overhead scan design approach for protection of secret key in scan-based secure chips. In: 25th IEEE VLSI test symposium, pp 455–460
11. Guin U, Tehranipoor M, DiMase D, Megrdichian M (2013) Counterfeit IC detection and challenges ahead. ACM SIGDA Newsl 43(3):1–5
12. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: 2013 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT), IEEE
13. Rahman MT, Forte D, Shi Q, Contreras GK, Tehranipoor M (2014) CSST: an efficient secure split-test for preventing IC piracy. In: 2014 IEEE 23rd North Atlantic test workshop (NATW), IEEE
14. Maes R et al (2009) Analysis and design of active IC metering schemes. In: IEEE international workshop on hardware-oriented security and trust, 2009. HOST'09, IEEE
15. Rahman MT, Forte D, Shi Q, Contreras GK, Tehranipoor M (2014) CSST: preventing distribution of unlicensed and rejected ICs by untrusted foundry and assembly. In: IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems
16. Lee J, Tehranipoor M, Patel C, Plusquellic J (2007) Securing designs against scan-based side-channel attacks. IEEE Trans. Dependable Secure Comput 4(4):325–336
17. Rolt D et al (2014) Test versus security: past and present. IEEE Trans Emerg Top Comput 2(1):50–62
18. Rolt J et al (2013) A novel differential scan attack on advanced DFT structures. ACM Trans Des Autom Electron Syst 18(4):58
19. Choosing the Right Scan Compression Architecture for Your Design. http://www.cadence.com/rl/Resources/white_papers/Test_Compression_wp.pdf
20. ARM inc. Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
21. Peeters E (2015) SoC Security Architecture: Current Practices and Emerging Needs, DAC
22. Rahman MT, Xiao K, Forte D, Zhang X, Shi J, Tehranipoor M (2014) Ti-TRNG: technology independent true random number generator. In: Proceedings of the 51st annual design automation conference, ACM

23. Banik S, Chowdhury A (2013) Improved scan-chain based attacks and related countermeasures. In: Paul G, Vaudenay S (eds) Proceedings of the 14th international conference on progress in cryptology, INDOCRYPT 2013 -, vol 8250., Vol 8250Springer, New York, pp 78–97
24. Tehranipoor M, Lee J (2012) Protecting IPs against scan-based side-channel attacks. Introduction to hardware security and trust. Springer, New York, pp 411–427
25. Hly D, Kurt R, Karri R (2011) Security challenges during VLSI test. In: Proceedings of the 9th IEEE NEWCAS conference
26. Goering R (2004) Scan Design Called Portal for Hackers. http://www.eetimes.com/news/design/-showArticle.jhtml/articleID=51200154
27. Scheiber S (2005) The Best-Laid Boards. http://www.reedelectronics.com/tmworld/article-/CA513261.html
28. Yang B, Wu K, Karri R (2004) Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: Proceedings of the IEEE international test conference, pp 339–344
29. Yang B, Wu K, Karri R (2005) Secure scan: a design-for-test architecture for crypto chips. In: Proceedings of the 42nd annual conference on design automation, pp 135–140
30. Nara R, Satoh K, Yanagisawa M, Ohtsuki T, Togawa N (2010) Scan-based side-channel attack against RSA cryptosystems using scan signatures. IEICE Trans Fundam Electron Commun Comput Sci E93–A(12):2481–2489
31. Liu Y, Wu K, Karri R (2011) Scan-based attacks on linear feedback shift register based stream ciphers. ACM Trans Des Autom Electron Syst (TODAES) 16(2):1–15
32. Da Rolt J, Das A, Di Natale G, Flottes M-L, Rouzeyre B, Verbauwhede I (2012) A new scan attack on RSA in presence of industrial countermeasures. In: Schindler W, Huss SA (eds) Proceedings of the third international conference on constructive side-channel analysis and secure design (COSADE'12). Springer, Heidelberg, pp 89–104
33. Rolt J et al (2012) Are advanced DfT structures sufficient for preventing scan-attacks. IEEE 30th VLSI test symposium (VTS). Hyatt Maui, HI, pp 246–251
34. Rolt J et al (2011) New security threats against chips containing scan chain structures. 2011 IEEE international symposium on hardware-oriented security and trust (HOST). San Diego, CA, pp 110–110
35. Liu C, Huang Y (2007) Effects of embedded decompression and compaction architectures on side-channel attack resistance. In: 25th IEEE VLSI test symposium (VTS'07). Berkeley, CA, pp 461–468
36. Rosenfeld K, Karri R (2010) Attacks and defenses for JTAG. IEEE Des Test Comput 27(1):36–47
37. Park K, Yoo SG, Kim T, Kim J (2010) JTAG security system based on credentials. J Electron Test 26(5):549–557
38. Dworak J, Crouch AL (2015) A call to action: securing IEEE 1687 and the need for an IEEE test security standard. In: IEEE 33rd VLSI test symposium (VTS). pp 1–4
39. Dworak J, Conroy Z, Crouch A, Potter J (2014) Board security enhancement using new locking SIB-based architectures. In: International test conference
40. Chakraborty R, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(10):1493–1502
41. Chakraborty R, Bhunia S (2010) RTL hardware IP protection using key-based control and data flow obfuscation. In: 23rd international conference on VLSI design, 2010. VLSID'10, IEEE
42. Jeyavijayan R et al (2013) Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security, ACM
43. Roy A, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In: Proceedings of the conference on design, automation and test in Europe, ACM
44. Clark A (2009) Preventing integrated circuit piracy using reconfigurable logic barriers
45. Rahman MT, Forte D, Fahrny J, Tehranipoor M (2014) ARO-PUF: an aging-resistant ring oscillator PUF design. In: Proceedings of the conference on design, automation and test in Europe, European Design and Automation Association

46. Xiao K, Rahman MT, Forte D, Huang Y, Su M, Tehranipoor M (2014) Bit selection algorithm suitable for high-volume production of SRAM-PUF. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2014, IEEE
47. Hosey A, Rahman MT, Xiao K, Forte D, Tehranipoor M (2014) Advanced analysis of cell stability for reliable SRAM PUFs. In: IEEE 23rd Asian test symposium (ATS). IEEE
48. Mazady A, Rahman MT, Forte D, Anwar M (2015) Memristor PUF: a security primitive: theory and experiment. IEEE J Emerg Sel Top Circuits Syst 5(2):222–229
49. Rahman MT, Rahman F, Forte D, Tehranipoor M, An aging-resistant RO-PUF for reliable key generation. IEEE Trans Emerging Topics Comput, PP(99):1-1. doi:10.1109/TETC.2015.2474741
50. Rahman MT, Forte D, Rahman F, Tehranipoor M (2015) A pair selection algorithm for robust RO-PUF against environmental variations and aging. In: 33rd IEEE international conference on computer design (ICCD), IEEE
51. Maes R, Schellekens D, Tuyls P, Verbauwhede I (2009) Analysis and design of active IC metering schemes. In: IEEE international workshop on hardware-oriented security and trust, (2009) HOST '09. Francisco, CA, pp 74–81
52. Ratanpal GB, Williams RD, Blalock TN (2004) An on-chip signal suppression countermeasure to power analysis attacks. IEEE Trans Dependable Secure Comput 1(3):179–188
53. Kocher, PC (1996) Timing attacks on implementations of diffieHellman, RSA, DSS and other systems. In: Proceedings on 16th annual international cryptology conference on advances in cryptology, pp 104–113
54. Karri R, Wu K, Mishra P (2001) Fault-based side-channel cryptanalysis tolerant architecture for Rijndael symmetric block cipher. In: Proceedings of the IEEE international symposium on defect and fault tolerance in VLSI systems, pp 427–435
55. Karri R, Wu K, Mishra P, Kim Y (2002) Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans Comput-Aided Des Integr Circuits Syst 21(12):1509–1517
56. Skorobogatov SP (2005) Semi-invasive attacks: a new approach to hardware security analysis. Ph.D. Dissertation, University of Cambridge
57. Kommerling O, Kuhn MG (1999) Design principles for tamper resistant smartcard processors. In: Proceedings of the USENIX workshop on smartcard technology. pp 9–20
58. Kmmerling O, Kuhn MG (1999) Design principles for tamper resistant smartcard processors. In: Proceedings of the USENIX workshop on smartcard technology
59. Ahmed N, Tehranipour MH, Nourani M (2004) Low power pattern generation for BIST architecture. In: Proceedings of the 2004 international symposium on circuits and systems, 2004. ISCAS '04, vol 2. pp II-689-92
60. Hafner K, Ritter HC, Schwair TM, Wallstab S, Deppermann M, Gessner J, Koesters S, Moeller W-D, Sandweg G (1991) Design and test of an integrated cryptochip. IEEE Des Test Comput 8(4):6–17
61. Zimmermann R, Curiger A, Bonnenberg H, Kaeslin H, Felber N, Fichtner W (1994) A 177 Mbit/s VLSI implementation of the international data encryption algorithm. IEEE J. Solid-State Circuits 29(3):303–307
62. Hely D, Flottes M-L, Bancel F, Rouzeyre B, Berard N, Renovell M (2004) Scan design and secure chip. In: Proceedings of the 10th IEEE international on-line testing symposium, pp 219–224
63. Sengar G, Mukhopadhyay D, Chowdhury DR (2007) Secured flipped scan-chain model for crypto-architecture. IEEE Trans CAD 26(11):2080–2084
64. Standard IEEE, 1149.1-2001, (2001) Standard Test Access Port and Boundary-Scan Architecture. Technical report, IEEE Standards Board
65. Inoue M, Yoneda T, Hasegawa M, Fujiwara H (2009) Partial scan approach for secret information protection. In: 14th IEEE European test symposium. Seville, pp 143–148
66. Ege B, Das A, Batina L, Verbauwhede I (2013) Security of countermeasures against state-of-the-art differential scan attacks. In: TRUDEVICE. Radboud University Nijmegen, Nijmegen

67. Mentor Graphics ST, Whitepaper YA (2010) High quality test solutions for secure applications. Wilsonville, OR, USA. Mentor Graph. Corp., Apr. 2010
68. Rolt J et al (2012) A new scan attack on RSA in presence of industrial countermeasures, COSADE 2012. Lect Notes Comput Sci 7275:89–104
69. Pierce L, Tragoudas S (2013) Enhanced secure architecture for joint action test group systems. IEEE Trans Very Large Scale Integr (VLSI) Syst 21(7):1342–1345
70. Chiu GM, Li JCM (2012) A secure test wrapper design against internal and boundary scan attacks for embedded cores. IEEE Trans Very Large Scale Integr (VLSI) Syst 20(1):126–134
71. Das A, Knezevic M, Seys S, Verbauwhede I (2011) Challenge-response based secure test wrapper for testing cryptographic circuits. In: IEEE European test symposium, ETS
72. Canniere C, Dunkelman O, Kneevi M (2009) KATAN & KTANTAN - A family of small and efficient hardware-oriented block ciphers, CHES
73. Hely D et al (2007) Securing scan control in crypto chips. J Electron Test 23(5):457–464
74. Novak F, Biasizzo A (2006) Security extension for IEEE std 1149.1. J Electron Test Theory Appl 22(3):301–303
75. Buskey RF, Frosik BB (2006) Protected JTAG. 2006 international conference on parallel processing workshops (ICPPW'06). Columbus, OH, pp 406–414
76. Tamir Y, Chi HC (1993) Symmetric crossbar arbiters for VLSI communication switches. IEEE Trans Parallel Distrib Syst 4(1):13–27

Part III
Finite State Machine (FSM) Based Hardware Obfuscation

Chapter 7
Active Hardware Metering by Finite State Machine Obfuscation

Farinaz Koushanfar

7.1 Introduction

The escalating cost of updating and maintaining silicon foundries has caused a major paradigm shift in the semiconductor business model. Many of the key design houses are entirely fabless (i.e., without a fabrication plant), outsourcing their fabrication to third-party providers. Several design companies that have traditionally fabricated their designs in house either have formed alliances to share the cost, or have moved parts of their fabrication offshore to third-party providers. In the older vertical market model, in-house fabrication together with the clandestine nature of the packaged chips was enough for protection of design IPs. In the present business model, however, fabrication outsourcing requires revealing the design intellectual property (IP) to external entities, creating many opportunities for IP infringements. The problem is exacerbated by contracting the offshore foundries to lower the labor and manufacturing cost, since many such fabrication plants are in countries with malpractice of IP enforcement laws [1]. The Alliance for Gray Market and Counterfeit Abatement has estimated that about 10% of the leading edge technology products available on the market are counterfeits [2]. The problem arises since the mass production of integrated circuits (ICs) m a common mask prohibits individual marking and tracking of each siliopiece. Several government and industry task force with members from the leading semiconductor companies are actively working to address the important problem of counterfeiting [3–5].

To enable tracking of the designer IPs and ICs, a suit of new security mechanisms and protocols called *hardware metering* was introduced [6–11]. Metering enables the designers to have post-fabrication control over their designed IPs by passively or actively monitoring or counting the number of produced ICs, by monitoring their properties and usages, and by remote runtime enabling/disabling. The term hardware

F. Koushanfar (✉)
University of California, San Diego, CA, USA
e-mail: farinaz@ucsd.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_7

metering was originally coined by Koushanfar, Qu, and Potkonjak to refer to methods for unique functional identification of ICs made by the same mask [6, 7]. Their metering methods were the first that could be used for specific functional tagging of ICs, or for monitoring and estimating the number of fake components in case of piracy detection. Methods for unique identification of chips based on process variations were suggested in 2000 [12], but hardware metering was the first to integrate the unique chip identifiers into the IC's functionality. Note that hardware metering is fundamentally different from IP fingerprinting, watermarking, and foundry identification [13, 14]. IP watermarking adds the same sign to all instances of a design [15–19], while fingerprinting gives a unique signature to each IP (typically on a reconfigurable hardware) [20, 21]. Foundry identification addresses the problem of finding the original fabrication facility of an IC by evaluating certain chip characteristics [22].

Passive methods for metering by post-silicon processing or adding programmable parts were proposed as early as 2001, but the first set of active methods for metering or tracking ICs was introduced in 2007 [9]. Several subsequent metering methods have been suggested since [10, 23–29]. For a comprehensive review and novel classification of metering, we refer the interested readers to survey on the topic [30–32]. What is most exciting about metering is that the mechanisms for integrating the unique IDs in functionality can be used to enable a host of other important hardware security applications [33–39], including but not limited to third-party IP and system protection [8, 40–46], obfuscation [47–49], FPGA security [45, 50, 51], anti-tampering/reverse engineering [52–54], as well as IoT security [55–58].

In hardware metering, each chip is uniquely and unclonably identified, for example, by using a physical unclonable function (PUF) module [59–61]. The PUF typically extracts the unique delay or current variations on each chip to assign a set of unclonable identifiers (IDs). To meter, the IDs are linked to parts of the IC's functional components, e.g., the combinational part or the sequential part of the computer circuitry. This way, a part of the design functionality is uniquely tailored to the unclonable properties (fingerprint) of the IC and is used to form a unique lock for each IC's functionality [9]. Only the designer who has the knowledge of the high-level design would be able to find the specific key to unlock each IC. Although we discuss a specific PUF in this chapter, it is important to note that one can build new metering and obfuscation systems by employing more novel and reliable PUF structures, e.g., [27, 60–77].

Security of the earlier active control methods is ensured by one of the two approaches: (1) expanding the finite state machine (FSM) of the functional specification such that the added states and transitions are hidden in high-level design and are only known to the designers [9, 23] or (2) employing a known cryptographic primitive such as public-key cryptography, secret sharing, or AES [24–26]. The surveys on active metering distinguish between the two methods [30, 31]. The first set of methods based on introducing locks embedded in the behavioral description of the design are called *internal active IC metering*. The second set of methods based on embedding locks within the design and interfacing (controlling) the access by an external cryptography function are termed *external active IC metering*.

This chapter presents a detail description and comprehensive security analysis of the FSM-based metering methods based upon the model and analysis suggested in [10]. The locking and controlling methods introduced here are internal and sequentially designed. Although combinational-only internal locks have been used in the context of external active hardware metering [24–26], those locks are interfaced with cryptographic hardware that is often sequentially implemented. Therefore, the control method as a whole is a sequential design. Furthermore, the power/area overhead of the cryptographic module interfaced with the combinational locks is greater than the overhead of the internally embedded control circuitry. We show that the construction of locks by finite state manipulation and compilation during the hardware synthesis and interfacing to a unique PUF state is an instance of an efficiently obfuscatable program under the random oracle model. The significance of this construction and security proofs for the obfuscated FSM goes beyond hardware metering and extends to the earlier work in information hiding and obfuscation of sequential circuits, e.g., [47, 48]. The highlights of this chapter are as follows.

- We comprehensively discuss the first known method for active IC metering and IC piracy prevention which allow uniquely locking each manufactured IC at the foundry. The locking structure is embedded during hardware synthesis by FSM modifications such that the IC would not be functional without a proper chip-specific passkey that can only be computed by the designer (IP rights owner).
- We show the analogy between the hardware synthesis transformations and program compilation. We pose the problem of extending the FSM for hiding the locks as an instance of the classic program obfuscation problem.
- We demonstrate a construction of the locks within FSM as an instance of a general output multi-point function family. This family is known to be effectively obfuscatable in the random oracle model. Therefore, the locks can be efficiently hidden.
- Automatic low overhead implementation of secure metering lock structure during synthesis is demonstrated by obfuscatable topology construction, secure passkey selection, and iterative synthesis.
- Potential attacks and security of the presented method against each attack are discussed.
- We show the low overhead and practicability of the suggested metering technique.

The remainder of this chapter is organized in the following way. The next section introduces the global flow. Background and assumptions are outlined in Sect. 7.3. Details of the active hardware metering method are presented in Sect. 7.4, where secure hiding of the locks within the FSM structure is discussed. We also briefly mention a number of potential applications of active metering. Section 7.5 discusses the details of the automatic synthesis and implementation. Attacks and countermeasures are discussed in Sect. 7.6. Overhead of this method is discussed in Sect. 7.7. We conclude in Sect. 7.8.

7.2 Flow

Figure 7.1 shows the global flow of the first known active hardware metering approach as described in [9]. Similar flows were later adopted for both internal and external active integrated circuits metering, e.g., [25, 26]. There are typically two main entities involved: (i) a design house (a.k.a., *designer*) that holds the IP rights for the manufactured ICs and (ii) a foundry (a.k.a., *fab*) that manufactures the designed ICs.

The steps of the flow are as follows. The designer uses the high-level design description to form the design's behavioral model in the FSM format. Next, the FSM is modified so that the extra locking structure composed of additional states and transitions is integrated within the FSM. The term boosted finite state machine (BFSM) is used to refer to the modified FSM. The subsequent design phases (e.g., RTL, synthesis, mapping, layout, and pin placement) take their routine courses. The foundry would receive the OASIS files (or GDS-II) and other required information for fabricating the chips, and also the test vectors. The test vectors include the challenge set (input) to be applied to the PUF unit on each IC [61]. The design house typically pays the foundry an upfront cost for a mask to be lithographed from the submitted OASIS files and for the required number of defect-free ICs to be fabricated.

Building a mask is a costly and complex process, involving multiple fine steps that should be closely controlled [78, 79]. Once the foundry lithographs a mask, multiple ICs could be fabricated from this mask. Because of the specific PUF responses integrated within the locks on the chips, each IC would be uniquely locked (nonfunctional) upon fabrication. During a start-up test phase, the fab inputs the challenge vectors to the chips that would run through the scan chains. The states stored in the chip FFs (a.k.a., *power-up state* or the *start-up state*) would be scanned at this point. The scanned FF values are sent back to the design house (IP rights owner) who has the full specifications of the hidden states. The design house is the only entity who could compute the unlocking sequence for each locked chip. This article introduces provably secure BFSM construction methods, so the transitions from each power-up state to the original reset state can be efficiently hidden. Additionally, the designer often computes the error correcting code (ECC) to adjust for any further changes to the start-up state because of the noise or other sources of physical uncertainty. The

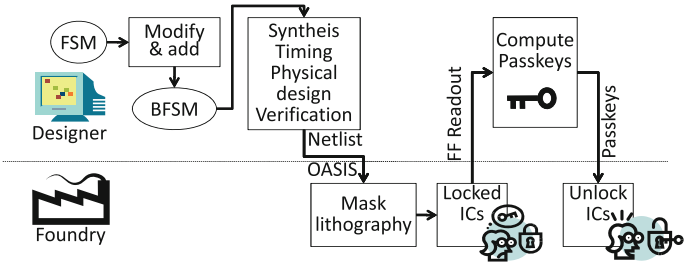


Fig. 7.1 The global flow of the suggested IC activation approach [9]

ECC is very important since a few of PUF response bits may be unstable and alter at a later time because of noise, environmental conditions (e.g., temperature), or circuit instability [80]. The key and the ECC would then be sent back to the fab.

The nonvolatile on-chip memories would be used to store the power-up test vectors (PUF challenges), the unlocking sequence (passkeys), and the relevant ECC bits on each pertinent activated IC. From this point on, every time the IC starts up, it would automatically read the passkey and error correcting codes and use them for traversing the chip to a working state. As it can be easily seen, active metering is nicely integrated within the regular phases of the hardware design, synthesis, and tape-out flow. The only added phase is the key exchange protocol for unique activation of each IC. The common functional and structural tests would be done on the unlocked ICs. We note that if a defect or a fault affects the states and sequences traversed during the start-up state or the traversal from the locked start-up state to the functional state, the IC cannot be unlocked and would be defective.

7.3 Background and Assumptions

We introduce a number of general terms and concepts that are used throughout the chapter. More specific definitions would be described as necessary.

Design description We consider the case where the sequential design represents a fully synchronous flow. The description of the design input/output functionality is publicly available. We assume that the functional description is fully fixed, and the I/O behavior is fully specified. Our metering technique is applicable when the IP is available in structural HDL description, or in form of a netlist that may or may not be technology dependent. Thus, it can protect both firmware and hardware [15]. During the IC design flow, the designer maps the circuit behavioral description to a specific technology provided by the target foundry. Several logic-level optimizations (including timing closure, power optimizations, and synthesis transformations) are applied by the designer. Very often, a designed IP is integrated within a larger circuit or a system-on-a-chip.

Finite State Machine Digital sequential circuits are commonly modeled by a finite state machine (FSM). An FSM is an abstract machine that can be in one of a finite number of states at a certain time instant. Transition between states can be triggered by an input. At each transition, the next state depends on the inputs and the current state. Two types of FSMs are distinguished in digital circuits: a Moore machine where the outputs are function of only the current state and a Mealy machine where the outputs are function of both the current state and the input.

A deterministic FSM is often formally defined by a sextuple, $FSM = (\Sigma, \Delta, S, s_0, \delta, \lambda)$, where

- $\Sigma \neq \Phi$ is a finite set of input symbols;
- $\Delta \neq \Phi$ is a finite set of output symbols;
- $S=\{s_0, s_1, \ldots\} \neq \Phi$ denotes a finite set of states;

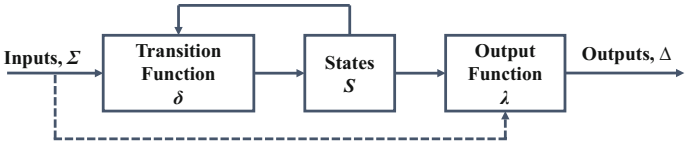


Fig. 7.2 Finite state machine

- s_0 is the FSM "reset" state;
- $\delta\ (s, i)$ is transition function on s and i $(S \times \Sigma \rightarrow S)$; and
- $\lambda\ (s, i)$ is output function on s and i $(S \rightarrow \Delta$ for Moore, $S \times \Sigma \rightarrow \Delta$ for Mealy).

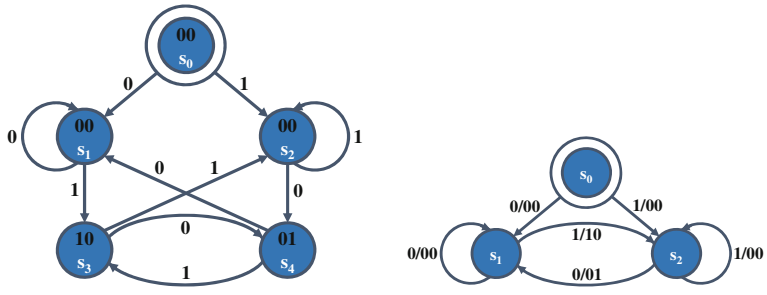
In δ and λ definitions, s is the state and i is the input.

Figure 7.2 illustrates the structure of an FSM. The transition function δ and the output function λ are designed with combinational logic. The dotted connection from the inputs to the output function is present only in the Mealy machine.

To represent the transitions and output functions of the FSM, we use the state transition graph (STG) with nodes corresponding to states and edges defining the transition conditions based on the current state and the edge inputs. Throughout the chapter, we use the terms STG and FSM interchangeably.

We demonstrate FSM with two simple examples. Figure 7.3 shows the STGs for an edge detector circuit that detects the flip of the input bit b. Input $\Sigma = \{b\}$ and output $\Delta = \{pos\_edge,\ neg\_egde\}$. The "reset" state is marked with double circle. In the Moore state machine in Fig. 7.3a, the output is displayed inside the circles depicting states since it is function only of the current state, while in the Mealy state machine in Fig. 7.3b, the output is displayed along the edges since, in this case, it depends on both the current state and the inputs.

The FSM is commonly used for realizing the control path of a circuit. The STG for a memory control unit placed between the processor and the memory is shown in Fig. 7.4. The input from the processor $\Sigma = \{access,\ r/w,\ burst\}$ and the output to the memory $\Delta = \{rd\_en, wr\_en\}$. The system powers up in the "reset" state and stays in that state as long as $access = 0$. Both rd_en and wr_en is set to 0 at this state.



(a) Moore machine (b) Mealy machine

Fig. 7.3 STG for edge detection

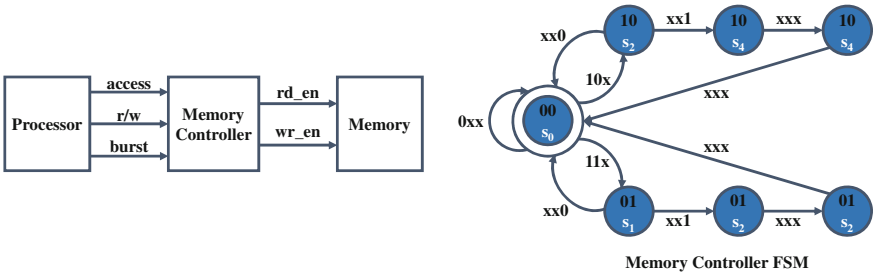


Fig. 7.4 STG for memory controller

If $access$ is set to 1, the FSM asserts either rd_en if $r/w = 1$ or wr_en if $r/w = 0$. If $burst = 0$, the system performs only one read/write operation and returns to the idle state. If $burst = 1$, the selected operation is performed three times before returning to the idle state. Only the Moore style implementation is presented here for space consideration.

Assumptions The hardware metering objective is to protect the ICs so they cannot be pirated or overbuilt by the foundry. There are a number of realistic assumptions that we make about the potential adversary (foundry). First, the foundry has access to layout (e.g., OASIS or GDS-II files) and the netlist but does not have access to FSM because: (i) the layout files, the netlist, and the test vectors are sufficient for fabricating the chip and a high-level behavioral description is not needed, and (ii) the details of the FSM behavioral description are a key part of the designer's IP and trade secret that maintains their revenue in the competitive semiconductor market. Second, a significant design modification would impact power, yield, and most importantly timing. In such scenarios, redoing timing, physical design, verification, and debugging would require an effort equivalent to designing a new IC and a new mask. Therefore, the attack cost and complexity do not justify its benefits. Third, the scan chains are available and it is possible to scan and read out the FF values storing the FSM states on each chip. Fourth, the designer's objective is to protect her/his design from piracy and other related tampering. The designer's objective is not to protect the overall functionality. For example, while designing a H.264 media player, the different submodules are known at the protocol level and at the block level. However, the specific details of a design are to be protected by the design house (so it cannot be readily reproduced) for competitive advantages reasons.

7.4 Secure Active Hardware Metering Methodology

During the IC design flow, a designer devises the behavioral specification in a FSM format. At this stage, the FSM will be modified to include multiple added states and transitions. This modified FSM is called the boosted finite state machine (BFSM). The initial power-up state of the BFSM is determined by the PUF and is unique for each IC.

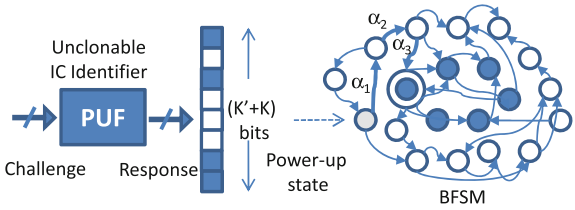


Fig. 7.5 The PUF response is fed to the FFs storing the states of the BSFM. The original states are shown in dark, and the added states are demonstrated in *white color* on the STG that represents the BFSM

7.4.1 Designing a BFSM

We first demonstrate the BFSM modification and active metering mechanisms using a small example. We then discuss parameter selection for ensuring the randomness and uniqueness of each activated IC. Comprehensive details of the parameter selection for a secure BFSM construction are discussed in later subsections.

On the small example in Fig. 7.5, the original FSM states are shown in dark color on the right side of the figure. The BFSM includes the original FSM, along with a number of added states that are shown in white. Assume that the original FSM has $|S|$ states. Therefore, it can be implemented using $K = log|S|$ FFs. Now assume that we add to the number of states in FSM to build a BFSM with $|S'| + |S|$ states that can be implemented by $K'' = log\{|S'| + |S|\}$ FFs. Observe that for a linear growth in the number of FFs denoted by $K' = K'' - K$, the number of states exponentially increases.

On the left side of Fig. 7.5, there is a PUF unit that generates random bits based on the unclonable process variations of the silicon unique to each chip. A fixed challenge is applied to the chip upon power-up. The PUF response is fed to the FFs that implement the BFSM. Since there are $K'' = log\{|S'| + |S|\}$ FFs in the BFSM, one would need K'' response bits from the PUF for a proper operation.

As shown in Fig. 7.5, upon the IC's power-up, the initial values of the design's FFs (i.e., *power-up state*) are determined by the unique response from the PUF on each chip. The PUF challenges are determined by fixed test vectors given by the designer. For a secure PUF design, the probability of the response should be uniformly distributed over the possible range of values [62]. The number of added FFs can be set such that the value $2^{K''} >> 2^K$. In other words, the value K'' is set by the designer such that for a uniform probability of selecting the state, the probability of selecting a state in the original FSM is extremely low. We will leverage more on this point in the next subsection.

Because there are exponentially large number of added states, there is a high probability that the unique PUF response on each chip sets the initial power-up state to one of the added states. Note that unless the design is in one of the original states, it would be nonfunctional. Therefore, the random FF states driven by the PUF response would place the design in a nonfunctional state. One would need to provide inputs to the FSM, so it can transition from this nonfunctional initial power-up state to the functional *reset state* of the original FSM shown by double circle on the example.

For the IP rights owners with access to the BFSM state transition graph, finding the set of inputs for traversing from the initial power-up state to the reset state (shown by double circle on the figure) is easy. All what is needed is to form a path on the graph and use the input values corresponding to the path transitions (from the STG description), so the chip transitions to the reset state. However, there is only one combination from exponentially large number of possibilities for the input corresponding to each edge transition. Thus, it would be extremely hard for anybody without access to the BFSM edge transition keys to find the exact inputs that cause traversal to the original reset states. The access to the full BFSM structure and the transition function on its edges is what defines the designer's secret. The passkey for unlocking the chip is the sequence of inputs that can traverse the BFSM states (describing the control component of the chip) from the initial random power-up state to the reset state. Note that although the initial power-up state is random, the assumption is that for a given PUF input (challenge), the response remains constant over time for one chip.

A set of passkeys $(\alpha_1, \alpha_2, \alpha_3)$ required for traversal from the power-up state to the reset state is shown on Fig. 7.5. This locking and unlocking mechanism provides a way for the designer to *actively control (meter)* the number of unlocked functional *(activated)* ICs from one blueprint (mask), and hence the name active hardware metering. In Sect. 7.4.3, we will describe a number of other important applications of this active control method.

While constructing the BFSM for hardware metering purposes, a number of requirements must be satisfied. The first set of requirements has to do with the probability of randomly powering up in a state that was not in the original FSM. Let us assume that by design, we require this probability to be lower than a given value ε. This low probability is satisfied by the following two conditions:
(i) The value $|S'|$ should be selected such that the probability of not powering up in an added state is smaller than ε:

$$P(\textit{power-up} \in S') = \frac{S' - S}{S' + S} \geq 1 - \varepsilon. \tag{7.1}$$

(ii) The value $|S'|$ should be selected so that the probability of two ICs having the same start-up states is extremely low. Assume that we need to have m distinct ICs each with a unique start-up state. Fortunately, for a linear increase in the number of FFs, we obtain an exponential increase in the number of states. The unclonable response from the PUF is used to set each IC in a unique random state. To achieve completely random and independent states, one can employ the birthday paradox to calculate this probability and to set it to low values. Consider the probability $P_{collision}(S', m)$ that no two ICs out of a group of m will have matching start-ups out of $|S'|$ equally possible states. Assume $S' >> S$. Start with an arbitrary chip's start-up. The probability that the second chip's start-up is different is $\frac{2^{K''}-1}{2^{K''}}$. Similarly, the probability that the third IC's start-up is different from the first two is $\frac{2^{K''}-1}{2^{K''}} \cdot \frac{2^{K''}-2}{2^{K''}}$. The same computation can be extended through the $2^{K''}$-th start-up. More formally,

$$P_{collision}(K'', m) = \frac{2^{K''}-1}{2^{K''}} \cdot \frac{2^{K''}-2}{2^{K''}} \cdots \frac{2^{K''}-(m-1)}{2^{K''}}$$
$$= \frac{2^{K''}!}{(2^{K''}-m)!2^{m.K''}}. \quad (7.2)$$

For a large value of $2^{|S'|}$, the formula can be asymptotically approximated [81]

$$m = \sqrt{2^{|S'|+1} \times ln\left(\frac{1}{1-P_{collision}(|S'|, m)}\right)} \quad (7.3)$$

This equation yields the approximate closed formula:

$$P_{collision}(|S'|, m) = 1 - e^{-\frac{m^2}{2^{|S'|+1}}}. \quad (7.4)$$

Another set of important requirements has to do with the BFSM edge traversal and state reachability. For an STG with the initial reset state (denoted by s_0), we call this graph *reset state reachable* if and only if for each and every state in STG there is at least one sequence of inputs $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ of arbitrary cardinality such that it can be applied to the pertinent state, and the final transition destination (after applying the input sequence) is the reset state. From this definition, it is clear that the desired BFSM needs to be reset state reachable.

7.4.2 Secure BSFM Construction

In this subsection, we show that our construction of finite state manipulation, compilation by hardware synthesis, and interfacing to the unique IDs coming from the PUF comprise an instance of an efficiently obfuscatable program.

7.4.2.1 Secure Program Obfuscation

Let us go through the steps of hardware design and synthesis flow. The designer starts at a functional description level that can be demonstrated by a FSM, whose states and transitions are known. Given the FSM model, for each input, the designer would be able to compute the corresponding output. The specifications are typically implemented in a hardware description language (HDL) format, e.g., VHDL, or Verilog. The register-transfer level (RTL) description of this HDL format along with the technology libraries and design constraints is then input to the hardware synthesis tool. The output of the synthesis tool may be either in form of a netlist with boolean gate types, or might be mapped to a specific library. The remainder steps of the design process use the resultant netlist.

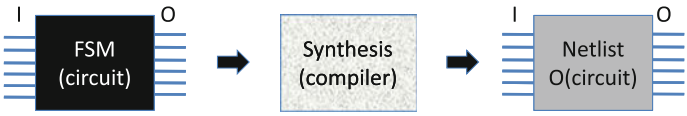


Fig. 7.6 Active hardware metering as a program obfuscation

Figure 7.6 demonstrates a black-box model of the metering synthesis flow from the high-level computational model (modeled by an FSM) to the lower level netlist. In effect, hardware synthesis is a compiler. Recall that a compiler provides a transformation from a source program written in a higher level computer language (source language) into another lower level computer language (target language). Here, the input source program is the description of the circuit behavior in finite state automata domain, and the target program is the circuit description in the netlist domain.

The FSM-based hardware metering attempts at hiding information in a source program by modifying the FSM. The modifications add multiple states to the STG, where the modified expanded state space is used for hiding the power-up state. The objective is to hide information such that the concealed data cannot be extracted from the target program presented in a lower level netlist. The two programs have (almost) the same functionality from the input/output program perspective. In effect, the objective of this type of active hardware metering is to build a *program obfuscation* that would conceal the secret passkey information (shown by α_is on Fig. 7.5), so the hidden state and its corresponding traversal cannot be retrieved from the netlist. Informally speaking, an obfuscator is a compiler that transforms a program (e.g., a Boolean circuit) into an obfuscated program (also a circuit), such that the obfuscated program has the same input/output relationships as the original program, but is otherwise obscure (unintelligible).

7.4.2.2 Secure Program Obfuscation

Let us discuss obfuscation in a more formal setting and outline the positive results on this subject. Under a random oracle model,[1] a function family $\mathscr{F}$ can be obfuscated when there is an algorithm $\mathscr{O}$ that takes an input in form of a Turing machine (i.e., a program, a circuit) computing $P \in \mathscr{F}$ and outputs another Turing machine (circuit) such that the *obfuscating requirements* are satisfied [82]. If the requirements are assured, $\mathscr{O}$ is an obfuscator for $\mathscr{F}$, and the obfuscation of the program is shown by $\mathscr{O}(P)$. $\mathscr{O}$ is called *efficient* if its computations are done in polynomial time, and then $\mathscr{O}(P)$ is said to be efficiently obfuscatable. Before we formally outline the requirements, let us define a number of notations. The notation k specifies the *feasibility parameter* that is associated with one family $\mathscr{F}_k$ of functions that we

[1] In cryptography, a random oracle is a mathematical abstraction used in proofs when no implementable function (except for an oracle) could provide the properties required.

obfuscate; the size of $P \in \mathscr{F}_k$ is polynomial in k. The family $\mathscr{F}$ is the union of the families $\mathscr{F}_k$.

1. *Approximate functionality.* There is a negligible function ν such that $\forall k$ and $\forall P \in \mathscr{F}_k$, we would have $\mathbf{Pr}[\exists$ inputs $x \in \{0,1\}^*$: $\mathscr{O}(P)(x) \neq P(x)] \leq \nu(k)$;
2. *Polynomial slowdown.* There is a polynomial p such that $\forall P \in \mathscr{F}$, $|\mathscr{O}(P)| \leq p(|P|)$ and for the Turing machine if P spends t time steps to compute on the input $x \in \{0,1\}^*$, $\mathscr{O}(P)$ would spend $p(t)$ time steps at most;
3. *Virtual black box.* For a probabilistic polynomial time Turing machine $\mathscr{A}$, there is another probabilistic polynomial time Turing machine $\mathscr{S}$ and a small negligible function denoted by ν such that $\forall P \in \mathscr{F}$ we have $|\mathbf{Pr}[\mathscr{A}(\mathscr{O}(P)) = 1]$ - $\mathbf{Pr}[\mathscr{S}^P(1^{|P|}) = 1]| \leq \nu(|M|)$, where the probabilities are over $\mathscr{A}$ and $\mathscr{S}$ randomness.

The work in [82] was the first to start a formal theoretical study of obfuscation. In particular, they showed a negative result for the possibility of having a generic obfuscator for all function classes by demonstrating a family of functions that could not be obfuscated. Thereafter, several theoretical studies and results for obfuscation have emerged [83, 84]. The work in [84] provided the first positive results for provable obfuscation of the family of point functions in the random oracle model.

The simplest example for a point function is a program that requires a password to login and operate. The password is typically hidden in the program and should be obfuscated such that nobody else with access to the program would be able to extract the password as long as it cannot be guessed. Password hiding can be modeled as an obfuscation of a point function under the random oracle model. More formally, a point function $\{\mathscr{P}_\alpha\}$ is defined as a function $\mathscr{P}_\alpha$ such that for all inputs x, the function $\mathscr{P}_\alpha = 1$ if the input is equal to the specific access key (password) α. The function would not do anything otherwise.

Reference [84] has demonstrated a point function $\mathscr{P}$ satisfies the three properties specified for an efficiently obfuscatable function. This result was further extended to the family of multi-point functions that have a general output. The function $\mathscr{P}_{(\alpha_1,\beta_1),\ldots,(\alpha_q,\beta_q)}$ on the domain $\{0,1\}^k \rightarrow (\{0,1\}^{\varsigma(k)})^q$ is a q-point function and has a general output of length $\varsigma(k)$ iff:

$$\mathscr{P}_{(\alpha_1,\beta_1),\ldots,(\alpha_q,\beta_q)} = \begin{cases} b \in (\{0,1\}^{\varsigma(k)})^q & \text{where } b_i = \beta_i \\ & \text{if and only if } x = \alpha_i; \\ \perp & \text{Otherwise.} \end{cases} \tag{7.5}$$

This latter proof is given by showing that the multi-point function can be self-decomposed to several smaller point functions. The generalized output multi-point function is a program with q passwords with a string output of length $\varsigma(k)$ (generalizing a single binary output case).

7.4.2.3 Provable Obfuscation of the Locks Within the BFSM

In this subsection, we show that a secure construction for the BFSM used in metering can be modeled as a multi-point function with a general output and thus can be efficiently obfuscated. The hardware metering approach introduced in [9] adds exponentially many states to the FSM such that the probability of powering up in one of the added states is extremely high. Because of the unique chip identifiers coming from the PUF, each chip would power up in one of the added states. An added state is nonfunctional with a very high probability, so the chip would be locked. The designer would be the only entity who can provide the *unique set of unlocking inputs* for transiting between the specific power-up state and the original "reset" state for each chip. In the remainder of the chapter, we use the term *passkey* to refer to the set of inputs corresponding to each traversed edge during unlocking. Let us more formally define the problem.

Input: Given an original sequential specification of a circuit P in form of an FSM = $(\Sigma,\Delta,S,s_0,\delta,\lambda)$, and given a compiler that transforms this functional specification to a netlist.

Objective: Construct a modified FSM denoted by BFSM=$(\Sigma + \Sigma',\Delta,S + S', s_0,\delta + \delta',\lambda + \lambda')$ such that transitions from a state $s' \in S'$ to one of the original FSM states $s \in S$ would require "strong" passkeys for transiting the edges. A passkey sequence of length l denoted by $\alpha=\{\alpha_1, \ldots, \alpha_l\}$ applied to the state s' would result in a sequence of transitions before it gets to reset state s. By strong passkeys, we mean they are random and long such that they cannot be guessed by the brute force attack. Without the loss of generality, let us assume that each edge passkey length is fixed to the value k. The reached state s then would be $s = \delta(s', \alpha) = \delta(\delta(\delta(\ldots\delta(s', \alpha_1)\ldots), \alpha_{l-1}), \alpha_l)$. The corresponding output would be $\lambda(s', \alpha) = \lambda(\delta(\delta(\ldots\delta(s', \alpha_1)\ldots), \alpha_{l-1}), \alpha_l)$.

Methodology: To address the above problem, we demonstrate a BFSM construction such that the addition of states and transitions to the original graph is an instance of a general output multi-point function that is efficiently obfuscatable. Using this result, we devise a strong passkey mechanism to build a secure hardware metering.

To show the obfuscatable BFSM construction, we form a set of extra state transition relations that are added to the original FSM. The addition is such that for reaching the states in the original FSM from each potential power-up state $s' \in S'$, one has to traverse one or more of the added states or transitions. Let us add a number of transitions to each state $s' \in S'$ such that each s' has at most t outgoing transitions denoted by $\delta'(s', \alpha_{s1}), \ldots \delta'(s', \alpha_{st})$. The upper bound t on the number of transitions from one added state is set such that there is a sequence of transitions to traverse from each added state to one of the states in the original FSM (our implicit assumption is that the "reset" state is reachable from all the states in the FSM). In other words, $\forall s' \in S', \exists \alpha = \{\alpha_1, \ldots, \alpha_l\}$ such that $s = \delta''(s', \alpha) = \delta''(\ldots\delta''(s, \alpha_1)\ldots), \alpha_{l-1}), \alpha_l)$, where δ'' may be either δ (transition on the original state machine) or δ' (transition on the added state machine).

Assume that the BFSM is at the state $s' \in S$ and the user inputs a vector of inputs $\{x_1, \ldots, x_l\}$. Let us first define a function $W(x_1, \ldots, x_l)$ as follows:

$$W(x_1, \ldots, x_l) = \begin{cases} s \in S \text{ if } \exists s'_0, \ldots, s'_l \text{ s.t. } s'_0 \equiv s' \text{ and} \\ \qquad s'_l \equiv s, \text{ and } s_j = \delta''(s_{j-1}, x_j) \\ \perp \qquad \text{Otherwise.} \end{cases} \tag{7.6}$$

Consider the family of functions $\mathscr{W}(.)$ over the set of all $W(.)$ that can be defined for the BFSM with parameters S' and t and the transition functions δ''. Our interest is in cases where there is a polynomial number of traversed edges in terms of the parameters S' and t. We notice that there are exponentially many possible valid sequences of input transitions that can traverse from the initial power-up state to one of the original states. Therefore, the function $\mathscr{W}(.)$ cannot be immediately shown to be obfuscatable by directly using the theoretical results for obfuscating general output multi-point functions.

Instead of discussing the $W(.)$ functions on the general BFSM structure, we decompose the transition path to state-to-state transition edges. Each state s_j can be represented by its incident transition functions denoted by ${\delta''}_j^{(t')}(s_j, \alpha_j^{(t')})$ for traversal to the "neighboring states". The neighboring states, denoted by $s_j^{(t')}$ are those reachable by applying a single passkey, i.e., $s_j^{(t')} = {\delta''}_j^{(t')}(s_j, \alpha_j^{(t')})$, $0 < t' \leq t$. For each state $s_j^{(t')}$, there is a unique passkey $\pi(t')$ from $\{0, 1\}^k$ (recall that k is the valid passkey length for the BFSM graph). Define the function $\omega(.)$ as follows:

$$\omega(s_j^{(t')}, \pi'', j) = \begin{cases} (s_j, {\delta''}_j^{(t')}(s_j, \pi(t'))) & \text{if } \pi'' = \pi(t') \,\&\, \exists s_j \\ & \text{incident to } s_j^{(t')} s.t. \\ & s_j^{(t')} = {\delta''}_j^{(t')}(s_j, \pi'') \\ \perp & \text{otherwise.} \end{cases}$$

The obfuscation of $W(.)$ consists of obfuscation of $\omega(.)$ that is a multi-point function with at most $t|S'|$ points where the output is not $\perp$ and hence can be efficiently obfuscated. Informally, this is an efficient obfuscation since the adversary (with a high probability) cannot guess a set of randomly selected keys that would result in a valid transition. Reference [84] has shown that a composition of obfuscatable functions is also efficiently obfuscatable under the random oracle model. Therefore, we conclude that $W(.)$ is also efficiently obfuscatable since it can be written as a composition of obfuscatable functions.

More importantly, it was demonstrated that for such functions that can be written by decompositions, exploring parts of the passkeys for transitions on the composite structure (in our case BFSM) would not reveal the remainder of passkeys, as long as the two passkey sets are not correlated and do not include all the same compositional unit (in our case the same state) [83, 84]. Therefore, as long as the traversal paths from the power-up state to the original set of states for a locked IC are at least different in one state from a previously unlocked IC, obfuscation of a multi-point function remains secure.

7.4.3 Additional Considerations and Applications

So far, we have described construction of a BFSM such that the structure can be used for obfuscating the initial power-up state. It is clear that to ensure randomness and protection of the scheme, it is desired to have an unbiased PUF that can generate an output bit of 0 or 1 with completely equal probabilities. The diversity of the power-up state is guaranteed if the PUF output is completely random. The other important factors to consider are stability of PUF responses and its vulnerability to operational and temperature conditions. As we mentioned earlier, to ensure robustness and full operability in presence of fluctuating operational and environmental conditions, we use error correction codes (ECC) in conjunction with PUF responses. As long as the PUF response is unclonable, the security of our method is not based on keeping the PUF output secret (it is the passkeys on the transition graph that are secret). Therefore, the ECC syndromes would not reveal much beyond the PUF value. The use of ECC for PUF is not new, and several other works have used ECC in conjunction with PUF to ensure its robustness. We refer the readers to the related work in this domain for the studies of added overhead and security of different ECC methods [80]. Another important issue that we address is storage of the passkeys. Once the passkey is given for unlocking one chip, its value would be stored on NVM inside the chip along with the ECC code. Therefore, every time the IC is powered up, it automatically reads the passkeys to transition to the original reset state.

An interesting and important observation is that the mechanisms described in the previous section for hiding a number of states within the FSM such that only the designer knows about the traversal to/from those states could be used as a basis for a suite of other security protocols. For example, this state hiding can be used for remote authentication or identification by the designer, online integrity checking, and real-time monitoring, controlling, enabling, or disabling the chip [23]. An application of the method for trusted integration of multiple IP cores was demonstrated in [40].

An example structure for disabling the chip, in case of tamper detection, is creation of *blackhole* states. The blackholes are states that cannot be exited regardless of the incident input sequence. Their design is very simple as shown in Fig. 7.7, where the blackhole states do not have a route back to the original reset state. A special case is

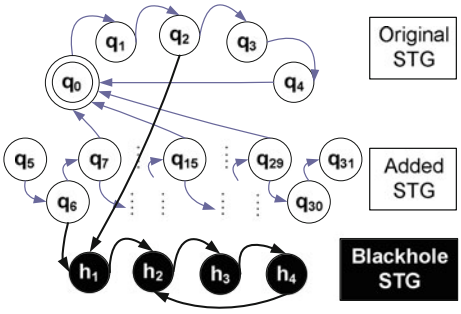


Fig. 7.7 An example of a *blackhole* FSM that can be used for online controlling and disabling the ICs. They can also provide a countermeasure against the brute force attacks (Sect. 7.6)

creation of trapdoor *grayhole* FSM. Grayhole FSMs are designed in such a way that only a long sequence of input signals (known by the passkey transition holder) can bring the control out of these states to the original functional states of the design. A greyhole construction is similar to a blackhole, with the exception of at least one edge in the greyhole that can take the design back to its functional states.

7.5 Automatic Synthesis and Implementation

In this section, we present the details of the implementation of secure BFSM construction that was introduced in the previous section. There is a need to devise a low overhead automatic construction for the high-level specification. Since the conventional tools are not optimized for synthesizing a relatively large state space, our solution for a low overhead implementation is careful selection of the BFSM topology by pre-synthesis and then performing *automatic iterative synthesis*. Our secure BFSM implementation method has three steps. The first step is BFSM graph topology construction described in Sect. 7.5.1. Second, transition (edge) passkey assignment is outlined in Sect. 7.5.2). Finally, Sect. 7.5.3 presents the iterative synthesis method.

7.5.1 BFSM Graph Topology Construction

The BFSM inputs are termed *primary inputs (PI)* and its outputs are termed *primary outputs (PO)* since they are the same as the PI and PO to the original design. The output at the PO would be correct for an unlocked chip. This value would be incorrect for a locked IC with a very high probability.

The number of states is increased to ensure randomness and uniqueness of the start-up state, as described in earlier equations. For a BFSM $=(\Sigma + \Sigma', \Delta, S + S', s_0, \delta + \delta', \lambda)$ with S' added states, our method first uses a partition approach. A partitioning method was introduced in [9], where the low overhead implementation of smaller partitioned FSMs is determined by pre-synthesizing and evaluating various random configurations of small FSMs (e.g., with 2^4 or 2^5 states). The partitions are then combined by randomly added edges to form the added state space and transitions. We refer to the added edges and logic for connecting and mixing the original FSM with the new partitions by the term *glue logic*.

An example for a partition is shown in Fig. 7.8, where the steps for construction of a partition STG with 8 states are demonstrated. A ring counter is used as the starting point as shown in Fig. 7.8a. Next, a few states are randomly selected and reconnected. Say on Fig. 7.8b, the state q_1 is reconnected, such that still there will be a path from each state to every other state. Finally, a few random transitions are added to STG. In the example shown in Fig. 7.8c, the states q_1 and q_4 are randomly reconnected, and the edges $\{q_4 \rightarrow q_1, q_7 \rightarrow q_3, q_2 \rightarrow q_2, q_7 \rightarrow q_7\}$ are randomly added. Therefore, the random perturbations on the ring counter are organized such

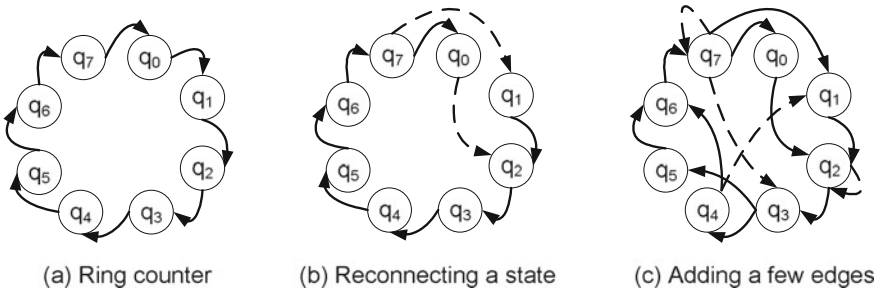


Fig. 7.8 A partition with $2^3 = 8$ states formed by random perturbations on a ring counter to store connectivity

that the graph connectivity is preserved. Note that our partitioning approach is more systematic and guided than [9], but it uses similar principles. The key difference is because our secure construction method constrains the number of directed edges incident to one added state to be t, such that t ensures the graph connectivity and multiplicity of keys.

In our implementation, connectivity in each group is ensured by construction, such that every state in the partitioned group of states is reachable from every other state. Next, transitions are added among the partitioned FSMs such that the reset state is reachable from all other states. For checking the connectivity of the added group of new states on a generic graph to the original reset state, we devise an algorithm as follows: For a state transition graph (STG) demonstrating the BFSM in graph format, form a corresponding graph STG$''$ whose vertices are a one-to-one mapping of the vertices in STG, but reverses the direction of the edges. This may be a cyclic graph. Next, use a spanning tree algorithm (e.g., Kruskal's algorithm [85]) to extract a directed acyclic graph (DAG) rooted at the initial reset state. If all nodes are reachable from the root node on this DAG (can be checked by a breadth first search (BFS) on the graph), then the connectivity condition is satisfied. Otherwise, more edges between the unreachable states and the reachable states are needed. For a graph with $|S| + |S'|$ vertices and E edges, the complexity of the Kruskal algorithm is $O(E \ \log\{|S| + |S'|\})$ and the complexity of the BFS is $O(|E| + |S| + |S'|)$. Since $\log \ \{|S| + |S'|\}$ is typically larger than $(1 + \frac{|S|+|S'|}{E})$, the overall complexity is in the order of $O(E \ \log\{|S| + |S'|\})$.

Now, a natural question that may arise is that for traversing to the original reset state, one has to pass one of its neighbors so there could only be at most t distinct passkeys which do not share the states. As we mention in our attacks and countermeasures section, this attack would be effectively avoided as long as there are enough number of added states and the passkeys, for some parts of the unlocking sequence are unique and hard to guess. Another important class of attacks that we discuss is the capturing and removal attacks.

7.5.2 Selecting and Computing the Transition Passkeys

Selecting strong passkeys is integral to the security of active hardware metering. Generally speaking, a random passkey is a vector of symbols of a certain length taken from a set of symbols by a random selection process. Each symbol must be equally likely to be selected. The strength of random passkeys depends on the entropy of the underlying random number generator. In our case, the selected passkeys can also serve a dual purpose: The designer can use them as a proof of ownership in addition to using them for unlocking.

For this reason, in order to generate the passkey (pre-synthesis in software), we use the hash value of the designer generated words that are signed by a private key (PrK) of a public-key cryptographic (PKC) system. Now, others with access to the public-key (PuK) of the same system can verify the ownership of the designer upon unlocking. However, an eavesdropper, who can unlock the chip using a stolen BFSM structure, would not be able to claim ownership.

The steps for selecting the transition passkeys for the edges on the added graph (E) are as follows. First, the IP rights owner selects E symbol strings and uses its own PrK to sign each of the strings. The strings must be long enough to be resilient against the brute force guessing attack. Next, a strong hash function (e.g., SHA-2) is applied to the signed strings, so a fixed-length digest message is obtained. Note that the edge transition passkeys are independent of the state encodings performed by the synthesis software. Therefore, reading out the state values from the FFs would not help in revealing the incident edge passkeys. Note that since for each chip a new transition passkey has to be found as a path on the BFSM, this computation needs to be efficient. To provide efficiency, binary decision diagrams (BDDs) can be used to store the BFSM and computing the key pairs.

7.5.3 Iterative Automatic Synthesis

Once the BFSM is selected and the transition passkeys are determined, the structure has to be synthesized. To perform the synthesis, we first modify the original FSM adding extra control inputs. These inputs represent the transitions from the BFSM with the correct passkeys. Next, we synthesize each partition of the BFSM with an output representing the incident edge transitions. After synthesizing each component separately, we connect the extra inputs of the original FSM to the outputs of the partitioned FSMs and resynthesize the whole system. This way, if the original design has K FFs and the BFSM has K'' FFs, then $|S| \leq 2^K$ and $|S'| = 2^{K''} - 2^K$ that implies $|S'|$ is much larger than $|S|$.

7.6 Attacks and Countermeasures

In this section, we state the attacks identified on active hardware metering and discuss how the newly proposed method is secure against the proposed attacks.

(1) Brute force attack. The adversary attempts at randomly generating inputs to randomly traverse from the power-up state until it reaches a state that was reached on the previously unlocked ICs, or it reaches the reset state.
– Under the construction of general output multi-point function, probability of finding the correct passkey resulting in a valid edge transition is extremely low, and therefore, this class of attacks would not be able to break the security. Another effective countermeasure against the brute force attack is creation of blackhole states as discussed in Sect. 7.4.3. All what is needed is to strategically place the blackhole states and their adjacent edges such that the probability of randomly entering them is high. To avoid the problem of starting up in one of the blackhole states, one can use the greyholes that can be exited by a long sequence of passkeys.

(2) BFSM reverse engineering. The adversary uses the states revealed by unlocking each chip to gradually build a BFSM model to enable finding the passkeys to unlock the new ICs.
– Our construction method is secure against this attack by the decomposition property (Sect. 7.4.2) [84]. The way to ensure security is by finding paths that are at least not intersecting on one edge. As an example, for a 128-bit input, each passkey has 2^{128} possibilities, so even finding the passkey string for one edge is of exponential complexity.

(3) PUF removal/tampering attack. The adversary removes/tampers the PUF on a locked IC and instead places a piece of SRAM that contains the PUF responses from a previously unlocked chip. Now, the passkeys for unlocking the previous chip can be used for unlocking a new IC.
– To protect against PUF removal, there are multiple measures that can be taken. One such measure is to use the time bound and single-cycle property of an authentic integrated PUF device (a memory lookup takes more cycles) [63, 66, 68, 69, 86, 87]. The PUF signal timing can be designed to be an integrated part of the timing path of the main functional description. Therefore, removing PUF would affect the circuit timing. Recall that in our attack model, redoing the timing closure is as hard as designing a chip from scratch and is not a valid attack. Another measure is to add obfuscated states within the FSM for PUF checking. The self-check signals from this test states would verify the existence of the PUF during unlocking and while the chip is in operation by testing for randomness (by adding a randomness test modules) or by supplying challenges such that the PUF response can be continuously checked or it's timing is taken into account [63, 66, 68, 69, 86, 87]. Also, the prohibitive cost of mask/retiming should be considered when targeting removal of an integrated PUF that is on the design's timing path.

(4) State capture and replay attack(s). The adversary captures the states from a previously unlocked chip and would try to force a new chip to power up in a similar state, or it forces the unlocked IC to start at the original reset state.
– To overcome this attack, in addition to the power-up states, also the traversal passkeys can be set to be a function of PUF. Now, the passkeys for one chip would not work on the next because of the uniqueness of PUF responses. All what is needed is to have a number of challenges (inputs) for the PUF and their associated ECC for the corresponding traversals. Since the responses from the PUF cannot be cloned on another chip, we have safeguarded our method against this attack. The PUF removal/tampering attack was already discussed.

7.7 Performance and Hardware Overhead

In this section, we analyze the overhead in terms of area, power, and timing of the synthesized circuits from the ISCAS benchmark suite. The hardware overhead is evaluated on the H.264/MPEG-4 or Advance Video Coding (AVC) decoder circuit, synthesized and implemented on FPGA.

7.7.1 Performance Overhead

As already stated, the finite state machine description is commonly used for realizing the control path of the circuitry. It is important to note that in modern designs, the control path is only a small part of the overall structure ($<<1\%$) [88]. Therefore, even doubling or tripling the control part would not have a significant impact on the overall design overhead in terms of area or power. The timing increase would impact the design delay though. As we will see in our evaluations, the timing overhead of our method is negligible and it could be even further suppressed by alternative synthesis methods.

The BFSM construction used in our simulations adds 20 flip-flops to the original design. As we mentioned earlier, the number of new states ought to be large. Let us assume that K is the number of FFs in the original design corresponding to $S = 2^K$ states, and $K + 20$ is the number of FFs in the modified design corresponding to $S' = 2^{K+20} - 2^K$ states. It is clear that the $S' >> S$ condition is satisfied. The length of the edge transition passkeys in our evaluations is set to 64 bits. An unlocking sequence on one chip would find a path of at least 8 edges on the BFSM state transition graph. Therefore, the minimum length of a passkey for a chip is 512 bits. This number could be fixed or could be any multiple of 64 bits.

The table in Fig. 7.9 demonstrates comprehensive performance overhead evaluations on the ISCAS benchmark suite. The first column denotes the benchmark circuit name (sorted by the benchmark number order). The next three columns (Columns 2–4) show the original design properties: the number of primary inputs, the num-

Circuit	In	Out	FFs	Orig. Area	Added Area	OH (%)	Orig. Power	Added Power	OH (%)	Orig. Delay	OH (%)
s820	18	19	5	769	+1411	186	2773	+5924	213.6	28.2	0
s1196	14	14	18	1009	+1524	151	2558	+8223	321.4	35.8	3
s1238	14	14	18	1041	+1472	141	2709	+7153	264.0	34.4	1
s1423	17	5	74	1164	+1393	120	4883	+5564	114.0	92.4	0
s1488	8	19	6	1387	+1261	91	3859	+2804	72.7	38	1
s1494	8	19	6	1393	+1591	114	3913	+9632	246.1	38.4	2
s5378	35	49	164	4212	+1203	28.6	12459	+1874	15.0	32.2	3
s9234	36	39	211	7971	+1425	17.9	19386	+6312	32.6	75.8	0
s13207	31	121	669	11241	+1571	14	37844	+9334	24.7	85.6	1
s15850	14	87	597	13659	+1281	9.3	40003	+3404	8.5	116	0
s35932	35	320	1728	28269	+1383	4.9	122048	+5279	4.3	299.4	0
s38584	12	278	1452	32910	+1226	3.7	112707	+2357	2.1	94.2	2

Fig. 7.9 Metering overhead on the benchmark suite

ber of primary outputs, and the number of flip-flops post-synthesis. Columns 5–7 demonstrate the design area (in terms of the number of gates in the ABC tool) in the following order: the original post-synthesis area, the added area post-BFSM synthesis after applying our method, and the ratio between the two former metrics (in %). The original design's power post-synthesis, the newly added power post-BFSM synthesis after applying the hardware metering, and the ratio between the two powers (in %) are reported in Columns 8–10. The post-synthesis delay of the original design and the ratio between the added delay (post-BFSM construction and synthesis) to the original delay are shown in the last two columns, respectively.

Let us start by analyzing the area overhead. As can be seen in Column 6, the added area by the BFSM seems to be independent of the benchmark circuit size, with a standard deviation of 131 around its mean of 1395. Observe that the circuits on the top of the table are the smaller ones in the benchmark set, with a limited number of inputs and outputs and FFs, occupying a small area (in the original circuit synthesis). Given this observation, it is natural that the overhead (%) is much more significant on the smaller circuits compared to the larger ones. The mean of the percentage overhead in the area is about 73%, with a standard deviation of 67% indicating large fluctuations among the circuits. Looking at the bottom half of the table that includes the larger designs, the mean of the overhead (%) is set at a much lower 13%, with still a relatively large standard deviation of 9%. These figures show that for most industrial designs that are of larger complexity (making them worthwhile to protect), the area overhead for the BFSM construction is quite low, especially since the control path is a small part of the overall design.

Next, we analyze the power consumption (Columns 8–10). Note that the units for this report are the same units reported by the ABC synthesis tool. We see that the added power (Column 9) has a large variation that seems not to follow the benchmark size. It has a standard deviation of 2650 around its mean of 5655. For the small circuits in the suite that are shown in the upper half of the table, the overhead ratio is very

large, with a mean of 205% and standard deviation of ∼95%. The overhead ratio is much lower for the larger circuits in the benchmark set in the lower half of the table, with a mean of 14.5% and a standard deviation of 12%. Since the circuits in the benchmark set are small compared to the industrial strength circuitry in design and use today, it is safe to say that the power overhead of the metering method is low, in particular, since the control path by itself is very small compared to the entire design.

Last but not least, we evaluate the impact of the BFSM construction methodology on the circuit timing (Columns 11–12). The unit for timing is the same unit reported by ABC. The ratio of the added critical path delay overhead compared to the original delay seems to be independent of the circuit size, with a mean of 1% and standard deviation of 1.15%. Therefore, we see that the overhead in the critical path delay introduced by our method is rather low.

It is also worthwhile to compare these figures to the hardware metering methodology introduced in [9]. In comparison, the secure BFSM construction method introduced here produces a visibly higher overhead. This is because the latest methodology introduced in this chapter follows the theoretical guarantees described in the previous sections. The heuristic-only solutions offered in [9] could not provide strong proof of security.

7.7.2 Hardware Implementation Overhead

The H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding) is a video compression standard. It is presently one of the most used formats for recording, compressing, and distributing the high-definition videos. It is widely used in Blu-Ray systems, YouTube, iTunes, and other Internet video distribution centers. Designing efficient H.264/MPEG-4 is both a big challenge and opportunity, especially with the move toward smartphones and other low power portable systems with video decoding capabilities. We selected the H.264/MPEG-4 for our hardware implementation and evaluation purposes. The original design was written in the Verilog hardware description language. The tool flow for synthesizing this design was described earlier in this section. Note that we do not report the overhead of implementing PUFs on the FPGA. The overhead for implementing PUF on FPGA on Xilinx boards is readily available in the contemporary literature [60, 63, 69, 80, 89].

The table in Fig. 7.10 shows the design area after synthesis to the FPGA in terms of the number of equivalent gates and the number of occupied look-up tables (LUTs), along with the percentage overhead. The second column reports the number of equivalent gates and the number of LUTs for the original design. The third column shows the overhead for a BFSM with 20 new FFs and a key length of 1024. It can be seen that the percentage of added equivalent gates is about 6.7%, and the percentage of added LUTs is about 13%. We also experimented with two other cases with 40 and 64 added FFs, as shown on the fifth and sixth columns, with key lengths of 2048 and 5120, respectively. As we mentioned earlier, the key length is determined by the

Fig. 7.10 Metering overhead for H.264/MPEG4 on FPGA

	Original H.264	(+20 states) Key: 1024b	(+40 states) Key: 2048b	(+64 states) Key: 5120b
Number of Gates	381,176	407,068	429,075	457,574
Gate overhead	-	6.79%	12.56%	20.04%
Number of LUTs	26,485	29,996	33,160	37,106
LUT overhead	-	13.25%	25.19%	40.09%

number of edge transitions required for unlocking and the passkey on each edge. The passkey on each edge is set to the number of inputs on the edge, $\alpha = 64$ bits. A key length of 1024 means that we require 16 edge transitions in our unlocking sequence.

We see that with the increase in the number of added FFs, the percentage overhead in terms of the number of equivalent gates and the number of LUTs increases linearly. Since adding to the number of FFs could yield exponentially stronger proofs for security, our protection method is relatively low overhead for very secure construction. As noted earlier, for MPEG4 decoding and many other data-intensive applications that are implemented in hardware for efficiency reasons, the bottleneck is in the data path optimization and not in the control path which is a much smaller part of the overall design. Note that the earlier work in hardware metering was only evaluated on benchmark simulations.

7.8 Conclusion

This chapter describes active hardware metering, a method which uniquely locks each integrated circuit (IC) at the fabrication facility. The locking mechanism works by embedding unique chip identifiers coming from a physical unclonable function (PUF) into the device's control function. The designer (IP rights owner) who has access to the full state encoding of the design is the only entity who can provide the passkeys for unlocking the chip. The unique identifiers are integrated within the states of the design's finite state machine (FSM). The FSM is boosted—in a way that does not alter the functionality of the original design—to include many added states.

We demonstrate provable security guarantees for active hardware metering. We show a construction for active hardware metering by modifying the behavioral description of the design in the finite state machine domain such that the modifications form an instance of a general output multi-point function that was shown to be efficiently obfuscatable. The hardware synthesis that takes the behavioral-level specification and transforms it to a netlist must be an obfuscating compiler for the metering to be secure so the passkeys cannot be guessed or attacked. The theoretical results on obfuscating the family of point functions are leveraged to ensure security of the new construction in the random oracle model. Automated synthesis methods for integration of the new secure metering construction are derived. We discuss the attacks and safeguards. The efficiency and practicality of the methods are

demonstrated by experimental evaluations on sequential benchmarks and by proof-of-concept hardware metering implementation on a H.264 MPEG decoder on Xilinx Virtex-5 FPGA.

Acknowledgements This work discussed in this article was in parts supported by the Defense Advanced Research Projects Agency (DARPA) grant No. W911NF-07-1-0198, Office of Naval Research (ONR) grant No. R16480, AFOSR-MURI grant on Nano-Hardware Security, and National Scient Foundation Trust-Hub. Dr. Golsa Ghiaasi-Hafezi, Dr. Azalia Mirhoseini, and Mr. Siam Hussain helped with reading and editing the chapter.

References

1. Defense science board (DSB) study on high performance microchip supply. http://www.acq.osd.mil/dsb/reports/2005-02-hpms_report_final.pdf
2. Pecht M, Tiku S (2006) Bogus! electronic manufacturing and consumers confront a rising tide of counterfeit electronics. IEEE Spectr 43(5):37–46
3. Pope S (2008) Trusted integrated circuit strategy. IEEE Trans Compon Packag Technol 31(1):230–235
4. Managing the risks of counterfeiting in the information technology industry. a white paper by KPMG and the Alliance for Gray Market and counterfeit Abatement (AGMA)
5. Defense industrial base assessment: Counterfeit electronics (2010) a report by Bureau of Industry and Security's (BIS)Office of Technology Evaluation (OTE). http://www.agmaglobal.org/
6. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. In: International workshop on information hiding (IH), pp 81–95
7. Koushanfar F, Qu G (2001) Hardware metering. In: Design automation conference (DAC), pp 490–493
8. Koushanfar F, Potkonjak M (2007) textscCAD-based security, cryptography, and digital rights management. In: Design automation conference (DAC), pp 268–269
9. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX security symposium, pp 291–306
10. Koushanfar F (2012) Provably secure active ic metering techniques for piracy avoidance and digital rights management. IEEE Trans Forensics Secur (TIFS) 7(1):51–63
11. M. Potkonjak and F. Koushanfar, "Identification of integrated circuits," Dec. 31 2013, uS Patent 8,620,982
12. Lofstrom K, Daasch WR, Taylor D (2000) Ic identification circuit using device mismatch. In: IEEE international solid-state circuits conference (ISSCC), pp 372–373
13. Qu G, Potkonjak M (2003) Intellectual property protection in VLSI design. Academic Publisher, Kluwer
14. Chang C-H, Potkonjak M (2016) Zhang L, Hardware in watermarking and fingerprinting. In: Secure system design and trustable computing. Springer, pp 329–368
15. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans Comput Aided Design 20(9):1101–1117
16. Koushanfar F, Hong I, Potkonjak M (2005) Behavioral synthesis techniques for intellectual property protection. ACM Trans Design Autom Electron Syst 10(3):523–545
17. Torunoglu I, Charbon E (2000) Watermarking-based copyright protection of sequential functions. IEEE J Solid-State Circuits (JSSC) 35(3):434–440
18. Kirovski D, Hwang Y-Y, Potkonjak M, Cong J (1998) Intellectual property protection by watermarking combinational logic synthesis solutions. In: International conference on computer-aided design (ICCAD), pp 194–198

19. Koushanfar F, Alkabani Y (2010) Provably secure obfuscation of diverse watermarks for sequential circuits. In: International symposium on hardware-oriented security and trust (HOST), pp 42–47
20. Lach J, Mangione-Smith W, Potkonjak M (1998) Signature hiding techniques for FPGA intellectual property protection. In: International conference on computer-aided design (ICCAD), pp 186–189
21. Qu G, Potkonjak M (2000) Fingerprinting intellectual property using constraint-addition. In: Design automation conference (DAC), pp 587–592
22. Wendt JB, Koushanfar F, Potkonjak M (2014) Techniques for foundry identification. In: Design automation conference (DAC). ACM, pp 1–6
23. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: International conference on computer-aided design (ICCAD), pp 674–677
24. Huang J, Lach J (2008) IC activation and user authentication for security-sensitive systems. In: International symposium on hardware-oriented security and trust (HOST), pp 76–80
25. Roy J, Koushanfar F, Markov I (2008) EPIC: ending piracy of integrated circuits. In: Design automation and test in Europe (DATE), pp 1069–1074
26. Roy J, Koushanfar F, Markov I (2008) Protecting bus-based hardware ip by secret sharing. In: Design automation conference (DAC), pp 846–851
27. Alkabani Y, Koushanfar F, Kiyavash N, Potkonjak M (2008)Trusted integrated circuits: a nondestructive hidden characteristics extraction approach. In: Information hiding (IH), pp 102–117
28. Dabiri F, Potkonjak M (2009) Hardware aging-based software metering. In: Design, automation and test in Europe conference and exhibition DATE, pp 460–465
29. Wei A, Nahapetian M, Potkonjak M (2011) Robust passive hardware metering. In: International conference on computer-aided design (ICCAD)
30. Koushanfar F (2011) Hardware metering: a survey. Book chapter in introduction to hardware security and trust. Springer, Berlin
31. Koushanfar F (2011) Integrated circuits metering for piracy protection and digital rights management: an overview. In: Great lakes symposium on VLSI (GLSVLSI), pp 449–454
32. Koushanfar F (2012) Hardware metering: a survey. In: Introduction to hardware security and trust. Springer, pp 103–122
33. Koushanfar F, Fazzari S, McCants C, Bryson W, Sale M, Song P, Potkonjak M (2012) Can eda combat the rise of electronic counterfeiting? In: Design automation conference (DAC), pp 133–138
34. Potkonjak M, Chen D, Kalla P, Levitan SP (2015) Da vision 2015: from here to eternity. In: International conference on computer-aided design (ICCAD). IEEE Press, pp 271–277
35. M. Potkonjak, "Secure authentication," May 12 2015, uS Patent 9,032,476. [Online]. Available: https://www.google.com/patents/US9032476
36. Rostami M, Koushanfar F, Karri R (2014) A primer on hardware security: models, methods, and metrics. Proc IEEE 102(8):1283–1295
37. Potkonjak M (2015) Usage metering based upon hardware aging. US Patent 9,177,119. https://www.google.com/patents/US9177119. Accesed 3 Nov 2015
38. Koushanfar F, Potkonjak M (2015) Methods and systems of digital rights management for integrated circuits. US Patent 8966660B2. Accesed 24 Feb 2015
39. Karri R, Koushanfar F (2014) Trustworthy hardware. Proc IEEE, vol 102(8)
40. Alkabani Y, Koushanfar F (2008) Active control and digital rights management of integrated circuit IP cores. In: International conference on compilers, architecture, and synthesis for embedded systems (CASES), pp 227–234
41. Chang C-H, Potkonjak M (2015) Secure system design and trustable computing. Springer, Berlin
42. Rostami M, Koushanfar F, Rajendran J, Karri R (2013)Hardware security: threat models and metrics. In: International conference on computer-aided design (ICCAD). IEEE Press, pp 819–823

43. Kong J, Koushanfar F (2014) Processor-based strong physical unclonable functions with aging-based response tuning. IEEE Trans Emerg Topics Comput 2(1):16–29
44. Koushanfar F, Karri R (2014) Can the shield protect our integrated circuits? In: Midwest symposium on circuits and systems (MWSCAS). IEEE, pp 350–353
45. Zhang J, Lin Y, Lyu Y, Qu G (2015) A puf-fsm binding scheme for fpga ip protection and pay-per-device licensing. IEEE Trans Inf Forensics Secur (TIFS) 10(6):1137–1150
46. Roy JA, Koushanfar F, Markov IL (2014) Protecting hardware circuit design by secret sharing. US Patent 8,732,468. Acessed 20 May 2014
47. Yuan L, Qu G (2004) Information hiding in finite state machine. In: Information hiding conference (IH), pp 340–354
48. Chakraborty R, Bhunia S (2008) Hardware protection and authentication through netlist level obfuscation. In: International conference on computer-aided design (ICCAD), pp 674–677
49. Guo Z, Tehranipoor M, Forte D, Di J (2015) Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In: Design automation conference (DAC). ACM, p 114
50. Zhang J, Lin Y, Qu G (2015) Reconfigurable binding against fpga replay attacks. ACM Trans Design Autom Electron Syst (TODAES) 20(2):33
51. Zhang J, Qu G (2014) A survey on security and trust of fpga-based systems. In: FPT, pp 147–152
52. Kiyavash N, Koushanfar F, Coleman TP, Rodrigues M (2013) A timing channel spyware for the csma/ca protocol. IEEE Trans Inf Forensics Secur (TIFS) 8(3):477–487
53. Rührmair U, Xu X, Sölter J, Mahmoud A, Majzoobi M, Koushanfar F, Burleson W (2014) Efficient power and timing side channels for physical unclonable functions. In: International workshop on cryptographic hardware and embedded systems (CHES), vol 8731, pp 476–492
54. Shahrjerdi D, Rajendran J, Garg S, Koushanfar F, Karri R (2014) Shielding and securing integrated circuits with sensors. In: International conference on computer-aided design (ICCAD). IEEE, pp 170–174
55. Xu T, Wendt JB, Potkonjak M (2014) Security of iot systems: design challenges and opportunities. In: International conference on computer-aided design (ICCAD). IEEE Press, pp 417–423
56. Abera T, Asokan N, Davi L, Koushanfar F, Paverd A, Sadeghi A-R, Tsudik G (2016) Invited-things, trouble, trust: on building trust in iot systems. In: Design automation conference (DAC). ACM, p 121
57. Mirhoseini A, Songhori EM, Koushanfar F (2013) Idetic: a high-level synthesis approach for enabling long computations on transiently-powered asics. In: Pervasive Computing and Communications (PerCom). IEEE, pp 216–224
58. Koushanfar F, Sadeghi A-R, Seudie H (2012) Eda for secure and dependable cybercars: challenges and opportunities. In: Design automation conference (DAC). ACM, pp 220–228
59. Gassend B, Clarke D, van Dijk M, Devadas S (2002) Silicon physical random functions. In: Conference on computer and communications security (CCS), pp 148–160
60. Suh G, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Design automation conference (DAC), pp 9–14
61. Rührmair U, Devadas S, Koushanfar F (2011) Security based on physical unclonability and disorder. Book chapter in introduction to hardware security and trust. Springer, Berlin
62. Majzoobi M, Koushanfar F, Potkonjak M (2008) Testing techniques for hardware security. In: International test conference (ITC), pp 1–10
63. Majzoobi M, Koushanfar F, Potkonjak M (2009) Techniques for design and implementation of secure reconfigurable pufs. ACM Trans Reconfig Technol Syst (TRETS) 2(1):5:1–5:33
64. Majzoobi M, Koushanfar F, Devadas S (2010) Fpga puf using programmable delay lines. In: International workshop on information forensics and security (WIFS)
65. Majzoobi M, Rostami M, Koushanfar F, Wallach DS, Devadas S (2012) Slender puf protocol: a lightweight, robust, and secure authentication by substring matching. In: Security and privacy workshops (SPW), pp 33–44
66. Meguerdichian S, Potkonjak M (2011) Matched public PUF: ultra low energy security platform. In: International symposium on low power electronics and design (ISLPED)

67. Majzoobi M, Ghiaasi G, Koushanfar F, Nassif SR (2011) Ultra-low power current-based puf. In: 2011 IEEE international symposium of circuits and systems (ISCAS), pp 2071–2074
68. Potkonjak M, Meguerdichian S, Nahapetian A, Wei S (2011) Differential public, physically unclonable functions: architecture and applications. In: Design automation conference (DAC)
69. Majzoobi M, Koushanfar F (2011) Time-bounded authentication of FPGAs. In: IEEE transaction on information forensics and security (TIFS)
70. Rostami M, Majzoobi M, Koushanfar F, Wallach DS, Devadas S (2014) Robust and reverse-engineering resilient puf authentication and key-exchange by substring matching. IEEE Trans Emerg Topics Comput 2(1):37–49
71. Xu T, Potkonjak M (2015) The digital bidirectional function as a hardware security primitive: architecture and applications. In: International symposium on low power electronics and design (ISLPED). IEEE, pp 335–340
72. Rajendran J, Rose GS, Karri R, Potkonjak M (2012) Nano-ppuf: a memristor-based security primitive. In: Annual symposium on VLSI. IEEE, pp 84–87
73. Potkonjak M, Goudar V (2014) Public physical unclonable functions. Proc IEEE 102(8):1142–1156
74. Xu T, Wendt JB, Potkonjak M (2014) Secure remote sensing and communication using digital pufs. In: Symposium on architectures for networking and communications systems. ACM, pp 173–184
75. Rostami M, Wendt JB, Potkonjak M, Koushanfar F (2014) Quo vadis, puf?: trends and challenges of emerging physical-disorder based security. In: Design, automation and test in Europe (DATE). European Design and Automation Association, p 352
76. Hussain SU, Yellapantula S, Majzoobi M, Koushanfar F (2014) Bist-puf: Online, hardware-based evaluation of physically unclonable circuit identifiers. In: International conference on computer-aided design (ICCAD). IEEE, pp 162–169
77. Hussain SU, Majzoobi M, Koushanfar F (2016) A built-in-self-test scheme for online evaluation of physical unclonable functions and true random number generators. IEEE Trans Multi-Scale Comput Syst 2(1):2–16
78. Mouli C, Carriker W (2007) Future fab: how software is helping intel go nano-and beyond. IEEE Spectr 44(3):38–43
79. Santo B (2007) Plans for next-gen chips imperiled. IEEE Spectr 44(8):12–14
80. Yu M, Devadas S (2010) Secure and robust error correction for physical unclonable functions. IEEE Design Test Comput 27:48–65
81. Ahmed SE, McIntosh RJ (2000) An asymptotic approximation for the birthday problem. Crux Math Math Mayhem 26(3):151–155
82. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K (2001) On the (im)possibility of obfuscating programs. In: International cryptology conference (CRYPTO), pp 1–18
83. Goldwasser S, Rothblum G (2007) On best-possible obfuscation. In: Theory of Cryptography (TCC), pp 194–213
84. Lynn B, Prabhakaran M, Sahai A (2004) Positive results and techniques for obfuscation. In: International conference on the theory and applications of cryptographic techniques (EUROCRYPT), pp 20–39
85. Corman T, Leiserson C, Rivest R, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge
86. Beckmann N, Potkonjak M (2009) Hardware-based public-key cryptography with public physically unclonable functions. In: Information hiding conference (IH), pp 206–220
87. Rührmair U, Chen Q, Stutzmann M, Lugli P, Schlichtmann U, Csaba G (2010) Towards electrical, integrated implementations of SIMPL systems. Inf Secur Theory Pract 6033:277–292
88. Hennessy J, Patterson D (2006) Computer architecture: a quantitative approach, 4th edn. Morgan Kaufmann, Massachusetts
89. Meguerdichian S, Potkonjak M (2011) Device Aging-Based physically unclonable functions. In: Design automation conference (DAC)

Chapter 8
State Space Obfuscation and Its Application in Hardware Intellectual Property Protection

Rajat Subhra Chakraborty and Swarup Bhunia

8.1 Introduction

Reuse-based System-on-Chip (SoC) design using hardware intellectual property (IP) cores has become a pervasive practice in the industry. These IP cores usually come in the following three forms: synthesizable register transfer level (RTL) descriptions in hardware description languages (HDLs) ("Soft IP"); gate-level designs mapped to a standard cell technology library ("Firm IP"); and GDSII design database ("Hard IP"). The approach of designing complex SoCs by integrating tested, verified, and reusable modules reduces the SoC design time and cost dramatically while drastically reducing the time to market [1].

Unfortunately, recent trends in IP piracy and reverse engineering efforts to produce counterfeit integrated circuits (ICs) have led to serious concerns in the IC design community [1–5]. IP piracy can take diverse forms, as illustrated by the following example scenarios:

- A chip design house buys the IP core from the IP vendor, and a rogue designer in the design house makes an illegal copy or "clone" of the IP. The IC design house then uses the IP without paying the required royalty or sells it to another IC design house (after minor modifications) claiming the IP to be its own design [3].
- An untrusted fabrication house makes an illegal copy of the GDS-II database supplied by a chip design house, and then manufactures and sells counterfeit copies of the IC under a different brand name [6].

R.S. Chakraborty (✉)
Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, West Bengal, India
e-mail: rschakraborty@cse.iitkgp.ernet.in

S. Bhunia
Department of Electrical and Computer Engineering, University of Florida, 336A Larsen Hall, Gainesville, FL 32611-6200, USA
e-mail: swarup@ece.ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_8

- A company performs post-silicon reverse engineering on an IC to manufacture its illegal clone [7].

These scenarios demonstrate that all parties involved in the IC design flow are vulnerable to different forms of IP infringement, which can result in the loss of revenue and market share.

In addition to the problem of IP piracy, the threat of hardware Trojans, i.e., malicious modifications to an IC design, is also on the rise [8]. Over the past decade, the semiconductor industry has gradually shifted to a horizontal business model in which design and manufacturing of ICs is distributed to different entities across the globe. Although such offshoring helps to reduce the enormous costs associated with IC manufacturing, it also causes the design house to lose control and oversight of the manufacturing process. Due to this, it becomes possible for adversaries in each stage of the distributed IC supply chain to engage in hardware Trojan insertion as well as IP piracy. Concerns about this kind of vulnerability in ICs and the resultant compromise of security have been expressed globally [9, 10]. Some have also attributed several unexplained military mishaps to the presence of malicious hardware Trojans [8, 11].

Several techniques have been recently proposed in order to counter the threats of IP piracy and hardware Trojans. Unfortunately, each of these techniques has their own limitations and cannot be applied to counter both the threats simultaneously. The authors in [7] have proposed gate-level design modifications to prevent illegal manufacturing of ICs by fabrication houses. However, such techniques are not useful in preventing hardware IP theft. For instance, they do not provide protection against possible IP piracy from the SoC design house. Hardware Trojan detection techniques also have their own set of challenges. Ideally, any undesired modification made to an IC should be detectable by presilicon verification/simulation and post-silicon testing. However, presilicon verification or simulation requires a golden model of the entire IC. This might not always be available, especially for IP-based designs where IPs can come from third-party vendors. Besides, a large multimodule design is usually not amenable to exhaustive verification. Post-silicon, the design can be verified either through destructive depackaging and reverse engineering of the IC [12], or by comparing its functionality or circuit characteristics with a golden version. However, the existing state-of-the-art approaches do not allow destructive verification of ICs to be scalable. Moreover, as pointed out in [12], it is possible for the adversary to insert Trojans in only some ICs on a wafer, not the entire population, which limits the usefulness of a destructive approach.

Traditional post-manufacturing logic testing is also not suitable for detecting hardware Trojans. This is due to the stealthy nature of hardware Trojans and inordinately vast spectrum of possible Trojan instances an adversary can employ. Typically, the adversary would design a Trojan that triggers a malfunction only under rare circuit conditions in order to evade the detection. Due to the finite size of the test set, the rare condition for activation of the Trojan might not be realized during the testing period, especially if the Trojan acts as a sequential state machine or "time bomb" [13]. Special algorithms must be developed to increase the accuracy of logic testing techniques [14]. On the other hand, the techniques for detecting Trojans by comparison

with the "side-channel parameters," such as power trace [15] or delay [16], are limited by the large process variation effect in nanoscale IC technologies, reduced detection sensitivity for ultrasmall Trojans, and measurement noise [15, 17].

In this chapter, we focus on obfuscation as a means to protect ICs against piracy and hardware Trojans. An effective obfuscation procedure needs to simultaneously achieve the following two goals: (1) It should affect the intelligibility of a design so that its true functionality is hidden, even from an adversary who could potentially have full access to the design. Doing so prevents the adversary from engaging in reverse engineering. It also makes it hard for the adversary to insert a carefully targeted hardware Trojan which would need full understanding of the design. (2) The obfuscation should prevent black-box usage of a design, so that an adversary cannot use it to fabricate properly functional ICs, thereby engaging in cloning or counterfeiting.

We present two low-overhead obfuscation techniques, each of which can serve as the basis for a secure SoC design methodology. We consider the circuits to be generic sequential circuits and follow a key-based state space obfuscation approach, where normal functionality is enabled only upon the application of a specific input initialization key sequence. As a result of the adopted state space obfuscation technique through the structural modifications, an adversary might find it difficult to correctly comprehend the circuit functionality. This in turn might cause an inserted Trojan to either not trigger, or become more vulnerable to logic testing-based Trojan detection techniques. Additionally, black-box usage of the IC or IP to perform piracy or counterfeiting is also prevented as the design is essentially locked to the party that does not possess the correct key. One of the major advantages of the proposed technique is that the state space obfuscation takes place without explicit enumeration of the circuit state space. Such an enumeration is usually an infeasible computational problem, even for moderately sized sequential circuits. As a result, the state space obfuscation becomes feasible for the implementation from a designer's perspective. At the same time, it also makes it impossible for an adversary to enumerate the state space while trying to reverse engineer the obfuscated design.

For the proposed approach, we initially develop the theory and design methodology for gate-level obfuscation. Later, we extend the technique to the RTL design descriptions. This ensures the applicability of the proposed methodology to a majority of commercial hardware IPs which come in the RTL ("soft") format. The RTL applicability also offers better portability by allowing design houses to map a circuit using a preferred tool flow to the technology library of a chosen manufacturing process [18]. We note that a conceptually similar protection scheme has also been suggested in [19] for software, whereby decreasing the comprehensibility of a program, malicious software modification is prevented. However, in general, the concepts of hardware obfuscation should not be confused with the term "obfuscation" as is used in the context of software; this will be explained later in this chapter.

This chapter is structured as follows. In Sect. 8.2, we discuss the proposed state space obfuscation technique. In this section, we first discuss some prior work on hardware and software obfuscation. We then provide details on how to implement the proposed state space obfuscation technique by state transition graph (STG) and

netlist modification. We also provide some sample results to show the efficiency of the approach and the overheads involved. In Sect. 8.3, we discuss how the proposed state space obfuscation can help in protecting against hardware Trojans by discussing the obfuscation's impact on Trojan effectiveness (e.g., potency and detectability). In Sect. 8.4, we discuss how the technique can be extended to the RTL designs. We conclude this chapter in Sect. 8.5.

8.2 State Space Obfuscation

8.2.1 Previous Work on Obfuscation

In obfuscation-based IP protection, the IP vendor usually affects the human readability of the HDL code [20], or relies on cryptographic techniques to encrypt the source code [21]. In [20], the code is reformatted by changing the internal net names and removing the comments, so that the circuit description is no longer intelligible to the human reader. The RTL obfuscation for VHDL descriptions has been explored in [22], where the authors use rudimentary transformations of the code such as variable name changes, inlining of code, loop unrolling, and statement order changing to make the code difficult to understand. However, usually, the IP interface and port names cannot be modified or obfuscated to comply with the specifications. As the above two techniques do not modify the functionality of the IP core, they cannot prevent an IP from being stolen by an adversary and used as a "black-box" circuit module. In [21], the HDL source code is encrypted and the IP vendor provides the key to decrypt the source code only to its customers using a particular secure design platform. A similar approach has been proposed in [23], where an infrastructure for IP evaluation and delivery for FPGA applications has been proposed based on the Java applets. However, the above techniques force the use of a particular design platform, a situation that might be unacceptable to many SoC designers who seek the flexibility of multiple tools from diverse vendors in the design flow. Also, none of the above techniques prevent possible reverse engineering effort at later stages of the design and manufacturing flow (to produce IC clones or insert hardware Trojan) and thus does not benefit the other associated parties (e.g., the SoC designers and the system designers). One important point to note is that a given hardware obfuscation technique should not affect any change in the standard interface of a hardware IP module. In other words, obfuscation of a hardware IP module should have minimum impact on the workflow of an IC designer.

Software Obfuscation: It has been shown that it is possible to recover software source code by decompilation of binary or byte code. One of the most popular approaches of preventing such reverse engineering is to obfuscate the control flow of a program to produce "spaghetti code" that is difficult to decompile from the binary form to a higher level program [24–27]. Another technique is the so-called code morphing [28], where a section of the compiled code is substituted with an

entirely new block that expects the same machine state when it begins execution of the previous section, and leaves with the same machine state after execution as the original. Other software obfuscation approaches include self-modifying code [29] (code that generates other code at run-time), self-decryption of partially encrypted code at run-time [30, 31], and code redundancy and voting to produce "tamper-tolerant software" (conceptually similar to hardware redundancy for fault tolerance) [32]. A general shortcoming of these approaches is that they do not scale well in terms of memory footprint and performance as the size of the program (or the part of the program to be protected) increases [33]. Hence, the RTL obfuscation approaches motivated along similar lines are also likely to result in inefficient circuit implementations of the obfuscated RTL with unacceptable design overhead. Also, the value of such techniques is the matter of debate because it has been theoretically proven that software obfuscation in terms of obfuscating the "black-box functionality" does not exist [34]. In contrast, we modify both the structure and the functionality of the circuit description under question; hence, the above result of impossibility of obfuscation is not applicable in our case.

8.2.2 Obfuscation Through State Transition Graph Modification

We start with the description of the scheme for gate-level designs. Later, we extend it for the RTL designs.

8.2.2.1 Goals of the Obfuscation Technique

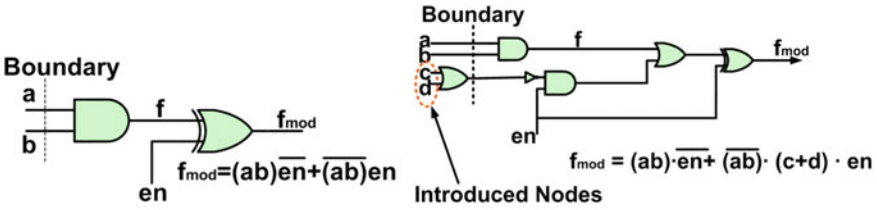
In order to achieve the comprehensive protection of hardware IPs, the proposed approach focuses on obfuscating the functionality and structure of an IP core by modifying the gate-level netlist, such that it both obfuscates the design and embeds the authentication features in it. The IC is protected from unauthorized manufacturing by the fact that the system designer depends on input from the chip designer to use the IC. Consequently, the manufacturing house cannot simply manufacture and sell unauthorized copies of an IC without the knowledge of the design house. In addition, by adopting a physically unclonable function (PUF)-based activation scheme, the security can be increased further since it ensures that the activation pattern is specific to each IC instance. The embedded authentication feature helps in proving illegal usage of an IP during litigation. Finally, the obfuscation remains transparent to the end user who has the assurance of using a product that has gone through an anti-piracy secure design flow.

8.2.3 Hardware IP Piracy: Adversary's Perspective

An adversary trying to determine the functionality of an obfuscated gate-level IP core can take resort to either (1) simulation-based reverse engineering to determine the functionality of the design or (2) structural analysis of the netlist to identify and isolate the original design from the obfuscated design. The proposed obfuscation approach targets to achieve the simulation mismatch for the maximum possible input vectors, as well as structural mismatch for maximum possible circuit nodes. To achieve the structural mismatch between the reference and the obfuscated design, we modify the state transition function as well as internal logic structure.

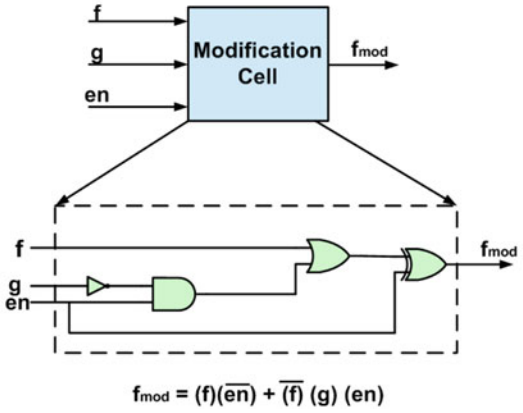
8.2.3.1 Modification by Input Logic-Cone Expansion

Consider the simple example shown in Fig. 8.1a. It shows a modified 2-input AND gate. If $en = 0$, it works as an ordinary AND gate; however, if $en = 1$, the original



(a) Simple scheme using XOR gate

(b) Scheme using expansion of logic cone



(c) Modification cell

Fig. 8.1 Schemes for boolean function modification and modification cell

functionality of the AND gate is obfuscated because the output is inverted. Simulation of the simple circuit of Fig. 8.1a against an ordinary 2-input AND gate will report 4 possible input vectors with $en = 1$ as failing patterns. To increase the number of failing patterns for this circuit, we must increase its input logic-cone size, while ensuring that it continues to function properly when $en = 0$. Figure 8.1b shows an alternative scheme, where the input logic cone has been expanded to include the nodes c and d. A complete enumeration of the truth table of the modified circuit will show failures for 13 input patterns.

The modification scheme of Fig. 8.1b can be generalized to a form shown in Fig. 8.1c. Here, f is the Boolean function corresponding to an internal node and g is any arbitrary Boolean logic function. It is worthwhile to note that the simple modification scheme of Fig. 8.1a is a special case with $g = 1$. As shown, the modified logic function is of the form:

$$f_{mod} = f \cdot \overline{en} + \overline{f} \cdot g \cdot en \tag{8.1}$$

Let us call the function g as the *Modification Kernel Function* (MKF). It is clear that for $en = 1$, if $g = 1$ for a given set of primary inputs and state element output state, $f_{mod} = \overline{f}$ and the test pattern is a failing test pattern. To increase the amount of dissimilarity between the original and the modified designs, we should try to make g evaluate to logic-1 as often as possible. At first glance, the trivial choice seems to be $g = 1$. However, in that case the input logic cone is not expanded, and thus, the number of failing vectors reported by a formal verification approach is limited. For any given set of inputs, this is achieved by a logic function which is the logical-OR of the input variables.

Selection of the Modification Kernel Function (g): Although it might be intuitive to select primary inputs or state element outputs to design the MKF (g), in practice, this could incur a lot of hardware overhead to generate the OR-functions corresponding to each modified node. An alternative approach is to select an internal logic node of the netlist to provide the Boolean function g. It should have the following characteristics:

1. The modifying node should have a very large fan-in cone, which in turn would substantially expand the logic cone of the modified node.
2. It should not be in the fan-out cone of the modified node.
3. It should not have any node in its fan-in cone which[4.] is in the fan-out cone of the modified node.

Conditions (2) and (3) are essential to prevent any *combinational loop* in the modified netlist. Such a choice of g does not, however, guarantee it to be an OR-function and is thus suboptimal.

8.2.4 System-Level Obfuscation

In this section, we present the secure SoC design methodology for hardware protection based on the analysis presented in the previous section.

8.2.4.1 State Transition Function Modification

The first step of the obfuscation procedure is the modification of the state transition function of a sequential circuit by inserting a small finite state machine (FSM). The inserted FSM has all or a subset of the primary inputs of the circuit as its inputs (including the clock and reset signals) and has multiple outputs. At the start of operations, the FSM is reset to its initial state, forcing the circuit to be in the *obfuscated* mode. Depending on the applied input sequence, the FSM then goes through a state transition sequence and only on receiving N specific input patterns in sequence, and goes to a state which lets the circuit operate in its *normal* mode. The initial state and the states reached by the FSM before a successful initialization constitute the "pre-initialization state space" of the FSM, while those reached after the circuit has entered its normal mode of operation constitute the "post-initialization state space". Figure 8.2 shows the state diagram of such a FSM, with $P0 \rightarrow P1 \rightarrow P2$ being the correct initialization sequence. The input sequence $P0$ through $P2$ is decided by the IP designer.

The FSM controls the mode of circuit operation. It also modifies selected nodes in the design using its outputs and the *modification cell* (e.g., M_1 through M_3). This scheme is shown in Fig. 8.2 for a gate-level design that incorporates the modifications of three nodes n_1 through n_3. The MKF can either be a high fan-in internal node (avoiding combinational loops) in the unmodified design, or the OR-function of several selected primary inputs. The other input (corresponding to the *en* port of the modification cell) is a Boolean function of the inserted FSM state bits with the constraint that it is at logic-0 in the *normal* mode. This modification ensures that when the FSM output is at logic-0, the logic values at the modified nodes are the same as the original ones. On the other hand, in the *obfuscated* mode, for any FSM output that is at logic-1, the logic values at the modified nodes are inverted if $g = 1$ and logic-0 if $g = 0$. Provided the modified nodes are selected judiciously, modifications at even a small number of nodes can greatly affect the behavior of the modified system. This happens even if the *en* signal is not always at logic-0. In our implementation, we chose to have the number of outputs of the inserted FSM as a user-specified parameter. These outputs are generated as random Boolean functions of the state element bits at design time with the added constraint that in the *normal* mode, they are at logic-0. The randomness of the Boolean functions adds to the security of the scheme. Such a node modification scheme can provide higher resistance to structural reverse engineering efforts than the scheme in [35].

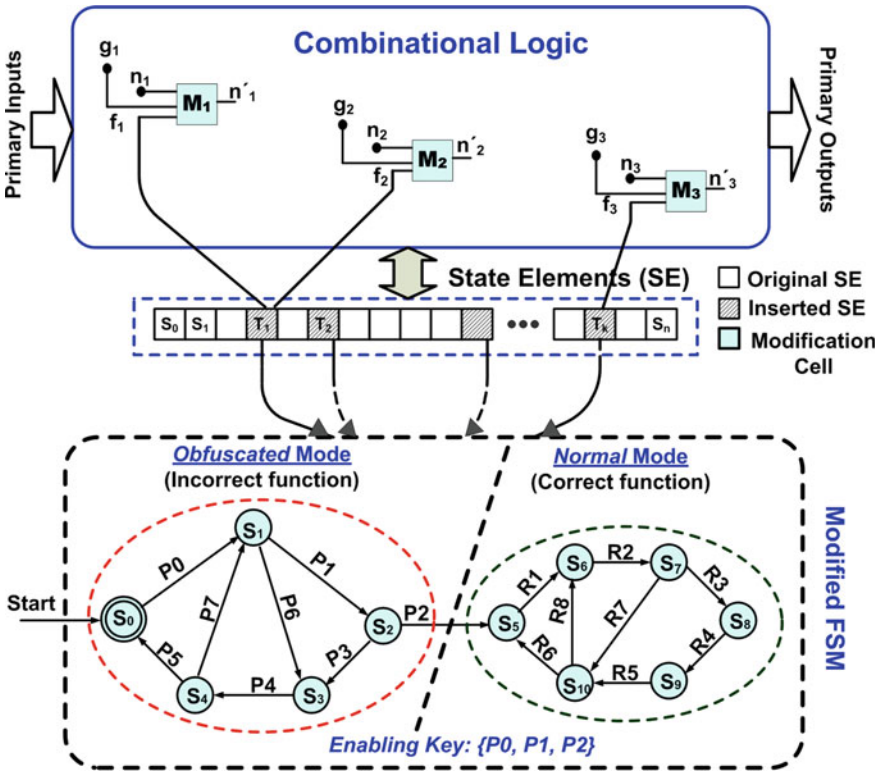


Fig. 8.2 The proposed functional and structural obfuscation scheme by modification of the state transition function and internal node structure

8.2.5 Embedding Authentication Features

The proposed obfuscation scheme allows us to easily embed authentication signature into a gate-level design with negligible design overhead. Such an embedded signature acts as a digital watermark and hence helps to prevent the attacks from trusted parties in the design flow with knowledge of the initialization sequence. Corresponding to each state in the *pre-initialization state space*, we arrange to have a particular pattern to appear at a subset of the primary outputs when a predefined input sequence is applied. Even if a hacker arranges to bypass the initialization stage by the structural modifications, the inserted FSM can be controlled to have the desired bit-patterns corresponding to the states in the *pre-initialization state space*, thus revealing the watermark. For post-silicon authentication, scan flip-flops can be used to bring the design to the obfuscated mode. Because of the prevalent widespread use of full-scan designs, the inserted FSM flip-flops can always be controlled to have the desired bit-patterns corresponding to the states in the *authentication FSM*, thus revealing the

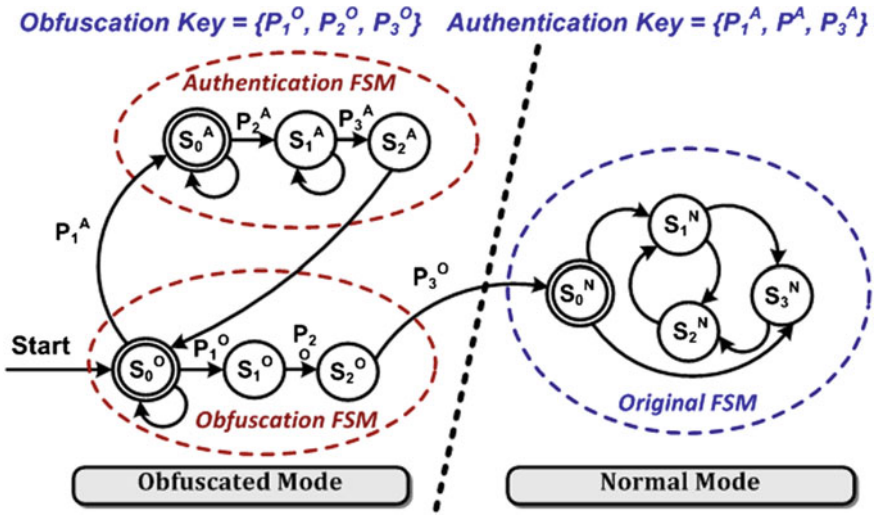


Fig. 8.3 Modification of the initialization state space to embed authentication signature

watermark. Figure 8.3 illustrates the modification of the state transition function for embedding authentication signature in the *obfuscated mode* of operation.

To mask or disable the embedded signature, a hacker needs to perform the following steps, assuming a purely random approach:

1. Choose the correct inserted FSM state elements (n_p) from all the total state elements (n_t). This has $\binom{n_t}{n_p}$ possible choices.
2. Apply the correct input vector at the n_i input ports where the vectors are to be applied to get the signature at the selected n_o output ports. This is one out of 2^{n_i} choices.
3. Choose the n_o primary outputs at which the signature appears from the total set of primary outputs (n_{po}). This has $\binom{n_{po}}{n_o}$ possibilities.
4. For each of these recognized n_o outputs, identify it to be one among the possible $2^{2^{(n_i+n_p)}}$ Boolean functions (in the *obfuscated* mode) of the n_i primary inputs and n_p state elements, and change it without changing the normal functionality of the IP.

Hence, in order to mask one signature, the attacker has to make exactly one correct choice from among $N = \binom{n_t}{n_p} \cdot 2^{n_i} \cdot \binom{n_{po}}{n_o} \cdot 2^{2^{(n_i+n_p)}}$ possible choices, resulting in a masking success probability of $P_{masking} \cong \frac{1}{N}$. To appreciate the scale of the challenge, consider a case with $n_t = 30$, $n_p = 3$, $n_i = 4$, $n_o = 4$, and $n_{po} = 16$. Then, $P_{masking} \sim 10^{-47}$. In actual IPs, the masking probability would be substantially lower because of higher values of n_p and n_t.

8.2.6 Choice of Optimal Set of Nodes for Modification

To obfuscate a design, we need to choose an optimal set of nodes to be modified, so that maximum obfuscation is achieved under the given constraints. We estimate the level of obfuscation by the amount of verification mismatch reported by a formal verification-based equivalence checker tool. Formal equivalence checker tools essentially try to match the input logic cones at the state elements and the primary outputs of the reference and the implementation [36]. Hence, nodes with larger fan-out logic cone would be preferred for modification since that will in turn affect the input logic of comparatively larger number of nodes. Also, large input logic cone of a node is generally indicative of its higher *logic depth*; hence, any change at such a node is likely to alter a large number of primary outputs. Thus, in determining the suitability metric for a node as a candidate for modification, we need to consider both these factors. We propose the following metric to be used as the *suitability metric* for a node:

$$M_{node} = \left(\frac{w_1 \cdot FO}{FO_{max}} + \frac{w_2 \cdot FI}{FI_{max}}\right) \times \frac{FO \cdot FI}{FI_{max} \cdot FO_{max}} \tag{8.2}$$

where FI and FO are the number of nodes in the fan-in and the fan-out cone of the node, respectively. FI_{max} and FO_{max} are the maximum number of fan-in and fan-out nodes in the circuit netlist and are used to normalize the metric. w_1 and w_2 are weights assigned to the two factors, with $0 \leq w_1, w_2 \leq 1$, and $w_1 + w_2 = 1$. We chose $w_1 = w_2 = 0.5$ which gives us the best results (verified experimentally). Note that $0 < M_{node} \leq 1$. Because of the widely differing values of FO_{max} and FI_{max} in some circuits, it is important to consider both the sum and the product terms involving $\frac{FO}{FO_{max}}$ and $\frac{FI}{FI_{max}}$. Considering only the sum or the product term results in an inferior metric that fails to capture the actual suitability of a node, as observed in our simulations.

8.2.7 Obfuscation-Based Design Methodology

The overall hardware obfuscation design flow is shown in Fig. 8.4. First, from the synthesized gate-level HDL netlist of an IP core, the fan-in and fan-out cones of the nodes are obtained. Then, an iterative ranking algorithm is applied to find the most suitable N_{max} modifiable nodes, where N_{max} is the maximum number of nodes that can be modified within the allowable overhead constraints. The ranking is a multipass algorithm, with the metric for each node being dynamically modified based on the selection of the node in the last iteration. The algorithm takes into account the overlap of the fan-out cones of the nodes which have been already selected and eliminates them from the fan-out cones of the remaining nodes. On the completion of each iteration, the top ranking node among the remaining nodes is selected, so that selection of N_{max} nodes would take N_{max} iterations. In this way, as the iterations

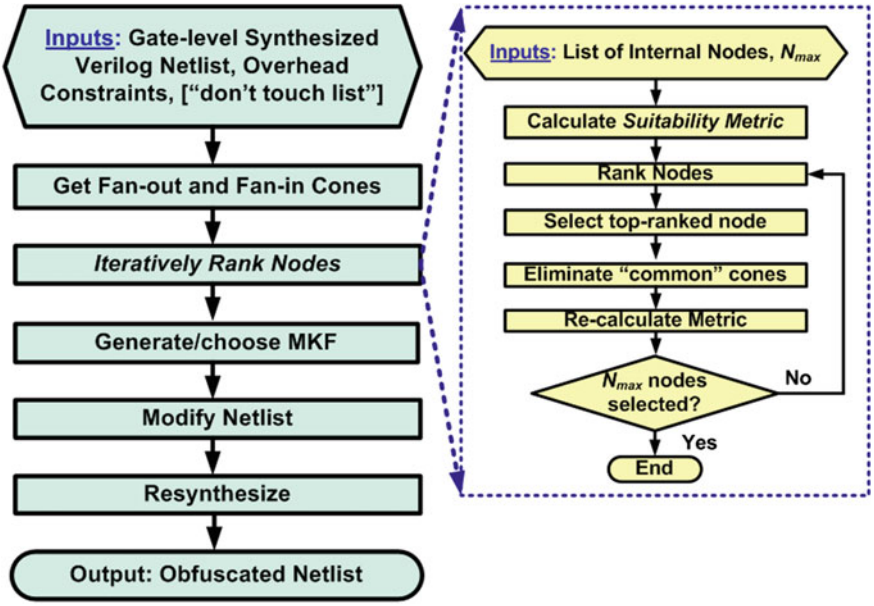


Fig. 8.4 Hardware obfuscation design flow along with steps of the iterative node ranking algorithm

progress, the nodes with more non-overlapping fan-out cones are assigned higher weight.

A "*don't touch*" list of nodes can be optionally input to the tool direct it not to modify certain nodes, e.g., nodes which fall in the critical path. In large benchmarks, we observed that there were sufficient nodes with high fan-outs, such that skipping a few "*don't touch*" nodes still maintains the effectiveness of node modification algorithm in achieving functional and structural obfuscation. For each node to be modified, proper MKF (g) is selected either on the basis of its fan-in cone size, or by OR-ing several primary inputs which were originally not present in its input logic cone. The FSM is then integrated with the gate-level netlist, and the selected nodes are modified. The modified design is resynthesized and flattened to generate a new gate-level netlist. The integrated FSM and the modification cells are no longer visually identifiable in the resultant netlist. This resynthesis is performed under timing constraint, so that it maintains circuit performance.

The IP vendor applies the hardware obfuscation scheme to create a modified IP and supplies it to the design house, along with the activating sequences. The design house receives one or multiple IPs from the IP vendors and integrates them on chip. To activate the different IPs, the designer needs to include a low-overhead *controller* in the SoC. This controller module can perform the initialization of the different IP blocks in two different ways. In the first approach, it serially steers the different initialization sequences to the different IP blocks from the primary inputs. This controller module will include an integrated FSM which determines the steering

of the correct input sequences to a specific IP block. Multiplexors are used to steer initialization sequences to the IP blocks, or the primary inputs and internal signals during normal operation. The chip designer must modify the test benches accordingly to perform block-level or chip-level logic simulations.

In the second approach, the initialization sequences is stored permanently on chip in a ROM. In the beginning of operations, the controller module simply reads the different input sequences in parallel and sends them to the different IP blocks for initialization. The advantage of this approach is that the number of initialization cycles can be limited. However, additional overhead is incurred for storing the input sequences in an on-chip ROM. To increase the security of the scheme, the chip designer can arrange an instance-specific initialization sequence to be stored in an one-time programmable ROM. In that case, following the approach in [37], we can have the activating patterns to be simple logic function (e.g., XOR) of the patterns read from the ROM and the output of a *physically unclonable function* (PUF) block. The patterns are written to the ROM post-manufacturing after receiving instructions from the chip designer, as suggested in [37]. Because the output of a PUF circuit is not predictable before manufacturing, it is not possible to have the same bits written into the programmable ROMs for each IC instance. Figure 8.5 shows this scheme.

The manufacturing house manufactures the SoC from the design provided by the design house and passes it on to the test facility. If a PUF block has been used in the IC, the test engineer reports the output on the application of certain vectors back to the chip designer. The chip designer then calculates the specific bits required to be written in the one-time programmable ROM. The test engineer does so and blows off an one-time programmable fuse, so that the output of the PUF block is no longer

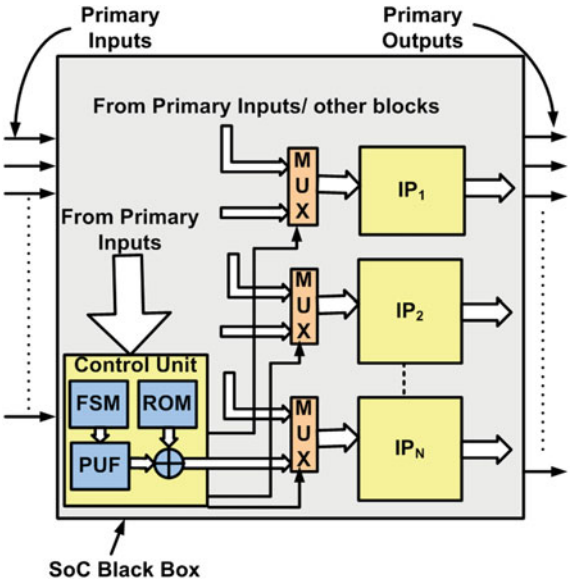


Fig. 8.5 SoC design modification to support hardware obfuscation. An on-chip controller combines the input patterns with the output of a PUF block to produce the activation patterns

visible at the output. The test engineer then performs post-manufacturing testing, using the set of test vectors provided by the design house. Ideally, all communication between parties associated with the design flow should be carried out in an encrypted form, using cryptographic protocols such as *Diffie–Hellman Key Exchange* [7]. The tested ICs are passed to the system designer along with the initialization sequence (again in an encrypted form) from the design house.

The system designer integrates the different ICs in the board-level design and arranges to apply the initialization patterns during "booting" or similar other initialization phase. Thus, the initialization patterns for the different SoCs need to be stored in read only memory (ROM). In most ASICs composed of multiple IPs, several initialization cycles are typically needed at start-up to get into the "steady-stream" state, which requires accomplishing certain tasks such as initialization of specific registers [38]. The system designer can easily utilize this inherent latency to hide the additional cycles due to initialization sequences from the end user.

Finally, this secure system is used in the product for which it is meant. It provides the end user with the assurance that the components have gone through a secure and piracy-proof design flow. Figure 8.6 shows the challenges and benefits of the design flow from the perspectives of different parties associated with the flow. It is worth noting that the proposed design methodology remains valid for a SoC design house that uses custom logic blocks instead of reusable IPs. In this case, the designer can synthesize the constituent logic blocks using the proposed obfuscation methodology for protecting the SoC.

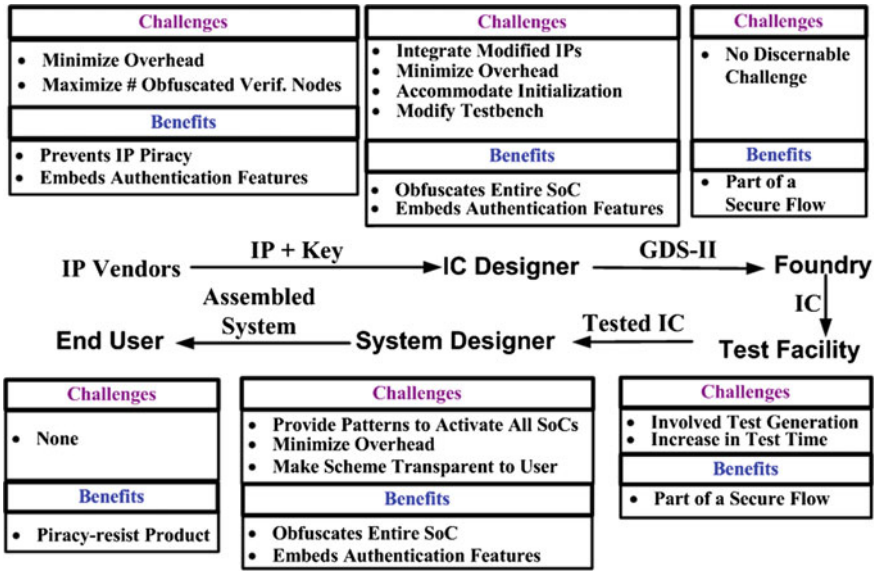


Fig. 8.6 Challenges and benefits of the *HARPOON* design methodology at different stages of a hardware IP life cycle

8.2.8 Results

In this section, we present simulation results to show the effectiveness of the proposed hardware obfuscation methodology for a set of ISCAS-89 benchmark circuits [39].

8.2.8.1 Simulation Setup

All circuits were synthesized using Synopsys *Design Compiler* with optimization parameters set for minimum area and mapped to a LEDA 250 nm standard cell library. The flow was developed using the TCL scripting language and was directly integrated in the *Design Compiler* environment. All formal verification was carried out using Synopsys *Formality*. The verification nodes considered by Formality constituted of the inputs of state elements (e.g., flip-flops) and primary outputs.

8.2.8.2 Effect of Obfuscation on ISCAS-89 Benchmark Circuits

A simple four-state FSM was designed for each of the benchmarks to achieve hardware and functional obfuscation. For each of the benchmarks, random internal nodes were selected while making sure that combinational loops were avoided. The benchmarks were then subjected to the hardware obfuscation design flow (including the iterative ranking algorithm). The modified and resynthesized benchmarks were then subjected to formal verification using Synopsys *Formality*. Figure 8.7 shows the observed percentage of verification nodes failing verification reported by *Formality* for the different benchmark circuits.

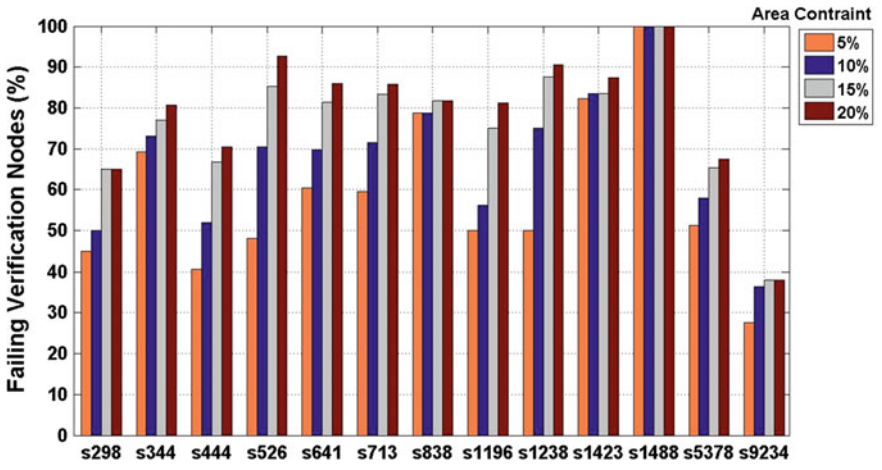


Fig. 8.7 Observed verification failures (with application of the *HARPOON* methodology) for ISCAS-89 circuits

Table 8.1 Design overheads (%) for different area constraints

Benchmark circuit	5% area constraint			10% area constraint			15% area constraint			20% area constraint		
	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power	Area	Delay	Power
s298	3.91	0.00	5.26	8.64	0.00	14.04	14.74	0.50	16.67	19.29	0.00	18.42
s344	3.45	−2.98	5.38	8.83	−8.96	9.87	14.89	−6.20	14.35	18.59	−7.90	18.83
s444	4.63	0.00	9.62	9.50	0.00	18.27	14.15	0.00	20.19	18.26	0.00	23.08
s526	3.64	0.00	7.06	8.12	0.00	16.17	14.89	−1.60	21.87	19.30	−0.78	25.28
s641	4.96	−3.66	8.20	9.74	−3.66	13.90	14.02	−4.60	19.59	19.63	−4.20	23.01
s713	4.06	−3.66	7.34	9.07	−3.66	14.69	14.43	−2.60	20.34	19.26	−2.60	22.88
s838	2.20	−2.39	6.92	9.00	−8.57	9.76	14.17	−5.50	12.93	19.13	0.00	14.00
s1196	3.96	0.00	6.04	6.06	0.00	16.09	14.45	−1.76	19.14	18.10	0.00	20.55
s1238	4.52	−0.42	6.52	9.99	−0.42	9.97	14.87	0.00	18.26	18.23	−0.90	23.79
s1423	4.70	−0.78	8.02	9.61	−2.64	14.91	14.97	−1.08	22.43	19.99	−2.42	26.19
s1488	3.27	−2.79	3.13	8.65	−0.93	8.33	13.19	0.00	10.49	18.17	0.00	13.62
s5378	4.34	0.00	8.91	9.87	0.00	13.80	13.84	0.00	20.43	19.93	0.00	23.70
s9234	4.74	0.00	5.80	8.82	3.60	12.37	8.82	3.50	15.52	14.29	3.80	19.72
Average	4.03	−1.28	8.83	8.92	−1.94	13.31	14.38	−1.49	17.81	19.06	−1.15	20.91

Table 8.1 shows the area, delay, and power overheads of the resynthesized benchmark circuits, following the application of our obfuscation scheme, for the projected area overheads of 5, 10, 15, and 20%, respectively. From the table, it is clear that the actual area overheads were smaller than the imposed constraints in all cases, while the timing overhead was negative; i.e., the timing constraint was met with positive slack in most cases. The power overheads in all cases were within acceptable limits. The design overhead is caused by the addition of both combinational (in the form of the *modification cells*) and few sequential elements to implement the inserted FSM. Although sequential memory elements do not scale as well as combinational ones, the percentage overhead is expected to remain unchanged for more advanced technology nodes because the combinational overhead forms the major fraction of the total overhead.

8.3 State Space Obfuscation for Protection Against Hardware Trojans

We now apply the concept of space obfuscation to ruggedize circuit netlists, such that an adversary finds it difficult to insert hard-to-detect combinational HTH instances. As before, the normal circuit functionality is achieved by the application of a sequence of secret "initialization keys" at the circuit output. Also, the state space obfuscation is achieved without the explicit enumeration of the sequential circuit state space.

8.3.1 Trojan Variants

Different methods of classifying hardware Trojans based on various characteristics have been proposed. In [40], the authors propose a simple classification of Trojans—*combinational* (whose activation depends on the occurrence of a particular condition at certain internal nodes of the circuit) and *sequential* (whose activation depends on the occurrence of a specific sequence of rare logic values at internal nodes). Although there are other ways of classifying hardware Trojans (based on the activation mechanism, threat model, etc.), we limit our discussion and experiments to combinational and sequential Trojans.

Figure 8.8a shows an example of a *combinationally triggered* Trojan where the occurrence of the condition $a=0$, $b=c=1$ at the trigger nodes a, b, and c, respectively, causes a payload node S to have an incorrect value at $S^{\star}$. Typically, an adversary would choose an extremely rare activation condition so that it is very unlikely for the Trojan to trigger during conventional manufacturing test.

Sequentially triggered Trojans (the so-called time bombs), on the other hand, are activated by the occurrence of a sequence or a period of continuous operation. The simplest sequential Trojans are counters, which trigger a malfunction on reaching

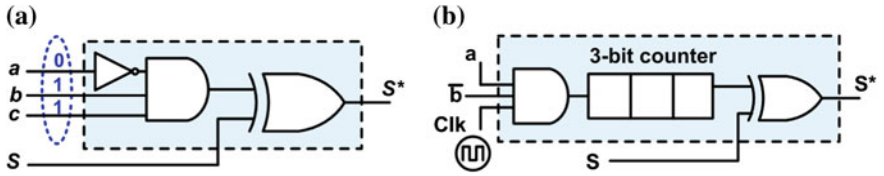


Fig. 8.8 Examples of **a** combinational and **b** sequential hardware Trojans that cause malfunction conditionally

a particular count. Figure 8.8b shows a 3-bit counter which increases its count if $a = 0$, $b = 1$ at the positive clock edges. The Trojan activates when the count reaches 7, by modifying the node S to have an incorrect value at node $S^{\star}$.

The obfuscation in our scheme is achieved by two important modifications of the *state transition graph* (STG) of the circuit:

- The size of the reachable state space is "blown up" by a large (exponential) factor using extra state elements.
- Certain states, which were *unreachable* in the original design, are used and made reachable only in the *obfuscated mode* of operation.

These two modifications make it difficult for an adversary to design a functionally potent and well-hidden Trojan, as shown through the analysis presented in Sects. 8.3.2, 8.3.3, and 8.3.4. As would be evident from the following, this modification of the STG is different in methodology and goals from the ones proposed in [6, 7, 37, 41], in which no state space explosion is attempted.

Figure 8.9a shows the proposed obfuscation scheme based on the change in the STG of the circuit. On power-up, the circuit is initialized to a state (S_0^O) in the *obfuscated mode*. On the application of an input sequence $K_1 \rightarrow K_2 \rightarrow K_3$ in order, i.e., the *initialization key sequence*, the circuit reaches the state S_0^N, which is the *reset state* in the *original state space*, allowing *normal mode* of operation. The states S_0^O, S_1^O, and S_2^O constitute the *initialization state space*. The application of even a single incorrect input vector during the initialization process takes the circuit to states in the *isolation state space*, a set of states from which it is not possible to come back to the *initialization state space* or enter the *original state space*. The *initialization state space* and the *isolation state space* together constitute the *obfuscation state space*. All state encodings in the *obfuscation state space* are done using unreachable state bit combinations for selected state elements of the circuit. This ensures that the circuit cannot perform its normal functionality until the correct initialization key sequence has been applied. As for the hardware IP obfuscation scheme, this initialization latency (typically < 10 clock cycles) can be easily hidden from the end user by utilizing the inherent latency of most ASICs during a "boot-up" or similar procedure on power-ON. We consider the post-manufacturing testing phase to be "trusted" such that there is no possibility of the secret initialization key to be leaked to the adversary in the fab. This is a commonly accepted convention which was first explicitly stated in [12]. To protect against the possibility of an user releasing

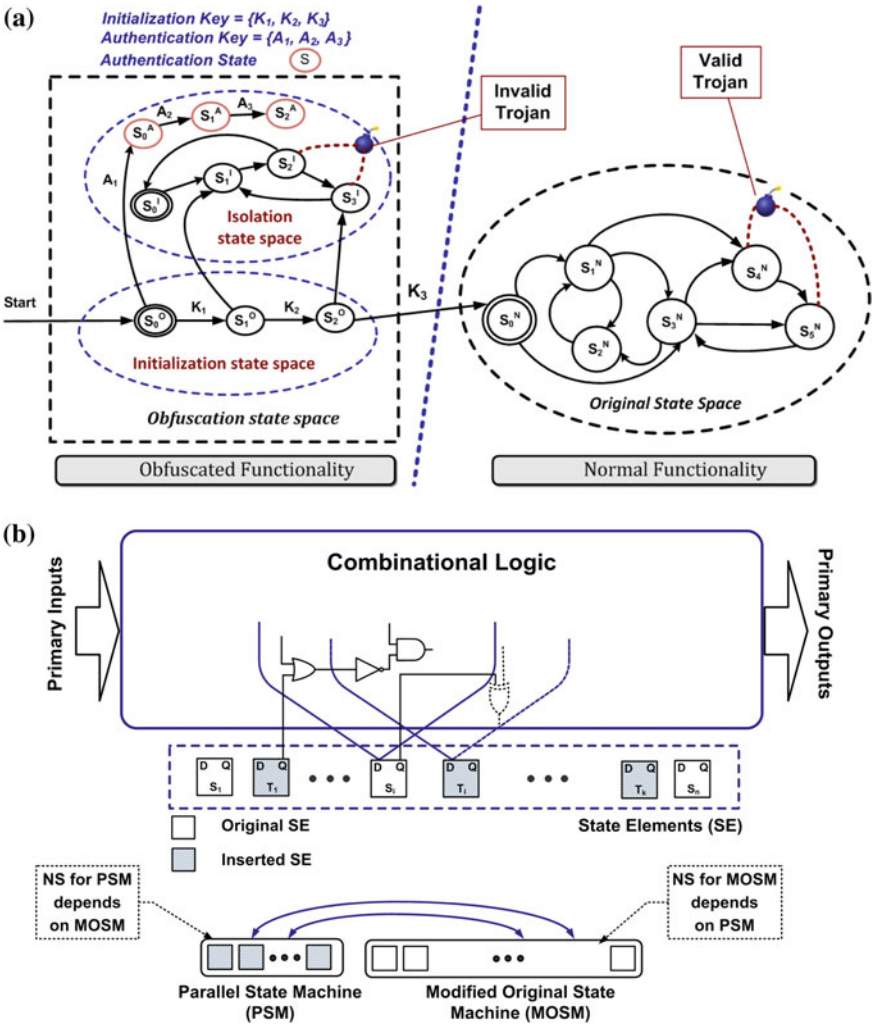


Fig. 8.9 The obfuscation scheme for protection against hardware Trojans: **a** modified state transition graph and **b** modified circuit structure

the initialization key sequence of the design in the public domain, user-specific initialization key sequence or in the extreme case instance-specific initialization key sequence might be employed.

To "blow up" the size of the obfuscation state space, a number of extra state elements are added depending on the allowable hardware overhead. The size of the obfuscation state space has an exponential dependence on the number of extra state elements. An inserted *parallel finite state machine* (PSM) defines the state transitions of the extra state elements. However, to hide possible structural signature formed by

the inserted PSM, the circuit description of the PSM is *folded* into the *modified state machine in the original circuit* (MOSM) (as shown in Fig. 8.9b) to generate an integrated state machine. A logic resynthesis step is performed, including logic optimization under design constraints in terms of delay, area, or power. In effect, the circuit structures such as the input logic cones of the original state elements change significantly compared to the unobfuscated circuit, making reverse engineering of the obfuscated design practically infeasible for a adversary. This effect is illustrated in Sect. 8.3.5 through an example benchmark circuit.

To increase the level of structural difference between the obfuscated and the original circuits, the designer can choose to insert *modification cells* as proposed earlier in the chapter at selected internal nodes. Furthermore, the level of obfuscation can be increased by using more states in the obfuscated state space. This can be achieved by the following: (1) adding more state elements to the design and/or (2) using more unreachable states from the original design. However, this can increase the design overhead substantially. In Sect. 8.3.9, we describe a technique to reduce the design overhead in such cases.

Selected states in the isolation state space can also serve the purpose of authenticating the ownership of the design, as described in [6]. Authentication for sequential circuits is usually performed by embedding a digital watermark in the STG of the design [42, 43], and our idea of hiding such information in the unused states of the circuit is similar to [44]. Figure 8.9 shows such a scheme where the states S_0^A, S_1^A, and S_2^A in the isolation state space and the corresponding output values of the circuit are used for the purposes of authenticating the design. The design goes through the state transition sequence $S_0^O \rightarrow S_0^A \rightarrow S_1^A \rightarrow S_2^A$ on the application of the sequence $A_1 \rightarrow A_2 \rightarrow A_3$. Because these states are unreachable in the normal mode of operation, they and the corresponding circuit output values constitute a property that was not part of the original design. As shown in [6], the probability of determining such an embedded watermark and masking it is extremely small, thus establishing it as a robust watermarking scheme.

8.3.2 *Effect of Obfuscation on Trojan Insertion*

As mentioned before, to design a functionally catastrophic but hard-to-detect Trojan, the adversary would try to select a "rare" event at selected internal "trigger nodes" to activate the Trojan. To select a sufficiently rare trigger condition for the Trojan to be inserted, the adversary would try to estimate the signal probability [45] at the circuit nodes by simulations with random vectors (and random starting states). However, the adversary has no way to know whether the starting state of the simulations is in the normal state space or the obfuscation state space. If the initial state of the simulations lie in the obfuscation state space, there is a high probability that the simulations would remain confined in the obfuscation state space. This is because the random test generation algorithm of the adversary most likely would be unable to apply the correct input vector at the correct state to cause the state transition

to the normal state space. Essentially, the STG of the obfuscated circuit has two *near-closed* (NC) set of states [46], which would make accurate estimation of the signal probabilities through a series of random simulations extremely challenging. An algorithm was proposed in [46] to detect the *NC sets* of a sequential circuit; however, the algorithm requires the following: (a) knowledge of the state transition matrix of the entire sequential circuit, which is not available to the adversary, and (b) a list of all the reachable states of the circuit, which is extremely computationally challenging to enumerate for a reasonably large sequential circuit. Hence, we can assume that the adversary would be compelled to resort to a random simulation-based method to estimate the signal probabilities at internal circuit nodes.

8.3.3 *Effect of Obfuscation on Trojan Potency*

To decrease the *potency* of the inserted Trojan, the designer of the obfuscated circuit must ensure that if the adversary starts simulating the circuit in the obfuscation state space, the probability of the circuit being driven to the normal state space is minimal. Consider a sequential circuit originally with N state elements and M *used states*, to which n state elements are added to modify the STG to the form shown in Fig. 8.9a. Let the number of states in the obfuscation state space be $S_i = f_1 \cdot 2^n \cdot \left(2^N - M\right) = f_1 \cdot 2^n \cdot B$, where $B = \left(2^N - M\right)$, and $f_1 < 1$ represents a *utilization factor* reflecting the overhead constraint.

Let I denote the set of states in the obfuscation state space, U denote the set of states in the normal state space, and T denote the set of states actually attained during the simulations by the adversary. Let p be the number of primary inputs (other than the clock and reset) where the initialization key sequence is applied, and let the length of the initialization key sequence be k. Then, it takes k correct input vectors in sequence to reach the normal state space from state S_0^O, $k-1$ correct input vectors from state S_1^O, and so on. Then, the probability that the simulation started in the initialization state space was able to reach the normal state space by the application of random input vectors:

$$P\left(T \subseteq \left\{I \bigcup U\right\}\right) = \frac{k}{S_i + M} \cdot \left(\frac{1}{2^p} + \frac{1}{2^{2p}} + \cdots \frac{1}{2^{pk}}\right) \tag{8.3}$$

$$\approx \frac{k \cdot 2^{-p}}{\left(f_1 \cdot 2^n \cdot B + M\right)\left(1 - 2^{-p}\right)} \tag{8.4}$$

assuming $2^{-pk} \ll 1$. The probability that the simulations started in the *normal state space* and remained confined there is:

$$P\left(T \subseteq U\right) = \frac{M}{f_1 \cdot 2^n \cdot B + M} \tag{8.5}$$

To maximize the probability of keeping the simulations confined in the obfuscation state space, the designer should ensure:

$$P\left(T \subseteq \left\{I \bigcup U^{'}\right\}\right) \gg P\left(T \subseteq U\right) + P\left(T \subseteq \left\{I \bigcup U\right\}\right) \tag{8.6}$$

Approximating $M \gg \frac{k \cdot 2^{-p}}{1-2^{-p}}$, and simplifying, this leads to:

$$f_1 \cdot 2^n \cdot B \gg M \tag{8.7}$$

This equation essentially implies the size of the obfuscation state space should be much larger compared to the size of the normal state space, a result that is intuitively expected. From this analysis, the two main observations are:

- The size of the obfuscation state space has an exponential dependence of the number of extra state elements added.
- In a circuit where the size of the used state space is small compared to the size of the unused state space, higher levels of obfuscation can be achieved at lower hardware overhead.

As an example, consider the ISCAS-89 benchmark circuit $s1423$ with 74 state elements (i.e., $N = 74$) and $> 2^{72}$ unused states (i.e., $2^N - M > 2^{72}$) [47]. Then, $M < 1.42 \times 10^{22}$, and considering 10 extra state elements added (i.e., $n = 10$), $f_1 > 0.0029$ for Eq. (8.7) to hold. Thus, expanding the state space in the modified circuit by about 3% of the available unused state space is sufficient in this case.

8.3.4 Effect of Obfuscation on Trojan Detectability

Consider a Trojan designed and inserted by the adversary with q trigger nodes, with *estimated* rare signal probabilities $p_1, p_2, \ldots p_q$, obtained by simulating the obfuscated circuit. Let the rare logic value probabilities of these internal nodes be $p_i + \Delta p_i$, for the ith trigger node. Then, the Trojan would be activated once (on average) by:

$$N^{'} = \frac{1}{\prod_{i=1}^{q}(p_i + \Delta p_i)} = \frac{N}{\prod_{i=1}^{q}(1 + \frac{\Delta p_i}{p_i})} \tag{8.8}$$

test vectors. The difference between the estimated and the actual number of test vectors before the Trojan is activated is $\Delta N = N - N^{'}$, which leads to a percentage normalized difference (assuming $\frac{\Delta p_i}{p_i} = f \quad \forall i = 1, 2 \ldots q$):

$$\frac{\Delta N}{N}(\%) = \left(1 - \frac{1}{(1+f)^q}\right) \times 100\% \tag{8.9}$$

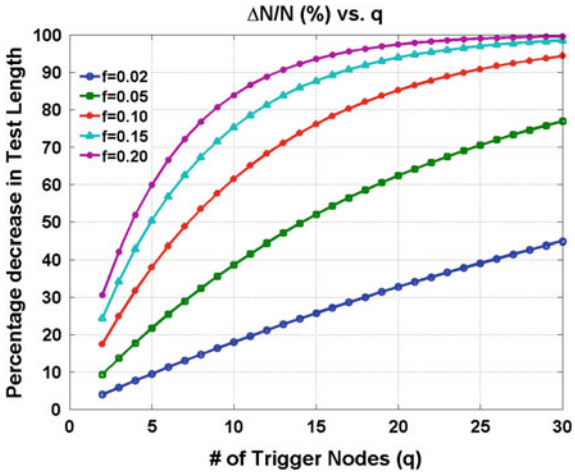


Fig. 8.10 Fractional change in average number of test vectors required to trigger a Trojan, for different values of average fractional mis-estimation of signal probability f and Trojan trigger nodes (q)

Figure 8.10 shows this fractional change plotted versus the number of trigger nodes (q) for different values of the fractional mis-estimation of the signal probability (f). From this plot and Eq. (8.9), it is evident that:

- The probability of the Trojan getting detected by logic testing increases as the number of Trojan trigger nodes (q) increases. However, it is unlikely that the adversary will have more than 10 trigger nodes, because otherwise as shown by our simulations, it becomes extremely difficult to trigger the Trojans at all.
- For values $2 \leq q \leq 10$, the number of random input patterns required to activate the trojan decreases sharply with q. The improvement is more pronounced at higher values of f. This observation validates the rationale behind an obfuscation-based design approach that resists the adversary from correctly estimating the signal probabilities at the internal circuit nodes.

8.3.5 *Effect of Obfuscation on Circuit Structure*

The resynthesis of the circuit after its integration with the RTL description corresponding to the obfuscation state space flattens the circuit into a single netlist, resulting in drastic changes to the input logic cones of the primary outputs and the state elements. This makes it infeasible to manually or automatically analyze and identify the modifications made to a practical circuit with reasonably large number of gates, even if the adversary is in possession of an unmodified version of the original circuit netlist. If the adversary is not in possession of an unmodified *reference* gate-level design, this task is even more difficult, as the adversary would have no idea about the netlist structure of the original design. The theoretical complexity of reverse engineering similar key-based obfuscation schemes for circuits has been ana-

lyzed in [48, 49], where such systems were shown to be "provably secure" because of the high computational complexity.

8.3.6 Determination of Unreachable States

The construction of the obfuscation state space requires the determination of *unreachable states* in a given circuit. First, for a set of S state elements in a given circuit, all the possible 2^S state bit combinations are generated for the S state elements. Then, each state of these 2^S states are subjected to *full sequential justification* at the inputs of the selected S state elements, using *Synopsys Tetramax*. The justified states are discarded, while the states which fail justification are collected to form the set $U = \{U_1, \ldots, U_N\}$ of structurally unreachable states.

8.3.7 Determination of Effectiveness

To determine the decrease in *potency* of the Trojans by the proposed scheme, we reduce a given vector set to eliminate those vectors with state values in the *obfuscation state space*. We then resimulate the circuit with the reduced test set to determine the Trojan coverage. The decrease in the Trojan coverage obtained from the reduced test set indicates the Trojans which are activated or effective only in the obfuscation state space and, hence, become benign.

To determine the increase in *detectability* of the Trojans, we compare the S_p values at the Trojan trigger nodes between two cases: (1) a large set of random vectors and (2) a modified set of random vectors which ensure operation of the obfuscated design in only *normal mode*. The increase in Trojan detectability is estimated by the percentage of circuit nodes for which the S_p values differ by a predefined threshold. The difference in estimated S_p prevents an adversary from exploiting the true rare events at the internal circuit nodes in order to design a hard-to-detect Trojan. On the other hand, true non-rare nodes may appear as rare in the obfuscated design, which potentially serve as *decoy* to the adversary. The above two effects are summed up by the increase in Trojan detection coverage due to the obfuscation. The coverage increase is estimated by comparing the respective coverage values obtained for the obfuscated and the original design for the same number of test patterns.

Figure 8.11 shows the steps to estimate the effectiveness of the obfuscation scheme [50]. In the figure, *RO-Finder* (**Rare O**ccurrence **Finder**), *ReTro* (**Re**duced pattern generator for **Tro**jans), and *TrojanSim* (**Trojan Sim**ulator) refer to the three C routines we developed to compute the internal signal probabilities, generate a reduced test set that can trigger rare conditions multiple times, and perform a cycle-accurate simulation of the circuit. Note that we assumed the Trojan model shown in Fig. 8.8a.

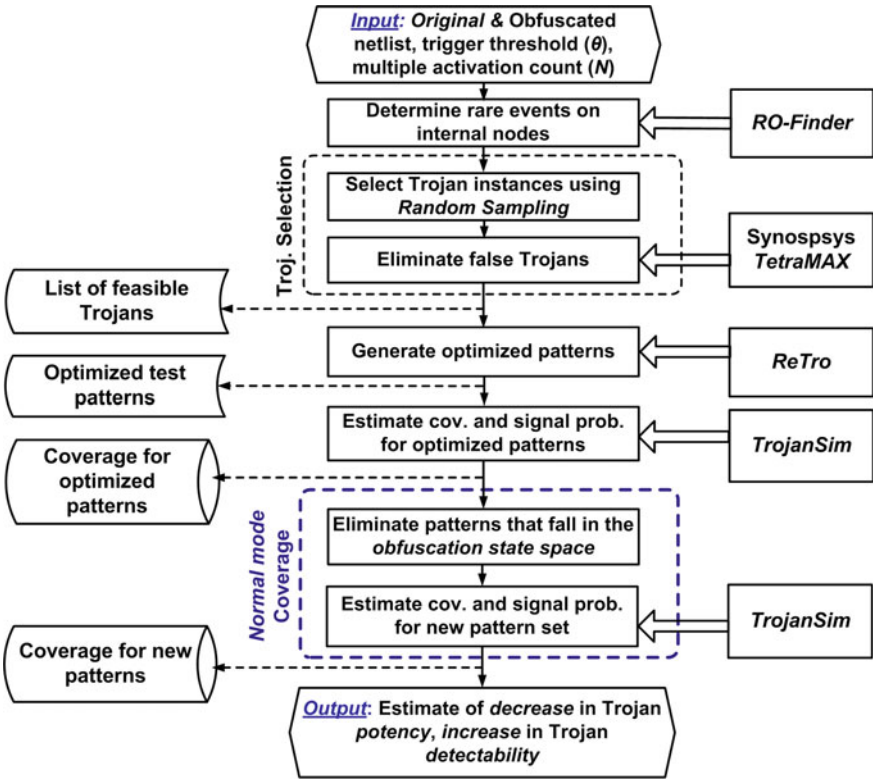


Fig. 8.11 Framework to estimate the effectiveness of the obfuscation scheme

8.3.8 Results

Tables 8.2 and 8.3 show the effects of obfuscation on increasing the security against hardware Trojans for a set of ISCAS-89 benchmark circuits with 20,000 random instances of suspected Trojans, trigger threshold (θ) of 0.2, and trigger nodes (q) 2 and 4, respectively. Optimized vector set was generated using N=1000. The same value of $n + S$ applies to both sets of results. The length of the initialization key sequence was 4 ($k = 4$) for all the benchmarks. The effect of obfuscation was estimated by three metrics: (a) the fraction of the total population of structurally justifiable Trojans becoming benign; (b) the difference between the signal probabilities at internal nodes of the obfuscated and original circuit; and (c) the improvement in the *functional Trojan coverage*, i.e., the increase in the percentage of valid Trojans detected by logic testing. Note that the number of structurally justifiable Trojans (as determined by *TetraMax*) decreases with the increase in the number of trigger nodes of the Trojan and increase in the size of the benchmark circuits. From the tables, it is evident that the construction of the obfuscation state space with even a relatively small number

Table 8.2 Effect of obfuscation on security against Trojans (100,000 random patterns, 20,000 Trojan instances, $q = 2, k = 4, \theta = 0.2$)

Benchmark circuit	Trojan instances	Obfus. flops (n + S)	Obfuscation effects		
			Benign Trojans (%)	False prob. nodes (%)	Func. troj. cov. incr. (%)
s1488	192	8	38.46	63.69	0.00
s5378	2641	9	40.13	85.05	1.02
s9234	747	9	29.41	65.62	1.09
s13207	1190	10	36.45	83.59	0.56
s15850	1452	10	40.35	68.95	2.65
s38584	342	12	33.88	81.83	0.45

Table 8.3 Effect of obfuscation on security against Trojans (100,000 random patterns, 20,000 Trojan instances, $q = 4, k = 4, \theta = 0.2$)

Benchmark circuit	Trojan instances	Obfuscation effects		
		Benign Trojans (%)	False prob. nodes (%)	Func. troj. cov. incr. (%)
s1488	98	60.53	71.02	12.12
s5378	331	70.28	85.05	15.00
s9234	20	62.50	65.62	25.00
s13207	36	80.77	83.59	20.00
s15850	124	77.78	79.58	18.75
s38584	11	71.43	77.21	50.00

of state elements (i.e., a relatively small value of $n + S$) still makes a significant fraction of the Trojans benign. Moreover, it obfuscates the true signal probabilities of a large number of nodes. The obfuscation scheme is more effective for 4-trigger node Trojans. This is expected since a Trojan with larger q is more likely to select at least one trigger condition from the obfuscation state space.

Table 8.4 shows the design overheads (at iso-delay) and the run-time for the proposed obfuscation scheme. The proposed scheme incurs modest area and power overheads, and the design overhead decreases with the increase in the size of the circuit. The results and trends are comparable with the STG modification-based watermarking schemes proposed in [42, 44]. As mentioned earlier, the level of protection against Trojan can be increased by choosing a larger $n + S$ value at the cost of greater design overhead. The run-time presented in the table is dominated by *TetraMax*, which takes more than 90% of the total time for sequential justifications.

Table 8.4 Design overhead (at iso-delay) and run-time[a] for the proposed design flow

Benchmark circuit	Overhead (%)		Run-time (mins.)
	Area	Power	
s1488	20.09	12.58	31
s5378	13.13	17.66	186
s9234	11.84	15.11	1814
s13207	8.10	10.87	1041
s15850	7.04	9.22	1214
s38584	6.93	2.63	2769

[a]The run-time includes the sequential justification time by Synopsys *Tetramax*, which in most cases was over 90% of the total runtime

8.3.9 Discussions

8.3.9.1 Protection Against Malicious CAD Tools

Besides protecting a design in foundry, the proposed obfuscation methodology can provide effective defense against malicious modifications (manual or automated) during the IC design steps. Compromised CAD tools and automation scripts can also insert Trojans in a design [12, 51]. Obfuscation can prevent insertion of hard-to-detect Trojans by CAD tools due to similar reasons as applicable in a foundry. It prevents an automatic analysis tool from finding the true rare events, which can be potentially used as Trojan triggers or payloads. Moreover, since large number of states belong to the obfuscation state space, an automation tool is very likely to insert a Trojan randomly that is only effective in the obfuscation mode. Note that since we obfuscate the gate-level netlist, protection against CAD tools can be achieved during the design steps following logic synthesis (e.g., during physical synthesis and layout).

To increase the scope of protection by encompassing the logic synthesis step, we propose a single, small modification in the obfuscation-based design flow. In the conventional design flow, the RTL is directly synthesized to a technology-mapped gate-level netlist, and obfuscation is applied on this netlist. However, in the modified design flow, the RTL is first *compiled* to a technology-independent (perhaps unoptimized) gate-level description, and obfuscation is applied on this netlist. Such a practice is quite common in the industry, and many commercial tools support such a compilation as a preliminary step to logic synthesis [52]. The obfuscated netlist is then optimized and technology mapped by a logic synthesis tool. Note that the logic synthesis step now operates on the obfuscated design, which protects the design from potential malicious operations during logic synthesis. Also, the RTL *compilation* (without logic optimization) is a comparatively simpler computational step for which the SoC design house can employ a trusted in-house tool. This option provides an extra level of protection.

The same kind of flow can also be applied to FPGA-based designs, where the CAD tool is assumed to be malicious. The RTL corresponding to the circuit can be

"compiled" to a unoptimized, technology-independent gate-level netlist. This netlist can then be obfuscated, and the obfuscated design can then be optimized and mapped by either third-party CAD tools or vendor-specific tools to a netlist in an intermediate format. This netlist is then converted to a vendor-specific bitstream format by the FPGA mapping tool to map the circuit to the FPGA.

8.3.9.2 Improving Level of Protection and Design Overhead

Equation (8.7) suggests that for large designs with a significantly large *original state space*, to attain satisfactory levels of design obfuscation, it is necessary to have the *obfuscation state space* much larger than the *original state space*. This can be achieved by either (a) addition of a large number of extra state elements, or (b) using a large number of unreachable states in the *obfuscation state space*. However, finding large number of unreachable states through sequential justification in a complex design is extremely computationally expensive. To keep the problem computationally tractable and reduce the design overhead, we propose a systematic approach to modify the state transition function as shown in Fig. 8.12. The n extra state elements are grouped into p different groups to form parallel FSMs PSM_1 through PSM_p, and the RTL code for each of them is generated separately. Similarly, the S existing state elements (corresponding to the unreachable states) used for state encoding in the *obfuscation state space* are grouped in q different groups PSM_1' through PSM_q'. The

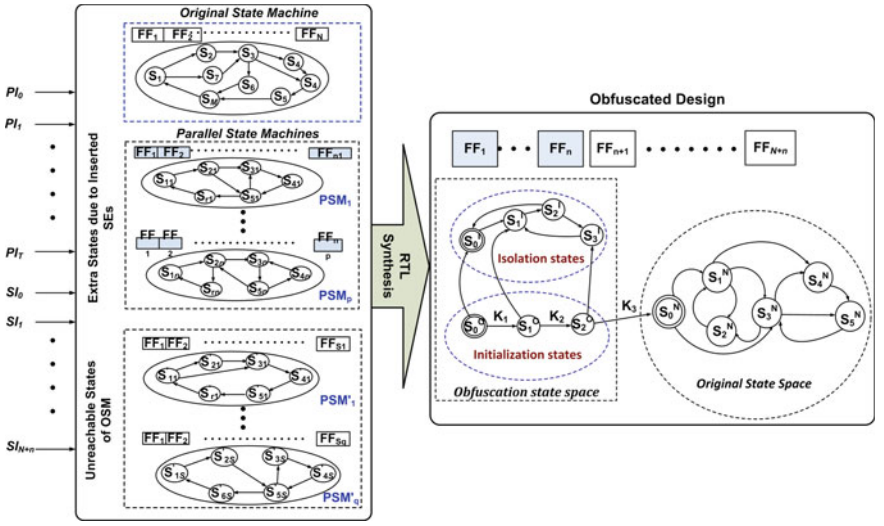


Fig. 8.12 Obfuscation for large designs can be efficiently realized using multiple parallel state machines which are constructed with new states due to the additional state elements as well as unreachable states of original state machine

RTL code for each of the parallel FSMs $PSM_1^{'}$ through $PSM_q^{'}$ is generated separately based on the unreachable states. Such a scheme of having multiple parallel FSMs to design the *obfuscation state space* achieves similar design obfuscation effects, without incurring high computational complexity and design overhead.

8.4 Extension to Register Transfer Level (RTL) Designs

While we have described the proposed design methodologies for gate-level designs, in reality, the majority of the commercial hardware IPs come in the RTL ("soft") format which offers better portability by allowing design houses to map the circuit to a preferred platform in a particular manufacturing process [18]. Two approaches have been proposed to extend the state space obfuscation technique to the RTL designs:

1. The first approach is based on extraction and modification of the state transition from the gate-level synthesized design, such that normal operation is possible only on the successful application of a correct initialization sequence. An obfuscated register transfer level (RTL) design can be generated by decompilation of the obfuscated netlist [53].
2. In the second approach, a register transfer level IP is obfuscated by manipulating its control and data flow graphs (CDFG) derived from the RTL [54].

Experimental results have demonstrated that both these techniques are effective, although the second approach results in lesser hardware overhead and is more scalable [54].

8.5 Conclusions

We have proposed and demonstrated the application of design obfuscation for active defense against IP infringement and IC overproduction at different stages of the SoC design and fabrication flow, and to resist the insertion of hardware Trojans in the manufactured ICs at untrusted fabrication facilities. The obfuscation steps operate on gate-level designs, can be easily automated and integrated in the IC design flow, and do not affect the test/verification of a SoC design for legal users. The methodology can be extended to the RTL designs also. We have shown that they incur low design and computational overhead and cause minimal impact on end-user experience while providing satisfactory levels of protection. The proposed approaches are easily scalable to large designs (e.g., processor) and in terms of level of security. Further improvement in hardware overhead can be obtained by utilizing normally unused states of the circuit.

References

1. Castillo E, Meyer-Baese U, García A, Parrilla L, Lloris A (2007) IPP@HDL: efficient intellectual property protection scheme for IP cores. IEEE Trans VLSI 15:578–591
2. Charbon E, Torunoglu I (2003) Watermarking techniques for electronic circuit design. In: IWDW'02: Proceedings of the international conference on digital watermarking, pp 147–169
3. Kahng A, Lach J, Mangione-Smith W, Mantik S, Markov I, Potkonjak M, Tucker P, Wang H, Wolfe G (2001) Constraint-based watermarking techniques for design IP protection. IEEE Trans CAD 20(10):1236–1252
4. Lach J, Mangione-Smith W, Potkonjak M (1999) Robust FPGA intellectual property protection through multiple small watermarks. Proceedings of the 36th annual ACM/IEEE design automation conference, DAC'99. ACM, New York, pp 831–836
5. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans CAD 20(9):1101–1117
6. Chakraborty RS, Bhunia S (2009) HARPOON: a SoC design methodology for hardware protection through netlist level obfuscation. IEEE Trans CAD 28(10):1493–1502
7. Roy JA, Koushanfar F, Markov IL (2008) *EPIC*: ending piracy of integrated circuits. In: DATE'08: Proceedings of the conference on Design, automation and test in Europe, pp 1069–1074
8. Adee S (2008) The hunt for the kill switch. IEEE Spectr 45(5):34–39
9. Australian Government DoD-DSTO: Towards countering the rise of the silicon trojan (2008). http://dspace.dsto.defence.gov.au/dspace/bitstream/1947/9736/1/DSTO-TR-2220%20PR.pdf
10. Defense Science Board: Task force on high performance microchip supply (2005). http://www.acq.osd.mil/dsb/reports/200502HPMSReportFinal.pdf
11. King ST, Tucek J, Cozzie A, Grier C, Jiang W, Zhou Y (2008) Designing and implementing malicious hardware. In: LEET'08: Proceedings of the Usenix workshop on large-scale exploits and emergent threats, pp 5:1–5:8
12. DARPA: TRUST in Integrated Circuits (TIC) - Proposer Information Pamphlet (2007). http://www.darpa.mil/MTO/solicitations/baa07-24/index.html
13. Wolff F, Papachristou C, Bhunia S, Chakraborty RS (2008) Towards Trojan-free trusted ICs: problem analysis and detection scheme,. In: DATE'08: Proceedings of the conference on design, automation and test in Europe, pp 1362–1365
14. Chakraborty RS, Wolff F, Paul S, Papachristou C, Bhunia S (2009) MERO: a statistical approach for hardware Trojan detection using logic testing. In: Clavier C, Gaj K (eds) Cryptographic Hardware and Embedded Systems - CHES 2009, vol 5737. Lecture Notes on Computer ScienceSpringer, Heidelberg, pp 396–410
15. Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B (2007) Trojan detection using IC fingerprinting. In: SP'07: Proceedings of the IEEE symposium on security and privacy, pp. 296–310
16. Jin Y, Makris Y (2008) Hardware Trojan detection using path delay fingerprint. In: HOST'08: Proceedings of the international workshop on hardware-oriented security and trust, pp 51–57
17. Narasimhan S, Du D, Chakraborty R, Paul S, Wolff F, Papachristou C, Roy K, Bhunia S (2010) Multiple-parameter side-channel analysis: a non-invasive hardware Trojan detection approach. In: HOST'10: Proceedings of the international workshop on hardware oriented security and trust, pp 13–18
18. Chinese firms favoring soft IP over hard cores (2011). http://www.eetasia.com/ART_8800440032_480100_NT_ac94df1c.HTM
19. Wang C, Hill J, Knight JC, Davidson JW (2001) Protection of software-based survivability mechanisms. In: DSN'01: Proceedings of the international conference on dependable systems and networks, pp 193–202
20. Thicket^{TM} family of source code obfuscators (2011). http://www.semdesigns.com
21. Methodology for protection and licensing of HDL IP (2011). http://www.us.design-reuse.com/news/?id=12745&print=yes

22. Brzozowski M, Yarmolik VN (2007) Obfuscation as intellectual rights protection in VHDL language. Proceedings of the 6th international conference on computer information systems and industrial management applications. IEEE Computer Society, Washington, DC, pp 337–340
23. Wirthlin MJ, McMurtrey B (2002) IP delivery for FPGAs using applets and JHDL. Proceedings of the 39th annual design automation conference, DAC'02. ACM, New York, pp 2–7
24. Hou T, Chen H, Tsai M (2006) Three control flow obfuscation methods for Java software. IEE Proc Softw 153(2):80–86
25. Huang YL, Ho F, Tsai H, Kao H (2006) A control flow obfuscation method to discourage malicious tampering of software codes. In: ASIACCS'06: Proceedings of the 2006 ACM symposium on information, computer and communications security, pp 362–362
26. Linn C, Debray S (2003) Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the ACM conference on computer and communications security, pp 290–299
27. Zhuang X, Zhang T, Lee H, Pande S (2004) Hardware assisted control flow obfuscation for embedded processors. In: CASES'04: Proceedings of the 2004 international conference on compilers, architecture, and synthesis for embedded systems, pp 292–302
28. Obfuscation by code morphing (2011). http://en.wikipedia.org/wiki/Obfuscated_code#Obfuscation_by_code_morphing
29. Joepgen H, Krauss S (1993) Software by means of the protprog method. Elecktronik 42:52–56
30. Aucsmith D (1996) Tamper resistant software: an implementation. In: IH'96: Proceedings of the international workshop on information hiding, pp. 317–333
31. Schulman A (1993) Examining the windows AARD detection code. Dr. Dobb's J 18 (1993)
32. Jakubowski M, Saw C, Venkatesan R (2009) Tamper-tolerant software: modeling and implementation. In: IWSEC'09: Proceedings of the international workshop on security: advances in information and computer security, pp 125–139
33. Chang H, Atallah M (2002) Protecting software code by guards. In: DRM'01: Revised papers from the ACM CCS-8 workshop on security and privacy in digital rights management, pp 160–175
34. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K (2001) On the (im)possibility of obfuscating programs. In: CRYPTO'01: Proceedings of the international cryptology conference on advances in cryptology, pp 1–18
35. Chakraborty RS, Bhunia S (2009) Hardware protection and authentication through netlist level obfuscation. In: ICCAD'08: Proceedings of the IEEE/ACM international conference on computer-aided design, pp 674–677
36. Wang F (2004) Formal verification of timed systems. Proc IEEE 92(8):1283–1305
37. Alkabani YM, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: ICCAD'07: Proceedings of the international conference on CAD, pp. 674–677
38. Moore WA, Kayfes PA (2007) US Patent 7213142 - system and method to initialize registers with an EEPROM stored boot sequence. http://www.patentstorm.us/patents/7213142/description.html
39. The ISCAS-89 Benchmark Circuits. http://www.fm.vslib.cz/~kes/asic/iscas/
40. Banga M, Hsiao MS (2008) A region based approach for the identification of hardware Trojans. In: HOST'08: Proceedings of the IEEE international workshop on hardware-oriented security and trust, pp 40–47
41. Alkabani YM, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: SS'07: Proceedings of USENIX security symposium, pp. 20:1–20:16
42. Oliveira A (1999) Robust techniques for watermarking sequential circuit designs. In: DAC'99: Proceedings of the ACM/IEEE design automation conference, pp 837–842
43. Torunoglu I, Charbon E (2000) Watermarking-based copyright protection of sequential functions. IEEE J Solid-State Circ 35(3):434–440
44. Yuan L, Qu G (2004) Information hiding in finite state machine. In: IH'04: Proceedings of the international conference on information hiding, IH'04, pp 340–354

45. Najm FN (1993) Transition density: a new measure of activity in digital circuits. IEEE Trans CAD 14(2):310–323
46. Chou T, Roy K (1996) Accurate power estimation of CMOS sequential circuits. IEEE Trans VLSI 4(3):369–380
47. Yotsuyanagi H, Kinoshita K (1998) Undetectable fault removal of sequential circuits based on unreachable states. In: VTS'98: Proceedings of the IEEE VLSI test symposium, pp 176–181
48. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans Inf Forensics Secur 7(1):51–63
49. Lynn B, Prabhakaran M, Sahai A (2004) Positive results and techniques for obfuscation. Cryptology ePrint Archive, Report 2004/060. http://eprint.iacr.org/
50. Chakraborty RS, Bhunia S (2009) Security against hardware Trojan through a novel application of design obfuscation. In: ICCAD'09: Proceedings of the international conference on CAD, pp 113–116
51. Roy JA, Kaushanfar F, Markov IL (2008) Extended abstract: circuit CAD tools as a security threat. In: HOST'08: Proceedings of the international workshop on hardware-oriented security and trust, pp 61–62
52. Systems, I.: Concorde – fast synthesis (2009). http://www.interrasystems.com/eda/eda_concorde.php
53. Chakraborty R, Bhunia S (2009) Security through obscurity: an approach for protecting Register transfer level hardware IP. In: HOST'08: Proceedings of the international workshop on hardware oriented security and trust, pp 96–99
54. Chakraborty R, Bhunia S (2010) RTL hardware IP protection using key-based control and data flow obfuscation. In: VLSID'10: Proceedings of the international conference on VLSI Design, pp 405–410

Chapter 9
Structural Transformation-Based Obfuscation

Hai Zhou

9.1 Introduction

A variety of techniques have been proposed for fighting against hardware piracy. There are two main classes of approaches. One approach is hardware metering [15], which enables design houses to have post-fabrication control on the produced ICs. By metering, the designer can count the number of fabricated ICs, monitor their usage, and even remotely lock/unlock the ICs. Hardware watermarking [21], as another popular approach to IP protection, is inspired by the traditional digital watermarking technique. It inserts certain identity information into behavioral specification or sequential structure of the design. Watermarking is more passive compared with metering. But since watermarking has a unified signature for all ICs and does not involve any designer–manufacturer interaction, it will usually be less expensive.

Both hardware metering and watermarking techniques are intimately related to program/circuit obfuscation. Informally speaking, an obfuscator is a probabilistic compiler O that transforms a source program/circuit F into a new program/circuit $O(F)$ that has the same functionality as F but less intelligible in some sense. The technique of obfuscation is often used to protect the secrets in programs by making them harder to comprehend. However, circuit (hardware) obfuscation is radically different from program (software) obfuscation. A program is usually obfuscated to hide its function, but the functionality of a commercial IC must be known to different parties other than the designer. The value of a hardware IP is determined by their efficiency of implementation in terms of performance, power consumption, reliability, etc. Thus, instead of hiding the information within the original circuit, circuit obfuscation usually tries to hide extra secret information (e.g., watermarking) that is intentionally added to the circuit in order to prevent illegal use of the IC.

H. Zhou (✉)
Northwestern University, 2145 Sheridan Rd, Evanston, IL, USA
e-mail: haizhou@northwestern.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_9

In this chapter, we discuss two popular notions of obfuscation: black-box obfuscation [3] and best-possible obfuscation [9]. Black-box obfuscation is stronger but has been proved impossible on general families containing point functions [3], while the best-possible obfuscation is weaker and possible to obtain [9]. Defined as disclosing only functionality, the best-possible obfuscation is more realistic in the context of hardware IP protection. Based on its definition, we show that any best-possible obfuscation of a sequential circuit can be accomplished by structural transformation composed of four types of operations: retiming, resynthesis, sweep, and conditional stuttering. We then develop a Key-Locked OBfuscation (KLOB) scheme for hardware IP protection. In KLOB, a circuit will first be inserted with a stuttering logic with conditions both on key checking and on the state of the circuit. The conditionally stuttered circuit will then be further obfuscated by a sequence of retiming, resynthesis, and sweep operations. In the presence of the correct key value, the obfuscated circuit will run in the same speed as the original circuit; without the key, it will run in much slower speed. An simple version of the KLOB has been implemented to measure its overhead, and the effectiveness of the approach is thoroughly discussed.

9.2 Related Approaches

Program/circuit obfuscation is a fundamental problem in computer security. Barak et al. [3] initiated the theoretical study of obfuscation and demonstrated that generic "virtual black-box" program obfuscator does not exist. Later Lynn et al. [20] proved the first positive result about obfuscation that the family of point and multi-point functions can be perfectly obfuscated under random oracle model. Goldwasser and Rothblum [9] argued that the black-box model be too strong for many real applications. They proposed a new notion of "best-possible" obfuscation under relaxed requirements and studied its properties. Yet there is still lack of common agreement on the definition of obfuscation.

The concept of hardware metering is first introduced by Koushanfar and Qu in 2001 [15]. The idea was to assign a unique signature to the IC's functionality by making a small part of the design programmable. There followed some works that exploit manufacturing variability to generate unique random ID for each IC to achieve metering [16, 18, 27, 28]. These methods are all passive. Alkabani and Koushanfar [1, 2, 12] proposed the first active hardware metering scheme. The method utilized physically unclonable function (PUF) [28] to generate the unique initial FF values (power-up state) for each IC. The power-up state will have very high probability to be in the non-functional part of an augmented FSM structure; thus, the IC will be locked. Only the designers who have knowledge about the augmented FSM structure would be able to send the key (transitions to legitimate reset state) to unlock the IC. According to a comprehensive survey about piracy avoidance [11], the methods based on embedding locks in the behavioral description of the design is also called internal active IC metering. In contrast, external active IC metering [10, 25, 26] usually embeds locks in the physical level of the design, which are further controlled

by external cryptography function. The latter set of methods tends to have larger power and area overhead due to the complexity of cryptographic modules interfaced with the locks.

Oliveira first proposed to hide a secret watermark in a sequential circuit [21, 22]. The watermarking was performed by modifying the State Transition Graph (STG) to go through a chosen path of state transitions with certain set of inputs (secret keys). The insertion of watermark will not have any effect on the IC's functionality. The proof of authorship is ensured by the fact that the displayed input-transition behavior would be extremely rare in non-watermarked circuit. Later Koushanfar and Alkabani [14] proposed to add multiple watermarks to further enhance security, and they showed that hiding multiple watermarks in the STG is an instance of obfuscating a multi-point function with a general output. Yuan and Qu proposed the idea of hiding information in the unused transitions of FSM [29]. They developed a SAT-based algorithm to find the maximal set of redundant transitions for a given minimized FSM and took advantage of this redundancy to hide the information in the FSM without changing the given minimized FSM. Hardware watermarking looks similar as passive hardware metering, but they have some critical differences. The watermarking signatures are uniform in all ICs of the same product, while metering will assign a specific signature for each IC. For this reason, watermarking cannot track the number of fabricated copies from one mask.

9.3 Structural Transformation for Best-Possible Obfuscation

9.3.1 Best-Possible Obfuscation

The definition of obfuscation had been intuitive but not vigorous before its theoretical study was initiated by Barak et al. [3]. Barak et al. defined obfuscation in very strong requests that 1) the obfuscated circuit computes the same function as the original circuit with at most a polynomial-time slow-down and 2) the obfuscated circuit should leak no more information than its "black-box" (input–output invocation) functionality. Formally, "black-box" obfuscation requires that anything that can be efficiently learned from the obfuscated circuit can also be learned efficiently from input–output access to the circuit. Barak et al. showed that the general "black-box" obfuscator does not exist. The proof comes from the intuition that even an obfuscated program provides a complete function description, while a "black-box" oracle access may not be able to help to learn the complete function. This is especially true for point functions defined as follows:

$$C_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha \\ 0 & \text{otherwise.} \end{cases}$$

More specifically, we can define another function $D_{\alpha,\beta}$ whose input is a function C:

$$D_{\alpha,\beta}(C) = \begin{cases} 1 & \text{if } C(\alpha) = \beta \\ 0 & \text{otherwise.} \end{cases}$$

Having the obfuscations of the two functions, $C_{\alpha,\beta}$ and $D_{\alpha,\beta}$, the adversary will apply the second function on the first one. The result is always one. However, if we only have "black-box" access to these functions, the probability for any simulator to get one is negligibly small. Therefore, there is no "black-box" obfuscation for any family that includes point functions.

The results of Barak et al. indicate that the "black-box" requirement in the definition may be too strong. In fact, it is indeed too strong in the context of circuit obfuscation for IP protection. When an IP block is provided, its functionality must be known and agreed up on by all parties. If it is a soft block, an obfuscated netlist is also visible to the parties. Thus, an IP block cannot be treated simply as a black-box. Following the study by Barak et al., Goldwasser and Rothblum [9] proposed a new definition of *the best-possible obfuscation* with a relaxed requirement in place of the "black-box" requirement. Intuitively, a best-possible obfuscation only leaks as much information as any circuit of the same function. In other words, it only leaks the functionality of the original circuit. While this relaxed notion of obfuscation gives no absolute guarantee about what information is hidden in the obfuscated circuit, it does guarantee that the obfuscation is literally the best-possible if the functionality is known.

Goldwasser and Rothblum also proved that there exists a best-possible obfuscation for a family of circuits that does not have "black-box" obfuscation. It shows that the definition of the best-possible obfuscation is strictly weaker than that of the "black-box" obfuscation. The family is the Polynomial-sized Ordered Binary Decision Diagrams (POBDD) [6]. Bryant [6] has shown that each OBDD has a canonical representation which can be efficiently computed. The best-possible obfuscation of any POBDD P is its canonical representation, which can be computed efficiently from any POBDD P' of the same function as P. Now, consider the point functions $C_{\alpha,1}(x)$ encoded in POBDD. As shown by Barak et al. [3], there is no "black-box" obfuscation for them.

9.3.2 Functional Equivalence of Finite State Machines

Starting from this section, we are going to show that the best-possible obfuscation can be computed by a sequence of structural transformations on the sequential circuit. Here, structural transformation means operations only on circuit netlist, not on state transition graph. Based on the definition, any obfuscated circuit must have the equivalent function as the original circuit. In this section, we will formally define *functional equivalence* between two circuits/FSMs.

Finite State Machine (FSM): FSM specifies how the system changes its states and produces outputs responding to inputs.

Definition 1 A FSM is quintuple $(Q, I, O, \lambda, \delta)$ where Q is a finite set referred to as the states, I and O are finite sets referred to as the set of inputs and outputs respectively, $\delta : Q \times I \rightarrow Q$ is the next-state function and $\lambda : Q \times I \rightarrow O$ is the output function.

Functional Equivalence: If we view a circuit as a black-box system, then its visible behavior can be described as its possible sequences of inputs and outputs. A circuit may exhibit an externally visible behavior like a sequence

$$\langle\langle E_0 = (I_0, O_0), E_1 = (I_1, O_1), E_2 = (I_2, O_2), \ldots\rangle\rangle$$

Note that in our specification every step in the sequence corresponds to a clock cycle in the sequential circuit. Traditionally, the equivalence of two FSMs [30] requires that their visible behavior should be precisely the same in every single clock cycle. In this chapter, we will define this strict form of equivalence as *cycle-accurate equivalence* to avoid ambiguity.

Definition 2 Two FSMs C and C' are cycle-accurate-equivalent if any sequence of external behavior $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ that is allowed by C will be also allowed by C'.

Nevertheless, the relation of two FSMs computing the same function may not be restricted to cycle-accurate equivalence. If there exists internal states for the circuit, we can also have the complete behavior

$$\langle\langle (E_0, S_0), (E_1, S_1), (E_2, S_2), \ldots\rangle\rangle$$

where S is the internal state (register values). In practice, sometimes only the internal state changes for example

$$\langle\langle (E_0, S_0), (E_1, S_1), (E_1, S_1'), (E_1, S_1''), (E_2, S_2), \ldots\rangle\rangle$$

Since the internal states are invisible to the users, the sequence of external behavior $\langle\langle E_0, E_1, E_1, E_1, E_2, \ldots\rangle\rangle$ and $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ compute the same function. Accordingly, we define the equivalence of two behavior sequences and derive the definition for equivalence of circuit behavior.

Definition 3 Two sequence of external behavior $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ and $\langle\langle E_0^*, E_1^*, E_2^*, \ldots\rangle\rangle$ are stuttering-equivalent if one can be obtained from the other by repeating states or deleting repeated states (by adding or removing finite amount of stuttering).

Definition 4 Two FSMs C and C' are functional-equivalent if for any sequence of external behavior $\langle\langle E_0, E_1, E_2, \ldots\rangle\rangle$ that is allowed by C, there exists a stuttering-equivalent sequence of external behavior $\langle\langle E_0^*, E_1^*, E_2^*, \ldots\rangle\rangle$ that is allowed by C'.

9.3.3 Structural Transformation

Previous approaches to circuit obfuscation usually operate on behavioral level of the design and require substantial modification on the STG of the design. The cost is potentially high since the STG will usually have exponential size in terms of the netlist. In this chapter, we will focus on operating on structural level of the design. Our approach has lower cost since we do not need to generate any STG. All the operations are done on the circuit netlist. We introduce four structural operations applied on sequential circuits: retiming, resynthesis, sweep, and conditional stuttering. An example of applying the first three of them to transform one circuit into another is shown in Fig. 9.1. The example of conditional stuttering will be shown later.

Retiming [17, 19] moves the registers in a sequential circuits while preserving its logic functionality. Two elementary operations can be applied: deleting a register from each input of a combinational node while adding a register to every output, or conversely adding a register to each input of a combinational node and deleting a register from every output. As can be seen from Fig. 9.1, retiming will change the state transition function and the state encoding while keeping the input/output functionality.

Resynthesis restructures the netlist within the register boundaries without changing its logic functionality. As seen from Fig. 9.1, resynthesis will not change the state transition, but it can create new signals in the circuit. These new signals can become new states if we move registers on them by retiming. The first resynthesis in Fig. 9.1 created two new signals for the subsequent retiming to use. Retiming becomes more powerful when combined with resynthesis due to new signals generated. Resynthesis also becomes more powerful when combined with retiming due to new combinational blocks generated by register moves.

Sweep adds or removes registers not having effect on the output. In Fig. 9.1, the sweep operation removes one register with the XOR gate since they do not affect the output. The sweep operation is necessary to change cycle lengths in the state transition graph. In Fig. 9.1 example, it reduces the length of the cycle in STG by half. Since synthesis normally simplifies the circuit structure, sweep is usually used as an operation to remove redundant registers and logic.

Conditional stuttering adds control logic to the circuit to stutter the registers, i.e., copy the register values in the current cycle to the next cycle, if a given logic condition is true. Stuttering is necessary if we want to transform a circuit into another that is not cycle-accurate-equivalent. It is easy to see that an obfuscated circuit can hide more information if it is not required to be cycle-accurate-equivalent to the original one. The simplest implementation is to add a multiplexer to the input of each register to select between the current register value and the next register value.

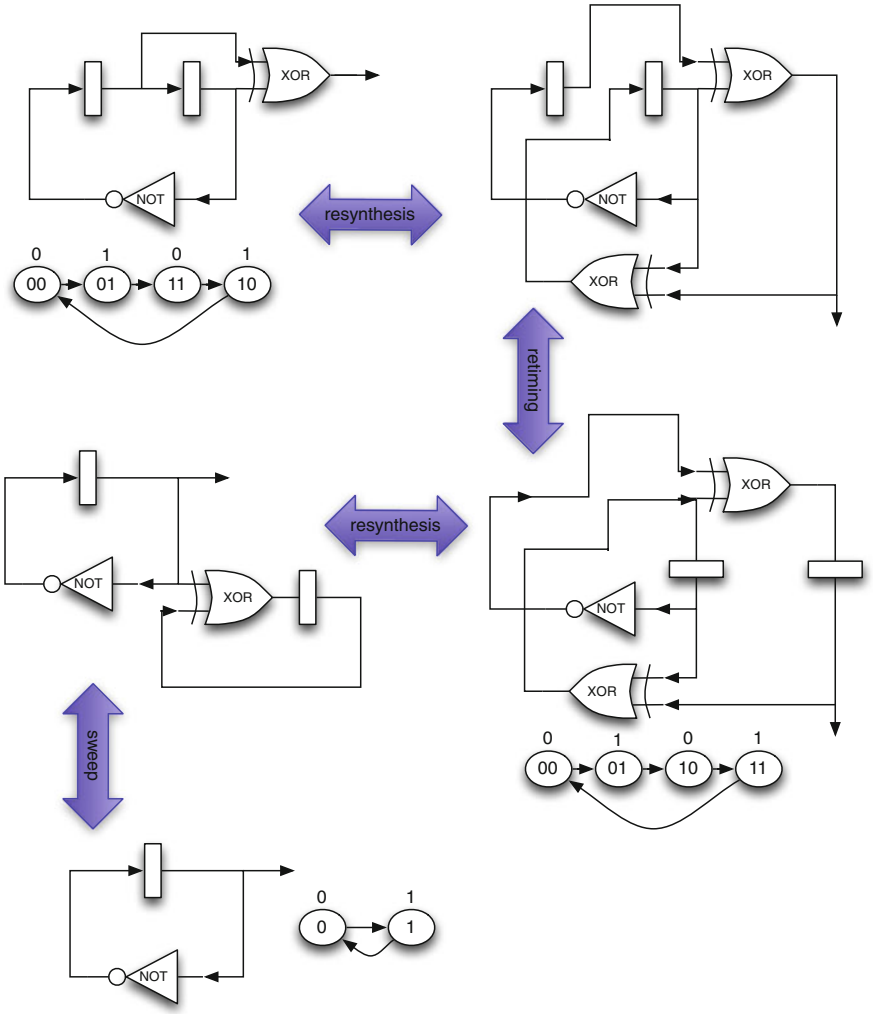


Fig. 9.1 Structural operations: retiming, resynthesis, and sweep

9.3.4 GCD Example for Conditional Stuttering

To better illustrate structural transformation for functional equivalence, we use two small circuits that compute the greatest common divisor (GCD) of two natural numbers as an example. They are different implementations of Euclid's algorithm. The two original circuits (dark lines) as shown in Fig. 9.2 have the same functionality but different netlists due to different resource allocation. Circuit *GCD_A* uses two subtracters, while *GCD_B* uses only one subtracter. Each circuit has registers for two integers. The basic iteration in Euclid's algorithm is to reduce the larger number

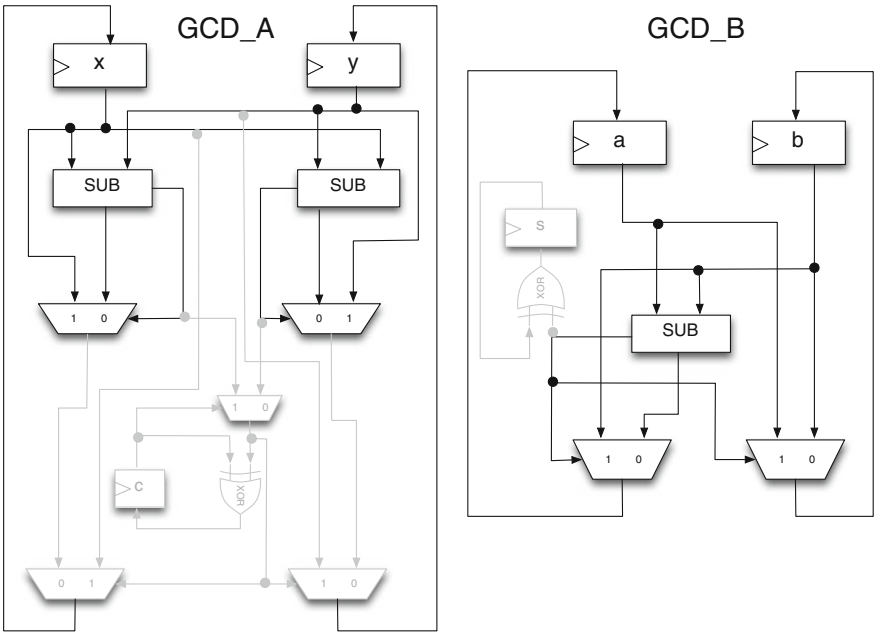


Fig. 9.2 The GCD example for conditional stuttering

to the difference of two numbers until they become the same. However, with only one subtracter, *GCD_B* may conduct a subtraction in wrong order, in which case, it needs to swap the two numbers. Because of it, *GCD_B* is slower than *GCD_A*. The circuits will output one number when they are the same. For simplicity, the outputs are not shown in the circuits.

Our first observation is that *GCD_B* will use more cycles than *GCD_A* for the same computation because it needs extra cycles to swap the two numbers if the subtraction result is negative. Thus *GCD_A* and *GCD_B* are functional-equivalent but not cycle-accurate-equivalent. In order to make them cycle-accurate-equivalent, *GCD_A* needs to be stuttering for one cycle when *GCD_B* is swapping the two numbers. Therefore, in order to know when *GCD_B* swaps, we need to keep track whether the number order in *GCD_A* is different from that in *GCD_B*. We introduce a register c for that purpose. Its value is 0 at the beginning and needs to be flipped when *GCD_B* swaps. The conditional stuttering in *GCD_A* is shown in gray lines in Fig. 9.2. With the conditional stuttering, the two circuits are cycle-accurate-equivalent. To make the mapping between the states of the two circuits explicit, we also introduce a history variable s in *GCD_B* by (inverse) sweep (shown in gray lines). It starts at 0 and flips when the number swaps. With these transformations, the mapping between the states of the two circuits is given as follows.

$$F: \begin{cases} a = & (c == 1)?y : x \\ b = & (c == 1)?x : y \\ s = & c \end{cases} \Leftrightarrow \quad F^{-1}: \begin{cases} x = & (s == 1)?b : a \\ y = & (s == 1)?a : b \\ c = & s \end{cases}$$

As an example, consider giving the input of 4 and 6 to the two GCD circuits. It means that $x = a = 4$ and $y = b = 6$ at the very beginning. The circuit *GCD_A* will generate a sequence of states (x, y) as (4, 6), (4, 2), (2, 2). The circuit *GCD_B* will generate a sequence of states (a, b) as (4, 6), (6, 4), (2, 4), (4, 2), (2, 2). As we can see that the two circuits are functional-equivalent since their final result is the same. However, they are not cycle-accurate-equivalent, since they take different numbers of cycles to reach the final states. But with the conditional stuttering in *GCD_A* and the history variable in *GCD_B*, the sequence of states (x, y, c) in *GCD_A* is (4, 6, 0), (4, 6, 1), (4, 2, 1), (4, 2, 0), (2, 2, 0), while the sequence of (a, b, s) in *GCD_B* is (4, 6, 0), (6, 4, 1), (2, 4, 1), (4, 2, 0), (2, 2, 0). It is easy to check that the corresponding states satisfy the mapping functions. The two cycle-accurate-equivalent circuits can be transformed from each other by a sequence of retiming and resynthesis. Therefore, we can transform *GCD_A* to *GCD_B* by a sequence of conditional stuttering, retiming, resynthesis, and sweep.

9.3.5 *Structural Transformation Sufficient for Best-Possible Obfuscation*

Now we will show that the best-possible obfuscation of a sequential circuit can be done by a sequence of structural transformations. As already demonstrated on a simple example in Fig. 9.1, we can transform any circuit into any other one that is cycle-accurate-equivalent to the original one. This is stated as the following lemma given by [30].

Lemma 1 *If two circuits are cycle-accurate-equivalent, then one of them can be transformed to the other by a sequence of sweep (inverse), resynthesis, retiming, resynthesis, and sweep, given that the second resynthesis operation is allowed to use one-cycle reachability.*

Similar to Fig. 9.2, it can be shown that conditional stuttering can transform a circuit into a circuit that is cycle-accurate-equivalent to any circuit that is functional-equivalent to the original one. Combined with the Lemma 1,

Lemma 2 *If two circuits C_1 and C_2 are functional-equivalent, then C_1 can be transformed into a new circuit C_1', and C_2 into a new circuit C_2', by conditional stuttering, such that C_1' and C_2' are cycle-accurate-equivalent.*

With Lemmas 1 and 2, we can show that any functional-equivalent transformation can be done if conditional stuttering is used in addition to retiming, resynthesis, and sweep.

Theorem 1 *Retiming, resynthesis, sweep, and conditional stuttering are complete for structural transformation between any functional-equivalent circuits.*

The following corollary shows the existence of structural transformations for any best-possible obfuscation of a sequential circuit. It is based on the above theorem that structural transformations can derive any functional-equivalent circuit from a given circuit, and the definition that the best-possible obfuscation reveals at most information as any other equivalent program. Given the current state of art in behavioral synthesis and logic synthesis, we can safely state that giving a program (no matter what computational model it is on) is the same as giving an equivalent sequential circuit.

Corollary 1 *Any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of retiming, resynthesis, sweep, and conditional stuttering.*

9.4 Key-Locked OBfuscation (KLOB)

The previous section shows the existence of structural transformation-based best-possible obfuscations. However, it does not provide a specific procedure, not even a guide, to do transformations for any best-possible obfuscation. The reason is that, even though Goldwasser and Rothblum [9] gave the definition of the best-possible obfuscation–one equivalent circuit, they did not show which one it is or not even which subset it belongs to. In this section, we will address this problem by developing a scheme called Key-Locked OBfuscation (KLOB).

9.4.1 KLOB Framework

We first argue that Key-Locked OBfuscation (KLOB) is the correct scheme for hardware IP protection. Based on the definition [9], the best-possible obfuscation of a circuit is one of equivalent circuits. Intuitively, in order to hide the original circuit, the obfuscation should be most different from the original one. But please note that it should not be *the* most different one if that helps to identify the original. Such a request helps to prevent the original to be understood or reverse-engineered. However, for hardware IP protection, that may not be sufficient: an adversary may not want to understand or modify the original, but to produce and use the circuit without permission. Therefore, an obfuscation that is very different from the original but with similar performance (speed, power consumption, etc.) is not very useful. However, if the obfuscation performs much worse than the original, then the legal users will suffer and complain. Therefore, any obfuscation for hardware IP protection should perform differently between an adversary and a legal user. And it is necessary to employ a secret key in the obfuscated circuit to differentiate the two modes, giving the Key-Locked OBfuscation (KLOB) scheme.

Behaviorally, the KLOB scheme works as follows. It uses a point function at the key value to select between two functional-equivalent circuits: the original one and its best-possible obfuscation with much worse performance. With the key, a legal user is served by the original circuit; without the key, an adversary is almost surely getting the much worse circuit. Of course, it is necessary to use obfuscation to mixed up the three parts of the circuit: the two versions of the circuit and the selection by the point function. Otherwise, an adversary may be able to extract the original circuit by analyzing the circuit.

Instead of starting with two equivalent circuits and then mixing them up, we will start with the original circuit and employ conditional stuttering to transform it. As shown in the previous section, stuttering based on circuit condition can mimic the behavior of any other equivalent circuit. If we only do this followed by a sequence of retiming, resynthesis, and sweep, what we can get is just a slower circuit of the best-possible obfuscation. In addition, KLOB does the stuttering also based on key checking. In other words, stuttering is happening in KLOB if and only if the key input is wrong *and* the circuit stuttering condition is true. Intuitively, the former encodes the selection by a point function at the key, while the latter encodes a slower circuit equivalent to the original one. The advantage of doing this is to make sure that the two circuits of the same function are tightly entangled together. After this conditional stuttering, a sequence of retiming, resynthesis, and sweep will be employed to make sure the three parts are inseparable.

The KLOB scheme after the conditional stuttering step is shown in Fig. 9.3. The components in dark color Ro and Co are the registers and combinational logic of the original circuit, respectively. The components in light color are the conditional stuttering logic, which includes the combinational circuit Cs to generate stuttering condition s based on the original state Ro and the extra registers Rs, and the key checking circuit Ck to generate the mismatching signal k. Only under the condition s AND k the original circuit is stuttering. Please note that Ck in a straight-forward design may not have the dashed connections in Fig. 9.3 and is only a combinational implementation of a point function that will only generate zero at a given key point.

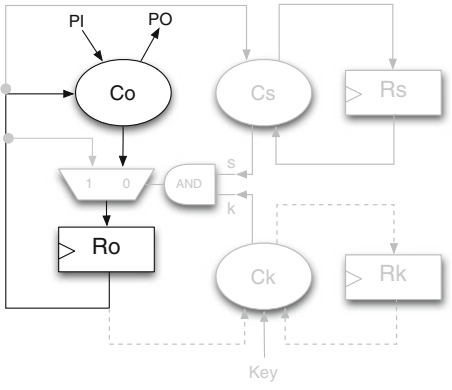


Fig. 9.3 Key-Locked OBfuscation (KLOB) scheme after conditional stuttering

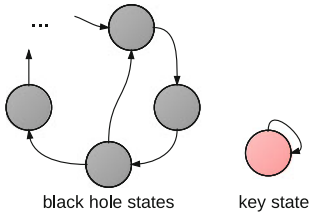


Fig. 9.4 Behavior of the key checking circuit

Such Ck is hard to be obfuscated by retiming and resynthesis, since there is only one bit and one-directional connection from Ck to the rest the whole circuit. In the next subsection, we will enhance the obfuscation of Ck by introducing extra registers Rk and the connection from Ro to Ck. With these modifications, it is easy to see that all the registers in Ro, Rs, and Rk can be retimed through or into Co, Cs, and Ck. It will greatly increase the security of the obfuscation by the followed retiming and resynthesis operations.

9.4.2 Stuttering Control Logic

This section will elaborate on the design of the stuttering control logic, including both Cs with Rs and Ck with Rk. As already mentioned, it is better to introduce extra registers Rk to Ck and to connect Ro to Ck to facilitate the obfuscation by later retiming and resynthesis operations. The idea is to make Rk to stay at the same state if and only if the correct key value is presented at the correct cycle, otherwise it will be trapped in the mismatch states (black hole states), as shown in Fig. 9.4. The transitions among the black hole states are dependent on Ro, making Rk depending on both Ro and Rk.

A simple design for stuttering signal s generation may make Cs with Rs as a counter such that $Rs' = (Rs + 1)\%2^w$ and set $s = (Rs\%t == 0)?0 : 1$, where $1 - 1/t$ is the frequency of stuttering. In other words, the slow circuit is approximately t times slower than the original circuit. However, such a design will make Cs independent of Ro, which will hurt the obfuscation of the whole circuit. From the structural point of view, it may limit the capability to retime registers on Cs; from the behavioral point of view, a very regular stuttering on the original circuit may only transform it into a very specific equivalent circuit, from which the original circuit may be easily derived.

As can be seen from the GCD example in Fig. 9.2, the right stuttering condition should be decided by the target circuit. The ideal approach is to follow the procedure shown in Fig. 9.2, that is, first come up with an equivalent circuit with much worse performance, then figure out a mapping between the states of the two circuits, and finally design the stuttering control logic to make the mapping one-to-one. An easier approach could first find an approximate formular of the reachable states in the original circuit (either by simulation or static analysis), then reformat it into a disjunction

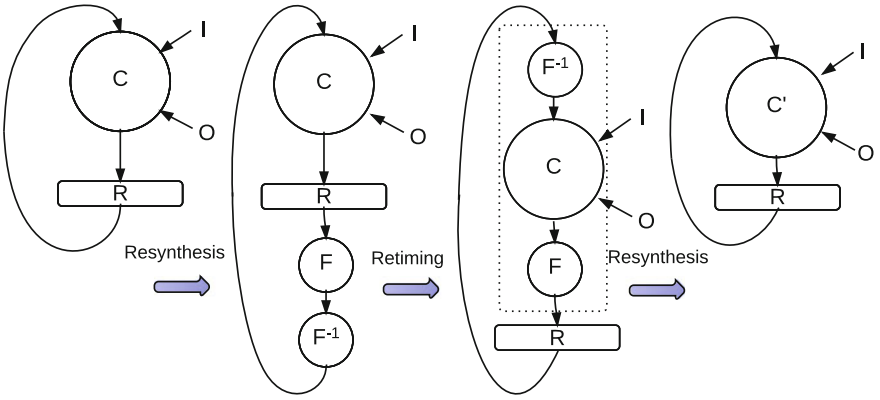


Fig. 9.5 Re-encoding of sequential circuits by retiming and resynthesis

of a few simple expressions, and finally give different numbers of stuttering cycles to different expressions.

Since the insertion of stuttering control logic results in extra delay, area, and power consumptions, its design has to consider not only obfuscation effect, but also the overhead. A good design must have a good trade-off between these effects.

9.4.3 Obfuscation by Retiming and Resynthesis

As shown in Sect. 9.3.5, conditional stuttering by itself is not sufficient for obfuscation. This can be easily seen on Fig. 9.3: by analyzing the circuit, an adversary can easily remove the stuttering control logic and get the original circuit! The conditionally stuttered circuit (shown in Fig. 9.3) has to be obfuscated by other structural transformations: retiming, resynthesis, and sweep. However, the transformation space by these operations is so huge, and it includes all cycle-accurate-equivalent circuits. We propose some basic ideas in this section. It should be note that any extra sequence of retiming and resynthesis operations can be applied on top of each other, and random operations of retiming and resynthesis can enhance the obfuscation.

It can be seen that one vulnerability of the conditional stuttered circuit in Fig. 9.3 is the relative independence of the three register groups Ro, Rs, and Rk. By carefully re-encoding the states, we can increase the dependency among them, thus make it harder to extract useful information from the netlist.[1] It is well known that any re-encoding of a sequential circuit can be done by a sequence of retiming and resynthesis, as shown in Fig. 9.5. The identity function at the register outputs is resynthesized to $F * F^{-1}$, where F is the one-to-one mapping from states of C to the target states of circuit C'. Then retiming moves the registers forward over F. The last step

[1] See Sect. 9.5 for detailed discussion.

resynthesizes $F^{-1} * C * F$ into C'. Note that retiming and resynthesis may also help to reduce the overhead caused by adding stuttering control logic. Different re-encoding functions may be evaluated and the one resulting in the least overhead will be chosen as the final re-encoding function.

A linear transformation can be used as the re-encoding function. An elementary linear transformation transforms the set of variables $X = \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n\}$ into the set of variables $X = \{x_1, \ldots, x_i, \ldots, x_i \oplus x_j, \ldots, x_n\}$. An arbitrary linear transformation can be obtained by a sequence of elementary linear transformations, each one of them implementable by two XOR gates, one gate in the transcoder before the registers (F) and one gate in the transcoder after the registers (F^{-1}).

9.4.4 Implementation Overhead

In this section, we report the overhead in terms of area, power, and timing of the synthesized circuits from the ISCAS89 benchmark suite. We first generate the original BLIF netlist of the benchmark circuits by ABC synthesis tool [5], which will be used as the baseline for obfuscated circuits. Then, we will generate the BLIF netlist of the stuttering control logic. Finally, the original circuit and stuttering control logic will be merged and obfuscated by resynthesis and retiming. All benchmark circuits are mapped to a standard cell library. In the experiments, we use 8 bits for the stuttering indicator and 24 bits for the key indicator.

Table 9.1 demonstrates comprehensive performance overhead evaluations on the ISCAS benchmark suite. The first column denotes the benchmark circuit name. The next three columns (Columns 2–4) show the original design statistics: the number of primary inputs, the number of primary outputs, and the number of FFs. Columns 5–7 demonstrate the design maximum delay in the following order: the original synthesized delay, the added delay post-obfuscation, and the percentage of increase. The original designs power post-synthesis, the added power post-obfuscation, and the ratio between the two are reported in Columns 8–10. The post-synthesis area of the original design, the added area post-obfuscation, and the ratio between are shown in the last three columns, respectively.

We first analyze the impact of obfuscation on the circuit timing. From Fig. 9.3, we can see that the critical path may be affected by newly added control signal, and the ratio of the added critical path delay overhead compared to the original delay seems to be independent of the circuit size. However, this overhead can be alleviated by the followed retiming and resynthesis optimization. Therefore, the actual overhead in the critical path delay introduced by our obfuscation is rather low, especially for large designs that have much flexibility for retiming and resynthesis to leverage. For the tested cases with small or modest design size, on average the delay overhead is 3.35%.

Table 9.1 Experiment results

Circuits	Stats			Delay (ns)			Power (μW)			Area ($literal$)		
	PI	PO	FF	Ori	Incr	%	Ori	Incr	%	Ori	Incr	%
s382	3	6	21	0.463	0.032	6.9	161.1	118.3	73.5	255	195	76.3
s400	3	6	21	0.493	0.016	3.2	167.2	106.0	63.3	264	174	65.9
s526	3	6	21	0.434	0.042	9.6	190.4	136.9	72.2	338	225	66.7
s838	34	1	32	1.227	0.019	1.6	341.0	149.1	43.8	531	253	47.6
s953	16	23	29	0.911	0.046	5.0	391.8	156.6	40.0	595	263	44.2
s5378	35	49	179	0.736	0.021	2.8	1411.0	256.3	18.2	2248	512	22.8
s9234	36	39	211	1.755	0.034	1.9	2157.1	267.0	12.4	3492	543	15.5
s13207	62	152	638	1.86	0.028	1.5	4110.2	501.0	12.2	6339	1114	17.6
s15850	77	150	534	2.78	0.031	1.1	4565.7	590.9	13.0	7104	1316	18.5
s35932	35	320	1728	1.18	0.042	3.6	17787	1271.3	7.2	24934	2998	12.0
s38417	28	106	1636	1.46	0.021	1.5	11731	1242.7	10.6	18417	2923	15.8
s38584	38	304	1426	1.90	0.029	1.5	12458	929.3	7.5	20920	2179	10.4
Average	–	–	–	–	–	3.35	–	–	31.2	–	–	34.4

The area and power overhead is closely related in our approach. In addition, they are not independent of the design size in the worst case since the control signal for all original FFs are changed. The overhead for area and power in our testcases are 31.2 and 34.4% on average. It can be seen that the overhead of our obfuscation scheme decreases as the size of the original design increases. Since our testcases are typically much smaller than current industrial designs, it can be estimate that the overhead for area and power will not exceed 10% for realistic designs.

9.5 Attack Resiliency

In this section, we enumerate possible attacks on KLOB scheme and discuss how the proposed method is secured against them.

- *Brute force attack:* The adversary attempts at guessing the key until the throughput of tested IC is obviously better. It is well known that such an approach could be successful with very tiny probability.
- *Stuttering control logic identification:* Assume that the adversary knows that the circuit is obfuscated by KLOB, thus will try to identify the stuttering control logic. Running without the key, the circuit in Fig. 9.3 must have many stuttering steps in Ro, but not in Rs or Rk. This may be explored by the adversary to identify Ro. However, in KLOB, re-encoding and other retiming and resynthesis steps has been done on this circuit. Suppose A is a stuttering register in Ro and B is a changing register in Rs, a linear transformation $A' = A \oplus B$ will make A' changing, defeating the suggested attack.

It is already mentioned in Sect. 9.4.2 that it is better to make Rs and Rk dependent on Ro. Otherwise, since every register in Ro is dependent on Rs and Rk, a register dependence analysis may separate Ro from the others. Here again, even we did a bad job such that the dependence of Rs and Rk on Ro is weak, a linear transformation $B' = A \oplus B$ will make register B' dependent on register A. Therefore, a general linear transformation on all the registers in Ro, Rs, and Rk will also prevent register dependence analysis.

- *Inverse structural transformation:* The adversary may attempt to inversely transform the obfuscated IC into the original IC via structural transformation. However, without any knowledge on the obfuscation transformations, the adversary can only randomly guess the reverse re-encoding and transformations and test correctness by stuttering control logic identification. In reality, this attack is too expensive and time consuming for the purpose of piracy.
- *Key-based de-obfuscation:* Here, we consider an extreme case where the key has been somehow leaked and want to check how easy the adversary can get the original circuit. If no re-encoding or other retiming and resynthesis is done on the circuit in Fig. 9.3, applying the key can identify Rk since they are not changing. This can further help to identify k and s, thus to get the original circuit. However, with a thorough linear transformation on Ro, Rs, and Rk together, all the registers are mixed up, and

it is impossible to identify Ck. Therefore, we can safely say that, even when the key is leaked, its damage to a KLOB circuit is limited since the adversary can only use the original circuit but cannot get the design.

9.6 Conclusion

This chapter presents a circuit obfuscation technique called KLOB (Key-Locked OBfuscation) based on structural transformations. It first shows that any best-possible obfuscation of a sequential circuit can be accomplished by a sequence of retiming, resynthesis, sweep, and conditional stuttering. Then the KLOB is presented for hardware IP protection. Starting with an original circuit, KLOB first adds stuttering with conditions both on key checking and on the original circuit, and then obfuscates the conditionally stuttered circuit by a sequence of retiming, resynthesis, and sweep. With the correct key, the circuit will run in the original speed; otherwise, it will run much slower. The efficiency of the method was demonstrated by evaluations on ISCAS89 benchmarks. We also discussed the possible attacks and how KLOB is secure against them.

As we already mentioned, the benefit of structural transformations is to avoid the expensive STG manipulation. Therefore, the structural transformation-based obfuscation is more efficient than STG-based obfuscations. Logic obfuscation (also known as logic encryption) is a technique that uses a key and extra logic to modify the combinational design of a given circuit [4, 7, 8, 13, 23–25]. Since it only modifies the combinational logic, logic obfuscation can be viewed as a subset of the structural transformation-based obfuscation, where the allowed operations is only resynthesis.

For hardware IP protection, circuit performance is the key treasure to be protected. If the obfuscated circuit performs similarly as the original, an adversary will be happy to take it. If it performs much worse, then no user wants it. This means that the KLOB scheme is the right choice for circuit obfuscation: with the key, legal users get a circuit with the same performance as the original one; without the key, an adversary gets the best-possible obfuscation—an equivalent circuit with much worse performance. The theory in Sect. 9.3.5 ensures that, even when the key is known, an adversary will still not be able to get the original circuit. Our future work will study more sequences of structural operations to better obfuscate the conditionally stuttered circuit, especially the key checking circuit. We will leverage existing obfuscation techniques for point functions since key checking is essentially a point function.

References

1. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: Proceedings of 16th USENIX security symposium on USENIX security symposium, pp 20:1–20:16

2. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: Proceedings of the 2007 IEEE/ACM international conference on computer-aided design, pp 674–677
3. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan SP, Yang K (2001) (im)possibility of obfuscating programs. In: Proceedings of the 21st annual international cryptology conference on advances in cryptology, pp 1–18
4. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. IEEE Design and Test 27:1
5. Brayton R, Mishchenko A (2010) ABC: an academic industrial-strength verification tool. In: Proceedings of the 22nd international conference on computer aided verification, pp 24–40
6. Bryant R (1986) Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 35:677–691
7. Chakraborty R, Bhunia S (2008) Hardware protection and authentication through netlist level obfuscation. In: IEEE/ACM international conference on computer-aided design
8. Dupuis S, Ba P-S, Natale GD, Flottes M-L, Rouzeyre B (2014) A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: IEEE international on-line testing symposium
9. Goldwasser S, Rothblum GN (2007) On best-possible obfuscation. In: Proceedings of the 4th conference on theory of cryptography, pp 194–213
10. Huang J, Lach J (2008) IC activation and user authentication for security-sensitive systems. In: Proceedings of the 2008 IEEE international workshop on hardware-oriented security and trust, pp 76–80
11. Koushanfar F (2011) Integrated circuits metering for piracy protection and digital rights management: an overview. In: great lakes symposium on VLSI, GLSVLSI '11, pp 449–454
12. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. Inf. forensics and secur., IEEE Trans. 7(1):51–63
13. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans. on Inform. Forensics and Secur. 7:1
14. Koushanfar F, Alkabani Y (2010) Provably secure obfuscation of diverse watermarks for sequential circuits. In: IEEE international symposium on hardware-oriented security and trust (HOST), pp 42–47
15. Koushanfar F, Qu G (2001) Hardware metering. In: Proceedings of the 38th annual design automation conference, pp 490–493
16. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. Inform. Hiding. Springer, Heidelberg, pp 81–95
17. Leiserson CE, Saxe JB (1991) Retiming synchronous circuitry. Algorithmica 6(1):5–35
18. Lofstrom K, Daasch W, Taylor D (2000) IC identification circuit using device mismatch. In: IEEE international solid-state circuits conference, pp 372–373
19. Lu Y, Zhou H (2013) Retiming for soft error minimization under error-latching window constraints. In: Design automation and test in Europe conference
20. Lynn B, Prabhakaran M, Sahai A (2004) Positive results and techniques for obfuscation. In: In EUROCRYPT 04
21. Oliveira AL (1999) Robust techniques for watermarking sequential circuit designs. In: Proceedings of the 36th annual ACM/IEEE design automation conference, pp 837–842
22. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. 20(9):1101–1117
23. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Design automation conference
24. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans. on Comput. 64:2
25. Roy JA, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In Design, Automation and Test in Europe
26. Roy JA, Koushanfar F, Markov IL (2008) Protecting bus-based hardware IP by secret sharing. In: Proceedings of the 45th annual design automation conference, pp 846–851

27. Su Y, Holleman J, Otis B (2007) A 1.6pj/bit 96% stable chip-ID generating circuit using process variations. In: IEEE international solid-state circuits conference, pp 406–611
28. Suh GE, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Proceedings of the 44th annual design automation conference, pp 9–14
29. Yuan L, Qu G (2004) Information hiding in finite state machine. In: Proceedings of the 6th international conference on information hiding, pp 340–354
30. Zhou H (2009) Retiming and resynthesis with sweep are complete for sequential transformation. In: Proceedings of 9th international conference on formal methods in computer-aided design, pp 192–197

Part IV
Hardware Obfuscation Based on Emerging Integration Approaches

Chapter 10
Split Manufacturing

Siddharth Garg and Jeyavijayan (JV) Rajendran

10.1 Introduction

The idea behind split manufacturing (or split fabrication) is to partition (or "split") an IC netlist into multiple "parts" and fabricate each part at a separate foundry. Intuitively, since no one foundry gets access to the full design of the IC, its ability to either pirate the design or maliciously modify it in a targeted way is hindered.

In its simplest instantiation, an IC is split into two parts. One part has of all the active components (transistors) and some of the interconnect (wires), while the other part has the remaining interconnections. As we will discuss, more sophisticated instantiations of split manufacturing might even involve splitting active components across gates.

Technologically, split manufacturing can be achieved in one of two ways: either using an FEOL/BEOL split, or using 3D integration technology. These are discussed below.

- FEOL/BEOL splitting: this technique, shown in Fig. 10.1a, involves splitting the front-end of line (FEOL) and the back-end of line (BEOL) fabrication steps across two foundries. The FEOL part consists of transistors as well as lower metal layers (for example Metal 4 and below), and the BEOL part consists of the upper metal layers [1]. The untrusted, high-end foundry manufactures the FEOL part (including the lower BEOL layers), since these steps involve the smallest feature sizes and require access to advanced fabrication technology. Next, the trusted, low-end, in-house foundry manufactures the remaining BEOL layers. Clearly, the attacker in the high-end untrusted foundry only has access to a partial netlist; he has only

S. Garg (✉) · J. Rajendran
New York University, New York City, NY, USA
e-mail: siddharth.garg@nyu.edu; sg175@nyu.edu

S. Garg · J. Rajendran
The University of Texas at Dallas, Richardson, TX, USA

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_10

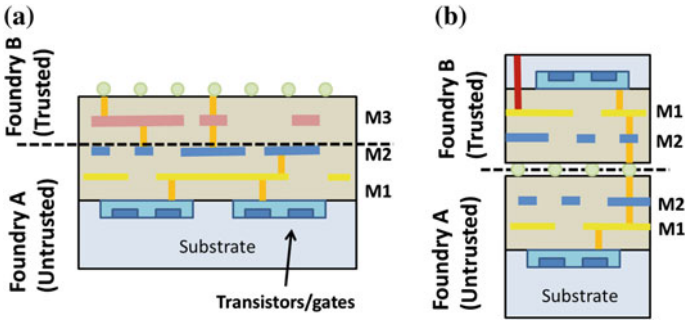


Fig. 10.1 Two approaches to split manufacturing. **a** FEOL/BEOL split manufacturing and **b** 3D integration-based split manufacturing

the FEOL part but not the BEOL part. The feasibility of this approach has been demonstrated by Vaidyanathan et al. [2] in a 0.13 μm technology node.

- 2.5D/3D integration: As shown in Fig. 10.1b, the netlist is split across two or more wafers, each containing a part of the netlist. The wafers are fabricated in different foundries and integrated through 3D integration technology. When the top part consists of only metal layers, the technology is more commonly referred to as 2.5D integration and the top tier is referred to as an interposer.

Split manufacturing is advantageous over other IP protection techniques as it does not require any key-storage mechanisms, as logic encryption does. In addition, as we will see, split manufacturing can be used to defend against strong attack models in which the attacker has access to the netlist the defender wishes to fabricate and aims to maliciously modify targeted parts of the netlist. On the other hand, hand split manufacturing is susceptible to proximity attacks that exploit physical design information, while logic encryption is not. Mitigating this vulnerability potentially introduces high overheads.

10.1.1 Split Manufacturing Flow

Figure 10.2 shows an exemplar split manufacturing flow that leverages 2.5D integration. The designer starts with the design netlist and first *partitions* the netlist into two tiers—the bottom tier consists of all gates and some wires, while the top tier consists of the remaining wires. The top tier is also referred to as the *hidden* tier since it is hidden from the view of the untrusted foundry. Note that partitioning for 2.5D/3D integration is a well-studied problem in the EDA community [3]. However, these partitioning strategies try to optimize metrics like delay and power, not security.

After partitioning, the next step is physical design. At the end of this step, the GDSII files for the bottom and top tiers are ready to be sent to their respective foundries. As mentioned earlier, the attacker should not have access to the layout

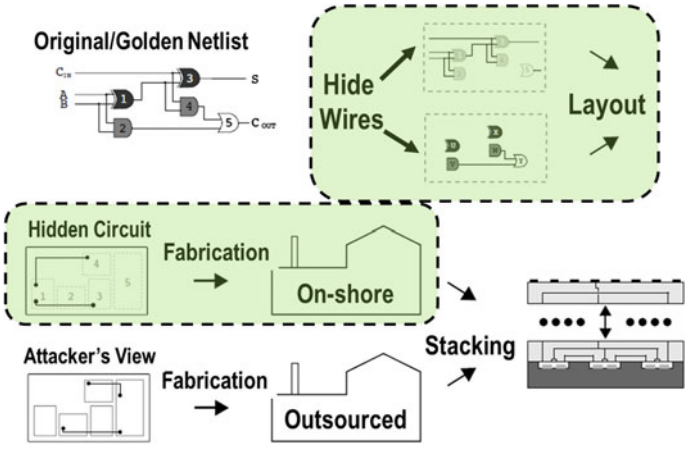


Fig. 10.2 Split manufacturing flow using 2.5D integration. The steps in the *green boxes* must be performed securely, while all others are potentially subject to attack. The flows for single-wafer and full 3D integration-based split manufacturing are similar

of wires in the top tier. Traditional physical design tools optimize for metrics like average wire length and can potentially leak information to the attacker, thus compromising security. In Sect. 10.4, we discuss a security-aware layout strategy for split manufacturing.

The two tiers are manufactured at their respective foundries. The assumption is that the two foundries cannot collude—for this reason, the top tier must be fabricated in a trusted, in-house foundry. Finally, the top and bottom tiers are stacked and packaged by a trusted integrator.

An inherent assumption in any split manufacturing process is that the IC has not been manufactured before, or at the very least, previously manufactured versions cannot be purchased commercially. If this were the case, the untrusted foundry could simply purchase the IC from the market and reverse engineer the wiring in the hidden tier.

The rest of this chapter is organized as follows. Section 10.2 details two specific threat models in the context of split manufacturing. The weak threat model assumes an attacker who is interested in thieving the designer's IP. In contrast, the strong threat model strengthens the weak attacker with apriori knowledge of the IC's netlist (hence, IP theft is moot)—the strong attacker wishes to maliciously modify the chip's functionality.

Next, Sect. 10.3 discusses a specific security metric, k-security, that quantifies the amount of security that split manufacturing buys in the context of the strong attack model. Intuitively, k-security measures the extent to which the attacker is confused as to the functionality of each gate in the (partial) netlist she observes.

Section 10.4 discusses how the traditional design automation tool flow should be modified to obtain a certain level of security while minimizing cost and information

leakage. In the secure partitioning step, the netlist is partitioned so as to maximize k-security within a cost constraint. In the secure layout step, the placement of gates in the netlist outsourced to the untrusted foundry is decided so as to ensure that the layout/placement does not reveal information about the wiring in the part of the netlist hidden from the attacker. We conclude in Sect. 10.6.

10.2 Threat Models

We now discuss two specific threat models in the context of split manufacturing. In the first, we assume that the attacker only has access to the GDSII file of the bottom tier. We call this the weak attack model. The attacker's goal is to reverse engineer the full netlist when the designer is fabricating. In the second, we assume that in addition to the GDSII file of the bottom tier, the attacker also has access to the full netlist when the designer is fabricating. Here, the attacker's goal is not IP theft, but instead is hardware Trojan insertion.

10.2.1 Weak Attack Model

In the weak attack model, the goal of the designer is to keep the IC's netlist hidden from the attacker. Formally, let the designer's private netlist be $C = (V, E)$. The part of the netlist sent to the untrusted foundry is referred to as $C_{PART} = (V_{PART}, E_{PART})$, where $|V| = |V_{PART}|$ for FEOL/BEOL splitting or 2.5D integration-based split manufacturing. We assume that the attacker can reliably recover the netlist C_{PART} from its layout file. Can a determined attacker recover C knowing only C_{PART}?

Rajendran et al. [4] have shown that attacker's can potentially infer the hidden connectivity in the top tier by leveraging the proximity of gates in the bottom tier. This is referred to as a *proximity attack*. Proximity attacks can be particularly successful if the defender uses conventional layout techniques. Such techniques try to minimize sum wire length; thus, connected gates are likely to be proximal. This is illustrated in Fig. 10.3 using a simple example in which connecting proximal bond points recovers the hidden wires correctly. Figure 10.4 shows the histogram of Euclidean distances between pairs of connected and unconnected gates obtained using a commercial layout tool—observe that connected gates are far more likely to be proximal than unconnected gates.

Using even a simple strategy in which the attacker connects each unconnected gate input to its closest unconnected gate output results in >90% correct recovery of hidden connections when a conventional netlist partitioning technique such as hMetis [5] and a commercial layout tool are used in the split fabrication flow.

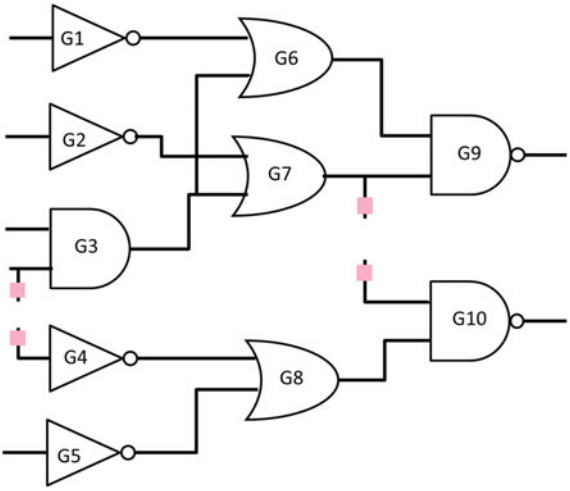


Fig. 10.3 c17 benchmark circuit with two hidden wires. Connecting proximal bond points recovers the correct netlist

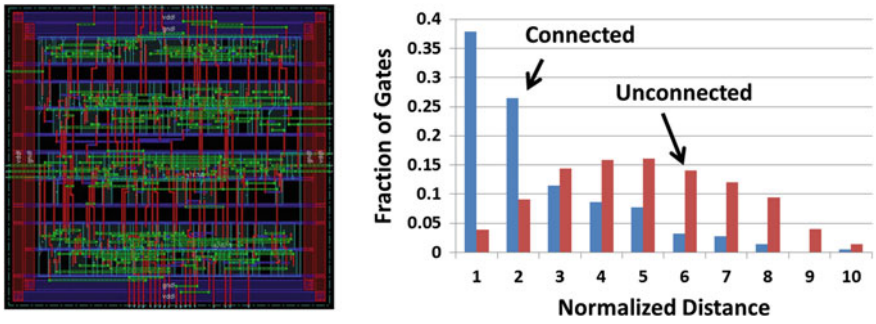


Fig. 10.4 Layout of a sample benchmark and corresponding wire length distribution for unconnected and connected gates

10.2.2 Strong Attack Model

In the weak attack model, the designer's netlist is private. However, what if the designer's goal is to fabricate logic for which the functionality, and perhaps even the netlist, is public knowledge? Examples of such functions are abundant. Most cryptographic protocols are publicly known, for instance advanced encryption standard (AES) and the data encryption standard (DES) protocols, and often have known optimized hardware implementations [6, 7]. The primary threat in such a scenario arises from hardware Trojan insertion. As noted by [8], hardware Trojans can have a disastrous impact on IC security, from unauthorized privilege escalation [9] to secret key leakage [10]. Hardware Trojans are broadly categorized into: (1) always active,

(2) trigger and payload, and (3) reliability-based [11], i.e., those that use device degradation as an implicit trigger.

Specifically, in the strong attack model, we assume that attacker has access to the full netlist, C, that the designer wishes to fabricate. In addition, as in the weak attack model, the attacker also has the partial netlist C_{PART}. We assume that the designer has scrubbed the layout file from which C_{PART} is derived of all identifying labels, and therefore, the labels in C_{PART} are arbitrary and unrelated to those in the original netlist C.

The attacker wishes to modify the design, i.e., insert a hardware Trojan, in a certain *targeted* way. For instance, for the privilege escalation attack [9], the attacker's goal is to modify the gates that control the bits that determine whether the processor executes in user or kernel mode. That is, the attacker needs to determine where in the design to insert the hardware Trojan payload. Similarly, to insert a Trojan that triggers when a certain sequence of instructions is observed [12], the attacker needs to identify certain wires/gates in the decode logic. As another example, the reliability attack discussed in [11] also requires modifications of certain targeted parts of the netlist.

To succeed in its objective, therefore, the attacker must first correctly identify the gate(s) in C_{PART} that it wishes to modify (recall that gates in C_{PART} and C are differently labeled). It does so by *matching* the gates in the partial netlist to those in the public netlist. If the match is correct, the attacker succeeds. The attacker's objective can therefore be formulated mathematically as follows. To match gates in C_{PART} to those in C, the attacker wishes to find a bijective mapping $\phi : V \rightarrow V_{PART}$ such that $< \phi(u), \phi(v) > \in E_{PART}$ only if $< u, v > \in E$. That is, the attacker knows that if an edge exists in C_{PART}, the corresponding edge must exist in C. On the other hand, if an edge does not exist in C_{PART}, the corresponding can still exist in C since it might be hidden.

The condition above is equivalent to the attacker determining a sub-graph of C, one that consists of all of the vertices in C but only some of the edges, that is isomorphic to C_{PART}. Two graphs are said to be isomorphic if they have the structure, i.e., there is a way to permute the vertices of the first graph to obtain the second. When a sub-graph of one graph is isomorphic to another, this is referred to as a **sub-graph isomorphism**.

The crux of using split fabrication as a defense mechanism in this setting is that *many such sub-graph isomorphisms might exist*, thus hindering the attacker in identifying a correct mapping.

Note that the proximity attacks discussed above are still relevant in the context of the strong attack model—that is, in addition to finding sub-graph isomorphisms, the attacker could use proximity information to match gates in C_{PART} to C. In this sense, the strong attack model subsumes the weak attack model. The rest of this chapter therefore focuses on the strong attack model.

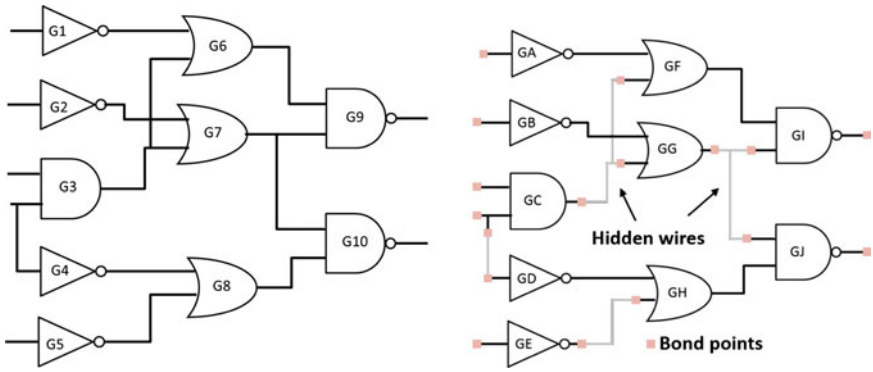
10.3 Security Metric

Imeson et al. [13] have proposed a security metric, *k-security*, that quantifies the security obtained from split manufacturing against targeted hardware Trojan insertion attacks. Consider, for example, the public netlist in Fig. 10.8a and the partial netlist sent to the untrusted foundry, shown in Fig. 10.8b. Five wires from the public netlist have been hidden, and bond points are added to allow these wires to be implemented in the top tier. Now observe that gate G6 in the public netlist can correspond to *either* gate GF and GG in the partial netlist. We thus say that gate G6 is 2-secure. To attack G6, the attacker can either pick one of GF/GG and fail with probability 0.5 (modifying both gates will change the nature of the attack). On the other hand, note that the attacker can uniquely identify that gate GC in the partial netlist is gate G3 in the public netlist. Gate G3 is therefore only 1-secure. The definition of k-security is formalized below (Fig. 10.5).

Definition 1 (*k-security*) A gate $u \in V_{PUB}$ is k-secure if there exist k distinct subgraph isomorphisms $\{\phi_1, \phi_2, \ldots, \phi_k\}$ between C_{PUB} and C_{PART} where $\phi_i(u) \neq \phi_j(u)$ for all $i, j \in [1, k]$ and $i \neq j$. A partial netlist C_{PART} is k-secure with respect to the public netlist C_{PUB} if each vertex $u \in V_{PUB}$ is k-secure.

10.3.1 Relevance of k-Security

To further understand the relevance of the k-security metric, consider the hardware implementation of the DES protocol shown in Fig. 10.6a. It has been shown that if an attacker can modify the LSB output of the 14^{th} round, then she can easily recover



(a) c17 Benchmark Netlist (b) Secure Partitioning

Fig. 10.5 The c17 benchmark netlist (**a**), and the part of the netlist sent to the untrusted foundry after secure partitioning (**b**)

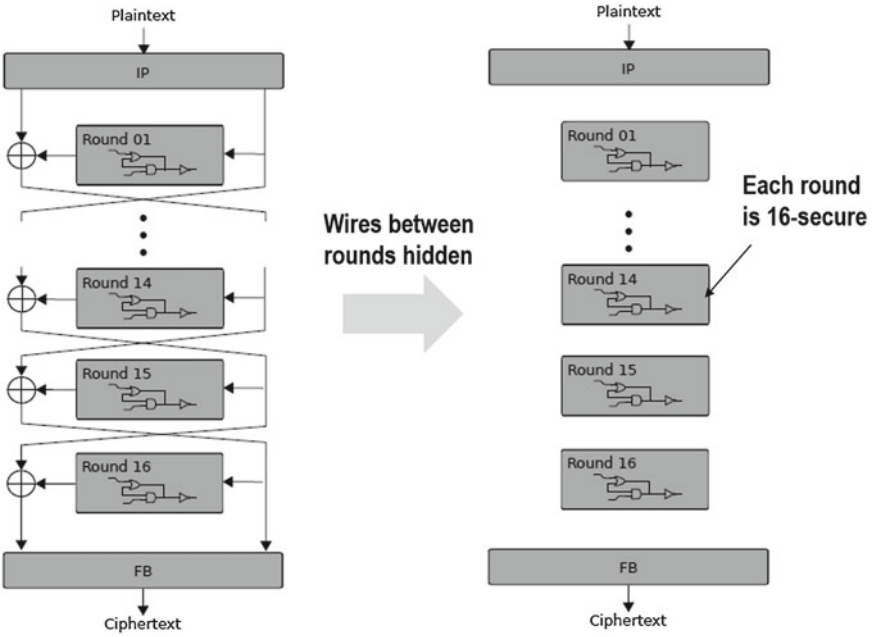


Fig. 10.6 A hardware implementation of DES encryption (*left*) and the partial netlist sent to the untrusted foundry after partitioning. Each round of the DES implementation is 16-secure since it cannot be distinguished from any other round

the DES key from plaintext–ciphertext combinations [14]. Without obfuscation, an untrusted foundry might try to leverage this vulnerability by maliciously modifying the DES implementation such that the least significant bit (LSB) output of the 14^{th} round flips when a certain trigger condition occurs. Obviously, to carry out such a targeted attack, the foundry must first identify the wire that corresponds to this vulnerable bit.

Now consider the implementation in Fig. 10.6b where all wires between rounds are hidden. Since the functionality (and netlist) of each round is identical, any one of the 16 modules could correspond to the one that implements the 14^{th} round. Indeed, in this case, the LSB output of the 14^{th} round is 16-secure.

Other examples of security-critical gates that an attacker might wish to target in a netlist include:

- The gate that outputs the privilege bit in a microprocessor. By modifying the output of this gate, the attacker can launch a privilege escalation attack [15].
- Bits that indicate the type of instruction in a processor decode unit. Rarely occurring instruction types, or more generally, gate outputs that rarely switch, can be used as triggers for attacks [16].

10.3.2 Computing k-Security

For a given partitioning, that is, given C and C_{PART}, how can the defender determine the security level k? As indicated by the definition of k-security, the problem determining the security level is closely related to the sub-graph isomorphism problem, which is NP-complete. Indeed, Imeson et al. [13] formally prove that the problem of determining whether a given partitioning meets a security constraint, k, is also NP-complete.

Having characterized the complexity of the problem, the next step is to devise a concrete algorithm to determine the security level for a given partitioning solution. To do so, the defender iterates through vertices in C. For each vertex in C, the defender iteratively checks if it can be mapped to each vertex in C_{PART}. Specifically, to check whether vertex $u \in V$ can map to vertex $v \in V_{PART}$, she checks if a sub-graph of C is isomorphic to C_{PART} with the constraint that u must map to v ($\phi(u) = v$).

The check above can be performed in one of two ways: (1) directly using sub-graph isomorphism solvers (since it is an instance of a sub-graph isomorphism problem); or (2) reducing the check to an instance of a CNF-SAT and calling a SAT solver. The reason to try the second approach is that fast, off-the-shelf solvers are available for the SAT problem.

The reduction to SAT approach (which is the one recommended by Imeson et al. [13]) introduces Boolean variables ϕ_{ij} that are 1 if node v_i in C_{PART} maps to node r_j in C, and 0 otherwise. Constraints are then introduced that ensure that a node in C_{PART} maps to only one node in C and vice versa. Finally, constraints are also introduced to ensure that an edge in C_{PART} only maps to an edge in C. The three sets of constraints are conjoined and input to a SAT solver.

Given graphs C and C_{PART}, and a bijective mapping ϕ as defined above, we now construct a Boolean formula that is true if and only if graphs C and C_{PART} are sub-isomorphic for the mapping ϕ. We will construct the formula in parts.

First, we ensure that each vertex in C maps to only one vertex in C_{PART}:

$$F_1 = \prod_{i}^{|V_{PART}|} \sum_{j}^{|V|} \left(\phi_{i,j} \prod_{k \neq i}^{|V|} \neg \phi_{i,k} \right)$$

and vice versa:

$$F_2 = \prod_{j}^{|V|} \sum_{i}^{|V_{PART}|} \left(\phi_{i,j} \prod_{k \neq i}^{|V_{PART}|} \neg \phi_{k,j} \right)$$

Finally, we need to ensure that each edge in C_{PART} maps to an edge in C. Let $E_{PART} = \{e_1, e_2, \ldots, e_{|E_{PART}|}\}$ and $E = \{f_1, f_2, \ldots, f_{|E|}\}$. Furthermore, let $e_k = \langle q_{src(e_k)}, q_{dest(e_k)} \rangle \in E_{PART}$ and $f_k = \langle r_{src(f_k)}, r_{dest(f_k)} \rangle \in E$. This condition can be expressed as follows:

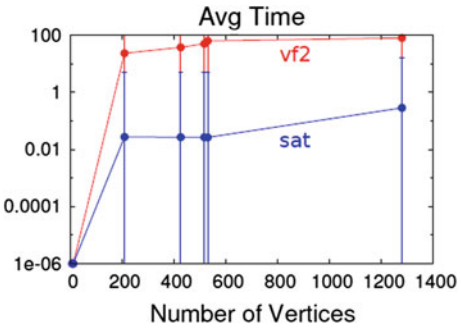


Fig. 10.7 Run-time of SAT-based a domain-specific sub-iso solver-based approaches for computing security. The sub-iso solver used is VF2

$$F_3 = \prod_{k}^{|E_{PART}|} \sum_{l}^{|E|} \phi_{src(e_k),src(f_l)} \wedge \phi_{dest(e_k),dest(f_l)}$$

The formula F that is input to the SAT solver is then expressed as a conjunction of the three formulae above: $F = F_1 \wedge F_2 \wedge F_3$. The formula F has $O(|V_{PART}||V|)$ variables and $O(|E_{PART}||E|)$ clauses.

Empirically (and perhaps surprisingly), the SAT approach is faster than using a domain-specific sub-iso solver. This is illustrated in Fig. 10.7.

10.4 Defense Mechanisms

Designer's must mitigate the threat from strong attackers in two ways: (1) secure partitioning to maximize k-security so as to defeat sub-iso attacks and (2) secure layout to defeat proximity attacks. Note that to defend against weak attackers, only the second method would be required. We now describe these defense mechanisms in detail.

10.4.1 Secure Partitioning

Split manufacturing incurs a cost—wires that cross from one tier to the other use large, capacitive bond points. This increases the area, delay, and power consumption of the chip. Thus, on the one hand, increasing the number of hidden wires increases security, but also increases area, delay, and power. The goal of secure partitioning is to explore the trade-offs between security and cost. As a starting point, we adopt a simple notion of cost, the number of hidden wires. Security is measured using the k-security metric described earlier.

Given C and constraint on maximum number of hidden wires, H, the goal of secure partitioning can be formulated as finding C_{PART} as follows:

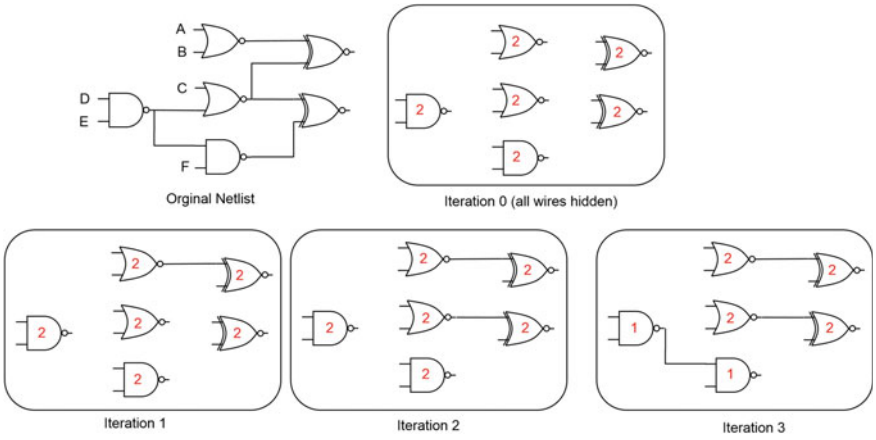


Fig. 10.8 Example of greedy secure partitioning heuristic. In each iteration, a new edge is added in a way to minimize the reduction in security. Each gate is annotated with its security level in each iteration

$$\max_{C_{PART}} k(C, C_{PART})$$

such that

$$|E| - |E_{PART}| \leq H$$

and

$$C_{PART} \subseteq C,$$

where $k(C, C_{PART})$ returns the security level of a partitioning solution.

El Massad [17] has shown that there exists no polynomial time approximation scheme for the secure partitioning problem. As an alternative, Imeson et al. [13] have proposed a greedy heuristic to solve this problem.

The greedy heuristic initializes C_{PART} to have all the gates in C but none of the wires. That is, C_{PART} is initialized to have maximum security, but at maximum cost. Then, in each iteration, a new edge/wire is added to C_{PART}, specifically, one that results in the smallest reduction in security. These iterations continue till $E_{PART} = E - H$, at which point the procedure terminates.

An example illustrating the greedy procedure is shown in Fig. 10.8. The original netlist has 6 gates. Each gate can be at best 2-secure because it can, at best, be confused for the other gate of the same type. Starting with the maximally secure netlist in which all wires are hidden, we observe that the new wire added in the first iteration does not reduce security. The same is true for the second iteration. In the third iteration, adding any new wire will result in a drop in security—the wire that is added results in the least drop in security (the security of the two NAND gates goes down to 1). Adding any other wire would have resulted in even larger reduction in

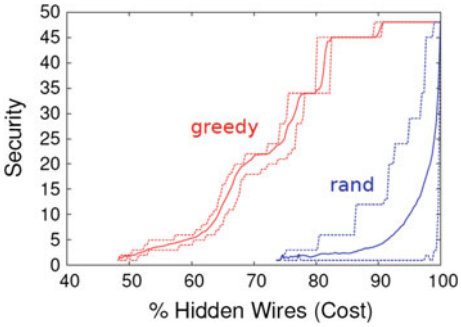


Fig. 10.9 Security versus cost trade-offs obtained using the greedy secure partitioning and random partitioning approaches

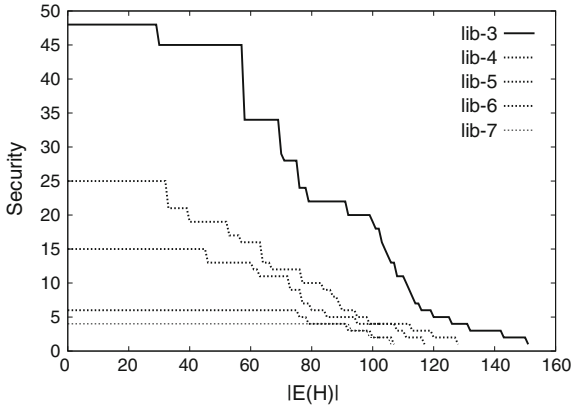


Fig. 10.10 Impact of choice of technology library on security. More diverse technology libraries yield greater security

security. Adding any further wires makes all the gates 1-secure, i.e., at this point we obtain no security at all.

Figure 10.9 shows the security versus cost trade-offs obtained by using the greedy approach on the c432 benchmark circuit from the ISCAS-85 benchmark suite. The results obtained from the greedy heuristic are compared to *randomly* hiding wires from the netlist, a strategy suggested in a white paper by Tezzaron. Note that the greedy secure partitioning approach significantly outperforms random partitioning.

The upper bound on security in Fig. 10.9 is set by the gate type that appears the *fewest* times in the netlist. In theory, if there is a unique gate in the netlist (one that appears only once), it will always be 1-secure, and correspondingly, the entire netlist would only be 1-secure. This suggests that the diversity of the technology library has a role to play in security—the more diverse the technology library, the less effective we would expect secure partitioning to be. This is indeed the case, as shown in Fig. 10.10.

10.4.2 Secure Layout

Figure 10.11 shows a secure layout tool flow that provably defends against proximity attacks [4]. In this flow, the bottom tier layout is performed independently of the top tier—since the layout tool has no information about connectivity in the top tier, this information cannot be embedded in the resulting layout. After layout, conventional routing is performed for the top and bottom tiers.

Figure 10.12 compares the results after conventional 2D layout to secure layout for the bottom and top tiers. Of note, the routing in the top tier is more "convoluted" than in the optimized 2D layout—this reflects the fact that the proximity of gates in the bottom tier reveals no information about their connectivity in the top tier. This assertion was empirically verified using a statistical hypothesis test for the layout in Fig. 10.12.

We note that although the secure layout tool flow guarantees security, it comes at extra cost because of increased wire length in the top tier. Increased wire length implies greater delay and power consumption. Table 10.1 reproduces data from Imeson et al. [13] that illustrates the relationship between delay, power, total wire

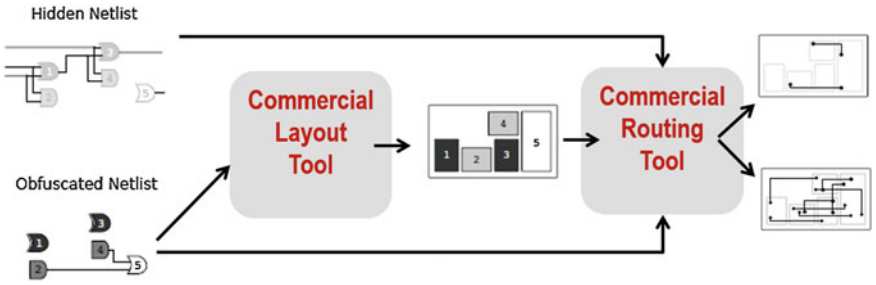


Fig. 10.11 Secure layout tool flow. The bottom tier layout is performed separately from the top tier after partitioning using a commercial layout tool. The resulting layout is independent of the connectivity of the hidden (top) tier

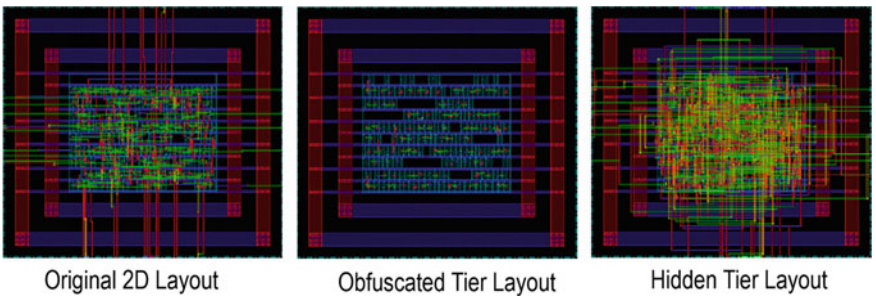


Fig. 10.12 Layout of the bottom and top tiers after secure layout, compared to the original, optimized 2D layout

Table 10.1 Power, delay, wire length, and area analysis for different levels of security on the c432 circuit. 1^* is the base circuit with no wires lifted and 48^* has all of the wires lifted

Security	Power ratio	Delay ratio	Total wire length (μm)	Total area (μm^2)
1^*	1.00	1.00	2739	1621
2	1.54	1.73	6574	4336
4	1.55	1.76	7050	4416
8	1.61	1.82	8084	4976
16	1.62	1.86	8161	5248
24	1.71	1.98	9476	6048
32	1.73	1.99	9836	6368
48^*	1.92	2.14	13058	8144

length and area as the k-security level is increased. These results are for a relatively small benchmark (c432); for large benchmarks, it is possible that beyond a certain security level the design becomes unroutable.

10.4.3 Raising the Bar on the Attacker

As a consequence of the defense mechanisms studied in this section, the attacker is now confounded. For any gate in the original netlist that the attacker wishes to modify, there are $k - 1$ other gates that can conceivably correspond to that specific gate.

An attacker could now choose one of those k gates at random and fail in her objective with a high probability. Or, the attacker could try and modify all gates, modifying one at a time (since modifying all together will not result in the malicious functionality the attacker desires). Figure 10.13 shows the minimal hardware the attacker would need to sequentially modify each of the k gates. The additional hardware significantly increases the Trojan area and likelihood of being detected in post-fabrication testing.

10.5 Future Opportunities and Challenges for Split Manufacturing

10.5.1 Reducing Cost

As we have seen, existing split manufacturing approaches can incur high cost, although they do provide strong security guarantees. Several opportunities exist to reduce the cost without compromising security.

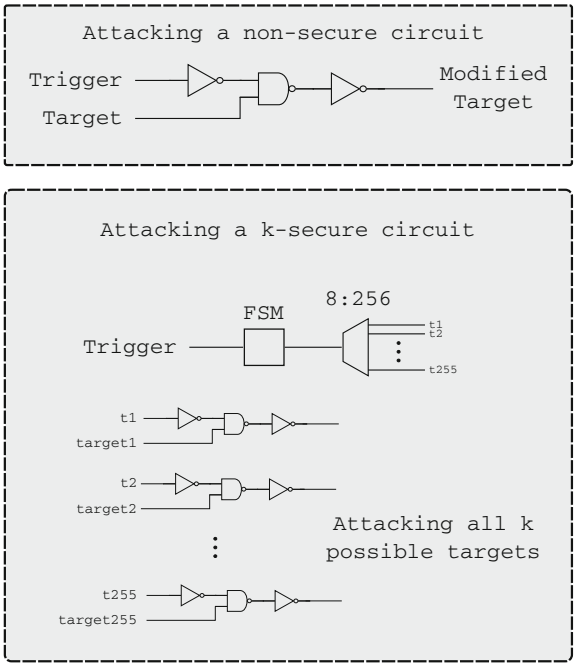


Fig. 10.13 Attack scenarios of 1- and k-secure circuits

Using decoy gates: simply duplicating a netlist and connecting only one of the netlists to the chip's IO pins (the IO pins are implemented in the hidden tier) automatically provides 2-security with low overhead in terms of number of bond points, power, and delay. The trade-off is in the area of the bottom tier, but these trade-offs might be acceptable especially in the era of dark silicon [18]. In this solution, one of the two netlists acts a decoy. The same idea can be deployed at the gate level instead of at the netlist level, i.e., the security level of individual gates can be increased by introducing decoys in the bottom tier.

Leveraging full 3D integration: Full 3D integration allows each tier to have both gates and wires. Using full 3D would allow gates to be hidden in the top tier along with wires. The top tier is still fabricated by a trusted foundry, but one with access to more mature CMOS fabrication technology. Security-critical gates in the design, for example, the sub-circuit that controls super-user privileges in a microprocessor, can be selectively implemented on the top tier.

Reducing cost of secure layout: A significant contributor to the cost of split manufacturing is secure layout, especially as the number of hidden wires exceed. The secure layout approach discussed so far guarantees that the placement of gates in the bottom tier leaks no information their connectivity—in practice, a solution that allows designers to trade off a limited amount of information leakage for reduced cost is desirable. Xie et al. [19] have taken in a step in that direction using simulated annealing-based layout flow. However, the authors do not quantify the amount of

information leaked by the proposed approach. More details on this approach can be found in Chap. 12.

10.5.2 Alternative Security Metrics

A criticism of the k-security metric is that it is perhaps too conservative. For one, it assumes that the attacker knows the original netlist, a potentially unrealistic assumption. Second, the metric requires that *every* gate in the netlist be k-secure while this might only be required of certain security-critical gates.

Jagasivamani et al. [20] have proposed alternative metrics for split manufacturing that might be relevant in the weak attack model. These metrics depend only on the partial netlist that the attacker observes, C_{PART} and not on the original netlist, C_{PUB}. Two specific metrics proposed by [20] and their relationship to k-security are discussed below:

Standard cell entropy: this metric measures the diversity of standard cells in the design using its entropy; a design that only uses cells of one type (say NANDs) has entropy 0, while one that has the same number of cells of each type has the highest possible entropy. Counter-intuitively, however, the authors advocate for lower entropy as being *beneficial* for security. This is antithetical to the traditional interpretation of greater entropy (i.e., greater disorder) being useful from a security perspective [21]. A simple example illustrates why using entropy in the way that Jagasivamani et al. suggest might be misleading.

Consider a netlist with $N/2$ gates of type 1 and $N/2$ gates of type 2 versus one with $N - 1$ gates of type 1 and a single gate of type 2. Based on the entropy metric, the former netlist has higher entropy than the latter and is therefore *less* secure. On the other hand, k-security suggests the opposite: The former netlist is $N/2$-secure while the latter is only 1-secure.

This discussion illustrates the potential danger in simply adopting metrics, like entropy, that are used in entirely different security contexts. While entropy is a useful metric for side-channel vulnerability assessment, for instance, it is not clear how it directly relates to the split manufacturing problem. In contrast, k-security has a precise attack model and relates directly to the success probability of the attacker in this model.

Neighbor connectedness: To address the threat from proximity attacks, Jagasivamani et al. [20] suggest the use of a metric that measures how likely proximal (neighboring) gates are to be connected. While this metric captures, abstractly, resilience against proximity attacks, metrics that can precisely estimate the attackers success probability are needed.

Table 10.2 Overview of split manufacturing-based obfuscation techniques

Work	Domain	Attack model	Attacker intent	Methods
Reference [13]	Logic	Strong attacker	Trojan insertion	2.5D integration Provably randomized layout
Reference [22]	Logic	Strong attacker	Trojan insertion	Obfuscated built-in self-authentication Optimized layout with filler extra filler cells
Reference [2]	Logic	Weak attacker	IP theft	FEOL/BEOL split (M1 and above) Optimized layout
Reference [23]	Logic	Weak attacker	IP theft	FEOL/BEOL split (poly and above) Obfuscated layout of standard cells
Reference [24]	SRAM	Weak attacker	Trojan insertion	FEOL/BEOL split (M1 and above) Partially randomized layout nonconventional design decoys
Reference [25]	RF	Weak attacker	IP theft	FEOL/BEOL split (M4–M7 on) obfuscated inductors and capacitors

10.5.3 Complementary Uses of Split Manufacturing

Split manufacturing can be used in a number of security-related setting that are complementary or orthogonal to the setting discussed in this chapter. Vaidyanathan et al. [24] for SRAM blocks and analog IP, while Bi et al. [25] have proposed similar ideas for RF ICs.

In particular, Vaidyanathan et al. [2] identify hard IP blocks such as SRAM arrays and analog IP as specific sources of weakness because they typically have very regular layout patterns. Even with only FEOL and M1 access, attackers can easily reverse engineer these patterns.

To address this vulnerability, the authors propose to use (a) randomized placement of peripheral logic (akin to the secure layout approach discussed before), (b) nonconventional design approaches for common logic blocks like decoders, and (c) nonstandard, application-specific features to confound the attacker (in part, similar to decoy cells discussed earlier).

Bi et al. [25] observe that for RF designs, removing the top metal layers has the effect of hiding inductors from the design. Further removing lower metal layers hides

capacitors from the design. As a consequence, the inductance and/or capacitance values in the design can be hidden from the attacker. The authors then posit that without that an attacker cannot recover these values without knowing the original design intent, for example, the center frequency. In many practical settings, RF designs for standard operating bands, for example, the designer's objectives are readily apparent from the standards documentation. Whether or not the hidden inductance and capacitance values can be reverse engineered for this stronger attack model remains to be addressed.

Otero et al. [23] have proposed techniques to obfuscate connections within standard cells, instead of across cells as we have discussed so far in this chapter. While this fine-grained level of obfuscation enables even distinct standard cells to look identical, it raises the bar on the capabilities of foundry entrusted with the BEOL connections.

Finally, Xiao et al. [22] have proposed to leverage split manufacturing in a different way, i.e., to obfuscate the implementation of built-in self-authentication circuits (BISA) on the chip. BISA cells occupy what would otherwise be nonfunctional filler cells and deter a foundry from using these cells for malicious purposes. Obfuscating BISA using split fabrication makes it even harder for a foundry to maliciously modify the original netlist without triggering an alert. More details of this approach are discussed in Chap. 11.

Table 10.2 provides a summary of the different proposals for the use of split manufacturing to achieve obfuscation.

10.6 Conclusion

Split manufacturing is an emerging technique to defend against the threat of outsourced semiconductor fabrication at untrusted foundries. By hiding a part of the design from the attacker, split manufacturing can be used to prevent IP theft and targeted hardware Trojan insertion. In this chapter, we have discussed existing threat models in the context of split manufacturing and presented state-of-the-art defense mechanisms and associated security metrics to mitigate these threats. We have also provided pointers to outstanding challenges that remain to be addressed and opportunities to further improve the effectiveness of split manufacturing.

References

1. Intelligence Advanced Research Projects Activity. Trusted Integrated Circuits Program
2. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ics using split fabrication. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2014. IEEE, pp 1–6
3. Goplen B, Spatnekar S (2007) Placement of 3d ics with thermal and interlayer via considerations. In: Proceedings of the 44th annual design automation conference. ACM, pp 626–631

4. Rajendran J, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Proceedings of IEEE/ACM conference on design automation and test in Europe, pp 1259–1264
5. Selvakkumaran N, Karypis G (2006) Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization. IEEE Trans Comput-Aid Des Integr Circuits Syst 25(3):504–517
6. Hoornaert F, Goubert J, Desmedt Y (1985) Efficient hardware implementation of the des. In: Advances in cryptology. Springer, pp 147–173
7. Ichikawa T, Kasuya T, Matsui M (2000) Hardware evaluation of the aes finalists. In: AES candidate conference, pp 279–285
8. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25
9. King ST, Tucek J, Cozzie A, Grier C, Jiang W, Zhou Y (2008) Designing and implementing malicious hardware. In: Proceedings of the 1st usenix workshop on large-scale exploits and emergent threats. USENIX Association, p 5
10. Lin L, Kasper M, Güneysu T, Paar C, Burleson W (2009) Trojan side-channels: lightweight hardware trojans through side-channel engineering. In: Cryptographic hardware and embedded systems-CHES 2009. Springer, pp 382–395
11. Shiyanovskii Y, Wolff F, Rajendran A, Papachristou C, Weyer D, Clay W (2010) Process reliability based trojans through nbti and hci effects. In: 2010 NASA/ESA conference on adaptive hardware and systems (AHS). IEEE, pp 215–222
12. Yang K, Hicks M, Dong Q, Austin T, Sylvester D (2016) A2: analog malicious hardware
13. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation. In: Proceedings of USENIX security, pp 495–510
14. Boneh D, DeMillo RA, Lipton RJ (1997) On the importance of checking cryptographic protocols for faults. In: Proceedings of the international conference on theory and application of cryptographic techniques, pp 37–51
15. Hicks M, Finnicum M, King ST, Martin MMK, Smith JM (2010) overcoming an untrusted computing base: detecting and removing malicious hardware automatically. In: IEEE symposium on security and privacy, pp 159–172
16. Bhunia S, Hsiao MS, Banga M, Narasimhan S (2014) Hardware trojan attacks: threat analysis and countermeasures. Proc IEEE 102(8):1229–1247
17. El Massad M (2014) On the complexity of the circuit obfuscation problem for split manufacturing
18. Shafique M, Garg S, Henkel J, Marculescu D (2014) The eda challenges in the dark silicon era. In: Design automation conference (DAC), 2014 51st ACM/EDAC/IEEE. IEEE, pp 1–6
19. Xie Y, Bao C, Srivastava A (2015) Security-aware design flow for 2.5 d ic technology. In: Proceedings of the 5th international workshop on trustworthy embedded devices. ACM, pp 31–38
20. Jagasivamani M, Gadfort P, Sika M, Bajura M, Fritze M (2014) Split-fabrication obfuscation: metrics and techniques. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 7–12
21. Cachin C (1997) Entropy measures and unconditional security in cryptography. PhD thesis, Swiss Federal Institute of Technology Zurich
22. Xiao K, Forte D, Tehranipoor MM (2015) Efficient and secure split manufacturing via obfuscated built-in self-authentication. In: 2015 IEEE international symposium on hardware oriented security and trust (HOST). IEEE, pp 14–19
23. Otero CTO, Tse J, Karmazin R, Hill B, Manohar R (2015) Automatic obfuscated cell layout for trusted split-foundry design. In: 2015 IEEE international symposium on hardware oriented security and trust (HOST). IEEE, pp 56–61
24. Vaidyanathan K, Liu R, Sumbul E, Zhu Q, Franchetti F, Pileggi L (2014) Efficient and secure intellectual property (ip) design with split fabrication. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE pp 13–18

25. Bi Y, Yuan J-S, Jin Y (2015) Split manufacturing in radio-frequency designs. In: Proceedings of the international conference on security and management (SAM), p 204. The steering committee of the world congress in computer science, computer engineering and applied computing (WorldComp)

Chapter 11
Obfuscated Built-In Self-authentication

Qihang Shi, Kan Xiao, Domenic Forte and Mark M. Tehranipoor

11.1 Introduction

As discussed in Chap. 1, changing economic trends have resulted in a global IC supply chain. For all but a few semiconductor companies, IC fabrication is now being performed by contract foundries and outside the purview of original intellectual property (IP) owners. There are serious concerns about whether trust between an IP owner and such fabs/foundries can be established [1]. A untrusted foundry with malicious intent could conduct a number of attacks including IP piracy [2], IC cloning/overproduction [3, 4], and hardware Trojan insertion [5].

A great deal of research has been performed to address the attacks associated with untrusted foundries. One approach introduced by DARPA [6] is split manufacturing. In this approach, an untrusted foundry manufactures the front-end-of-line (FEOL) part of the IC (the transistors and lower metal layers) and then ships it to a trusted foundry to deposit back-end-of-line (BEOL) layers, which includes the remaining metal layers (see Fig. 11.1). By concealing complete layout information, split manufacturing prevents the untrusted foundry from stealing IP information or committing attacks that require reverse engineering of the design.

Q. Shi (✉)
ECE Department, University of Connecticut, Storrs, CT, USA
e-mail: qihang.shi@engr.uconn.edu

K. Xiao
Intel Corporation, Santa Clara, CA, USA
e-mail: kan.xiao@intel.com

D. Forte · M.M. Tehranipoor
ECE Department, University of Florida, Gainesville, FL, USA
e-mail: dforte@ece.ufl.edu

M.M. Tehranipoor
e-mail: tehranipoor@ece.ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_11

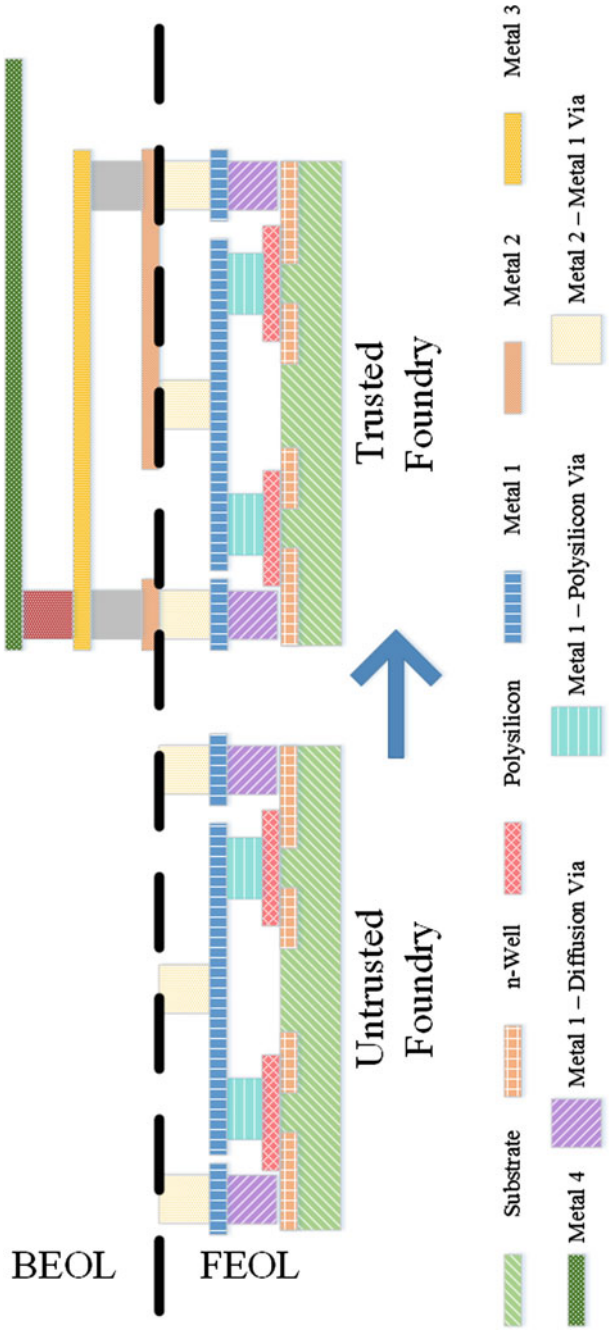


Fig. 11.1 Typical split manufacturing arrangement (assuming split made between Metal 2 and Metal 1 layers)

There has also been a lot of work on protection against the threat of hardware Trojans. Techniques against hardware Trojan insertion can be grouped into two categories, depending on how they address the issue.

- *Detection*: The first category includes Trojan detection techniques, such as functional verification, side-channel signal analysis, or by new front-end design techniques such as design-for-trust [7–15]. Such techniques detect the existence of hardware Trojans by generating a signature of the circuit under test (CUT) and then classifying the CUT with this signature. To perform classification, they require a golden model, i.e., signature of a copy of the same circuit that is known to be free from hardware Trojans. Unfortunately, it remains doubtful whether golden models can be acquired for real-world applications (e.g., commerical off-the-shelf parts). In addition, process variations introduce errors in classification, especially for small, hard-to-detect Trojans.
- *Prevention*: The second category includes hardware Trojan prevention techniques that stop an adversary from inserting a Trojan in the first place and also do not require a golden model. Built-in self-authentication (BISA) is the first proposed technique to prevent hardware Trojan insertion in the circuit layout and mask. By occupying all available spaces for Trojan insertion and detecting malicious removal through built-in self test, BISA is able to deter hardware Trojan insertion without the requirement of golden models and classification errors introduced by process variation.

However, problems remain. Techniques against IP piracy do not usually consider the threat of hardware Trojan insertion. Conversely, techniques against hardware Trojan insertion (such as BISA) do not consider IP theft. Unfortunately, IP piracy and Trojan insertion often go hand-in-hand, with the same adversary capable of pirating the IP as well as inserting a Trojan.

An untrusted foundry is characterized by two attributes: (1) The service of an untrusted foundry is imperative (e.g., due to the high cost associated with advanced nodes). Otherwise, security could be ensured by simply using a trusted foundry and (2) an untrusted foundry cannot be trusted with the security of the intellectual property (IP). Hence, additional measures need to be taken to prevent potential IP piracy. Since both attributes need to be present for split manufacturing to be necessary, we can assume that all untrusted foundries in the adversarial model possess both attributes.

Therefore, in order to ensure the security of fabrication with split manufacturing, we must ensure security against all possible attacks from an untrusted foundry. Due to attribute (1), it is likely that the untrusted foundry is aware of its own criticality, so there is no disincentive for the untrusted foundry to refrain from all possible attacks given its technical capability. At the same time, as a result of attribute (2), the IP owner has no reason to trust the untrusted foundry. It can be reasonable to assume that the untrusted foundries will likely try all attacks in their arsenal, and IP owners will desire an overall solution that can secure their design against all attacks. Therefore, a complete security solution to address the threats from an untrusted foundry is needed. Unfortunately, both split manufacturing and BISA are limited when it comes to comprehensively countering the problem of an untrusted foundry.

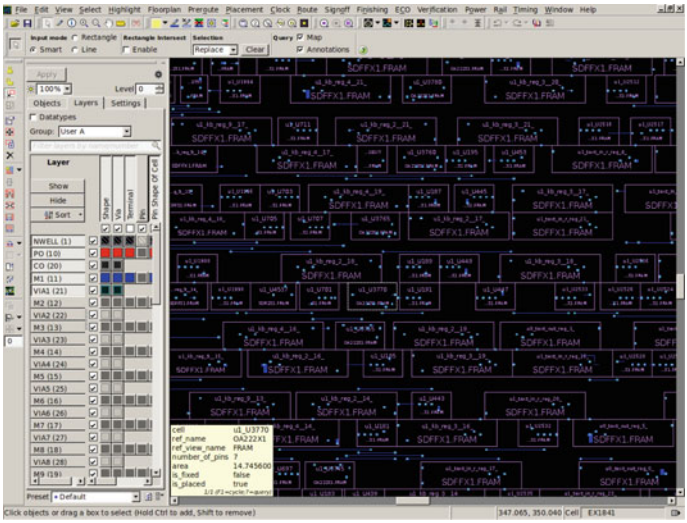
11.1.1 Limitations of Built-In Self-authentication (BISA)

To explain how built-in self-authentication (BISA) works, we need to first establish how normal back-end design of an IC works. Back-end design of an IC is usually in the form of a netlist, i.e., a list of gates and how the nets connect them. This netlist is used to build a layout, which can be used to generate a photolithography mask, which in turn is used to fabricate the IC. The layout phase consists of at least two steps: (i) placement of the gates and (ii) routing of the nets. Normally, during the placement step of the back-end design, gates in the circuit are placed at optimized locations based on density and routability [16]. This leaves so-called *white spaces*, i.e., spaces in the layout that are not filled by standard cells in the layout (see Fig. 11.2a). White spaces have to exist, because gates placed too close to each other will make routing of nets very difficult or impossible. Power dissipation as well as cross talk due to high-frequency gate operations in close vicinity could also generate enough heat and noise in the IC to render it useless.

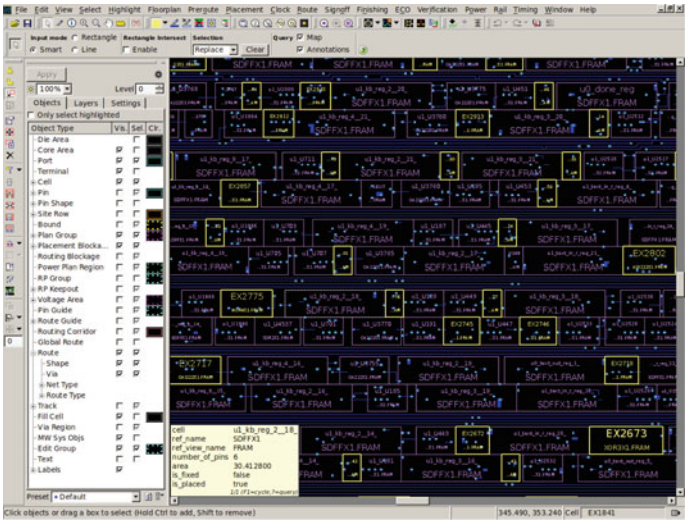
For security-oblivious design purposes, these white spaces are usually filled with *filler cells* to serve as decoupling capacitors and/or extension of power tracks [17]. For such purposes, a filler cell design containing only power tracks, or power tracks and decoupling capacitors, is usually adopted, since they consume less leakage power than standard cells. However, such a simple and unsupervised design also makes these filler cells prone to malicious removal by Trojan inserters in order to make room for hardware Trojans. This is because white spaces are not monitored by any logic. Decoupling capacitors serve performance purposes rather than functional needs. The very reason white spaces exist is because these spaces cannot be occupied with normal functional logic. The problem is that Trojan gates are mostly dormant throughout the host IC's life span. If white spaces or decoupling capacitors are replaced by Trojan gates, it would likely incur a mild level of performance loss (e.g., slight drop in operating frequency). However, the magnitude of a Trojan impact could be so low that designers could simply attribute it to transient conditions. The symptom would be comparable to the case of a mild fever in humans: The patient usually attributes it to stress or other temporary factors, without suspecting any major problem at play.

BISA prevents hardware Trojan insertion by occupying white spaces with testable standard cells instead of non-functional filler cells (see Fig. 11.2b). All inserted BISA cells are organized to form a built-in self test (BIST) circuitry, so that they can be tested to verify that no BISA cells have been removed. BISA is designed so that removal of its member cells will lead to a BIST failure, so that no attempt to make room for hardware Trojans will evade detection. As a technique against Trojan insertion, BISA is highly effective.

Unfortunately, simply securing the design against all hardware Trojan insertion is insufficient at addressing the intended adversary. As we have discussed above, untrusted foundries can and should be expected to commit all kinds of attacks within its capabilities. Unfortunately, the intended attacks BISA is capable of securing against are quite limited in scope. It is unwise to assume an untrusted foundry willing to attempt Trojan insertion will not attempt to commit other attacks, e.g., steal IC



(a) Layout of an AES crypto-core, with unfilled white spaces.



(b) Layout of an AES crypto-core, with white spaces filled with BISA cells.

Fig. 11.2 Layout of an AES crypto-core, with unfilled white spaces

layout in order to perform IP piracy or IC cloning [2, 3]. Therefore, BISA is not a complete solution. Moreover, specific attacks exist for BISA. For example, if the attacker can distinguish BISA cells from original circuitry, it is theoretically possible to perform a "redesign attack" so that BISA detection could be evaded. We shall discuss limitations of BISA in more details in Sect. 11.2.2.

11.1.2 Limitations of Split Manufacturing

Split manufacturing prevents all attacks that require complete knowledge of the whole layout, which also includes attacks against BISA such as identification of BISA cells. However, not all attacks require complete knowledge of the whole layout. One example of these kinds of attacks is untargeted Trojan insertion [18].

A "targeted" hardware Trojan is designed to trigger at certain specified states of the original circuit or to maliciously modify a specific function of the original circuit, or both. Hence, it requires knowledge of the related modules in the original circuitry. In the case where the adversary is an untrusted foundry, this knowledge will also require their locations on the FEOL layout. In a split fabricated IC, at least some nets consist of BEOL interconnects, whose information is thus denied to the untrusted foundry. This will complicate or deter targeted Trojan insertion by making it harder for the untrusted foundry to identify the site that he wants to insert the Trojan trigger or payload. This deterrence is most significant when the split is optimized to maximize the effect of obfuscation, e.g., through the use of wire lifting to maximize security rating k as described earlier in the chapter on 3D IC- based obfuscation. To overcome this obfuscation, the untrusted foundry will have to insert Trojan payloads at all possible sites and/or generate trigger signals using net values from all possible sites, proportional to the design's security rating k. This will cause the Trojan to be proportionally larger and easier to discover and/or trigger.

On the other hand, untargeted Trojan does not require knowledge of the original circuitry [19] and can still pose a threat to split manufactured ICs. Such a Trojan can be designed as long as a sufficiently rare triggering condition can be produced. For example, consider an original circuitry where only front-end-of-line (FEOL) part of the layout is visible, i.e., the portion of the layout visible to an untrusted foundry under split manufacturing. In this scenario, all cells of the original circuitry are visible; therefore, all pins are available for the hardware Trojan. Granted, split manufacturing is effective in denying the untrusted foundry access to backend-of-line (BEOL) information, and the untrusted foundry would not be able to distinguish the functionality the signals at these pins may serve; however, hardware Trojan payloads do not depend on knowing the functionality of the original circuitry, and therefore neither does it require such access to begin with. As long as the Trojan trigger is hard enough to trigger during manufacturing test, the Trojan is capable of evading detection and can cause security concerns.

One way to do this is to choose structurally hard-to-reach nets for the Trojan trigger inputs. Since such nets can be expected to be difficult for manufacture tests to place values at without knowing what functions they serve, it is also unlikely for the manufacture test patterns to trigger the Trojans. This is shown in Fig. 11.3. In this example, D-pins (i.e., input pins) of flip-flops are used to generate the trigger signal. D-pins are a suitable option because they are structurally hard to reach for manufacture test patterns. As is shown in Fig. 11.3, D-pins of flip-flops are at one end of timing paths. During manufacture test, test patterns are fed into the flip-flops with scan paths and launched from the other end of timing paths as shown in the

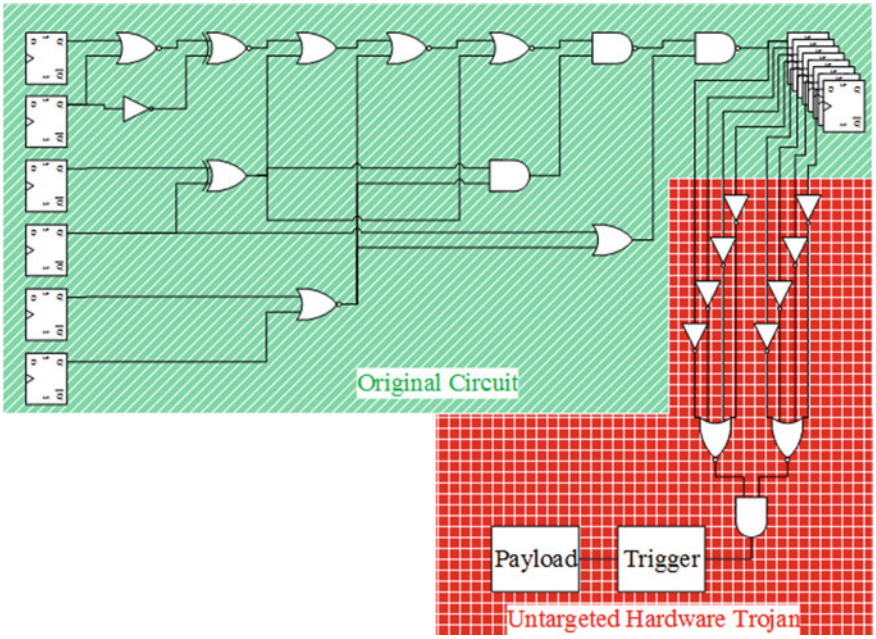


Fig. 11.3 One approach to insert untargeted hardware Trojan into split manufacturing protected layout

figure. This naturally makes D-pins the furthest away from the origin of manufacture test patterns, in terms of number of gates in between. Since each gate in between exponentially increases the number of inputs necessary to control any net (i.e., to place a desired logic state at a given net), this makes D-pins hardest to control from a test coverage point of view. In other words, the number of tests necessary to completely traverse the complete state space of the D-pin nets in order to trigger the Trojan will be too large for manufacture test to implement. Consequently, using D-pins as inputs to generate trigger signal makes the hardware Trojan very hard to discover with manufacture tests.

Another possible way for the proposed hardware Trojan to be discovered during manufacture test is through its impact on path delay. Since D-pin nets are used as inputs to the Trojan's trigger, the trigger adds load capacitance to the net, which adds delay to paths that contain that net. There exists a realistic possibility that this additional delay overhead will change the timing of the affected paths significantly enough to lead to discovery during delay test. Another realistic constraint is the distance between available space for a Trojan gate to be inserted and the D-pin. If the minimum distance is too large, it could also lead to a large resistance on the interconnect between the D-pin net and the trigger gate. This can also add delay, possibly resulting in Trojan detection.

A simple solution to these issues is to add the smallest gate to buffer the trigger input. This is done in the example shown in Fig. 11.3 with inverters. An inverter is the smallest cell in a standard cell library. It can help in minimizing the distance and input capacitance that contributes to delay change to paths that contain D-pin nets. Its weakness is its fan-out load capacity, a quality essential in meeting timing requirement, but this is rather unimportant for inserting hardware Trojans since Trojans usually do not have rigid timing requirements. Therefore, to evaluate the possibility of untargeted Trojan insertion in a layout, one may simply insert filler cells into it, but use an inverter instead of usual filler cell models. This will insert inverters at all available white spaces for it, and a simple geometric search using a reasonably set radius centered at the coordinate of D-pins will yield all possible inverter site candidates that are reasonably close to the D-pins. The layout editor may then be employed to create an interconnect between the inverter inputs and the D-pins to study the impact on delays of paths that ends at those D-pins. Indeed, in one such experiment, we performed with an untargeted Trojan insertion on an open source advanced encryption standard (AES) crypto-core, insertion at 2601 of all 2602 D-pins does not even impact worst-case delays of all paths containing them. In a real implementation where parametric drift of devices due to fabrication can lead to path delay variation of 10 % or more [20], the timing impact will be even less distinguishable.

Untargeted hardware Trojans do not target specific functions of the original circuitry and therefore cannot commit attacks that require knowledge of such functions. Nevertheless, untargeted hardware Trojans are still capable of degrading the performance and/or reliability of manufactured ICs or triggering a denial-of-service (DoS) attack in critical control systems [21]. These are threats that need to be addressed.

Like BISA, split manufacturing has its own share of issues, in addition to the problem of lacking security against untargeted Trojan insertion. As has been introduced in Chap. 10, k-security is an existing security metric of split manufacturing. It evaluates the effectiveness of obfuscation (via denial of BEOL information) by calculating the least number (the security rating k) of mutually indistinguishable gates or nets that exist for any observable gate or net in FEOL. This definition is mathematically sound, but places some rather heavy restrictions on the original circuitry design. For example, unconnected nets are indistinguishable from each other by default, but unconnected gates are not. An inverter is distinguishable from an AND gate and so is an AND gate from another AND gate with twice as much fan-out capacity. As mentioned in Chap. 10, normal synthesis and netlist optimization yield many standard cell models with very few numbers of instances, which will seriously restrict how high k can reach. Consequently, a design that optimizes its k-security rating will have to restrict the standard cell models it uses. Indeed, the authors in [22] restricted the standard cell model count in their experiments between 3 and 7, while an otherwise unconstrained synthesis of an AES core yields 37. This restriction will undoubtedly lead to elevated area and power overhead as well as performance loss. Moreover, white spaces also exist in split fabricated ICs, whose existence and spatial distribution on the FEOL layout could be used to deduct BEOL connections as part of proximity attack [23] (see Chap. 10).

11.1.3 Chapter Overview

While split manufacturing and BISA are excellent techniques, they are still incomplete. An apparent solution is to create a technique that combines the best of both worlds. In this chapter, we term this combined technique as the *obfuscated* built-in self-authentication (OBISA). This chapter provides background information to OBISA, investigation of different ways in which OBISA can be implemented, as well as how protection against both IP theft and hardware Trojan insertion can be better implemented by OBISA than BISA or split manufacturing alone. The rest of this chapter is organized as follows: Sect. 11.2 provides background on BISA and elaborates on existing problems and weakness of BISA that could be improved; Sect. 11.3 reviews split manufacturing techniques introduced in the previous chapter, comments on their relevance to BISA, and investigates possible benefits of their integration with BISA; Sects. 11.4 and 11.5 investigate two possible ways that OBISA can be implemented and provide their implementation flow, their respective strengths, and potential attacks; and finally, Sect. 11.6 concludes this chapter.

11.2 Built-In Self-authentication (BISA)

The purpose of BISA, as we briefly visited in Sect. 11.1, is to prevent hardware Trojan insertion. This objective is achieved by implementing two major features:

1. occupying all white spaces in the layout that could be used for Trojan insertion and
2. ensuring no inserted filler cell for the purpose of preventing Trojan has been removed.

As was discussed in Sect. 11.1, existing design flow already occupies white spaces in the layout with filler cells. That alone is not sufficient to deter malicious Trojan insertion because conventional filler cells are not under any kind of surveillance to prevent them from being removed by an attacker in order to make room for hardware Trojans. Therefore, the second feature is really essential to prevent hardware Trojan insertion. BISA implements this by replacing filler cells with combinational logic cells from the standard cell library and then organizing them into a BIST. Removal of any BISA cell will lead to changes in BIST test signatures and thus detection.

As shown in Fig. 11.4a, BISA consists of three parts: the BISA circuit under test, the test pattern generator (TPG), and the output response analyzer (ORA). The BISA circuit under test is composed of all BISA cells that have been inserted into unused spaces during layout design. In order to increase its stuck-at fault test coverage, the BISA circuit is divided into a number of smaller combinational logic blocks, called BISA blocks as shown in Fig. 11.4a. Each BISA block can be considered as an independent combinational logic block. The TPG generates test vectors that are shared by all BISA blocks. The ORA will process the outputs of all BISA blocks

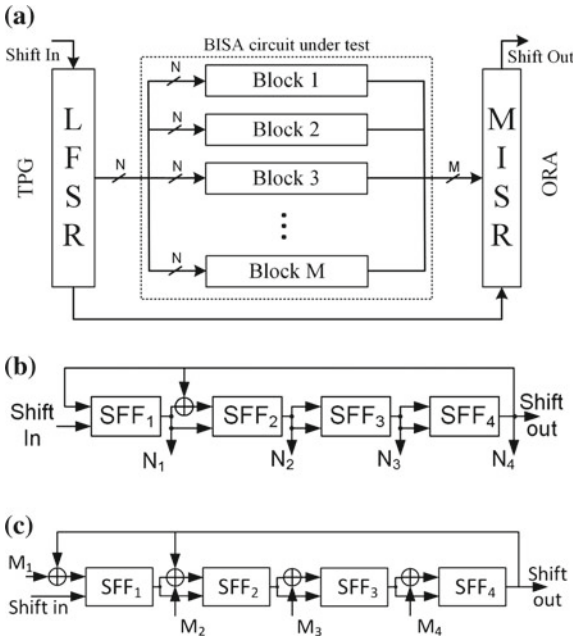


Fig. 11.4 Structure of **a** BISA, **b** four-stage LFSR, and **c** four-stage MISR

and generate a signature. TPG has been implemented with linear feedback shift register (LFSR), while ORA has been implemented with multiple input signature register (MISR) in prior work [24]. Examples of four-stage LFSR and four-stage MISR are shown in Fig. 11.4b, c. They are used in the generation of random vectors and compression of responses into a signature. SFF in the figure represents a scan flip-flop. Other types of TPG and ORA can also be applied [25].

The main advantage of BISA is that it does not require a golden chip/model. Most other researches on addressing the issue of Trojan insertions have focused on the development of:

1. hardware Trojan detection techniques using functional verification and side-channel signal analysis (applied post-silicon) or
2. new design techniques to improve detection capabilities (applied to front-end design) [26].

Most detection approaches need golden chips either fabricated by a trusted foundry or verified to be Trojan-free through reverse engineering, both of which are prohibitively expensive, if not impossible in many scenarios. Since BISA relies on logic testing, process variation is not a factor either, as compared to Trojan detection techniques based on side-channel analysis. As an additional advantage, impact of BISA on original design in terms of area and power is also negligible.

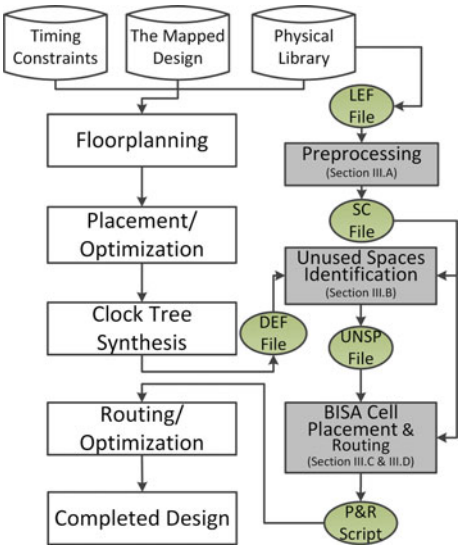


Fig. 11.5 BISA design flow

11.2.1 Implementation Flow

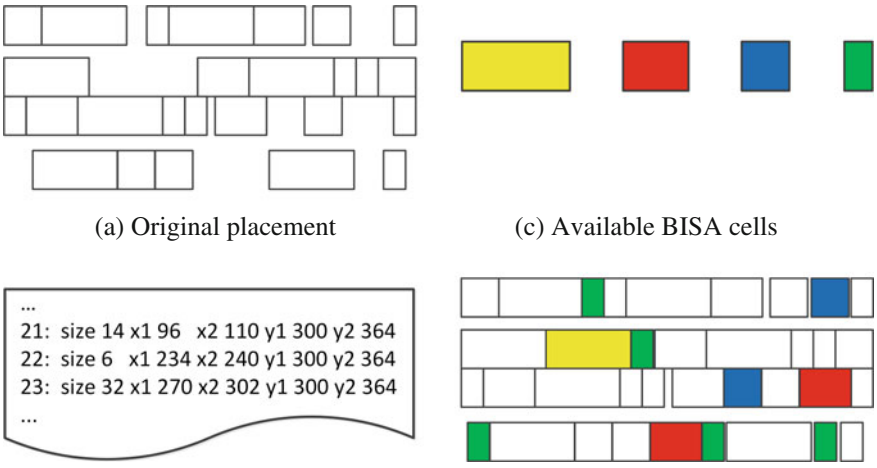
Figure 11.5 shows the BISA design flow and where it fits within the conventional ASIC design flow. The white rectangles in the figure are steps taken in a conventional ASIC design flow, and the gray ones are the additional steps for inserting BISA circuitry.

The first step in BISA design flow is called *preprocessing*, where information such as dimensions of each standard cell, the number of input pins, and the name of a cell is acquired from the standard cell library for use in later steps.

After obtaining the necessary information for all standard cells, BISA cells will be selected from them and marked according to the following criteria:

1. BISA cells must be the minimum-sized cell for every logic function, so they are resistant to a resizing attack by the adversary (see Sect. 11.2.2).
2. The amount of decoupling capacitance the cells can provide and the input count should be considered as well. Fewer inputs help to improve test coverage; therefore, a normalized input count is used here to represent the number of inputs of a standard cell if the same cell has the same area of the minimum-sized cell (e.g., INVx0 in Synopsys 90 nm library).
3. The smallest cell in the library must also be included in order to ensure that no cell can be inserted in any remaining unused space.

The second step is called *unused space identification*, where the BISA flow identifies white spaces by using a matrix to record the state of each point in the layout. Every standard cell placed in the layout will be processed one by one, and eventually,



(a) Original placement (c) Available BISA cells

(b) Unused spaces file (.unsp) (d) Placement after BISA insertion

Fig. 11.6 BISA cell insertion and placement

the matrix reveals the location and size of unused spaces. The matrix is then used to insert BISA cells into these spaces, as shown in Fig. 11.6.

The final step in the flow is to *place and route* BISA cells. Placement solutions can be found and optimized (e.g., dynamic programming algorithm was used in [24]) based on white space identified in the previous step. Optimization of BISA placement is an interesting problem; however, it is not of central importance in BISA. On the other hand, all placed BISA cells need to be connected into a number of combinational BISA circuits (referred to as BISA blocks) to ensure test coverage of the BISA circuit. Test coverage is a key issue for BISA since its security relies on its capability to discover tampering of its constituent cells. A higher test coverage leads to a higher credibility of results from BISA. Several approaches are employed to enhance stuck-at fault test coverage:

- First, create as many BISA blocks as possible to make each BISA block with fewer gates so that higher test coverage is easier to achieve. Since the output of every BISA block will connect to MISR, the number of BISA blocks is determined by the size of MISR. If M is the size of MISR, all placed BISA cells are divided into M groups (BISA blocks).
- Second, redundant gates could deteriorate controllability and observability of the circuit and lower the test coverage significantly, so a tree-structure circuit is constructed to eliminate redundant gates, as shown in Fig. 11.7. If every input is independent of other inputs in a tree-structure circuit, every net is controllable and observable, so the theoretical test coverage of stuck-at fault is 100 %. Here, a tree-structure BISA block is constructed according to the sequence of cells in a block set.

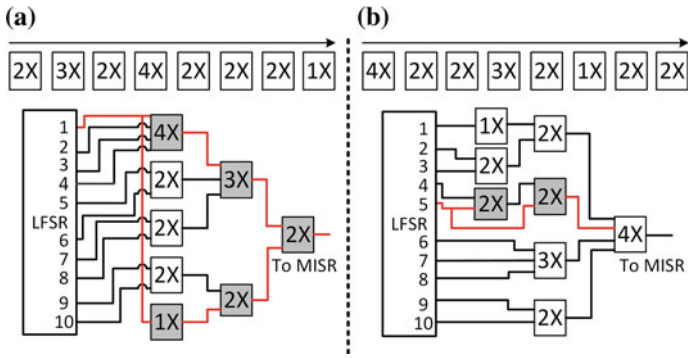


Fig. 11.7 Routing a BISA block

Figure 11.7 shows that two different sequences lead to two different tree-structure circuits. The first gate becomes the root of the tree-structure circuitry, i.e., it is on the top (first) level of tree. The outputs of the next x cells (x being the number of inputs of the root cell) are connected to its inputs as its children cells, on the second level. The same is repeated on the third level to connect new cells to cells on the second level. Cells are sequentially connected to cells on upper levels until all of them are processed, as shown in Fig. 11.7a, b. After complete routing in each block, all inputs of each block should connect to the LFSR sequentially to avoid sharing of inputs. In the end, the M bit outputs from M BISA blocks connect to a MISR with size of M.

11.2.2 Possible Attacks Against BISA

To attack BISA, an attacker would have to find a way to remove enough cells to make room for his Trojan insertion, without triggering detection by BISA. Depending on targets and methods used in this removal, several possible strategies exist to attack BISA:

1. attack TPG or ORA of BISA,
2. directly remove cells from BISA or original circuitry. This is known as a *removal attack*,
3. Replace BISA or original circuitry with a smaller functionally equivalent circuit. This is known as a *redesign attack*. In particular, if standard cells of greater fan-out are replaced with their equivalent counterparts of lower fan-out, this is known as a *resizing attack*.

Of the three possible attacks against BISA, attacking TPG or ORA is the least likely to succeed. BISA uses pseudo-random pattern to perform BIST, which makes it very easy to increase pattern count and consequently very difficult for the attacker to make sure all responses of the modified TPG and/or ORA will stay the same for arbitrarily many patterns. Similarly, direct removal of BISA cells is unlikely

to succeed as they are covered by BISA test coverage during BISA insertion. It is indeed possible to remove cells from original design as long as they do not serve crucial functions. However, design optimization and test coverage will minimize this opportunity for the attacker.

The attack that is most likely to succeed against BISA is the redesign attack. Both BISA and the original circuitry can be targeted in this attack. Redesign attack on original circuitry is restricted by manufacture test as well as other detection-based anti-Trojan approaches. If attackers redesign the original layout for Trojan insertion, moving gate locations and altering wire interconnections will result in significant changes in the electrical parameters, such as power and path delay. These can be detected much more easily by delay-based and power-based techniques [27–36].

It is more likely for the removal attack to succeed against BISA cells since the BISA cells cannot be expected to meet a uniform timing constraint: Their insertion has to prioritize area occupation. The attacker may try to first reverse engineer BISA circuitry—a monumental effort, but not impossible—and then perform logic optimization, hoping to remove redundant BISA cells. It is possible to further secure BISA cells by performing this optimization on BISA design to prevent this particular attack. However, the attacker can also choose to design a custom cell functionally equivalent to several BISA cell at the cost of fan-out and/or delay in order to make room for Trojan insertion. Prevention of such an attack would require anticipation of all possible custom cell designs that are functionally equivalent to any combination of BISA cells. That is not likely feasible except for very small BISA circuitry. Therefore, this attack, called the *custom cell attack*, is also not a likely threat to BISA security.

11.2.3 Limitations of BISA

BISA adversarial model is more or less limited to Trojan attacks after the back-end design. This is because Trojan attacks on the original circuitry can be expected to be detected by other techniques such as functional and delay tests. This leaves untrusted foundry as the most likely adversary to BISA.

One valid limitation of BISA is its inability to prevent IP piracy or IC cloning—a task perfect for split manufacturing to tackle. Another small limitation is that due to the possibility of resizing attacks (see Sect. 11.2.2), all BISA cells have to be of the smallest variant in area among standard cells of the same function, which might make it easier for the attacker to identify them.

11.3 Combining BISA with Split Manufacturing

In light of the respective limitations of BISA and split manufacturing, it makes sense to combine them so that benefits of both techniques can be reaped. We henceforth term the combined technique as obfuscated BISA (OBISA).

The most apparent advantage of the resulting technique is the security against untargeted hardware Trojan insertion, as well as security against IP piracy and IC cloning, both of which are primary strengths of BISA and split manufacturing, respectively. Combining with split manufacturing can also make the resulting OBISA technique secure against redesign attack, since the attacker must first identify which existing cells are connected together before designing a functionally equivalent circuit to replace these existing cells. This will be much harder if the designer lifts the wires that connect them to BEOL, so that BISA structure becomes indistinguishable from the original circuitry.

The obfuscation effect from split manufacturing can further enhance OBISA beyond protection against redesign attack. As mentioned previously in Sects. 11.2.2 and 11.2.3, conventional BISA requires functional and delay tests as well as detection-based anti-Trojan techniques for the security of the original circuitry against redesigning. Although this does not make BISA insecure, detection-based anti-Trojan techniques do rely on golden models for effectiveness. Reliance on these techniques erodes BISA's advantage of not requiring a golden model. Combining with split manufacturing makes the threat of redesign attack much less of a problem due to security of BEOL information and therefore reduces the necessity of using detection based anti-Trojan techniques (which often require a golden model that is not always available). In addition, obfuscation also deters reverse engineering. A relaxed threat from reverse engineering could allow relaxation of other limitations that was not possible with BISA alone, e.g., the requirement of only using the smallest standard cells may not be necessary if the designer can be reasonably confident that OBISA cells will not be identified.

On the other hand, obfuscation in OBISA could also benefit from BISA insertion, owing to additional cells and FEOL interconnects that BISA insertion introduces to the layout. Since the purpose of split manufacturing is to hide BEOL information, most theorized attacks and security metrics (see Chap. 10) define split manufacturing security as anonymity of broken interconnects in FEOL layout [22, 23, 37]: In other words, even and uniform distribution of FEOL features help split manufacturing security. Additional cells and interconnects introduced by BISA circuitry can be very helpful here, because they can be used to compensate rare gate models and interconnect types so that signatures of original circuitry can be hidden. Without such additional obfuscating material, cells of rare gate models have to be banned during synthesis optimization of original circuitry [22], leading to performance loss. Further, proximity attack based on FEOL-observable distribution of gates as well as white spaces could also be foiled by occupying white spaces and compensating spatial distribution of gate types with BISA cells.

To summarize, a combined OBISA technique could derive advantages from both split manufacturing and BISA, as shown in Fig. 11.8.

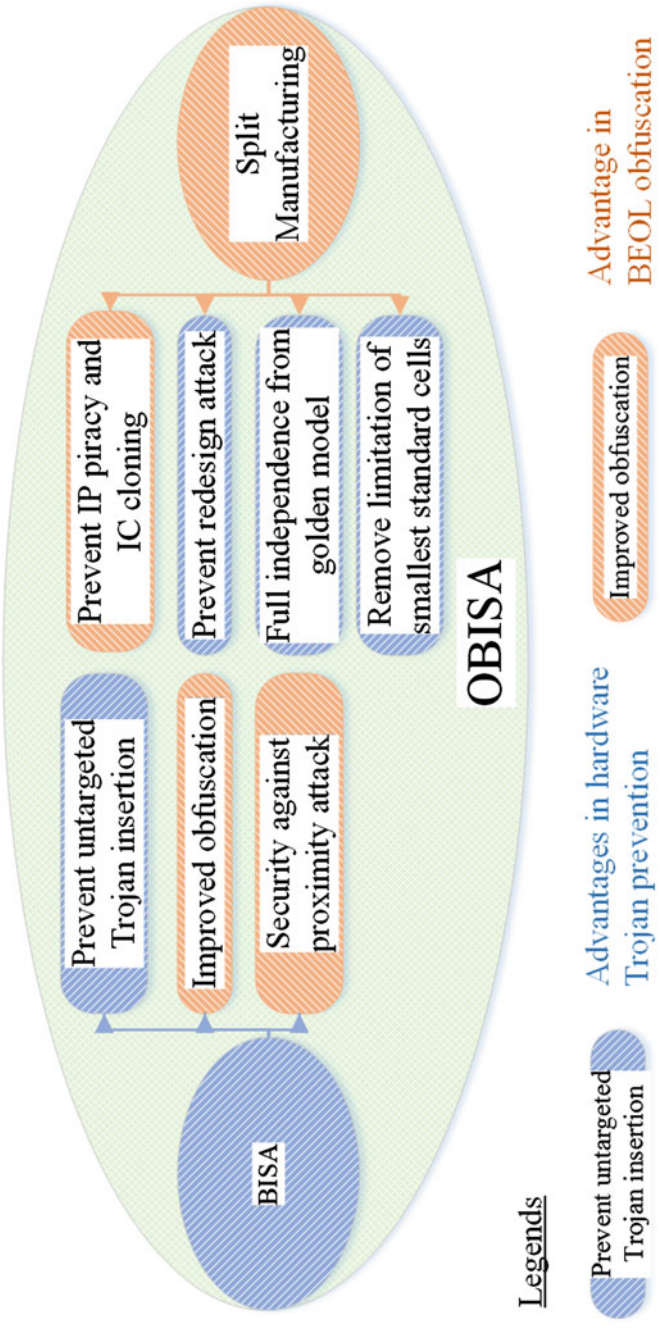


Fig. 11.8 Possible advantages of obfuscated BISA (OBISA) technique

11.3.1 Trade-Off Between BEOL Security and Computational Cost

Split manufacturing techniques, as discussed in Chaps. 10 and 12, come in a number of different implementations. It can be implemented simply by separating FEOL from BEOL at a certain layer without any modification to the design flow; alternatively, wires and/or cell placements in FEOL can be modified to avoid information leakage [23, 37]. In the most secure form of implementation, a list of wires to be elevated to BEOL fabrication can be optimized to obtain a mathematically provable metric of obfuscation [22]. Unfortunately, this is also computationally the most complicated. Generally speaking, there is a trade-off between security and computational cost among split manufacturing techniques.

Depending on how much complexity is dedicated to the split manufacturing side of the technique, OBISA can have different implementation approaches. In this chapter, we introduce two sample approaches:

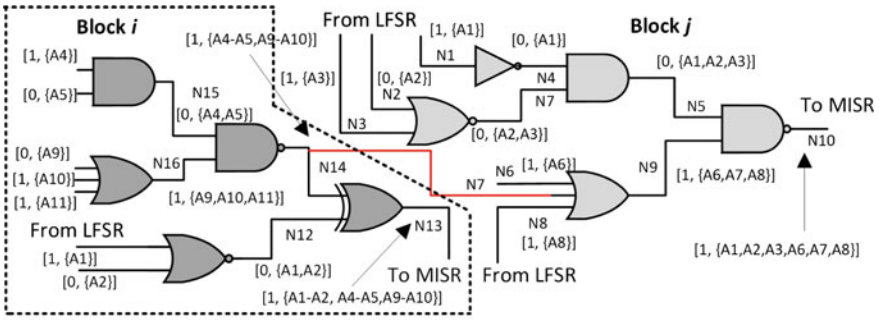
1. Approach A assumes minimum computational cost is dedicated to split manufacturing (i.e., assume split manufacturing is simply implemented by separating FEOL from BEOL at a certain layer) and
2. Approach B assumes maximum level of security is desired (i.e., wire lifting—as was introduced in Chap. 10—is optimized using the notion of k-security).

11.4 Approach A: Obfuscated Connection

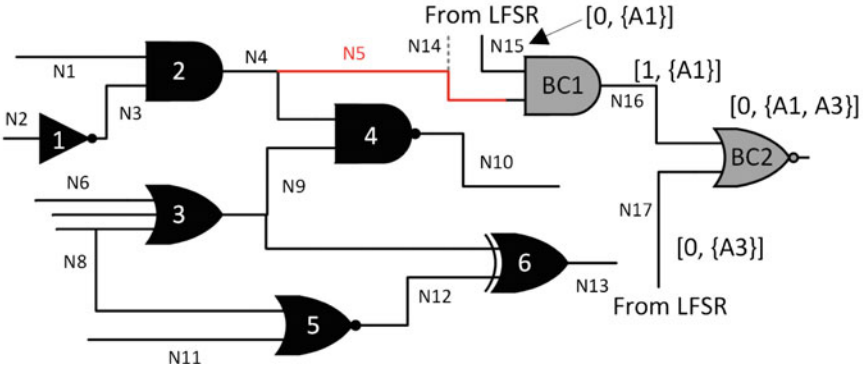
As has been discussed in Sect. 11.3.1, there is a trade-off between security and computational cost among split manufacturing techniques. There is also a trade-off between security and fabrication difficulty, in terms of which layer is used to split the design between FEOL and BEOL. Generally speaking, splitting at higher layers would lead to easier fabrication, higher yield, and lower requirements on the technical capability of the trusted foundry, but would likely leak more interconnect information to the FEOL and thereby the untrusted foundry. From an industrial point of view, a higher split layer is more desirable. In this approach, we assume a split layer at or higher than M3 [18].

To maintain the security of this approach despite reduced obfuscation (due to splitting at higher metal layers), modifications are performed based on the classic BISA structure (which we introduced earlier in the chapter). Specifically, two new types of connections are introduced (see below), and critical wires are lifted to BEOL.

1. The *inter-BISA-block fan-outs*: This refers to an input of a OBISA block being driven by a net in another OBISA block. By doing this, the typical tree-like structure of OBISA blocks can be broken, so that it will become more complicated for an attacker to identify OBISA cells.



(a) A fan-out is made between two OBISA blocks.



(b) An obfuscation connection is made.

Fig. 11.9 Two new types of connections to improve obfuscation

2. The *obfuscation connection (OC)*: This refers to an input of a OBISA block being driven by a net in the original circuitry. By doing this, logic cones in original circuitry are obfuscated with OBISA cells, and identifiable logic patterns are broken.

Adding *inter-OBISA-block fan-outs* (henceforth called "fan-outs") could potentially produce redundant gates and thereby lower controllability of gates. To avoid this, fan-outs can be created using following the rules:

- The fan-out is created between a net in one block i and an input pin of another block j ($i \neq j$) and
- The net in block i and the root output in block j have no common related inputs from OBISA LFSR.

If these two conditions are satisfied, the net and the input pin can form a candidate pair for a fan-out. Figure 11.9a shows an example of the fan-out creation. The net N14 in OBISA block i has completely different related inputs of LFSR from the root output N10 in OBISAv block j, so N14 can have a fan-out to connect to any input in

OBISA block j. In Fig. 11.9a, the pin for the net N7 is selected. Note that a fan-out cannot be made on the net N13, because the net N13 and net N10 share related inputs of LFSR, A1 and A2.

The creation of *obfuscation connections* involves two issues. First, activity from the original circuitry must not propagate into OBISA. Otherwise, it could cause unnecessary power consumption. This is ensured by choosing the right kind of gates that have the same controlling value as the LFSR's idle states so that they can isolate OBISA from the original circuitry when OBISA is idle. As shown in Fig. 11.9b, cells BC1 and BC2 are both gated in idle state. BC1 is selected since it is a leaf cell in the tree-structure OBISA block. The other issue that needs consideration is that it will inevitably add capacitive load to the original circuitry net it is attached to. The added capacitance could potentially cause paths to fail in the original circuitry. Thus, we must select target nets in the original circuitry for the obfuscation connection very carefully to avoid timing violations. One way to do this is to perform static timing analysis (STA) prior to the creation of obfuscation connections and only choose nets whose worst-case paths are faster than the critical path by a margin.

A final step of this approach is to perform wire lifting to restrict FEOL interconnect information. The main problem with wire lifting is that finding the best solution requires high computational cost. Since Approach A is geared toward minimizing computational cost, wire lifting in this approach is limited to security critical block key to OBISA functions, for example, the mode select net, the feedback nets in LFSR/MISR, and nets connecting flip-flops in LFSR/MISR.

11.4.1 Implementation Flow

Figure 11.10 presents the design implementation flow of Approach A. The flow fits within the conventional ASIC design flow and is compatible with current commercial physical design tools. OBISA insertion procedure begins after clock tree synthesis. At that point, the whole original circuit has been placed and no more cells will be added in conventional flow (the most left column in Fig. 11.10). The unused spaces would be identified in DEF file, and various standard cells are inserted depending on size of each unused space. Once all unused spaces are filled with OBISA cells, all OBISA cells in each geometric region will be connected to construct an OBISA block. These steps as shown in the middle column were developed in [24, 38]. New steps, as shown in the third column in Fig. 11.10, are introduced to strengthen obfuscation, including fan-out creation, adding obfuscation connections, and lifting secure-critical paths within OBISA. After the OBISA process, the flow resumes the procedures in the conventional design flow. The physical design tool will perform routing for the entire design including original circuit and OBISA circuit. All constraints for the original design can be taken care of by the physical design tool during routing process. Once the timing and sign-off of the design are successful, the last step involves the generation of a GDSII format of the design for final tape-out.

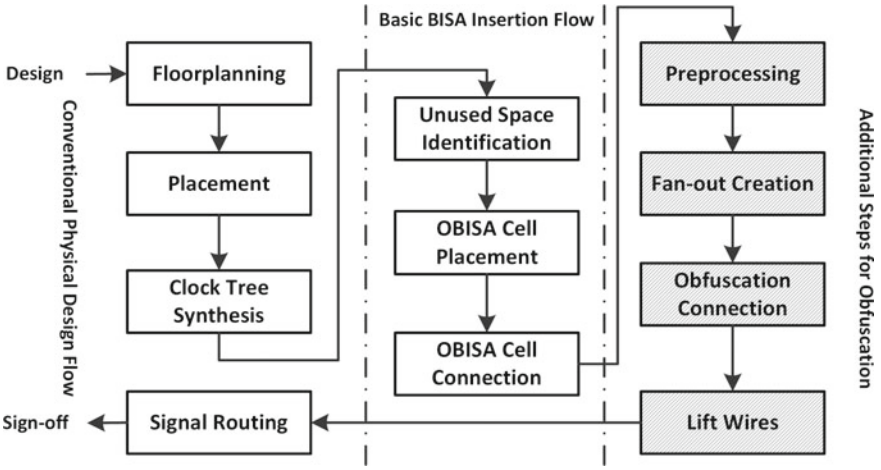


Fig. 11.10 Implementation flow of OBISA technique, Approach A

11.4.2 Design Issues and Trade-Offs

The main trade-off of Approach A results from the two new types of connections it adds to the original BISA design. Both fan-outs and obfuscation connections modify BISA's built-in self test architecture, which could lead to redundant gates, redundant nets, and loss of controllability. These translate into test coverage loss, which means that "blind corners" of OBISA gates are created, whose removal cannot be detected by OBISA. As long as reverse engineering of the OBISA structure can be prevented, this will not be a security issue, but a lowered test coverage nevertheless hurts the credibility of the authentication result. Comparatively, fan-outs between OBISA blocks are easier to handle as they can be made to be fully controllable, while the obfuscation connections will cause more severe loss of test coverage, since their value comes from and is subjected to the status of the original circuitry. However, this can be remedied by performing multiple tests, each starting from an independent state of the original circuitry. Indeed, results in [18] show that even for obfuscation connections which are as many as 25 % of all OBISA inputs, 100,000 random patterns yield just above 60 % coverage, while 10,000 random patterns applied from 10 iterations of the original circuitry can boost the test coverage to almost 100 %.

One optional feature of Approach A is it can choose to use only minimum-sized cells for OBISA structure to keep its security against resizing attacks or dropping that requirement for better obfuscation between OBISA cells and original circuitry cells. This is due to the fact that Approach A does not perform optimized wire lifting as is done in Approach B, nor does it need to introduce compensation OBISA cells that may not have minimum sizes, as we shall see in Approach B. In this regard, Approach A is closer to the classic BISA design.

11.4.3 Potential Attacks

One possible attack against Approach A of OBISA technique is redesign attack. Owing to a higher split layer, it is likely that many small-scale logic blocks such as adders, decoders, and finite-state machines can be identified and optimized by an untrusted foundry. Wire lifting could prevent this. However, large-scale wire lifting would not be possible under our assumption that Approach A is supposed to serve as an example of OBISA technique with minimum computational cost dedicated to the split manufacturing side. Obfuscation connections and inter-OBISA-block fan-outs could help in reducing such signatures. Unfortunately, without a metric dedicated to computational complexity, it is hard to say how much this could help.

11.5 Approach B: OBISA with Wire Lifting

In Sect. 11.4, we discussed an approach to implement OBISA with minimum computational cost. In this section, we discuss the possibility on the other end of the cost-security trade-off axis, which is how maximum security could be achieved with large-scale wire lifting.

To achieve maximum security, it is necessary to first define security. In this approach, we use the k-security definition as was introduced in [22]. k-security has been discussed in more detail in Chap. 10, so we are only giving a brief revisit of the idea here. Consider an IC secured with split manufacturing. Its netlist can be modeled as a graph, where each of its gates is represented by a vertex and each interconnect by a number of edges connecting such vertices. Assuming limited number of custom cells, all models of the gates can be represented by coloring the vertices. Now, remember that we are considering an IC secured with split manufacturing. The FEOL part of its layout contains netlist information of all the gates and a subset of all interconnects. The corresponding graph of the FEOL part, compared to the graph of the complete netlist, will look like the second graph with some of its edges hidden away. Those hidden edges correspond to interconnects reserved to the BEOL part of the IC.

An example of this process is shown in Fig. 11.11. Shown on top is a netlist of a full adder, where black lines mark interconnects visible in FEOL layout, while gray lines mark interconnects in BEOL layout. If we further assume FEOL layout of the full adder splits at its input and output pins, the FEOL layout of the full adder is represented by the graph below.

From the figure, we notice it is impossible to distinguish the two XOR gates (represented by vertices shaded in red slash) in FEOL layout. According to k-security, XOR gates in this full adder have a $k = 2$ security. If the same can be said for all other gate models, k-security metric dictates the FEOL layout has at least $k = 2$ security. Unfortunately, as we can see from the figure, it is impossible for the full adder to

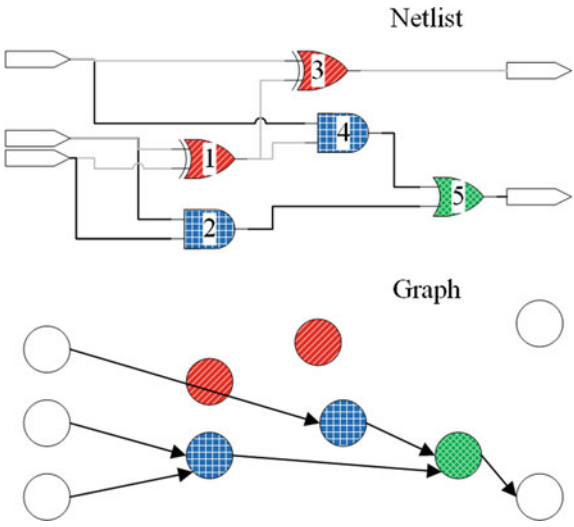


Fig. 11.11 Example: netlist and graph of a split manufactured full adder

reach $k = 2$, even if we lift all edges to BEOL, simply because it has only one OR gate. For this full adder, any optimization of wire-lifting solution is futile.

There are a few ways to address this issue. In [22], only 3 to 7 gate models are allowed during design synthesis, in order to prevent rare gate models from restricting wire-lifting optimization. From a designer's point of view, however, this approach seriously impacts the performance of the original circuitry and could cause serious overhead in area and power.

However, an OBISA technique that performs wire lifting does not have to submit to this restriction, because the number of instances of rare gate models can be compensated with OBISA cells. For example, an AES crypto-core netlist after unconstrained synthesis and optimization has more than 26,000 gates. Among them, 20 gate models have less than 100 instances. Meanwhile, simple BISA insertion into its layout at a normal 0.7 utilization ratio typically yields about 5,000 BISA cells. In other words, OBISA cell count under typical utilization ratio is more than sufficient to compensate rare gate models to 100 instances, a number much higher than what existing wire-lifting algorithm will likely be able to work at within realistic processing time. In [22], the highest reported k is 48 and was only achieved on a much smaller benchmark circuit (c432 from ISCAS, gate count 147).

An example of this advantage is shown in Fig. 11.12. The same full adder is used as shown in Fig. 11.11. In this example, OBISA cells and interconnects are added, as shown in dashed lines. We can see from the example how the bottleneck in previous example—the single OR gate—is compensated with OBISA cells. In the shown wire-lifting example, $k = 2$ security is reached. If we consider a more extreme solution, for example, lifting all wires to BEOL, at maximum, the layout could reach $k = 4$ security rating.

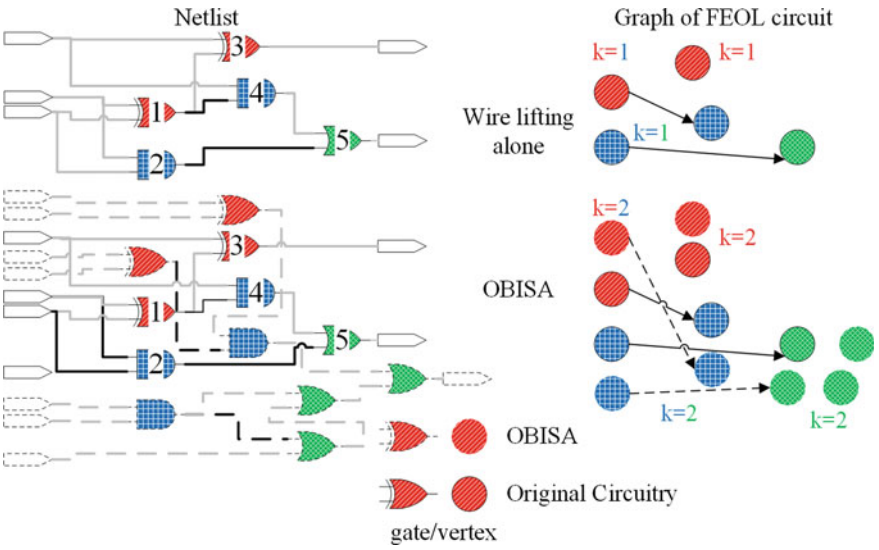


Fig. 11.12 Example: With OBISA insertion, the same full adder can benefit from wire-lifting optimization to achieve higher k-security rating

We can also see from the example that to reach a high security rating k, a large percentage of wires has to be lifted to BEOL for almost every gate. One advantage of this result is that most logic blocks will unlikely be distinguishable in the resulting FEOL layout. This will greatly deter IP piracy as well as redesign attacks, making this approach much more secure. On the other hand, this will also result in a lot of vias having to be matched between FEOL and BEOL, making fabrication more complicated.

Another inference from this example is that the previous limitation (to only use the smallest standard cell models for BISA cells to prevent resizing attacks) must be dropped. This is because it is unlikely for all rare gate models to be of the smallest size. However, as long as obfuscation due to wire lifting holds (i.e., the circuit has a high k-security rating), the attacker will not be able to distinguish OBISA cells from original circuitry cells. Resizing original circuitry cells runs the risk of being discovered by a simple delay test, and this will likely be a sufficient deterrent against resizing attacks.

11.5.1 Implementation Flow

The implementation flow of the OBISA technique (Approach B) is shown in Fig. 11.13. As shown in the diagram, boxes shaded with blue slashes represent procedures already existing in the BISA flow, while boxes shaded with red crosses repre-

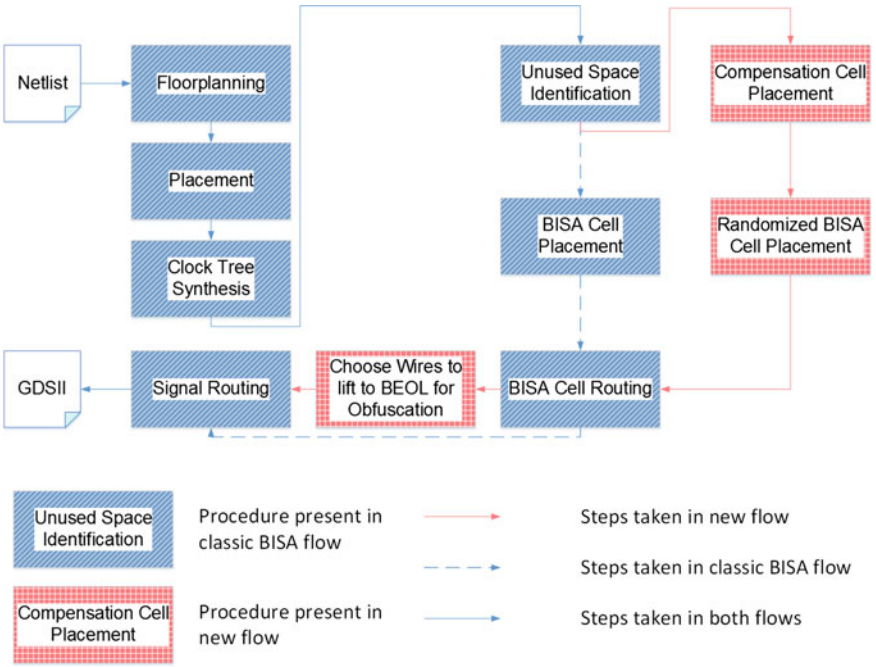


Fig. 11.13 Implementation flow of OBISA technique, Approach B

sent new procedures in this approach. This technique departs from classic BISA after unused space identification: Instead of performing BISA cell placement, cells of the rare gate models are placed first to compensate gate model distribution. After these cells are inserted, placement of BISA cells with random gate models is performed to fill remaining white spaces. After that, normal BISA cell routing is performed. Before signal routing, an optimized wire-lifting solution is found for the complete layout. Wire lifting can then be performed with the help of layout editor, for example, by simply elevating routed interconnects to BEOL metal layers. The rest of the design flow does not differ from existing back-end design flow.

11.5.2 Design Issues and Trade-Offs

The main issue with this approach is cost. Searching for optimal wire-lifting solution is computationally costly, and lifting a sizable percentage of wires to BEOL results in cost in yield loss, elevated requirement on trusted foundry, etc. Between the two costs, another trade-off exists: Obviously, if we choose to lift all wires to BEOL, computational cost will be minimal, security will be maximum, but fabrication cost will be astronomical. The dilemma here is that computational complexity of wire-

lifting solution optimization is a Sharp-P problem, because it consists of a greedy algorithm that exhaustively traverses and verifies the entire solution space, where each step is a Boolean satisfiability problem—a known NP-hard problem. Meanwhile, fabrication cost is not exactly easy to accommodate either. Unless an improvement on at least one of both problems emerges, the cost of Approach B is likely caught in between rock and hard places.

11.5.3 Potential Attacks

One possible attack is based on spatial distribution of cells in FEOL layout. Although k-security metric is defined as at least k mutually indistinguishable instances of each gate model, it goes without saying that this does not account for where those instances are located on FEOL layout. For example, it is certainly logical to assume that a NAND gate very close to a memory cell array is more likely to be a part of the load-store unit than the execution unit. The problem with this kind of information leakage is that little has been established on how the attacker can utilize it. Further, although it is still unclear how the attacker could utilize this information, methods to restrict this leakage of information nevertheless exist. In [22], it is proposed that performing layout design *after* wire lifting could anonymize the layout and prevent any information leakage. Unfortunately, this will likely make back-end optimization of timing a nightmare. Besides, this is not applicable in OBISA since BISA insertion has to come after placement. Still, OBISA in Approach B can partition the complete layout into a number of smaller sublayouts and perform wire lifting on each of them so that wire lifting in each sublayout is relevant to cells in close vicinity. This technique may limit the maximum achievable security rating, but it makes it possible to parallelize the wire-lifting algorithm.

11.6 Conclusion

In this chapter, we first reviewed both BISA and split manufacturing techniques in terms of their adversarial models and pointed out their common adversary, the untrusted foundry. We also explained why the untrusted foundry cannot be trusted to not try attacks that are beyond the scope of a BISA-only design flow (IP piracy) or split manufacturing-only approach (Trojan insertion). We then provided a detailed background for the BISA technique and showed how a combination of both techniques, which we termed as "OBISA," could be expected to be effective against both kinds of attacks. We then investigated two possible approaches to implement the OBISA technique depending on the trade-offs between security and computational/fabrication cost. We provided details on their implementation, discussed the design trade-offs involved, introduced their respective strengths and weaknesses, and theorized how either approach could be attacked.

References

1. Guin U, Forte D, Tehranipoor M (2013) Anti-counterfeit techniques: from design to resign. In: 14th international workshop on microprocessor test and verification, pp 89–94. IEEE
2. Tehranipoor MM, Guin U, Forte D (2015) Counterfeit integrated circuits. Springer, Switzerland, pp 15–36
3. Guin U, Shi Q, Forte D, Tehranipoor MM (2016) Fortis: a comprehensive solution for establishing forward trust for protecting ips and ics. ACM Trans Des Autom Electron Syst (TODAES) 21(4):63
4. Guin U (2016) Establishment of trust and integrity in modern supply chain from design to resign
5. Xiao K (2015) Techniques for improving security and trustworthiness of integrated circuits
6. IARPA Trusted Integrated Circuits (TIC) program announcement. http://www.fbo.gov
7. Salmani H, Tehranipoor M, Plusquellic J (2012) A novel technique for improving hardware trojan detection and reducing trojan activation time. IEEE Trans Very Large Scale Integr (VLSI) Syst 20(1):112–125
8. Li J, Lach J (2008) At-speed delay characterization for ic authentication and trojan horse detection. In: IEEE international workshop on hardware-oriented security and trust, 2008. HOST 2008. IEEE, pp 8–14
9. Jin Y, Kupp N, Makris Y (2010) Dftt: design for trojan test. In: 2010 17th IEEE international conference on electronics, circuits, and systems (ICECS). IEEE, pp 1168–1171
10. Rajendran J, Jyothi V, Sinanoglu O, Karri R (2011) Design and analysis of ring oscillator based design-for-trust technique. In: 29th VLSI Test Symposium. IEEE, pp 105–110
11. Salmani H, Tehranipoor M (2012) Layout-aware switching activity localization to enhance hardware trojan detection. IEEE Trans Inf Forensics Secur 7(1):76–87
12. Chakraborty RS, Bhunia S (2009) Security against hardware trojan through a novel application of design obfuscation. In: Proceedings of the 2009 international conference on computer-aided design. ACM, pp 113–116
13. Banga M, Hsiao MS (2011) Odette: a non-scan design-for-test methodology for trojan detection in ics. In: 2011 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 18–23
14. Chakraborty RS, Bhunia S (2009) Harpoon: an obfuscation-based soc design methodology for hardware protection. IEEE Trans Comput Aided Des Integr Circ Syst 28(10):1493–1502
15. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Proceedings of the 49th annual design automation conference. ACM, pp 83–89
16. Yang X, Choi B-K, Sarrafzadeh M (2003) Routability-driven white space allocation for fixed-die standard-cell placement. IEEE Trans Comput Aided Des Integr Circ Syst 22(4):410–419
17. Charlebois S, Dunn P, Rohrbaugh G (2008) Method of optimizing customizable filler cells in an integrated circuit physical design process, 28 October 2008, uS Patent 7,444,609. https://www.google.com/patents/US7444609
18. Xiao K, Forte D, Tehranipoor MM (2015) Efficient and secure split manufacturing via obfuscated built-in self-authentication. In: 2015 IEEE international symposium on hardware oriented security and trust (HOST). IEEE, pp 14–19
19. Xiao K, Forte D, Jin Y, Karri R, Bhunia S, Tehranipoor M (2016) Hardware trojans: lessons learned after one decade of research. ACM Trans Des Autom Electron Syst 22(1):6:1–6:23. http://doi.acm.org/10.1145/2906147
20. Shi Q, Tehranipoor M, Wang X, Winemberg L (2014) On-chip sensor selection for effective speed-binning. In: 2014 IEEE 57th international midwest symposium on circuits and systems (MWSCAS). IEEE, pp 1073–1076
21. Turk RJ et al (2005) Cyber incidents involving control systems. Idaho National Engineering and Environmental Laboratory
22. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In: Presented as part of the 22nd USENIX security symposium (USENIX Security 13), pp 495–510

23. Jagasivamani M, Gadfort P, Sika M, Bajura M, Fritze M (2014) Split-fabrication obfuscation: metrics and techniques. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 7–12
24. Xiao K, Forte D, Tehranipoor M (2014) A novel built-in self-authentication technique to prevent inserting hardware trojans. IEEE Trans Comput Aided Des Integr Circ Syst 33(12):1778–1791
25. Bushnell M, Agrawal VD (2000) Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits, vol 17. Springer Science & Business Media, New York
26. Jha S, Jha SK (2008) Randomization based probabilistic approach to detect trojan circuits. In: 11th IEEE high assurance systems engineering symposium, 2008. HASE 2008. IEEE, pp 117–124
27. Wang X, Tehranipoor M, Plusquellic J (2008) Detecting malicious inclusions in secure hardware: challenges and solutions. In: IEEE international workshop on hardware-oriented security and trust, 2008. HOST 2008. IEEE, pp 15–19
28. Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B (2007) Trojan detection using ic fingerprinting. In: 2007 IEEE symposium on security and privacy (SP'07). IEEE, pp 296–310
29. Narasimhan S, Wang X, Du D, Chakraborty RS, Bhunia S (2011) Tesr: a robust temporal self-referencing approach for hardware trojan detection. In: 2011 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 71–74
30. Zhang J, Yu H, Xu Q (2012) Htoutlier: hardware trojan detection with side-channel signature outlier identification. In: 2012 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 55–58
31. Wei S, Meguerdichian S, Potkonjak M (2010) Gate-level characterization: foundations and hardware security applications. In: Proceedings of the 47th design automation conference. ACM, pp 222–227
32. Aarestad J, Acharyya D, Rad R, Plusquellic J (2010) Detecting trojans through leakage current analysis using multiple supply pad s. IEEE Trans Inf Forensics Secur 5(4):893–904
33. Alkabani Y, Koushanfar F (2009) Consistency-based characterization for ic trojan detection. In: Proceedings of the 2009 international conference on computer-aided design. ACM, pp 123–127
34. Jin Y, Makris Y (2008) Hardware trojan detection using path delay fingerprint. In: IEEE international workshop on hardware-oriented security and trust, 2008. HOST 2008. IEEE, pp 51–57
35. Xiao K, Zhang X, Tehranipoor M (2013) A clock sweeping technique for detecting hardware trojans impacting circuits delay. IEEE Des Test 30(2):26–34
36. Cha B, Gupta SK (2013) Trojan detection via delay measurements: a new approach to select paths and vectors to maximize effectiveness and minimize cost. In: Design, automation & test in Europe conference & exhibition (DATE). IEEE, pp 1265–1270
37. Rajendran JJ, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, 2013, pp 1259–1264
38. Xiao K, Tehranipoor M (2013) Bisa: built-in self-authentication for preventing hardware trojan insertion. In: 2013 IEEE international symposium on hardware-oriented security and trust (HOST). IEEE, pp 45–50

Chapter 12
3D/2.5D IC-Based Obfuscation

Yang Xie, Chongxi Bao and Ankur Srivastava

12.1 Introduction

Physical limit of transistor miniaturization has driven chip design into the third dimension. 3D integration technology emerges as a viable option to improve chip performance in a direction orthogonal to costly device scaling [10]. A typical stacked 3D IC structure is illustrated in Fig. 12.1a. It expands the circuit design space by vertically stacking multiple device layers and interconnecting them using vertical connections called Through-Silicon-Vias (TSVs). This emerging technology improves chip performance in various aspects. The vertical stacking structure is an attractive option for increasing transistor density. It breaks new ground in system-level integration by integrating more devices and resources into one chip. Besides, 3D integration reduces interconnect wirelength because two distant devices in a conventional 2D design can be placed vertically close to each other and connected with a shorter connection in 3D ICs. The reduction in wirelength scales down interconnect power and delay, which can be leveraged by implementing a more highly connected architecture such as the high-bandwidth Memory-on-Chip architecture [10]. Moreover, 3D integration allows heterogeneous integration. Separate layers can be fabricated using disparate materials and technologies. Heterogeneous integration optimizes existing System-on-Chip (SoC) designs by integrating components of different novel technologies into a single chip. Another structure of 3D IC is called interposer-based 3D IC (or 2.5D IC), as shown in Fig. 12.1b. In this structure, multiple dies are placed side-by-side on a silicon interposer, which provides chip-scale interconnections

Y. Xie (✉) · C. Bao · A. Srivastava
University of Maryland, College Park, MD 20742, USA
e-mail: yangxie@umd.edu

C. Bao
e-mail: borisbcx@umd.edu

A. Srivastava
e-mail: ankurs@umd.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_12

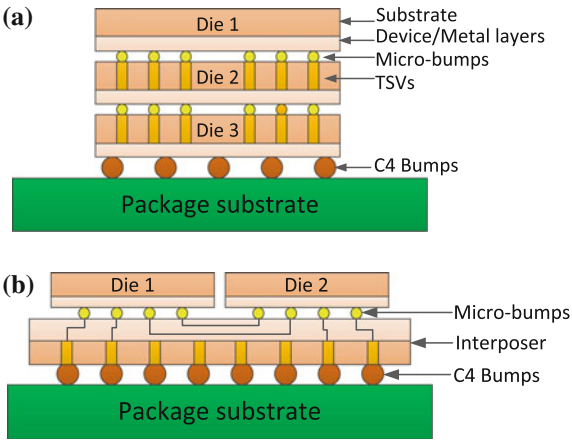


Fig. 12.1 Two common structures of 3D ICs: **a** Stacked 3D IC and **b** Interposer-based 3D IC (2.5D IC)

among different dies. 2.5D integration offers a better thermal cooling option than stacked 3D ICs while still enjoys comparable performance benefits, hence it is viewed as a step stone to stacked 3D integration.

As 3D/2.5D integration is becoming a promising technology for next-generation chip design, researchers have started to investigate it from a hardware security perspective [46]. One line of research focuses on utilizing 3D/2.5D IC technology to protect IC designs from being pirated or tampered during outsourced fabrication [4, 15, 26, 38, 45, 47]. Nowadays, IC designs are increasingly outsourced to an offshore fabrication foundry due to the increasing complexity of modern IC designs and the huge capital expenditure for developing an advanced semiconductor foundry [11]. This poses new security threats on the outsourced designs since the offshore foundry might not be trustworthy. Potential attacks include intellectual property (IP) piracy, overproduction, and malicious modification (hardware Trojans), as discussed in previous chapters. With 3D/2.5D integration, a designer can choose a portion of layers at his discretion and fabricate them in a trusted foundry while outsourcing the rest to untrusted foundries for advanced fabrication technology. This split fabrication strategy of 3D/2.5D ICs creates a new opportunity to obfuscate the outsourced designs. Without the knowledge of the layers that are fabricated in the trusted foundry, an attacker in the untrusted foundry can only observe an incomplete netlist that is a part of an original design. Therefore, it is difficult for him to pirate or counterfeit the complete design, or insert hardware Trojans at a targeted place. 3D/2.5D-based obfuscation enables the access of offshore semiconductor foundries while reducing the security threats in outsourced fabrication.

3D/2.5D integration not only boosts chip performance, but also unlocks new opportunities to thwart security threats in a global IC supply chain. At the same time, it also brings new design and security challenges. This chapter presents the current state of 3D/2.5D IC-based obfuscation techniques and highlights potential security opportunities and challenges of this technology in hardware intellectual property (IP) protection. The outline of this chapter is as follows. Section 12.2 gives an overview of

3D/2.5D integration technology. Section 12.3 discusses 3D/2.5D IC-based obfuscation enabled by 3D/2.5D split fabrication strategy. Section 12.4 summarizes different design objectives, metrics and granularities of 3D/2.5D split fabrication. Section 12.5 introduces a security-aware 2.5D IC design flow that aims at thwarting hardware IP piracy. Section 12.6 discusses various security challenges in 3D/2.5D ICs. Finally, Sect. 12.7 summarizes the implications of 3D/2.5D-based obfuscation on the design of computer-aided design (CAD) tool and Sect. 12.8 concludes this chapter.

12.2 3D/2.5D Integration Technology

3D integration is a technology that vertically integrates multiple 2D dies to create a single high-performance chip, referred to as 3D IC. In general, 3D ICs can be fabricated in two ways. Conventional *die-stacking-based 3D fabrication* utilizes existing 2D IC fabrication process to fabricate multiple 2D dies separately on different substrates and then stack them vertically. Vertical interconnects between different layers are enabled by TSVs. TSVs are vertical electrical connections which are typically made of copper or tungsten. They penetrate through a silicon substrate to connect device layers of different dies, as shown in Fig. 12.1a. TSVs are essential components in die-stacking-based 3D ICs because they provide inter-layer signal communication, thermal conducting and power delivery. Another emerging 3D IC fabrication technology is *monolithic 3D fabrication* [5]. Unlike die-stacking-based 3D fabrication, it grows multiple device layers vertically on the same substrate in a serial order, so it does not require die alignment and bonding while die-stacking-based 3D fabrication does. Because die-stacking-based 3D fabrication has received more attention from both academia and industry due to its fabrication compatibility, we focus on this technique in the following discussion.

Two common structures of 3D ICs are stacked: 3D IC and interposer-based 3D IC (also known as 2.5D IC). Figure 12.1a illustrates the structure of a stacked 3D IC. Multiple TSV-penetrated dies are stacked and bonded vertically. The stacking structure offers various performance advantages as discussed in Sect. 12.1. However, the increased device density in stacked 3D ICs brings about thermal, power and reliability issues. To alleviate these issues while still enjoying the performance benefit, 2.5D IC has been proposed (as shown in Fig. 12.1b). Unlike stacked 3D ICs, 2.5D IC places multiple dies side-by-side and bonds them on a silicon interposer through fine-pitch micro-bumps. The interposer contains both horizontal chip-scale interconnect wires between dies as well as vertical interconnect TSVs to external I/O pins. However, TSVs are not required for inter-die communication in 2.5D ICs. The absence of TSVs in the dies of 2.5D IC makes it easier to design and fabricate than TSV-penetrated stacked 3D IC. Although 2.5D ICs might not achieve the same amount of performance improvement as 3D ICs, it offers better cooling options, which is essential for high-performance computing systems. While commercial large scale 3D IC is still being developed, large-volume commercial 2.5D products are already in the market, such as the Xilinx Virtex-7 2000T FPGA [34].

3D integration can be done at different granularities [22]. Coarse-grained 3D integration can be implemented at *core level*, such as the 3D memory-on-chip architectures. This approach could offer significant improvements to performance and power and alleviate the memory bandwidth wall problem, a situation in which the chip-to-memory bandwidth is becoming a performance and throughput bottleneck. But the core-level integration does not take full advantage of the benefits of 3D ICs. A finer grained *functional block level* integration allows functional blocks to be distributed across multiple layers but maintain each functional block as a 2D circuit. This can reduce intra-core wirelength and allow reduced clock period or power. To take this idea even further, 3D integration at the *logic-gate level* offers even more savings in power and delay. It implements an individual functional block across multiple layers so as to reduce intra-block delays and power. A recent study [17] of full-chip 3D design of a SPARC chip multiprocessors (CMP) showed that a 3D design using 2D functional blocks can reduce power by 14% compared to a baseline 2D design, however when the logic-gate level 3D integration is applied this reduction in power becomes 20%. Even finer grained integration at the *transistor level* (e.g. separate layer for NMOS and PMOS) has been considered [22], but the ability to manufacture TSVs at the size and pitch required for such a scheme is yet to be realized. Moreover, the reliability and yield implications of such an approach are expected to be prohibitive [23].

12.3 3D/2.5D IC-Based Obfuscation

As 3D/2.5D integration is becoming a promising technology for next-generation chip design, researchers have started to investigate it from a hardware security perspective. One line of research focuses on utilizing 3D/2.5D ICs to mitigate security threats in outsourced fabrication [4, 15, 26, 38, 45, 47]. In order to access advanced semiconductor technology at a lower cost, most IC design companies that once possessed their own foundries are now adopting a fabless model: they concentrate their resources and efforts on IC designs while outsourcing the fabrication. Although such outsourcing model is cost-effective, it poses new security threats on the outsourced designs since the offshore foundry might not be trustworthy. Without close monitoring and direct control, the outsourced designs are vulnerable to various attacks such as piracy, overproduction and hardware Trojan insertion, as discussed in previous chapters. These attacks (also known as supply chain attacks) pose not only an economic risk to commercial IC design companies, but also security threats for sensitive electronic systems. In this section, we discuss how to utilize 3D/2.5D integration to mitigate these attacks.

12.3.1 3D/2.5D Split Fabrication

In 3D/2.5D integration, multiple dies (active layers) are fabricated independently on separate substrates and then integrated together into a single chip. This fabrication process offers inherent support for *split fabrication*, where different dies can be fabricated in different foundries. A designer can choose a portion of the design at his discretion and manufacture it in a trusted foundry for security while manufacturing the rest in an untrusted foundry for state-of-the-art fabrication technology. Because a portion of the original design will be hidden from the untrusted foundry, *3D/2.5D split fabrication* creates a new opportunity to access offshore fabrication foundries while preventing potential security threats.

3D split fabrication can take place in two different forms [38, 42]. In one embodiment, some dies (active layers) of a stacked 3D IC are fabricated in a trusted foundry, referred to as *trusted tier* while others are outsourced to one or more untrusted foundries, referred to as *untrusted tier*, as shown in Fig. 12.2a. The final integration is also implemented in the trusted foundry. With that, each untrusted foundry can only obtain one portion of design and thus it is difficult to reverse-engineer the original design or insert hardware Trojans at a desired place. Even if we assume all untrusted foundries are colluded (as one untrusted foundry), the portion of IC design in the trusted tier is not directly accessible to the untrusted foundries and hence it is protected from potential attacks by the adversary. In another embodiment, all active layers are outsourced to the offshore foundries and then securely routed and bonded in a trusted foundry. By doing so, the vertical connections between layers are kept secret. Although the offshore foundries can reverse-engineer the netlist of each layer, the resultant incomplete netlist (lacking the inter-layer connections) is incomprehensible if a design is intelligently partitioned into different layers in an obfuscated manner.

For 2.5D split fabrication [15, 45, 47], the most common split fabrication strategy is to fabricate the silicon interposer in a trusted foundry as the trusted tier while

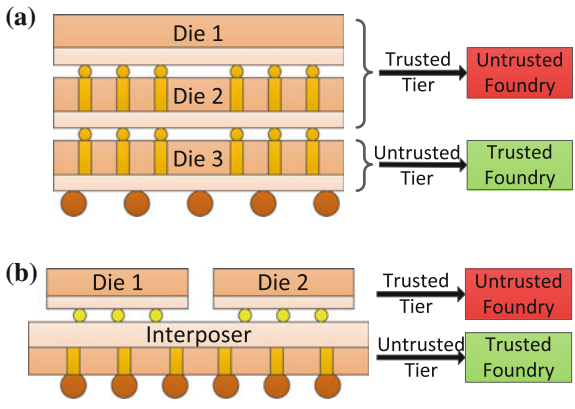


Fig. 12.2 3D split fabrication for **a** stacked 3D IC and **b** 2.5D IC

outsourcing the dies as the untrusted tier, as shown in Fig. 12.2b. If all untrusted foundries are independent (not colluded), an attacker in one untrusted foundry can only obtain the netlist of a die that is fabricated in this foundry. Even if the offshore foundries are colluded, they can at most obtain an incomplete design that lacks these interconnect wires. The incomplete netlist will be incomprehensible if the wires in the interposer layer are intelligently selected. As discussed in Sect. 12.2, 2.5D integration has less severe thermal and reliability challenges while offering a comparable performance improvement compared to the stacked 3D integration. Moreover, leveraging this technology requires only minor modification to current IC design flow and fabrication process. As a result, recent research work on 3D IC-based obfuscation [15, 45, 47] focuses more on 2.5D split fabrication than stacked 3D IC-based split fabrication.

12.3.2 Comparison Between 3D/2.5D and 2D Split Fabrication

Notice that the split fabrication strategy can also be applied to conventional 2D IC technology [16, 29, 40, 41]. As shown in Fig. 12.3, 2D IC-based split fabrication (also known as *split manufacturing*) splits a 2D IC into a Front-End-Of-Line layer (FEOL) that contains active devices and lower metal layers, and a Back-End-Of-Line (BEOL) layer that contains higher metal layers. The FEOL layer is outsourced to an untrusted foundry for advanced fabrication technology while the fabrication of BEOL layer and final integration are securely implemented in a trusted foundry. Thus, interconnect wires in BEOL layer of a split 2D IC are kept secret from the untrusted foundry.

Compared to 2D split fabrication, 3D/2.5D split fabrication requires less strict fabrication compatibility between the untrusted foundry and the trusted foundry and can provide more flexibility on obfuscation design. The difference between 2D and 3D/2.5D split fabrication are summarized as follows.

- *Alignment and integration*: for a split 2D IC, the alignment and integration are more challenging especially when the 2D IC is split from a low metal layer [45]. In general, a low-layer split 2D IC has smaller pitch length (eg 0.1–1.6 μm [45]) and

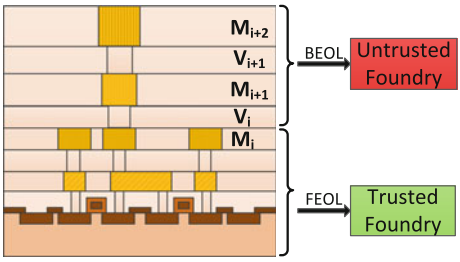


Fig. 12.3 2D split fabrication. A 2D IC is split into a FEOL layer that contains active devices and lower metal layers, and a BEOL layer that contains higher metal layers

dense connections across trusted BEOL and untrusted FEOL layer, which requires more precise alignment and integration techniques. In contrast, the alignment of TSVs of 3D/2.5D IC is less challenging because of larger pitch size (eg 5 μm [24]) and less number of connections.

- *Fabrication process*: the split fabrication strategy of 3D/2.5D IC is adaptable to off-the-shelf 3D/2.5D IC fabrication process. Each die is an individual component that can be fabricated separately and then integrated together, either in a single foundry or in different foundries. Interconnecting separately made dies using 3D integration is already a proven technology [42]. Thus, the extra effort for 3D/2.5D IC to adapt the split fabrication is lower than that of 2D IC.
- *Obfuscation flexibility*: in terms of obfuscation, the trusted tier of 3D IC consists of active layers which can be used to hide logical gates and functional circuits while for 2D ICs, the trusted BEOL layer is restricted to be metal wires. As a result, the complexity for an adversary to reconstruct the whole design for 3D/2.5D split fabrication is much higher than 2D split fabrication [20].

12.3.3 Comparison Between 3D/2.5D Split Fabrication and Logic Locking

Logic locking [3, 18, 21, 27, 28, 31, 33, 43] is another hardware IP protection technique that hides the functionality of an IC by inserting additional key-controlled logic gates (eg XOR/XNOR) and key-inputs into a circuit, as introduced in previous chapter. The locked circuit preserves the correct functionality only when the key-inputs are set correctly. Although both logic locking and 3D/2.5D split fabrication aim at obfuscating the outsourced design to prevent security threats in outsourced fabrication, these two techniques differ in various aspects:

- *Obfuscation Approach*: Obfuscation by logic locking is implemented by "adding" extra logic gates to make the original circuit become a key-controlled reconfigurable circuit. On the contrary, 3D/2.5D split fabrication is implemented by "subtracting" a portion of gates/wires and hiding them in the trusted tier so as to prevent the complete exposure of the original design.
- *Attack Resistance*: 3D/2.5D split fabrication is believed to be more attack-resistant than logic locking [20]. For logic locking, although the outsourced design is locked with additional key-gates, its layout and hence netlist are completely exposed to the untrusted foundry. Once the correct key is known, the correct functionality and netlist are accessible to an attacker. Various attacks have been proposed to infer the correct key of logic locking techniques [27, 28, 36, 48]. On the contrary, 3D/2.5D split fabrication hides a portion of design in the trusted tier, so the untrusted foundry does not have access to the complete netlist. The trusted tier behaves as a black box and thus it is more difficult for an adversary to infer the functionality of the trusted tier.

- *Fabrication Compatibility*: 3D/2.5D split fabrication requires the usage of emerging 3D/2.5D integration technology while logic locking can utilize existing well-developed 2D IC technology.

12.4 Design of 3D/2.5D Split Fabrication

3D/2.5D IC technology offers a new opportunity to obfuscate the outsourced IC designs by hiding partial circuitry into a trusted tier that's fabricated in a trusted foundry. To fully exploit this idea, one important challenge is to determine the portion of circuit design that needs to be hidden and protected in the trusted tier. In this section, we introduce different design objectives, metrics and granularities for 3D/2.5D split fabrication.

12.4.1 Design Objectives and Metrics

12.4.1.1 Functionality Obfuscation

Functionality obfuscation by 3D/2.5D split fabrication aims at obfuscating the functionality of an outsourced design (the untrusted tier). It hides gates/wires into the trusted tier such that the functionality of the untrusted tier (or a *reconstructed circuit* that is inferred based on the untrusted tier) differs substantially from the original functionality. By obfuscating the functionality, an attacker who has the knowledge of the untrusted tier cannot infer or utilize the functionality of the original complete design, thereby protecting the outsourced design from piracy and overproduction.

Hamming distance (HD) is widely used to quantify the security level of functionality obfuscation [29, 30, 32, 47]. It is defined as the number of different output bits between original netlist and reconstructed netlist on applying a same input vector. Given one input vector $\mathbf{X_i}$, the function of original netlist F will produce an output vector $\mathbf{Y_i} = F(\mathbf{X_i})$, while the function of reconstructed netlist $F^{'}$ will produce another output vector $\mathbf{Y_i}^{'} = F^{'}(\mathbf{X_i})$, the HD between two outputs $HD(\mathbf{Y}_i^{'}, \mathbf{Y}_i)$ is the number of different bits in two output vectors, and the normalized HD of two functions can be calculated as follows:

$$HD(F, F^{'}) = \frac{1}{n}\sum_{i=1}^{n}\frac{HD(\mathbf{Y}_i^{'}, \mathbf{Y}_i)}{|\mathbf{y}|} \times 100\% \tag{12.1}$$

where n is the number of input vectors and $|\mathbf{Y}|$ is the number of output bits. Since the objective of functionality obfuscation is to restrain the attacker's ability to infer or utilize the correct functionality, HD approaching 50% is desirable, which indicates that the functionality of the reconstructed netlist deviates substantially away from the original functionality.

12.4.1.2 Netlist Obfuscation

Netlist obfuscation by 3D/2.5D split fabrication aims at obfuscating the netlist structure of the untrusted tier (eg the connection degree of each gates and the gate types) so that an attacker is not capable of identifying a desired place to insert hardware Trojans in the untrusted tier. These hardware Trojans are referred to as *targeted hardware Trojans*, because they aim at modifying specific targeted gates/wires to achieve some purposeful attacks such as privilege level escalation [15] or tampering hardware Trojan detection circuitries [26, 45]. By hiding enough gates/wires into the trusted tier, a targeted circuitry is partially (or completely) hidden in the trusted tier, thereby preventing the attacker for identifying the target gates/wires to attack.

Imerson et al. [15] proposed a security metric called *k-security* for evaluating the netlist obfuscation level of a 2.5D split fabrication design under targeted hardware Trojan insertion. An incomplete netlist in the untrusted tier is said to be k-secure if every gate in the original netlist can be mapped to at least k indistinguishable gates in the incomplete netlist. The k-security ensures that an attacker cannot find out the targeted gate out of the k indistinguishable gates to attack. As a result, he can either insert Trojans at one gate but has only $1/k$ success probability, or he can attack all k gates but at the risk of being detected since he needs to modify more gates.

12.4.1.3 Layout Obfuscation

The security of 3D/2.5D split fabrication rests upon the assumption that the attacker does not know the hidden portion (trusted tier) and cannot infer it based on the exposed portion of design (untrusted tier). Otherwise, the attacker can reconstruct the complete design and continue to conduct the supply chain attacks. For example, Rajendran et al. [29] proposed an attack called *proximity attack* that can be utilized to infer the hidden connection in 2.5D split fabrication. In a split-fabricated 2.5D IC, a portion of wires are hidden in the trusted tier (interposer), and they are not accessible to the untrusted foundry. However, modern floor planning and placement (F&P) tool will place two connected pins closely in the untrusted tier so as to reduce the wirelength, thereby leaking the information of the hidden connections. Since the layout information for each die is known to the attacker, he can iteratively connect an output pin in one die to its closet input pin in other die and thus reconstruct the circuit. Therefore, it is of great significance to obfuscate the layout (by placing two connected pins far away) in order to prevent the leakage of the trusted tier, especially in the case of 2.5D split fabrication.

Proximity-attack correctness is a security metric that is used to quantify the layout obfuscation level under the proximity attack. For 2.5D split fabrication, it is defined as the percentage of correct connections that are recovered by the proximity-attack algorithm. Attack correctness approaching 0% is desirable for a secure layout design, which indicates that the attacker cannot infer the correct connections in the trusted tier of a split 2.5D design.

12.4.1.4 Trusted Tier Protection

In Sect. 12.4.1.3, we introduce the proximity attack that utilizes the layout information of the trusted tier to infer the connections in the untrusted tier of a 2.5D IC. Another potential attack to infer the trusted tier can be implemented by reverse-engineering the final product obtained from the open market. The attacker can purchase the IC from the open market and utilize state-of-the-art reverse-engineering technique [39] to obtain the design of the trusted tier by delayering and extracting the chip. Therefore, tamper-resistant techniques should be applied to protect the trusted tier.

The percentage of gates correctly extracted from a layout is one of the security metrics for IC reverse-engineering [32, 39]. Thus, the security metric for trusted tier protection against reverse-engineering can be defined as the percentage of gates/wires extracted from the layout of the trusted tier, referred to as the *reverse-engineering correctness*.

12.4.1.5 Performance and Fabrication Cost

Noticing that usually the untrusted offshore foundries support more advanced technology nodes and operate at a lower cost than the trusted foundries, the choice of which part to hide is actually a trade-off among security, performance and fabrication cost. If more gates/wires need to be fabricated in the trusted foundry, the overall fabrication cost will be increased (if same technology is used). If a less advanced semiconductor technology is used for the trusted tier to reduce cost, the performance of the overall circuit will be compromised.

Area, wirelength, delay and *power* are widely used to quantify the performance of an IC design. In summary, the objective of a secure 3D/2.5D split fabrication design is to increase the circuit obfuscation level and prevent the leakage of the trusted layer at an acceptable performance/fabrication cost. The summary of design objectives and metrics is shown in Table 12.1.

Table 12.1 Design objectives and metrics for 3D/2.5D split fabrication

Design objectives	Metrics
Functionality obfuscation	HD [29, 47]
Netlist obfuscation	k-security [15]
Layout obfuscation	Proximity-attack correctness [29, 47]
Trusted tier protection	Reverse-engineering correctness [32, 39]
Performance	Area, wirelength, delay, power [15, 29, 47]

12.4.2 Design Granularities

3D/2.5D split fabrication can be designed at different granularities.

At *block-level*, the trusted tier can conceal the whole security-critical circuit blocks such as hardware Trojan detection sensors in order to protect them from being tampered or removed by the attacker. Recent years have seen a huge proliferation of hardware Trojan detection research based on functionality verification [35], side-channel signatures [2], built-in self-authentication (BISA) [44] and so on. Most of these techniques require additional circuits to assist in Trojan activation and/or detection, including dummy flip-flops, sensors and authentication circuits, which are referred to as design-for-security (DfS) circuitries. However, these DfS circuitries may also be tampered or bypassed, which undermines the system's security. With 3D/2.5D split fabrication, the DfS circuitries can be placed in the trusted tier, thereby preventing them from being identified and tampered.

At *gate-level*, the trusted tier can withhold a portion of original wires and/or gates that can maximally obfuscate the functionality and/or netlist of the exposed untrusted tier in order to prevent piracy or targeted hardware Trojan, as discussed in Sects. 12.4.1.1 and 12.4.1.2.

With technology progresses, future 3D/2.5D split fabrication might be implemented at *transistor level*. Although such fine-grained integration has not yet been realized, it offers a novel opportunity for obfuscating a lower-level component such as standard cells.

12.5 Security-Aware 2.5D IC Design Flow Against IP Piracy

Due to the advantages in thermal cooling and fabrication compatibility, 2.5D-based obfuscation has been investigated more in recent research work [15, 45, 47] compared to 3D-based obfuscation. By fabricating the interposer of 2.5D IC in a trusted foundry while outsourcing the rest to an untrusted foundry, an attacker in the untrusted foundry can only obtain an incomplete netlist which lacks the wires in the trusted tier (interposer). However, this does not imply that a conventional performance-driven 2.5D IC design flow followed by a split fabrication strategy is security optimal. In a performance-driven 2.5D IC design flow, a netlist is first partitioned in a way that minimizes the number of cut-wires to reduce the number of wires that need to be routed in the trusted tier. Then, corresponding layouts are generated using placement and routing tools to minimize layout area and routing wirelength. Although a min-cut partitioning has a lower performance overhead, it might not hide enough wires to fully obfuscate the functionality of outsourced designs. Also, a performance-driven placement might place two connected pins/gates close-by, thereby leaking the information about the hidden connections that can be exploited by the proximity-attack algorithm, as introduced in Sect. 12.4.1.1.

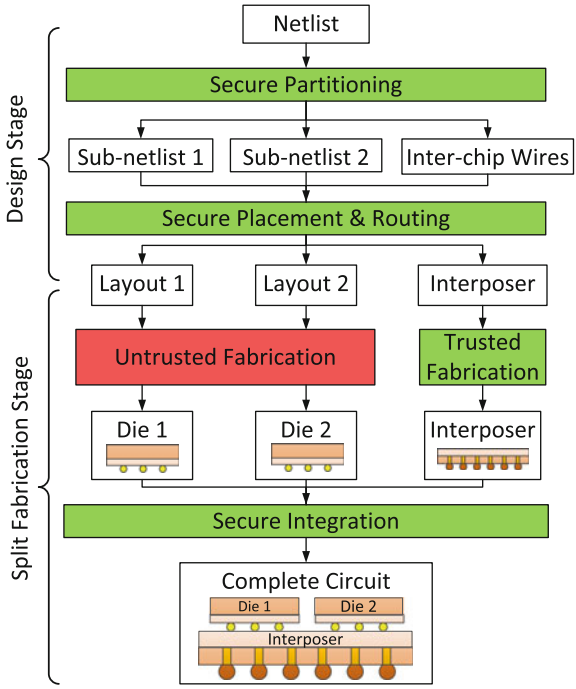


Fig. 12.4 A security-aware 2.5D IC design flow [47]

In this section, we introduce a security-aware 2.5D IC physical design flow that aims at thwarting hardware IP piracy. The security-aware 2.5D IC design and split fabrication flow of is shown in Fig. 12.4. The objective of this design flow is to thwart IP piracy by producing a security-aware partitioning and placement solution that can obfuscate the original functionality while defending the proximity attack. The *secure 2.5D design flow problem* can be defined as follows:

Given a netlist of a combinational circuit and the Boolean function F that maps its primary outputs (PO) $\mathbf{Y}$ to its primary inputs (PI) $\mathbf{X}$: $\mathbf{Y} = F(\mathbf{X})$, the objective of a security-aware 2.5D IC design flow is to find a bi-partition and a corresponding gate-level placement result, so that the placement result of two partitions will disclose the least functionality and netlist of the original circuit at a minimum performance cost.

Notice that this design and fabrication flow assumes only one untrusted offshore foundry that is responsible for fabricating two dies. However, it is possible that two dies can be outsourced to different foundries, and if these foundries are completely independent (no collusion), the information leakage to each foundry can be reduced. Moreover, this design flow focuses only on bi-partitioning for simplicity, but it would be possible to partition into more layouts and use more "independent" foundries for better security.

12.5.1 Security Metrics and Objectives

Two security metrics are utilized in order to quantify the security level of a 2.5D IC design flow, namely HD and proximity-attack correctness, as discussed in Sects. 12.4.1.1 and 12.4.1.3.

- *HD*, as defined in Eq. 12.1, is widely used to quantify the security level of functionality obfuscation [29, 30, 32]. To ensure that the functionality of a reconstructed netlist deviates substantially away from the original functionality, HD approaching 50% is desirable.
- *Proximity-attack correctness* is defined as the percentage of correct connections under proximity-attack algorithm. Attack correctness approaching 0% is desirable for a secure layout design, which indicates that the attacker cannot infer the correct connections in the trusted tier.

Based on these two security metrics, the objective of our problem can be formulated as follows:

$$\text{minimize } |HD - 50\%| + Correctness \tag{12.2}$$

A secure design flow for 2.5D IC should achieve two objectives: (a) incorrect outputs will be produced on applying incorrect connections between two partitions, i.e., the HD between the functionalities of the original netlist and the netlist reconstructed using proximity-attack algorithm is 50%; (b) the proximity-attack algorithm has 0% attack correctness.

12.5.2 Secure Partitioning

The partitioning phase plays a pivotal role in functionality obfuscation because it determines the hidden wires in interposer layer. Figure 12.5 illustrates a bi-partitioning of the c17 circuit from ISCAS85 benchmark. The cut-wires are selected as the hidden wires that will be routed in the interposer layer. The resulting cut-wires have a significant impact on the incorrectness of output logics of reconstructed netlist, because they decide whether faults can be generated and propagated to primary outputs (POs) when incorrect connections are made.

To evaluate the capability of fault occurrence and fault propagation for a cut-set, we utilize the concepts of *controllability* and *observability*. As discussed in previous chapters, controllability and observability are the two characteristics that are widely used in IC testing and security techniques. Controllability of an internal wire is the sensitivity of the wire w.r.t. the logic transition of primary inputs (PIs). It quantifies the ability of setting a wire to some values (1 or 0) through PIs in order to activate a fault (due to incorrect reconnections) inside a circuit. Observability of a wire is the sensitivity of POs w.r.t. the logic transition of the internal wire. It quantifies the ability of observing faults in POs when the logic value of a wire inside the circuit is

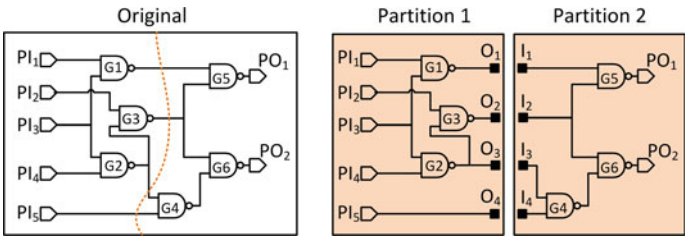


Fig. 12.5 A bi-partitioning of the c17 circuit from ISCAS85 benchmark. The cut-wires are selected as the hidden wires that will be routed in the interposer

flipped. In order to activate and produce more faults when incorrect connections are made between two partitions, we aim at selecting cut-wires with high controllability and observability. The controllability $CTRL(w)$ and observability $OBS(w)$ of a wire w can be simulated and normalized to a value between 0 to 1 [47], where 1 indicates high controllability/observability.

12.5.2.1 Secure Min-Cut Algorithm

The secure min-cut problem is to find a bi-partitioning with minimum cut-size while satisfying balance constraint and security constraint. The balance constraint ensures that two partitions have roughly equal sizes while the security constraint enforces that the controllability and observability of the wires in the cut-set are relatively large. The overall algorithm is based on Fiduccia-Mattheyses (FM) algorithm [9], a linear time heuristic approach to solve hypergraph bi-partitioning problem. The overall algorithm is as follows:

- *Initialization*: a balanced partitioning is randomly initialized so that two partitions have roughly equal sizes. PI pins and PO pins are separated into two partitions. Moreover, the controllability and observability of all wires are simulated.
- *Maintenance*: after initialization, the FM algorithm will iteratively move a gate that has the maximum cut-size drop from one block to another while maintaining the following two constraints:
  - Balance constraint: $\frac{|A(P_1)-A(P_2)|}{A(P_1)+A(P_2)} \leq B_{th}$, where $A(P_1)$, $A(P_2)$ are the sizes of two partitions P_1 and P_2, and B_{th} is a pre-defined balance threshold $0 \leq B_{th} \leq 1$.
  - Security constraint: if a gate's output wire w is in the cut-set and it has high controllability/observability $CTRL(w) + OBS(w) \geq S_{th}$, then do not move this gate. S_{th} is a pre-defined security threshold $0 \leq S_{th} \leq 2$.
- *Termination*: After all possible gate moves, the algorithm obtains a series of moves that will result in the most cut-size reduction, which produces a new partitioning solution. The algorithm is continued until it cannot find a partitioning solution with smaller cutsize. Then, a final partitioning solution is generated and each gate is assigned to a partition.

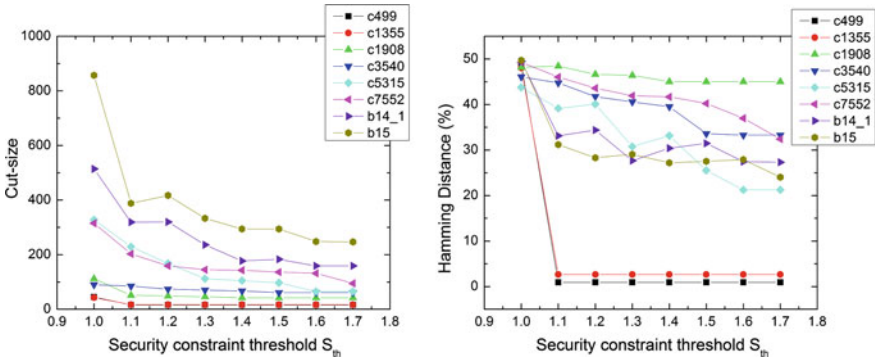


Fig. 12.6 Impact of security constraint S_{th} on **a** cut-size and **b** HD [47]

We normally run the FM algorithm multiple times with random initial partitioning solution and select the best partitioning solution with minimum cut-size as the final solution.

12.5.2.2 Trade-Off Between Cutsize and HD

In partitioning phase, we set the balance threshold B_{th} to be 0.1 to allow a slight imbalance between two partitions. Since a new security constraint is added in the secure partitioning algorithm, the feasible solution space is normally reduced. As a result, the cut-size of a partitioning solution will be increased when the security constraint is tight (S_{th} is small). The impact of S_{th} on cut-size and HD is shown in Fig. 12.6. The experiment is conducted on 8 combinational circuits from ISCAS85 and ITC99 benchmark suites. As S_{th} increases (security constraint becomes loose), the cutsize and HD decreases for all benchmarks, since a large S_{th} indicates that only few wires with large controllability and observability will be locked in the cut-set to prevent cut-size reduction. Based on this simulation results, we define **secure partitioning (SecPart)** as the partitioning with S_{th} that makes HD larger than a predefined threshold (eg 40%). Also, we define **normal partitioning (NormPart)** as the partitioning that does not consider the security constraint.

Table 12.2 shows the partitioning results of three partitioning settings, namely normal partitioning (NormPart), secure partitioning (SecPart) and normal partitioning with cut-size lower-bound that is set to the cut-size of secure partitioning solution (NormPart_LargeSize). Comparing NormPart and SecPart, we can see that HD increases from 13.24 to 46.35% on average. This is because that we have enforced the security constraint to select enough cut-wires with high controllability/observability so that more faults will be produced for an incorrectly reconstructed netlist. However, the security constraint inevitably increases the cut-size of secure partitioning. As seen, the cut-size of SecPart is 3.4× the cut-size of NormPart on average. The extra cut-wires will increase the performance overhead such as area and wirelength

Table 12.2 Benchmark information and partitioning results of normal partitioning (NormPart), normal partitioning with large cut-size (NormPart_LargeCutsize) and secure partitioning (SecPart) [47]

Benchmark	#PIs	#POs	#Gates	NormPart		NormPart_LargeSize		SecPart	
				Cutsize	HD (%)	Cutsize	HD (%)	Cutsize	HD (%)
c499	41	32	202	16	0.86	45	48.20	45	49.84
c1355	41	32	546	16	7.08	43	45.01	43	49.96
c1908	33	25	880	35	20.09	37	33.46	37	44.79
c3540	50	22	1669	57	32.82	74	33.28	74	42.67
c5315	178	123	2307	30	8.65	168	19.13	168	41.07
c7552	207	108	3512	25	5.46	155	14.34	155	48.55
b14_1	277	299	4048	99	14.85	386	19.14	386	44.76
b15	485	519	7022	168	16.14	625	27.76	625	49.12
Average	–	–	–	–	13.24	–	30.04	–	46.35

in the placement phase. To validate the efficiency of the security constraint, we compare the partitioning results of SecPart and NormPart_LargeSize. It can be seen that although these two cases have the same cut-size, SecPart can ensure 46.35% HD while NormPart_LargeSize can only achieve 30.04% HD. Therefore, with security constraint, the secure partitioning algorithm can achieve 50% HD more efficiently.

12.5.3 Secure Placement

The placement phase is designed to thwart the proximity attack by obfuscating the layouts of the untrusted tier so as to mislead the proximity-attack algorithm into making wrong connections. The goal of secure placement is to minimize the area, intra-chip wirelength, inter-chip wirelength and proximity-attack correctness.

12.5.3.1 Secure Placement Algorithm

The secure 2.5D IC placement algorithm is based on a B*-tree and simulated annealing (SA)-based 2.5D IC placement algorithm proposed by Ho et al. [13]. Figure 12.7 shows the overall flow of the secure placement algorithm.

The placement algorithm utilized the B*-tree to represent a compacted placement solution [6]. Two B*-trees are firstly constructed to represent the geometry relationship for all gates and I/O pins of two sub-netlists. A node in the B*-tree represents a gate or an I/O pin and each B*-tree represents a compacted placement for one sub-netlist. Using two B*-trees allows us to optimize the placement of two sub-netlists simultaneously. Three node perturbation operations are implemented in the SA process, as defined in [13]:

- *Rotation*: the rotation of a gate or I/O pin.
- *Move within a B*-tree*: the move of a gate or an I/O pin within same die.
- *Swap two nodes within a B*-tree*: the swap of two gates or I/O pin within same die.

After perturbation, two new B*-trees are formed and corresponding compact placements for two chips can be obtained. Based on the placement solution, we can calculate its area, inter-chip wirelength and intra-chip wirelength and perform the proximity attack to obtain the proximity-attack correctness.

The cost function of SA optimization is defined as:

$$\Phi = \alpha \times Area + \beta \times WL_{intra} + \gamma \times WL_{inter} + \theta \times Correctness \tag{12.3}$$

where α, β, γ and θ are user-specified weighting parameters, *Area* is the total area of two chips, WL_{intra} is the total intra-chip wirelength, WL_{inter} is the total inter-chip wirelength and *Correctness* is the proximity-attack correctness obtained by proximity-attack algorithm. Two SA processes are used to generate an effective and secure placement, as shown in Fig. 12.7. The first performance-driven ($\theta = 0$) SA process creates an initial placement that has optimized area and total wirelength. Based on this initial placement, the second security-driven ($\theta \neq 0$) SA process attempts to trade-off between performance and security.

In placement phase, in order to determine the optimal weights in cost function, we tested different setups on all benchmarks and define the setup $\alpha = 0.2$, $\beta = 0.7$, $\gamma = 0.1$, $\theta = 0$ as **normal placement (NormPlace)** since it can obtain relatively optimal results in area and total wirelength. For **secure placement (SecPlace)**, we increase θ to 0.05 and decrease γ to 0.05.

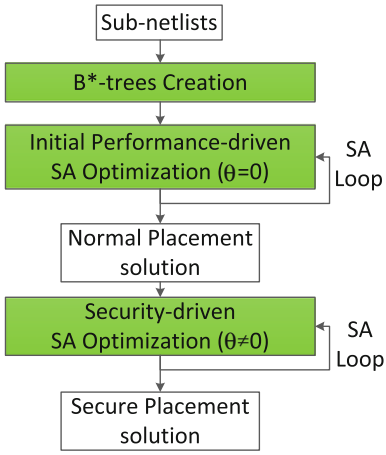


Fig. 12.7 B*-tree and SA-based secure placement algorithm flow [47]

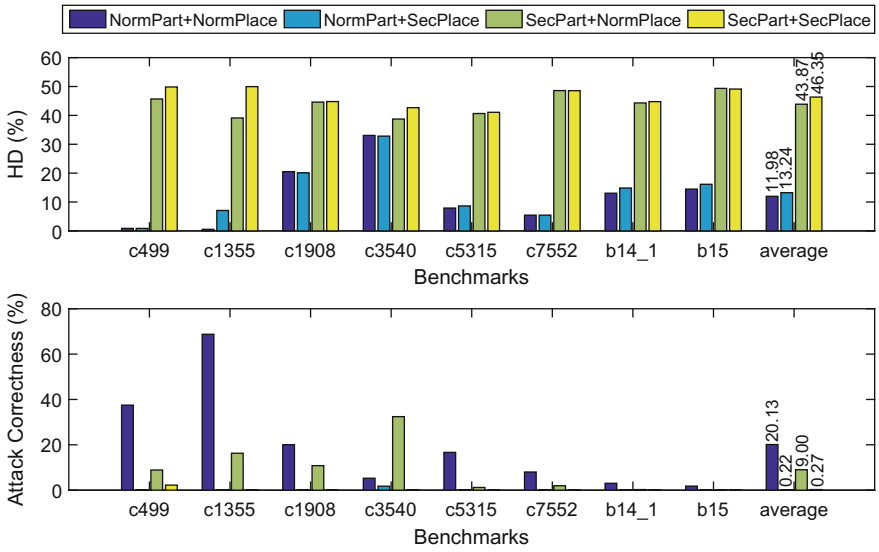


Fig. 12.8 HD and attack correctness for four design flows (NormPart + NormPlace, NormPart + SecPlace, SecPart + NormPlace, SecPart + SecPlace) [47]

12.5.4 Security and Performance Trade-Off

In order to evaluate the overall security-aware 2.5D design flow (SecPart + SecPlace), we compare four possible combinations, namely NormPart + NormPlace, NormPart + SecPlace, SecPart + NormPlace and SecPart + SecPlace in terms of attack correctness, Hamming distance, area and total wirelength.

Figure 12.8 shows the correctness and HD of proximity-attack for four cases. For 'NormPart + NormPlace', the attack correctness is 20.13%, and HD is only 11.98% because no security constraint is enforced in the NormPart to conceal the functionality, and the NormPlace does not minimize attack correctness during SA optimization. When SecPlace is performed on NormPart, we noticed that the attack correctness is limited to 0.22%, and the HD increases to 13.24%, which is still far below 50% as a large amount of functionality is exposed due to the normal min-cut partitioning. For the case 'SecPart + NormPlace', the HD increases to 43.87%, which proves the effectiveness of SecPart in concealing the functionality of a design. Finally, if we perform SecPlace on top of SecPart, compared to the 'SecPart + NormPlace' case, the attack correctness is reduced from 9.00% to 0.27% and the HD increases from 43.87% to 46.35%. The 'SecPart + SecPlace' design flow achieves the optimal security among four design flows. Overall, the SecPart algorithm is capable of approaching 50% HD, and the SecPlace algorithm can effectively achieve 0% attack correctness.

Figure 12.9 shows the area and total wirelength for four cases. Chip area and wirelength are two metrics that are commonly used to evaluate the performance of gate placement algorithm [13]. The 'NormPart+NormPlace' design flow is considered as

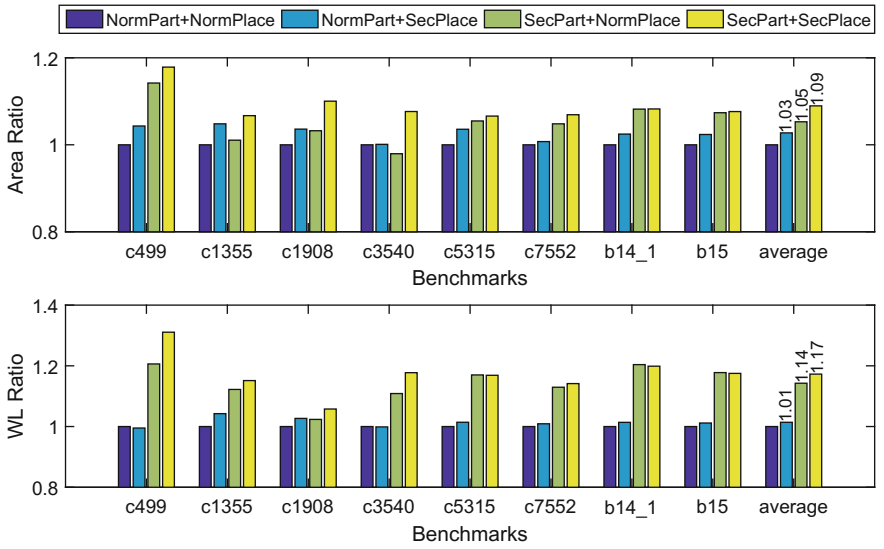


Fig. 12.9 Area and total wirelength overhead for four design flows (NormPart + NormPlace, NormPart + SecPlace, SecPart + NormPlace, SecPart + SecPlace) [47]. The NormPart + NormPlace is considered as the baseline design flow for calculating overheads, hence its overhead is 0% for all benchmarks

a baseline for calculating overheads. As seen, the main overheads come from the SecPart, as it requires a larger cut-set than NormPart to ensure 50% HD, which will inevitably increase the area and wirelength. The average overheads for SecPart are 5.29% on area and 14.27% on total wirelength. The SecPlace algorithm contributes to additional overhead because it perturbs the layout geometry to produce a placement with 0% attack correctness. Overall, the average overheads for 'SecPart+SecPlace' design flow are 8.95% on area and 17.27% on total wirelength.

12.6 Security Challenges in 3D/2.5D ICs

While providing the great promise in terms of performance and security, 3D/2.5D integration technology might also bring about adverse security impacts. In this section, we discuss various security challenges in 3D/2.5D ICs.

12.6.1 3D/2.5D IC Testing

IC testing is significant for detecting counterfeit components [12] and hardware Trojans [1, 19, 35] introduced in a global IC supply chain. The challenge of 3D IC testing stems from three aspects [24]: (1) a complicated test flow that consists of

pre-bond test, mid-bond test and post-bond test; (2) new test contents such as TSVs; (3) limited internal test accesses and mismatch between probe (50 μm) and fine-pitch micro-bumps (20 μm) for external test accesses.

Split fabrication further complicates the testing of 3D/2.5D IC. Pre-bond test performed at the untrusted foundries before die-bonding and post-bond test performed at the trusted foundry after die-bonding are both important to ensure the correctness, reliability and authenticity of an IC. However, the pre-bond test performed at the untrusted foundries might not be trustworthy. If a functional test is performed, a set of correct input–output patterns (for a die) are given to the untrusted foundries and thus there is information leakage that could help the attackers.

Emerging solutions to these challenges have been proposed. A test cost analysis has been performed to develop an economic and effective 3D test flow [37]. Redundant TSV has been proposed to reduce the yield loss due to TSV defects during fabrication [14]. Additional probe pads are integrated into each die to enable external test access and novel DfT architecture [25] for internal test access has been demonstrated. For 2.5D ICs, corresponding interposer-centric DfT architecture and post-bond testing strategy have also been proposed [7]. Moreover, secure test mechanisms such as the secure split test [8] might be employed to ensure the security of 3D/2.5D IC testing.

12.6.2 3D/2.5D IC Authentication

3D/2.5D IC is designed and fabricated by stacking/connecting multiple conventional 2D dies. How these 2D dies are connected, and how secure the dies are, will determine the vulnerability of a 3D/2.5D design. These 2D layers may contain functional IPs that are provided by third-party IP vendors and may be fabricated by different foundries. The complicated global supply chain introduces new chances for attackers to insert inauthentic (counterfeit and maliciously modified) designs to compromise the performance and security of the whole chip. Once all the layers are bonded, it is difficult to detect an inauthentic layer in the middle since the stacking structure of 3D ICs complicates physical testing and electrical testing. Thus, security-aware authentication techniques before and after bonding are of great significance. More design-for-security run-time mechanisms can also be developed to detect and/or isolate the inauthentic layers during runtime.

12.7 Implications of 3D/2.5D-Based Obfuscation on CAD Tool

While 3D/2.5D-based obfuscation offers new security opportunities to thwart various attacks, it also brings about new design challenges to the CAD tool designers since corresponding security-aware design techniques have not been well developed for the emerging technology. We summarize some of the implications on different phases of a 3D/2.5D design flow as follows:

1. *Logic Synthesis:* When 2.5D split fabrication strategy is utilized, logic synthesis poses a new impact on security [15]. Since different gate types (e.g. a NAND gate or a NOR gate) are distinguishable by their layouts, the number of gate types used in a design actually affects the difficulty of netlist obfuscation. A netlist that is synthesized using a limited number of gate types will be easier to be obfuscated using 2.5D split fabrication. However, it restricts the optimization space for logic synthesis and will result in a less optimal synthesis solution.
2. *Partitioning:* Partitioning is the core of a security-aware 3D/2.5D design flow because it determines the portion of design that is hidden from the attacker. A gate-level partitioning selects wires and/or gates into the trusted tier that can maximally obfuscate the netlist and/or functionality. Designing an optimal partitioning that can balance performance and security is challenging.
3. *Placement and Routing:* With 3D/2.5D split fabrication, a security-aware P&R algorithm is important for maintaining the secrecy of hidden portion in the trusted tier. Conventional P&R algorithm will place two connected gates/pins close-by in order to reduce wirelength. Eliminating the relationship between connectedness of two gate/pins and their physical layout proximity demands a security-aware P&R algorithm.
4. *Design Verification:* IC testing is essential to ensuring not only the correctness and reliability of an IC, but also its integrity and authenticity. 3D/2.5D integration technology complicates the testing process by introducing more layers, new contents such as TSVs while providing limited test accesses. The split fabrication strategy introduces additional complexity into the test process. The development of efficient test flow, direct test access and effective design-for-test circuitries such as build-in self-test (BIST) circuits would mitigate the testing challenge for 3D/2.5D ICs.

12.8 Summary

The stacking structure of 3D/2.5D ICs enables a new split fabrication strategy to obfuscate and protect outsourced design from supply chain attacks. A secure split fabrication-enhanced 2.5D/3D IC design flow consists of two important phases: netlist partitioning (wire and/or gate lifting) and placement. The core of the design flow is partitioning, which determines the secret information hidden from the attacker. Overall, it requires a comprehensive analysis and optimization to obtain a secure 2.5D/3D IC design flow to prevent the supply chain attacks. The true potential of 3D ICs in presence of modern security challenges has not been investigated in substantial depth. With the effort made in 3D IC security characterization and modelling, future chip designers can take security into consideration at an early phase of the design while optimizing the chip for performance and power. Moreover, novel architectures such as memory-on-chip enabled by 3D integration offer new opportunities to apply aggressive (ie high-performance overhead) security policies and mecha-

nisms to obfuscate the information flow between memory and CPU. Future 3D CPU design can incorporate security and performance advantages in 3D integration while tackling the challenges in power management, thermal dissipation and testing.

References

1. Bao C, Forte D, Srivastava A (2014) On application of one-class SVM to reverse engineering-based hardware trojan detection. In: 2014 15th International symposium on quality electronic design (ISQED). IEEE, New York, pp 47–54
2. Bao C, Forte D, Srivastava A (2015) Temperature tracking: toward robust run-time detection of hardware trojans. IEEE Trans Comput-Aided Des Integr Circuits Syst 34(10):1577–1585
3. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing ic piracy using reconfigurable logic barriers. IEEE Des Test Comput 27(1):66–75
4. Bilzor M (2011) 3D execution monitor (3D-EM): using 3D circuits to detect hardware malicious inclusions in general purpose processors. In: Proceedings of the 6th international conference on information warfare and security, Academic Conferences Limited, p 288
5. Bobba S, Chakraborty A, Thomas O, Batude P, Pavlidis VF, De Micheli G (2010) Performance analysis of 3-D monolithic integrated circuits. In: IEEE international 3D systems integration conference (3DIC), 2010, IEEE, pp 1–4
6. Chang YC, Chang YW, Wu GM, Wu SW (2000) B*-trees: a new representation for non-slicing floorplans. In: Proceedings of the 37th annual design automation conference, ACM, pp 458–463
7. Chi CC, Marinissen EJ, Goel SK, Wu CW (2011) Post-bond testing of 2.5 d-sics and 3d-sics containing a passive silicon interposer base. In: IEEE international test conference (ITC), 2011, IEEE, pp 1–10
8. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT), 2013, IEEE, pp 196–203
9. Fiduccia CM, Mattheyses RM (1982) A linear-time heuristic for improving network partitions. In: 19th conference on design automation, 1982, IEEE, pp 175–181
10. Garrou P, Bower C, Ramm P (2011) Handbook of 3d integration: volume 1-technology and applications of 3D integrated circuits. Wiley, New York
11. Gartner Inc (2012) Market trends: Rising costs of production limit availability of leading-edge fabs. https://www.gartner.com/doc/2163515
12. Guin U, Huang K, DiMase D, Carulli JM, Tehranipoor M, Makris Y (2014) Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proc IEEE 102(8):1207–1228
13. Ho YK, Chang YW (2013) Multiple chip planning for chip-interposer codesign. In: 2013 50th ACM/EDAC/IEEE design automation conference (DAC), IEEE, pp 1–6
14. Hsieh AC, Hwang T (2012) TSV redundancy: architecture and design issues in 3-D IC. IEEE Trans Very Large Scale Integr (VLSI) Sys 20(4):711–722
15. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In: Presented as part of the 22nd USENIX security symposium (USENIX Security 13), pp 495–510
16. Jagasivamani M, Gadfort P, Sika M, Bajura M, Fritze M (2014) Split-fabrication obfuscation: metrics and techniques. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2014, IEEE, pp 7–12
17. Jung M, Song T, Wan Y, Peng Y, Lim SK (2014) On enhancing power benefits in 3d ICs: block folding and bonding styles perspective. In: Proceedings of the 51st annual design automation conference, ACM, pp 1–6

18. Khaleghi S, Da Zhao K, Rao W (2015) IC piracy prevention via design withholding and entanglement. In: The 20th Asia and south Pacific design automation conference, IEEE, pp 821–826
19. Li J, Lach J, (2008) At-speed delay characterization for IC authentication and trojan horse detection. In: IEEE international workshop on hardware-oriented security and trust (HOST), 2008, IEEE, pp 8–14
20. Liu B, Qu G (2016) VLSI supply chain security risks and mitigation techniques: a survey. Integr. VLSI J 55:438–448
21. Liu B, Wang B (2014) Embedded reconfigurable logic for asic design obfuscation against supply chain attacks. In: Proceedings of the conference on design, automation & test in Europe, European Design and Automation Association, p 243
22. Loh GH, Xie Y, Black B (2007) Processor design in 3d die-stacking technologies. IEEE Micro 27(3):31–48
23. Lu T, Srivastava A (2015) Electromigration-aware clock tree synthesis for tsv-based 3d-ics. In: Proceedings of the 25th edition on Great Lakes symposium on VLSI, ACM, pp 27–32
24. Marinissen EJ (2012) Challenges and emerging solutions in testing TSV-based 2 1/2D-and 3D-stacked ICs. In: Proceedings of the conference on design, automation and test in Europe, EDA Consortium, pp 1277–1282
25. Marinissen EJ, De Wachter B, O'Loughlin S, Deutsch S, Papameletis C, Burgherr T (2014) Vesuvius-3D: a 3D-DfT demonstrator. In: IEEE international test conference (ITC), 2014, IEEE, pp 1–10
26. Narasimhan S, Yueh W, Wang X, Mukhopadhyay S, Bhunia S (2012) Improving IC security against trojan attacks through integration of security monitors. IEEE Des Test Comput 29(5):37–46
27. Plaza SM, Markov IL (2015) Solving the third-shift problem in ic piracy with test-aware logic locking. IEEE Trans Comput-Aided Des Integr Circuits Syst 34(6):961–971
28. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: Proceedings of the 49th annual design automation conference, ACM, pp 83–89
29. Rajendran J, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Design, automation & test in Europe conference & exhibition (DATE), 2013, IEEE, pp 1259–1264
30. Rajendran J, Sinanoglu O, Karri R (2014) Regaining trust in VLSI design: design-for-trust techniques. Proc IEEE 102(8):1266–1282
31. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64(2):410–424
32. Rostami M, Koushanfar F, Rajendran J, Karri R (2013) Hardware security: threat models and metrics. In: Proceedings of the international conference on computer-aided design, IEEE Press, pp 819–823
33. Roy JA, Koushanfar F, Markov IL (2008) Epic: ending piracy of integrated circuits. In: Proceedings of the conference on design, automation and test in Europe, ACM, pp 1069–1074
34. Saban K (2011) Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, and power efficiency. Xilinx, White Paper
35. Salmani H, Tehranipoor M, Plusquellic J, (2009) New design strategy for improving hardware trojan detection and reducing trojan activation time. In: IEEE international workshop on hardware-oriented security and trust (HOST'09), 2009, IEEE, pp 66–73
36. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: IEEE international symposium on hardware oriented security and trust (HOST), 2015, IEEE, pp 137–143
37. Taouil M, Hamdioui S, Beenakker K, Marinissen EJ (2010) Test cost analysis for 3D die-to-wafer stacking. In: 19th IEEE asian test symposium (ATS), 2010, IEEE, pp 435–441
38. Tezzaron (2008) 3D-ICs and integrated circuit security. http://www.tezzaron.com/about/papers/3D-ICs_and_Integrated_Circuit_Security.pdf
39. Torrance R, James D (2009) The state-of-the-art in IC reverse engineering. In: Cryptographic hardware and embedded systems-CHES 2009, Springer, Berlin, pp 363–381

40. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ics using split fabrication. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2014, IEEE, pp 1–6
41. Vaidyanathan K, Liu R, Sumbul E, Zhu Q, Franchetti F, Pileggi L (2014) Efficient and secure intellectual property (IP) design with split fabrication. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2014, IEEE, pp 13–18
42. Valamehr J, Sherwood T, Kastner R, Marangoni-Simonsen D, Huffmire T, Irvine C, Levin T (2013) A 3-D split manufacturing approach to trustworthy system development. IEEE Trans Comput-Aided Des Integr Circuits Syst 32(4):611–615
43. Wendt JB, Potkonjak M (2014) Hardware obfuscation using puf-based logic. In: Proceedings of the 2014 IEEE/ACM international conference on computer-aided design, IEEE Press, pp 270–277
44. Xiao K, Tehranipoor M (2013) BISA: Built-in self-authentication for preventing hardware trojan insertion. In: IEEE international symposium on hardware-oriented security and trust (HOST), 2013, IEEE, pp 45–50
45. Xiao K, Forte D, Tehranipoor MM (2015) Efficient and secure split manufacturing via obfuscated built-in self-authentication. In: IEEE international symposium on hardware oriented security and trust (HOST), 2015, IEEE, pp 14–19
46. Xie Y, Bao C, Serafy C, Lu T, Srivastava A, Tehranipoor M (2015) Security and vulnerability implications of 3d ics. IEEE Trans Multi-Scale Comput Syst 2(2):108–122
47. Xie Y, Bao C, Srivastava A (2015) Security-aware design flow for 2.5 d ic technology. In: Proceedings of the 5th international workshop on trustworthy embedded devices, ACM, pp 31–38
48. Yasin M, Saeed SM, Rajendran J, Sinanoglu O (2016) Activation of logic encrypted chips: Pre-test or post-test? In: 2016 design, automation & test in Europe conference & exhibition (DATE), IEEE, pp 139–144

Part V
Other Hardware Obfuscation Building Blocks

Chapter 13
Obfuscation and Encryption for Securing Semiconductor Supply Chain

Ujjwal Guin and Mark M. Tehranipoor

13.1 Cryptographic Primitives

With the advent of the Internet, security has become an integral part of our day-to-day lives. Modern cryptography is used in almost every application in communication systems including department of defense, mobile, banking, medical, and many more applications to provide security and protection against unwanted access by an adversary. However, it is less widely used in securing semiconductor supply chain from various different attacks which include IP piracy, IP overuse, and IC overproduction. In the following, we will first introduce various cryptographic primitives and then use them to secure semiconductor supply chain.

Figure 13.1 shows a taxonomy of cryptographic primitives. These primitives are broadly classified based on their usage—(*i*) encryption and decryption, and (*ii*) authentication. There are symmetric and asymmetric ciphers for the encryption and decryption of text messages. Symmetric ciphers are classified into stream ciphers and block ciphers. On the other hand, we have discrete logarithm algorithm [1], Rivest-Shamir-Adleman (RSA) algorithm [2], and elliptic curve algorithms [3, 4] for asymmetric ciphers. Message Authentication Codes [5, 6] and digital signatures [7] are widely used for integrity, authentication, and non-repudiation. We will briefly describe these primitives below.

U. Guin (✉)
Auburn University, Auburn, AL, USA
e-mail: ujjwal.guin@auburn.edu

M.M. Tehranipoor
University of Florida, Gainesville, FL, USA
e-mail: tehranipoor@ece.ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_13

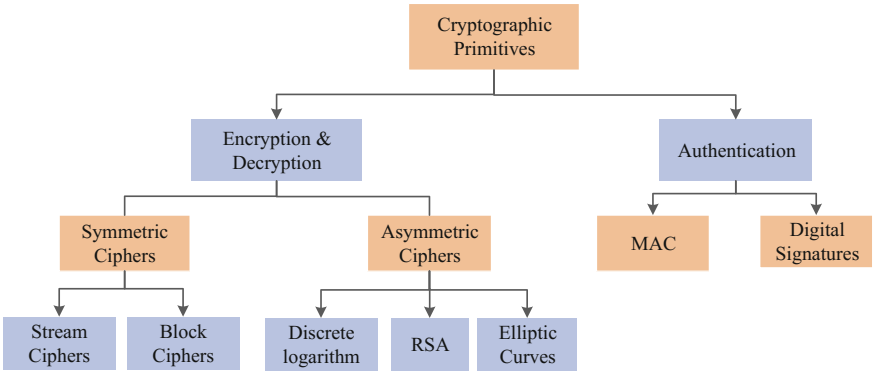


Fig. 13.1 A taxonomy of cryptographic primitives used in modern communication systems

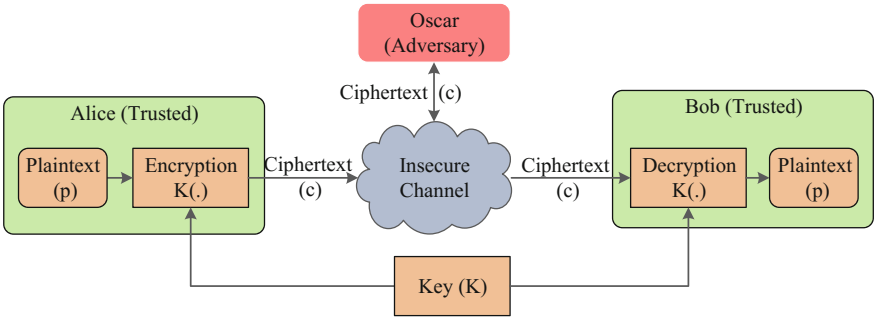


Fig. 13.2 Symmetric key cryptographic system

13.1.1 Symmetric Ciphers

Symmetric ciphers are used in symmetric key cryptographic systems (cryptosystems), where two trusted parties (Alice and Bob) want to communicate privately with each other. Figure 13.2 shows a symmetric key cryptosystem, where Alice encrypts the plaintext (p) with a key (K) and transmits the ciphertext ($c = K(p)$) resulting in a secure communication channel. Bob receives the ciphertext, c, and then performs decryption to construct the plaintext p, where $p = K(c)$. An adversary, Oscar, observes the ciphertext and cannot construct the original plaintext as he does not possess the key, K. Note that both the encryption and decryption process use the same unique key (K) for these cryptosystems. The security of these systems lies in the fact that Alice and Bob's shared key remains a secret. This also implies that Bob and Alice must have previously agreed upon a key or secretly transferred the key from one party to the other.

Encryption. The encryption process can be expressed as:

$$\begin{aligned} c &= f_K(p) \\ &= K(p) \end{aligned}$$

where, p is the plaintext; c is the ciphertext; and $f_K(.)$ or simply $K(.)$ is the encryption function.
Decryption. The decryption process can be expressed as:

$$p = f_K^{-1}(c)$$

where, $f_K^{-1}(.)$ is the decryption function.

For a symmetric key cryptosystem, the encryption key must be the key for decryption. Thus, we can simply write:

$$p = K(c).$$

13.1.1.1 Stream Ciphers

Stream ciphers are the first type of ciphers that are used in symmetric key cryptosystems. Stream ciphers encrypt plaintext in bitwise fashion, and decryption of the ciphertext is performed similarly. As stream ciphers are small and fast, they are mostly used in lightweight applications. The encryption and decryption are performed by modulo 2 additions (XOR equivalent).
Encryption. The encryption process can be written as follows:

$$\begin{aligned} c_i &= p_i + k_i \; mod \; 2 \\ &= p_i \oplus k_i \end{aligned} \tag{13.1}$$

Decryption. The decryption process is as follows:

$$\begin{aligned} p_i &= c_i + k_i \; mod \; 2 \\ &= c_i \oplus k_i \end{aligned} \tag{13.2}$$

where,

p_i is the i^{th} bit of the plaintext, p
k_i is the i^{th} bit of the key, K
c_i is the i^{th} bit of the ciphertext, c
$p_i, k_i, c_i \in \{0, 1\}$

Gilbert S. Vernam presented a perfectly secret cipher, which was granted the US Patent 1310719 in 1919 [8]. This cipher is most widely known as the one-time pad (OTP) but is also referred to Vernam's cipher in honor of its inventor. Interested

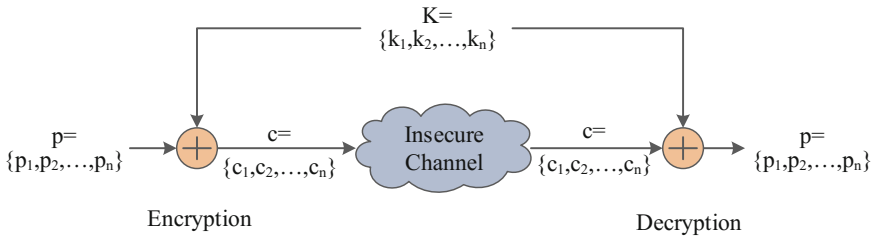


Fig. 13.3 Symmetric key cryptosystem using one-time pad

readers can find details of OTP and its perfect secrecy in the book, *Introduction to Modern Cryptography* [9].

Figure 13.3 shows a symmetric key cryptosystem that uses an OTP. The encryption and decryption are performed by using Eqs. 13.3 and 13.4. Even though it provides the perfect secrecy, the application of one-time pad in a symmetric key cryptosystem possesses two severe limitations. First, the length of the key must be equal to the data (plaintext) to be encrypted. It is practically impossible to maintain a key of such length for long messages. For example, a 4GB key would be needed to encrypt a movie or a computer game of size 4GB. Second, the key can only be used once. Otherwise, the cipher leaks information. Both of these limitations become even more problematic given that Alice needs to send the key securely to Bob beforehand. If Alice can already securely send a key (equal to the data) to Bob (e.g., in person) then why not send the data itself by the same method. For these reasons, OTPs are rarely useful in practical applications. OTP tends to be used in applications where the length of the plaintext is very small (few bits). In this chapter, we will use OTP in Sect. 13.4 to provide security against IP overuse and IC overproduction since it provides a very low area overhead compared to block-based symmetric ciphers.

Various other stream ciphers were used in practical applications over the years. For example, GSM mobile phones widely use A5/1 stream cipher for voice encryption. However, there are various attacks reported on the stream ciphers due to their shorter key length. It is computationally feasible to break a stream cipher by brute force with current computing resources. We shall not discuss stream ciphers in further detail as it is beyond the scope of this chapter. Interested readers can find more in any modern cryptography book, such as [9, 10].

13.1.1.2 Block Ciphers

Block ciphers are the symmetric ciphers that are widely used in modern cryptographic systems. These ciphers encrypt plaintext in blocks of n bits, unlike stream ciphers where the encryption takes place bitwise. For example, if the block size in 128 bits ($n = 128$), the plaintext needs to be divided into 128-bit blocks, which will be encrypted independently. The decryption of the ciphertext is also performed in the similar fashion like encryption. Data encryption standard (DES) [11] and advance

encryption standard (AES) [12] are the two popular block ciphers already employed in modern systems.

The data encryption standard (DES) encrypts a plaintext of a block of 64 bits (8 bytes). The key size for DES is 56 bits. The encryption process is performed in 16 rounds. Each round uses the Feistel Network, which performs the confusion and diffusion. The famous mathematician Claude Shannon proposed that confusion and diffusion are the key elements to achieve a strong encryption process. In confusion stage, the relationship between the key and ciphertext is obscured, whereas the effect of plaintext is spread out in the ciphertext at the diffusion stage [10]. DES was widely used in various cryptosystems until 1999, when its short key length made it susceptible to brute force attacks. The US National Institute of Standards and Technology (NIST) then recommended triple DES (3DES) where the key size is 112 bit. However, due to its implementation challenges and shorter key size, NIST called for proposals for a new block cipher, *Advance Encryption Standard*, in 1997.

In 2001, NIST selected *Rijndael* proposed by Joan Daemen and Vincent Rijmen as the new Advanced Encryption Standard (AES) [12]. The block size for AES encryption is 128 bits (16 bytes). There are three key sizes (128 bit, 192 bit, and 256 bit) that one can select for encryption and decryption processes depending on the desired level of security. There are 10, 12, and 14 internal rounds for 128 bit, 192 bit, and 256 bit keys, respectively. The number of the internal rounds also depends on the key size, allowing its security to scale more easily than DES. Each round consists of Byte Substitution Layer, ShiftRows Layer, MixColumns Layer, and Key Addition Layer, except for the final round. The detailed description of AES can be found in [10, 12].

13.1.2 Asymmetric Ciphers

The symmetric ciphers (OTP, DES, AES, etc.) cannot provide security over a communication channel even though they are inherently secure. The major challenges associated with the symmetric ciphers are key management and end-point authentication. For key management, a secure channel must be set to transfer the key between Alice and Bob. In addition, if Alice wants to securely communicate with both Bob and Tom, she needs to store the different keys for Bob and Tom. Thus, communication based solely on symmetric cryptography scales poorly when it is needed between multiple different parties. In end-point authentication, both Alice and Bob must verify the identity of each other. There is no way that Bob knows the identity of Alice when they use symmetric ciphers. For example, an adversary, Oscar, can replay an old ciphertext to Bob. Due to these shortcomings, Diffie and Hellman introduced an algorithm widely known as Diffie–Hellman Key Exchange based on discrete logarithm [1] that laid the foundation for public key cryptography or asymmetric key cryptography. At the same time, Rivest, Shamir, and Adleman proposed an algorithm based on integer factorization (widely known as RSA) [2] to achieve something similar.

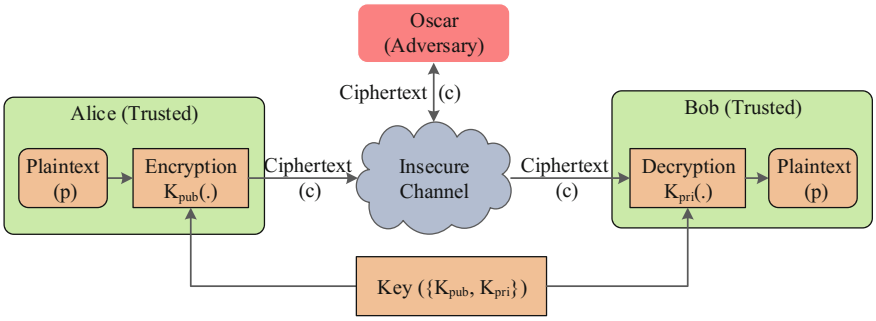


Fig. 13.4 Asymmetric key cryptographic system

Figure 13.4 shows an asymmetric key cryptographic system, where Alice encrypts the plaintext (p) with the public key (K_{pub}) of Bob and transmits the ciphertext (c) through an insecure channel. Bob receives the ciphertext, c, and then performs decryption using his private key (K_{pri}) to construct the plaintext p. An adversary, Oscar, observes the ciphertext and cannot construct the original plaintext as he does not possess the private key of Bob, K_{pri}.

Encryption. The encryption process can be expressed as:

$$\begin{aligned} c &= f_{K_{pub}}(p) \\ &= K_{pub}(p) \end{aligned}$$

where, $f_{K_{pub}}(.)$ or simply $K_{pub}(.)$ is the encryption function.

Decryption. The decryption process can be expressed as:

$$\begin{aligned} p &= f_{K_{pri}}(c) \\ &= K_{pri}(c) \end{aligned}$$

where, $f_{K_{pri}}(.)$ or simply $K_{pri}(.)$ is the decryption function.

The RSA algorithm has gained the popularity over the years due to its simplicity and has become almost synonymous with public key cryptography. It is worth noting, however, that RSA and public key cryptography in general are not intended to replace the symmetric ciphers (e.g., AES), but rather to provide support for end-point authentication (see Sect. 13.1.4). RSA requires two keys $K_{pub} = \{n, e\}$ and $K_{pri} = d$.

Encryption. The encryption process is as follows:

$$\begin{aligned} c &= K_{pub}(p) \\ &= p^e \ mod \ n \end{aligned} \tag{13.3}$$

Decryption. The decryption process is as follows:

$$\begin{aligned} p &= K_{pri}(c) \\ &= c^d \ mod \ n \end{aligned} \tag{13.4}$$

where $e, d \in \mathbb{Z}_n$, and $\mathbb{Z}_n$ is an integer ring with value $\{0, \ 1, \ \ldots, \ n-1\}$. Interested readers can find more description on RSA and its relevant mathematical background in [2, 10].

13.1.3 Message Authentication Codes

Message Authentication Codes (MACs) are widely used in message integrity verification and message authentication. MACs are commonly known as keyed hash functions which provide the cryptographic checksum [5]. MACs are formed based on secret symmetric keys, where the signing (Alice) and verifying (Bob) parties must share a secret key (k) before the communication. MACs are often generated by using secure hash functions and are called as a keyed-hash message authentication code (HMAC).

$$\begin{aligned} MAC &= HMAC(p, k) \\ &= H\Big[(k^+ \oplus opad)||H\Big[(k^+ \oplus ipad)||p\Big]\Big] \end{aligned}$$

where,

p is the message or plaintext;
$H(.)$ is the secure hash function (e.g., SHA-256) [13];
k^+ is the expanded symmetric key, k;
opad is outer pad; and
ipad is the inner pad.

One can find the detailed description of *HMAC* in [10, 14]. MACs can also be generated from block ciphers. One of the popular approach is to use AES in cipher block chaining (CBC-MAC) and counter with cipher block chaining (CCM) [15].

13.1.4 Digital Signature

A digital signature is a cryptographic technique widely used for authenticating an end user. It ensures the authenticity of a message originated from a unique user, who is the sole owner of it. Based on the scheme used for generating digital signatures, they can be of different types, such as RSA digital signature, Elgamal digital signature, and elliptic curve digital signature. In the following, we will only describe the digital signature generated from the RSA algorithm for simplicity.

Let us assume that Alice wants to communicate with Bob. Alice generates two keys ($KA_{pri} = d$ and $KA_{pub} = \{n, e\}$) and publishes her public key (say in her Web site). She now encrypts the message (plaintext, p) with her private key and forms the signature (sig) and sends $\{p, sig(p)\}$ to Bob.

$$\begin{aligned} sig &= KA_{pri}(p) \\ &= p^d \ mod \ n \end{aligned}$$

Alice is the only person, who can claim the sole ownership of sig as KA_{pri} is only known to her. Bob can retrieve the message by performing:

$$\begin{aligned} KA_{pub}(sig) &= (p^d \ mod \ n)^e \ mod \ n \\ &= p^{de} \ mod \ n \\ &= p \ mod \ n \end{aligned}$$

A simple equality check of $KA_{pub}(sig)$ and p provides necessary proof of the origination of the message. Due to the large size of the message, a small fixed size message digest (e.g., hash of p) is computed first and then a signature of the message digest is generated for end-point authentication.

13.2 Vulnerabilities in SoC Design and Fabrication Processes

So far, we have discussed various cryptographic primitives for securing a communication channel. In this section, we will now focus our discussion on:

- Design and fabrication stages of the semiconductor supply chain; and
- Lack of forward trust between various entities involved with these stages.

13.2.1 Design Process

The persistent trend of device scaling has enabled designers to fit more and more functionality on an SoC while reducing the overall area and cost of a system. The design of large complex integrated circuits (ICs) has evolved to a stage where it is extremely challenging to complete the entire design in-house. Therefore, the semiconductor industry has shifted gears to the concept of design reuse rather than designing the whole SoC from scratch. Nowadays, the SoC designers obtain licenses for various functional blocks (known as intellectual properties or IPs) for their SoCs to optimize the design process and decrease time-to-market. These IPs can be hard IPs (GDSII layout level designs), firm IPs (netlists and HDL designs with parameterized

constraints), and soft IPs (synthesizable register-transfer level (RTL) designs). In addition, the flow from RTL to GDSII is performed in many different places (even in different countries) mainly to reduce development costs and time-to-market.

13.2.2 Fabrication Process

The increased complexity of the fabrication process has resulted in a majority of SoC designers no longer maintaining a fabrication unit (or foundry) of their own. Building and maintaining such fabs for modern SoCs are reported to cost more than several billions of dollars and increasing as technology further scales [16]. Given the increasing cost, the semiconductor business has largely shifted to a contract foundry business model (horizontal business model) over the past two decades. In this business model, the SoC designers outsource their SoC design to the foundries and assemblies (mostly located offshore) for fabrication and packaging. At fabrication, foundries first fabricate defect-free wafers, which contains hundreds of working SoCs, which are commonly known as die. After fabrication, the foundry sends tested wafers to an assembly unit to cut the wafers into die, package the die, and perform final tests before shipping them to the market.

13.2.3 Forward Trust Issue

Counterfeiting and piracy have been recognized as a major concern in the SoC design and fabrication processes [17–19]. To be more specific, entities participating in this design and fabrication process must share their intellectual property (IP) with those further down the chain, and trust that they use it appropriately. We refer to this type of trust as 'forward trust' in this chapter. As an example, forward trust includes the trust of SoC designers by IP owners and the trust of foundries/assemblies by the IP owners and/or SoC designers. Note that forward trust is distinct from the more common notion of trust used in the literature when discussing the supply chain. This includes the trust of IP owners by SoC designers. To more carefully distinguish the two, we refer to the latter as 'backward trust' in this chapter since it involves trust of entities in the opposite direction of the supply chain. Ensuring backward trust is beyond the scope of this chapter. Interested readers can find more information on the detection and prevention of hardware Trojans in [20], which we believe is the key to ensuring backward trust.

The lack of forward trust may lead to the following vulnerabilities, which are highlighted as red in Fig. 13.5.

- *IP overuse*: Due to the increased complexity of SoCs and continuous demand for shorter time-to-market, the SoC designers use third party IPs in their design and pay a licensing fee for these 3PIPs. However, an untrusted SoC designer may

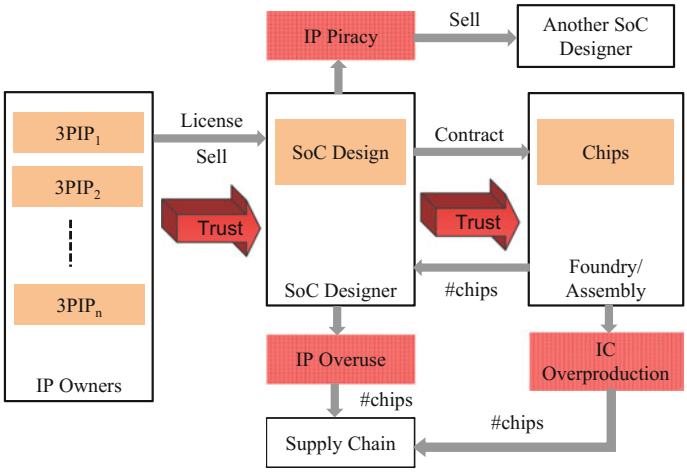


Fig. 13.5 Lack of trust between various entities involving in SoC design and fabrication

produce more ICs and report a lesser amount to the IP owners. This results in an illegal advantage and reduction in licensing cost. At the same time, the SoC designer may illegally use an IP that was licensed to be used in a different design. In short, the IP owners have no or little means to verify how many chips have been fabricated with their IPs and where their IPs have been used. When an untrusted SoC designer overuses the IPs by selling extra ICs in the market, the IP owners lose any possible revenue that could have been generated from those chips.

- *IP piracy*: IP piracy occurs when an untrusted entity in the supply chain acquires an IP legally or illegally and then gains undue advantages from it [21–24]. The IPs are of different forms, such as hard IPs (layout level designs), firm IPs (netlists and HDL designs with parameterized constraints), and soft IPs (synthesizable register-transfer level (RTL) designs). The attacks on these IPs are of different types depending on the nature of these IPs. An untrusted foundry can sell illegal copies of the hard IPs (GDSII files) that they receive from SoC designers for fabrication. An untrusted SoC designer can—(*i*) resell an IP 'as is' like untrusted foundries; (*ii*) strip the IP of certain features and sell it as a new IP; and (*iii*) add some extra features, which can be malicious (e.g., a backdoor) or nonmalicious, to the soft and firm IPs. Aside from the financial loss of the original IP owners, security is at stake whenever one uses one of these pirated IPs as the IC may leak secret information to the attacker or disable a system at some critical point in time.
- *IC overproduction*: Due to the complex fabrication process and extremely high cost of maintaining a fabrication unit, the SoC designers provide contracts to the foundries to fabricate their SoCs. The foundry agrees to manufacture certain number of chips and in return the SoC designer incurs all the cost for developing the masks and fabrication costs. An untrusted foundry/assembly can produce more than the number of chips they are contracted to manufacture [24–26]. As no

R&D cost is incurred for these chips, they can receive illegitimately larger profits by selling these chips with the name of SoC designer. In addition, an untrusted foundry/assembly can also overbuild chips practically at zero cost by reporting a lower yield (i.e., percentage of defect-free chips to the total number of chips) to the SoC designer [27, 28]. Like IP overuse, the IP owners or the SOC designers lose revenue due to the overproduction of ICs. Overproduced ICs may have reliability concerns as they simply end up in the market with minimal or no testing for evaluating reliability and/or functionality. Since these ICs have the same name of the SoC designers, their failure would tarnish company reputation.

13.3 Establishing Forward Trust in SoC Design and Fabrication Processes

Figure 13.6 shows the solution for establishing forward trust between the IP owners, SoC designers, and foundries/assemblies, which was introduced in [29, 30]. The entry point for sourcing out-of-contract chips (overused IPs and overproduced ICs) can be blocked by obfuscating the design. Obfuscation is a technique where a design is transformed to a different one often by using a unique key to obfuscate the inner details of the original design, thus preserving the original functionality. Whenever an untrusted entity wants to sell a chip to the market, it requires a unique key. On the other hand, IP piracy can be prevented by encrypting the design. In the following, we will discuss the prior works on netlist obfuscation and netlist encryption.

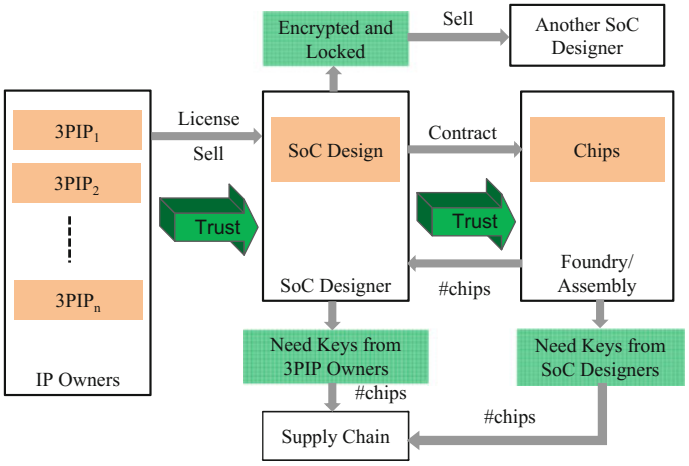


Fig. 13.6 Establishment of forward trust between various entities involving in SoC design and fabrication

13.3.1 Netlist Obfuscation

Roy et al. first proposed to obfuscate a netlist by using a lock (a set of XOR/XNOR gates), and it can only be unlocked by using a unique unlock key [26]. We will refer to this key as the chip unlock key or *CUK* for the rest of the chapter. The XOR and XNOR gates in the obfuscated netlist indicate 0 and 1 at *CUK* location, respectively. The basic problem with this approach is the direct relationship of the bits in *CUK* to the XOR/XNOR gates. An adversary can reverse engineer a netlist to identify these key gates and can easily obtain the key. In addition, *CUK* can be leaked to the primary output of the chip while manufacturing test [31]. Rajendran et al. addressed those problems by proposing different logic obfuscation techniques, where the identity of the key gates are hidden [31]. Chakraborty et al. proposed a methodology which can be integrated into the SoC design and manufacturing flow to simultaneously obfuscate and authenticate a design [23]. In this approach, the circuit operates in a normal mode when it receives a predefined sequence of patterns, known as a key, at its input. However, it is not clear how this key will be hidden from the foundries or assemblies as it is necessary to prevent overproduction. In addition, this technique does not address IP overuse.

13.3.2 Netlist Encryption

The Design Automation Standards Committee of the IEEE recently developed the P1735 standard [32] to provide the guidance for encryption and management of IPs. P1735 has been adopted by most IP and EDA vendors. In the encryption approach, the IP is encrypted with a random symmetric session key. This session key is then encrypted with the public keys of different EDA vendors and attached to the IP such that these vendors can later reconstruct the original IP.

Figure 13.7a shows a very simple IP (written in Verilog) which performs *AND* operation in every clock cycle. The IP is encrypted by using Synopsys *encryptP1735.pl* script [33] to protect it from any unwanted modification. The code inside the *'pragma protect* block (encircled in red in Fig. 13.7a) will be encrypted when we run *encryptP1735.pl* script. The encrypted IP is shown in Fig. 13.7b, where the code inside the *'pragma protect* block (encircled in red) is not recognizable. Unfortunately, this encryption approach cannot prevent placing additional features to an existing IP as it does not provide any integrity verification. Figure 13.7c shows this modified encrypted IP where an adversary adds an extra feature (*OR* operation) to the existing *AND* operation.

In this chapter, we will present a solution by adding an IP digest resulting from a cryptographic hash function [13] in the IP header (see Sect. 13.5) to prevent any unauthorized modifications. In addition, the encrypted IP does not provide any protection against copying of the whole IP to make an exact clone. However, combining the obfuscation and encryption can solve the threat of making a perfect clone as the

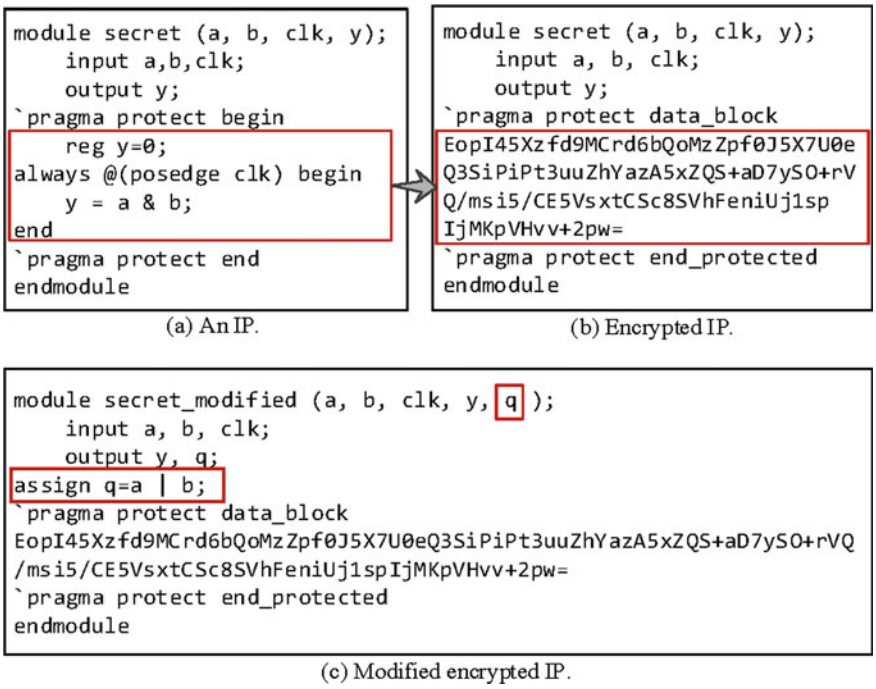


Fig. 13.7 Modification of an encrypted IP to add extra features

adversary needs to know the correct *CUK*. He can copy the netlist but cannot make a chip operational.

13.3.3 Flow for Establishing Forward Trust

Establishing forward trust in the semiconductor supply chain by ensuring the prevention of IP overuse, IP piracy, and IC overproduction requires the implementation of four separate measures. First, it is necessary to obfuscate the netlist of original IPs and SoCs, which can only be functional when they receive proper chip unlock keys, *CUKs*. Second, the secure transfer of these *CUKs* to the chips without being intercepted by the foundries/assemblies is required. A foundry/assembly can activate as many chips as it wants once it finds the proper *CUK*. Third, the testing is necessary before the activation of chips. An untrusted foundry/assembly can manipulate the manufacturing yield to build a stockpile of defect free chips by simply reporting a lower value to the SoC designer. Finally, the IPs need to be encrypted in such a way that the SoC designers cannot find the inner details of an IP. In addition, the IPs need to be tamper resistant and should be rendered useless if any modifications are performed on them.

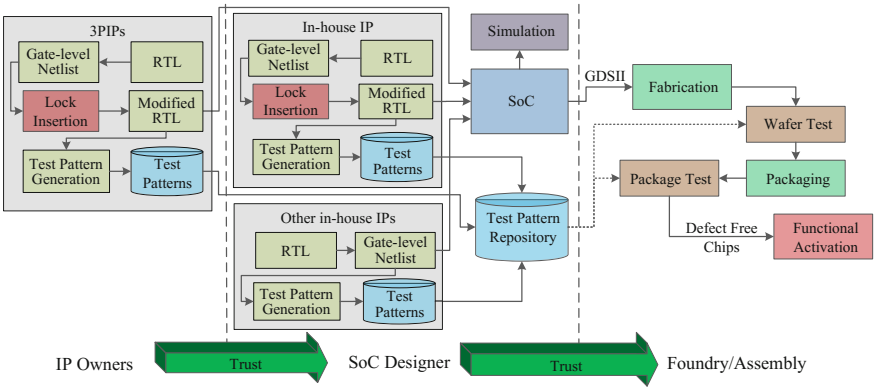


Fig. 13.8 FORTIS for enabling IC/3PIP metering to ensure forward trust in the SoC design and fabrication

Figure 13.8 shows our proposed design flow for establishing forward trust between IP owners, SoC designers, and foundries/assemblies. The design flow is very similar to the existing IC design flow. Two additional steps are required, and they are lock insertion and functional activation highlighted in red. The design process starts with the insertion of locks by using a set of key (XOR/XNOR) gates using an existing secure logic obfuscation technique. The chip produces functionally correct output when it is activated with a proper *CUK*. The number of XOR or XNOR gates depends on the level of security one wants to achieve. The gate level netlist is modified to enable manufacturing tests before the activation of chips (see details in Sect. 13.3.4).

Each 3PIP owner inserts key gates to lock their design to protect their IPs from overuse, and then generates test patterns. The SoC designer receives all these locked IPs and integrates them into the design. The SoC designer can also generate the test patterns for these locked IPs, as they do not need the *CUK* during test patterns. The SoC designer also inserts a lock in one of the in-house IP to protect against IC overproduction. The SoC designer collects all the test patterns from different IP owners or generates the test patterns for all these IPs and stores them in a pattern repository for future manufacturing tests (e.g., wafer and package tests). As all the 3PIPs are locked, the simulation may be a challenge for an SoC designer. Simulation support is described in Sect. 13.5 for these locked IPs.

The SoC designer sends the GDSII file corresponding to the SoC design to the foundry for fabrication and assembly. The foundry first processes wafers, which generally contains hundreds of dies in a single wafer. Foundry then performs wafer test to inspect dies to find gross defects. It rejects the whole wafer if too many dies are defective. After wafer tests, the defect-free dies are sent to assembly for packaging. The good chips are then sorted out by using package tests and the chips that have been damaged during the packaging process are discarded. Finally, each chip is unlocked using a valid *CUK* by the entity who performs the final manufacturing test (foundry, assembly, or SoC designer) before being sent to the market.

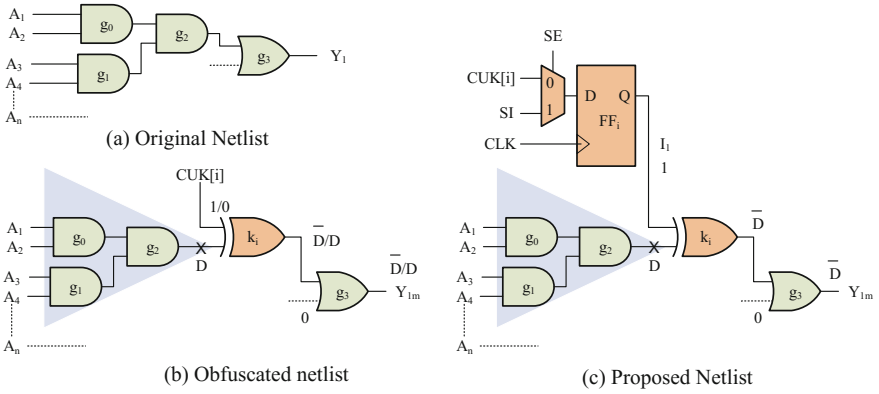


Fig. 13.9 Modification of an obfuscated netlist to enable manufacturing test before the activation of chips

13.3.4 Obfuscation Key Requirements: Enabling Structural Test Before Activation

Enabling manufacturing tests is one of the key requirements for preventing IP overuse and IC overproduction. A foundry or assembly can perform tests right after the chips are manufactured and reject the defective ones. If the activation takes place before the test, an untrusted foundry or assembly can pile up defect free chips by hiding actual yield to the SoC designer. In this section, we will present an architecture that enables structural tests before the activation of chips.

In the previously proposed architectures, the structural test patterns used in manufacturing tests are generated considering a predetermined *CUK* value. This is due to the existence of forward implication of the key gates. A forward implication exists when the inputs of a logic gate are assigned in a way that the output is uniquely identified [34]. For a two input XOR gate, one of the inputs will be transferred to the output when the other input is either 1 or 0. If we do not assign a value at $CUK[i]$, the ATPG tool will consider this input as unknown (X), and all the faults before the gate k_i (logic cone shown in shaded grey color in Fig. 13.9) will be untestable due to the nonexistence of the forward implication.

Let us illustrate this point with an example by considering a fault D, shown in Fig. 13.9b. This fault will be testable if it is being propagated to the output Y_{1m}. If $CUK[i]$ is 1 then the output of the gate k_i becomes $\bar{D}$, otherwise it becomes D. The corresponding Y_{1m} will be $\bar{D}$ or D depending on the $CUK[i]$.

$$Y_{1m} = \begin{cases} D \; if \; CUK[i] = 0 \\ \bar{D} \; if \; CUK[i] = 1 \end{cases}$$

To maintain a forward implication, it is required to provide the CUK during test pattern generation. Thus, the previously proposed designs need a CUK (for example, $CUK[i] = 0$ or $CUK[i] = 1$) before the structural test pattern generation to test the logic cone before the key gate. It is now necessary to load the same key into the chips before the manufacturing test begins, which is basically the activation of chips. An untrusted foundry/assembly can overbuild chips by asking for more keys and reporting a lower yield to the SoC designer if the chips are activated before the manufacturing tests.

To solve this problem, a flip-flop is added in between the $CUK[i]$ and the key gate to reach the key gate from the primary input (PI) or pseudo primary input (PPI). The ATPG tool assigns a unique value at the I_1 input during the scan shift through SI input (e.g., 1) for the key gate to transfer the fault $\bar{D}$ to the output Y_{1m}. Thus, the ATPG tool can generate test patterns without knowing the exact key. In this chapter, we refer structural or scan test patterns as patterns.

Figure 13.9c shows the desired netlist, where the key bit $CUK[i]$ is connected to a scan flip-flop (FF_i). The output of FF_i drives the key gate k_1. In the test mode, when the scan enable (SE) signal is asserted, this flip-flop becomes a part of the scan chain. The ATPG tool generates test pattern for this modified netlist with $n + 1$ inputs $(A_1, A_2, \ldots, A_n, I_1)$ rather than the original netlist (Fig. 13.9a) with n inputs $(A_1, A_2, \ldots, A_n)$ or obfuscated netlist (Fig. 13.9b) with n inputs $(A_1, A_2, \ldots, A_n)$ and $CUK[i] = 0/1$.

13.3.5 Attack Analysis and Countermeasure

Let us now consider an attack on the design presented in Fig. 13.9c, where an adversary wants to recover the key (CUK) by observing the test responses. As $CUK[i]$ is directly connected to the scan flip-flop, it will be propagated to the output and an adversary can recover the key by observing the non-changing response values for different test patterns. However, this attack may not be feasible in any design, which uses an on-chip test response compaction module. On-chip test response compaction is very common in today's designs [35–37]. Almost every chip uses response compaction to significantly reduce test data not out of preference, but of necessity.

Figure 13.10 shows an example of a compressor logic structure with a compression ratio 2 (8 scan chains and 4 outputs). Let us consider the key bits as Xs (an adversary does not know the value, but it will be 1 or 0 in real chips). The effect of Xs will be suppressed at the output $dout$ if at least two of the inputs of the XOR gates in the compressor are Xs. In this example, we can select scan chain 3, 4, and 5. At i^{th} clock cycle three key bits $(k - 1, k, k + 1)$ will be at $dout$ simultaneously and their individual effect cannot be separated.

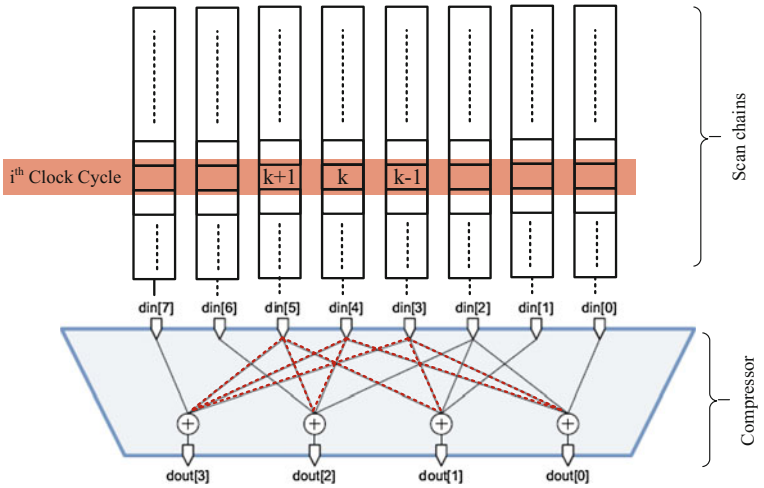


Fig. 13.10 Compressor logic structure example for 8-to-4 compressor [38]

$$
\begin{aligned}
dout[0] &= din[4] \oplus din[3] \oplus din[2] \oplus din[0] \\
&= k \oplus (k-1) \oplus \ldots \\
&= X \oplus X \oplus \ldots \\
dout[1] &= din[5] \oplus din[3] \oplus din[2] \oplus din[1] \\
&= (k+1) \oplus (k-1) \oplus \ldots \\
&= X \oplus X \oplus \ldots \\
dout[2] &= din[6] \oplus din[5] \oplus din[4] \oplus din[2] \\
&= (k+1) \oplus k \oplus \ldots \\
&= X \oplus X \oplus \ldots
\end{aligned}
$$

$$
\begin{aligned}
dout[3] &= din[7] \oplus din[5] \oplus din[4] \oplus din[3] \\
&= \ldots \oplus (k+1) \oplus k \oplus (k-1) \\
&= \ldots \oplus X \oplus X \oplus X
\end{aligned}
$$

The key propagation will fail as there is no forward implication for these XOR gates (two of the inputs are *Xs*). Thus, by selecting the scan chains carefully and place key gates at the same location on these scan chains, we can circumvent this attack.

One could argue that the diagnostics done for failure analysis may be impacted due to the compressed test responses. However, modern EDA tools provide diagnostic support (high defect coverage and accurate fault diagnostics) with compression in place [35–37]. The compacted responses collected during the test can be used for diagnostics without going back to the traditional DFT (without compressions). So

with this added feature, we do not see any reason why the SoC designers will not use test compression in their SoCs.

It is worthwhile to mention here that this key insertion flow does not impact the test process using JTAG [39] in the field as the test patterns are generated after the insertion of the key gates and has no impact on *CUK*. No modifications to the design are made after test pattern generation.

13.4 Secure Key Exchange Between IP Owner, SoC Designer, and Foundry

Thus far, we have discussed how locks inserted in a netlist can provide protection against IP overuse and IC overproduction. However, a major question, *"How can the IP owners or SoC designer transfer CUK to the chips without interception by any untrusted entity in the supply chain?"*, remains unanswered. In this model, an untrusted foundry or assembly becomes the adversary and attempts to find *CUK* such that it can activate any number of chips. In this section, we will present a secure communication protocol to transfer the *CUK* from SoC designer to the chips to prevent IC overproduction. Then, we will extend this communication protocol to transfer *CUKs* from 3PIP owners to the chips to prevent IP overuse.

To ensure the safe transfer of *CUK* from the IP owners or SoC designers to the chips, the following are required:

- *Message integrity*: The IP owners or SoC designers must ensure the integrity of the request received from the chips. If they detect an altered request, either modified by an attacker or errors in the transmission, it is necessary to stop the transmission of the encrypted *CUK*s.
- *End-point authentication*: The IP owners or SoC designers must verify that the request was initiated by the chips and not by an untrusted foundry or any other entity in the supply chain. As the chip cannot communicate by its own, the foundry or assembly only gets the information from the chip and forwards it to the SoC designer.
- *Confidentiality*: Only the IP owners or SoC designers and the chip should understand the contents of the transmitted messages.

All these can be achieved by using a combination of asymmetric and symmetric key encryption. The widely used Rivest-Shamir-Adleman (RSA) algorithm [2] is used as the asymmetric key encryption algorithm to provide message integrity and end-point authentication. Note that Discrete logarithm or elliptic curve algorithms [10] can also be used instead of RSA. Depending on the area budget, one can select one of the algorithms from the above. One-time pad (OTP) [10] is used for symmetric key encryption to provide the confidentiality. OTP has low area overhead as it only requires a simple XOR network for the encryption and decryption.

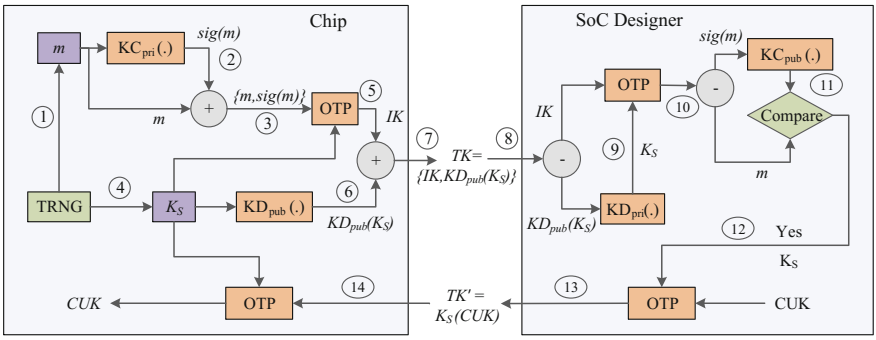


Fig. 13.11 Communication protocol for secure transfer of *CUK* from the SoC designer to the chip

13.4.1 Protection Against IC Overproduction

Untrusted foundries/assemblies produce more number of chips beyond the contract from the SoC designers and commonly known as IC overproduction. Here, the relevant parties are the SoC designers and untrusted foundries/assemblies. Our objective is to provide a secure transfer of *CUK* from SoC designer to the fabricated chips without being intercepted by an untrusted foundry or assembly.

Figure 13.11 shows our proposed protocol to securely transfer *CUK* from SoC designer to the fabricated chips. To achieve this, we need the keys (public key of the SoC designer (KD_{pub}) and private key (KC_{pri}) of the design) to be embedded in the design. Thus all the fabricated chips have the same CUK, KD_{pub}, and KC_{pri}. The SoC designer has the other two keys, KD_{pri}, and KC_{pub}. The steps for transferring the *CUK* from the SoC designer to the chip are listed below:

Step 1: The on-chip TRNG generates a message (m) which is unique for each and every chip.

Step 2: The message m is encrypted with the private key KC_{pri} stored in the chip to form a signature, i.e., $sig(m) = KC_{pri}(m)$. This signature will be used to validate message integrity and verify end-point authentication.

Step 3: The message m and its signature $sig(m)$ are concatenated.

Step 4: The TRNG generates a random session key (K_S), which is unique for every communication. This session key can be stored in a nonvolatile memory for future decryption to receive CUK. If the entire activation is performed while the chips are powered on, we can even store K_S in a volatile memory. This unique session key helps us to prevent replay attacks.

Step 5: A one-time pad (OTP) encrypts the concatenated message (m) and its signature ($sig(m)$) with K_S.

$$IK = K_S(\{m, sig(m)\}) = K_S \oplus \{m, sig(m)\}$$

Step 6: The session key, K_S, is encrypted with the public key, KD_{pub}, of the SoC designer.

Step 7: The transmission key is formed by concatenating encrypted K_S and IK. $TK = \{KD_{pub}(K_S), IK\}$. The foundry receives TK from the chip and forwards it to the SoC designer.

Step 8: Upon receiving the TK from the foundry, the SoC designer separates encrypted K_S and IK.

Step 9: Session key K_S is retrieved by decrypting $KD_{pub}(K_S)$ with KD_{pri}.

$$K_S = KD_{pri}(KD_{pub}(K_S))$$

Step 10: A one-time pad is used to decrypt IK to retrieve the concatenated m and its signature $sig(m)$.

$$IK \oplus K_S = K_S \oplus \{m, sig(m)\} \oplus K_S = \{m, sig(m)\}$$

Step 11: The SoC designer retrieves the message from the signature by using chip's public key, KC_{pub}.

$$KC_{pub}(sig(m)) = KC_{pub}(KC_{pri}(m)) = m$$

Step 12: A comparison is performed to match m and decrypted signature $sig(m)$. This step verifies the integrity of m and end-point authenticity. The SoC designer now knows that the TK is originally coming from the chip if m equals to the $KC_{pub}(sig(m))$, not from an attacker.

Step 13: After verifying the authenticity of the sender, the SoC designer encrypts CUK by using an OTP with the session key K_S and sends another transmission key (TK') to the foundry.

$$TK' = K_S(CUK) = K_S \oplus CUK$$

Step 14: The foundry applies this TK' to the chip. The chip now reconstructs the correct CUK after decrypting TK' by using the OTP with its stored session key, K_S.

$$K_S(TK') = K_S \oplus CUK \oplus K_S = CUK$$

This correct CUK is then stored in a nonvolatile memory (NVM) [40] to provide inputs to the key gates. The size of the NVM depends on the size of the CUK. One needs to make sure that the CUK values are not accessible by the JTAG [39] in the field.

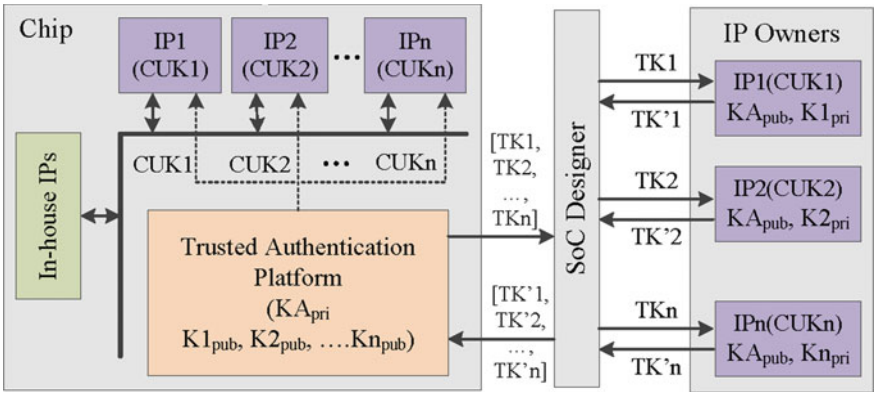


Fig. 13.12 Architecture and communication flow of FORTIS to prevent IP overuse

13.4.2 Protection Against IP Overuse

The overuse of IP occurs when an SoC designer makes a foundry manufacture extra chips (including IC overproduction) without the knowledge of the 3PIP owners, which results in a loss of revenue for the IP owners. The relevant parties are the IP owners, SoC designers and untrusted foundries/assemblies. An IP owner obfuscates his/her IP in such a way that it requires a unique *CUK* to become completely functional. This *CUK* has to be completely unknown to the SoC designer; otherwise, there will be no point of obfuscating an IP. Now the question becomes *how can an SoC designer simulate an SoC in the design phase if he does not know the CUK of a 3PIP?* We address this simulation problem in Sect. 13.5.

To control the number of chips from a manufacturing unit, our objective is to provide a secure transfer of *CUKs* from the different IP owners to the fabricated chips without being intercepted by an untrusted SoC designer, foundry, or assembly.

Figure 13.12 shows the architecture and communication flow to prevent IP overuse. Each IC contains a trusted authentication platform (TAP), which is introduced in the SoC design in order to reduce the area of each 3PIP by eliminating individual encryption/decryption blocks for each IP block and is trusted by all the 3PIPs in that SoC. In addition, TAP can be encrypted by our propose approach (see Sect. 13.5) such that inner details are hidden to the SoC designer and it is tamper resistant. The connection details between the TAP and 3PIPs are also obfuscated by the EDA tool such that SoC designers cannot add additional circuitry to observe *CUKs* and provide them to the 3PIPs directly. Note that we assume trusted EDA tools throughout the chapter, and it cannot be modified to get an undue advantage by the SoC designers.

Each IP contains a lock (i.e., the key gates) which can only be unlocked by using the correct chip unlock key CUK_i of IP i. This CUK_i is only known by the i^{th} IP owner. The IPs only receive CUK_is from the TAP for the activation. TAP holds its

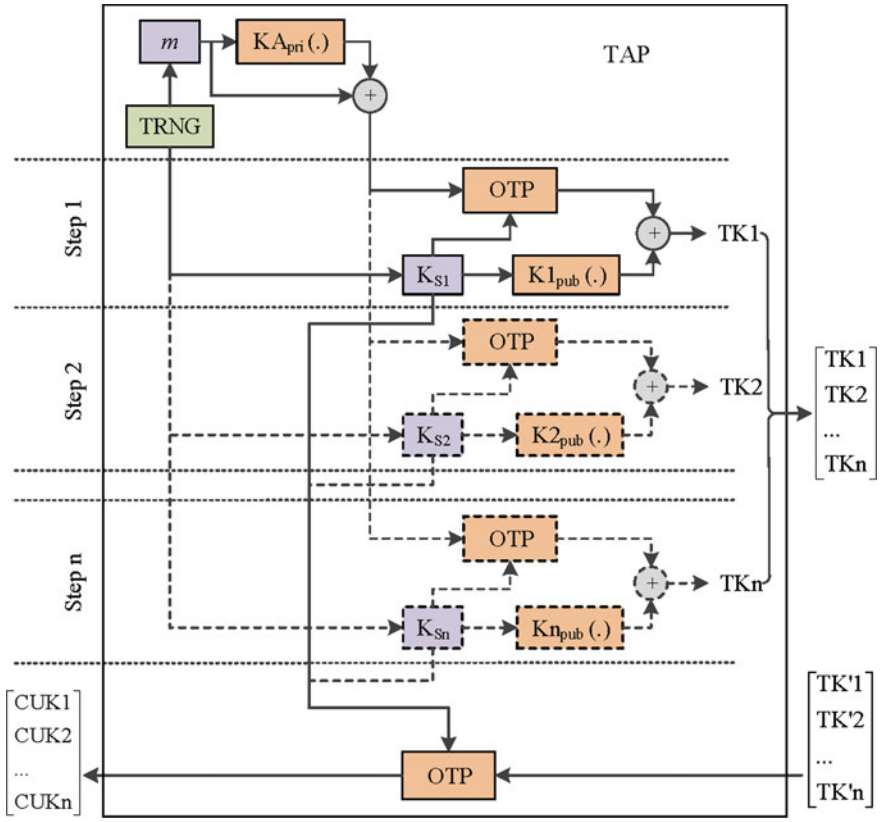


Fig. 13.13 Architecture of TAP and communication flow to reconstruct *CUKs* for all 3PIPs in a SoC

own private key (KA_{pri}) and public keys ($\{Ki_{pub}\}$) for all the IPs in the design. TAP generates the transmission keys ($TK1, TK2, \ldots, TKn$) using Steps 1 to 7 of Fig. 13.11 and sends them to the SoC designer. The SoC designer forwards each transmission key (TKi) to the corresponding IP owner. In return, the IP owners send the encrypted chip unlock key ($TK'i$) to the SoC designer. Upon receiving all the $TK'i$s from the IP owners, the SoC designer sends them to the foundry to unlock each IP in the fabricated chips.

Figure 13.13 shows the generation of transmission keys by the trusted authentication platform. TAP has a built-in TRNG, which generates a message (m) and separate session keys (K_S) for all different IP owners. First, the signature of m is generated and then concatenated with its signature. This ensures the message integrity and end-point authentication for all the IP owners and also that the request is indeed coming from the trusted platform module. TAP then generates one transmission key in each step. At step 1, a session key (K_{S1}) for IP owner 1 is obtained from the TRNG. This session key helps to encrypt $\{m, sig(m)\}$ and the encrypted output is concatenated

with the encrypted K_{S1} to form $TK1$. At step 2, a different session key (K_{S2}) for IP owner 2 is received from the TRNG. This session key is then used to encrypt $\{m, sig(m)\}$, and the encrypted output is concatenated with the encrypted K_{S2} to form $TK2$. In a similar fashion, all the transmission keys (TKi) are generated. Then the foundry receives all the TKi, sends them to the SoC designer, and waits for the encrypted $CUKs$.

After receiving the transmission keys ($TK'is$), the foundry applies them to the TAP, which decrypts these $TK'is$ by using its session keys, K_Ss, to generate the chip unlock keys, CUK_is, for all different IPs.

13.4.3 Area Overhead Analysis

The purpose of the area overhead analysis is to analyze the overhead from different modules of the communication protocol described in Sect. 13.4.1. As the activation of chips is performed only once, the time for the activation (the end-point authentication and then transfer of keys from IP owners/SoC designers to the chips) is not the major concern, rather than the area overhead. We need to optimize the area by selecting minimum size crypto modules. The below modules are the contributors for the area overhead:

(1) *RSA module:* The RSA crypto primitive is used in the design to encrypt the session keys and generate the signature, which makes up a major part of the area overhead. However, this overhead can be reduced significantly, if we choose a slower RSA module. As the speed of encryption is not our major concern, we can select a slower, but more area efficient RSA module. It is reported that a minimum size RSA datapath can be implemented by using only 861 gates [41].
(2) *OTP module:* The size of the one-time pad depends on the size of the CUK. For a 128 bit CUK, we need 128 XOR gates. The same OTP can be used in multiple passes for the encryption of $\{m, sig(m)\}$ and decryption of TK', where the typical size of the RSA signature is 1024 or 2048 bits.
(3) *Keys gates:* The size due to keys also depends on the CUK. To implement one key bit, we need one XOR/XNOR gate and a scan flip-flop.
(4) *RSA Keys:* Extra storage or logic is needed to keep or generate at least 1024 or 2048 bits (80-bit or 112-bit block cipher equivalent security, respectively) KC_{pri} for chips or KA_{pri} for TAP.
(5) *TRNG:* A single TRNG is used for generating the message, m and session keys, K_Ss. We propose the use of an area efficient cryptographically secure pseudo-random number generator [42] or [43] depending on the implementation choice.
(6) *Nonvolatile memory:* The size of the nonvolatile memory depends of the session keys, K_Ss. We need nonvolatile memory of $|m||sig(m)|$ bits to store K_S.

There is no area overhead of any 3PIPs for preventing IP overuse except for the key gates. The trusted authentication platform (TAP) provides the $CUKs$ to all

different 3PIPs. The primary motivation for implementing TAP in any design for a SoC designer is that they need to prevent IC overproduction.

13.4.4 Security Analysis

The security of our proposed protocol is of prime importance to prevent the overproduction of ICs and overuse of 3PIPs. In the following, we will perform the security analysis of our proposed approach.

(1) *Exhaustive key search*: The length of a chip unlock key, *CUK*, should be long enough such that it can withstand exhaustive key search or brute-force attacks. We need to achieve at least 80 bits of security as this is the lower minimum requirement for exhaustive key search [10]. To achieve this, we require 80 key gates (XOR/XNOR). However, the key size may be increased up to 256 bits for higher security, which will hardly impact the overall area of a modern design.
(2) *Encryption*: In our approach, we use RSA to encrypt the session key and generate signature. One can achieve 80 bit of security while the key length is 1024 bits. However, 128 bit security can be achieved with the key length of 3072 bits [44]. Depending on the area budget, one can select a desired security level of n bits. We have used one-time pad to encrypt $\{m, sig(m)\}$. As the session keys, K_Ss, are generated from a TRNG, a perfect secrecy can be achieved. Thus, we can achieve an overall RSA equivalent secrecy in our proposed protocol.
(3) *Man-in-the-middle attack*: As the key-pairs for the RSA are generated by the IP owners and reside in the circuit, no key transfer is required. This prevents an attacker (e.g., untrusted foundry) from becoming a man-in-the-middle.
(4) *Replay attack*: In this attack scenario, the attacker copies a message between two parties and replays that message to one or more of them. Our proposed protocol is inherently resistant to replay attacks as a new session key, K_S, is generated every time during encryption, which will be used later to decrypt the encrypted *CUK*.
(5) *Reverse engineering*: It is extremely hard for an attacker to find *CUK* by reverse engineering for modern designs manufactured with a latest technology node (22 nm or lower). Even if we assume that reverse engineering is possible to find the key, an attacker cannot feed the *CUK* to a chip, as they do not know the private key of the SoC designer (KD_{pri}) to retrieve K_S. As the session key, K_S, is unique for every chip, it is not economical for the attackers to retrieve K_S for each chip by reverse engineering. We also assumed that the attacker cannot model the TRNG to predict its output after observing certain K_Ss.
(6) *Tampering RSA Keys*: In this attack scenario, an untrusted foundry reconstructs new masks to replace the keys, KC_{pri} and KD_{pub}, with its own. This enables the foundry to unlock unlimited number of chips when it receives the *CUK*s from the IP owners. Fortunately, this attack can easily be prevented by the IP owners. The SoC designer can request only one locked chip and then verify the correct

keys. If the foundry replaces KC_{pri} and KD_{pub} by its own, the SoC designer will not be able to unlock the chip and consequently, it can detect mask modification.

(7) *Tampering TRNG*: An untrusted foundry can modify the masks to bypass the TRNG and write a permanent value for K_Ss and m. Once it knows the CUK, it can unlock any number of chips. Fortunately, this attack can also be detected by the IP owners and can be prevented. Like before, the SoC designer can request few locked chips to monitor the message, m and the session key, K_S. If either m or K_Ss from these chips are the same or biased, it will definitely be the indication of the tampering of TRNG. As it is extremely expensive to design a new set of masks, there is little economic incentive for an untrusted foundry to manufacture a product with two different set of masks.

13.5 IP Piracy

We have discussed so far the secure transfer of the chip unlock keys from the IP owners or SoC designers to the untrusted foundries and assemblies to prevent IP overuse and IC overproduction. This final section will introduce a SoC design flow to prevent IP piracy, such as cloning, and unwanted modification of IPs by the untrusted SoC designers and foundries.

Let us first discuss IEEE P1735 standard to prevent IP piracy. Figure 13.14 shows the normal practice for the generation of encrypted IPs and retrieval of the original IPs inside the EDA tool [33]. An IP owner encrypts a part of his/her IP (referred as *IP Data*) to protect from the SoC designers and other IP owners by using a random symmetric session key (K_S). K_S is then encrypted with the public key of the EDA tool to form EnK_S and is attached to the encrypted IP such that the EDA tool can later reconstruct the original *IP data*. In this model, the EDA tools are always trusted. One

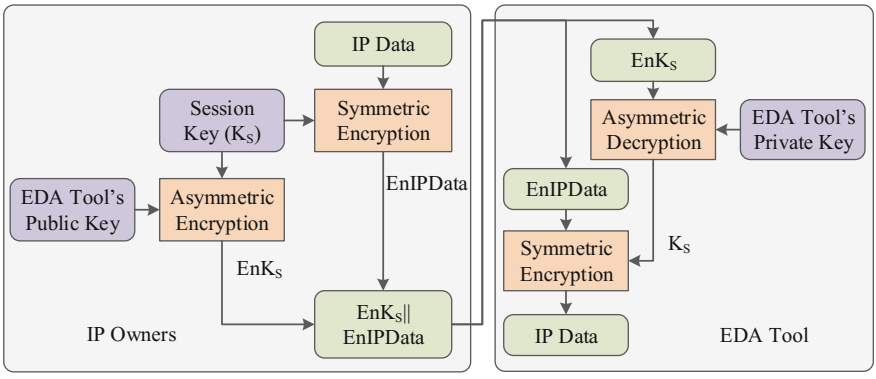


Fig. 13.14 Encryption and decryption process of IP using IEEE standard 1735 [33]

can find the public key of different EDA tools in the corresponding EDA vendors' Web site.

During the synthesis or simulation processes, the EDA tool first separates EnK_S from the encrypted IP. It then recovers the session key, K_S, by using its private key. Once the K_S is recovered, the tool uses the session key to recover *IP data* from the encrypted IP.

This IP encryption process prevents an adversary from finding the inner details of an IP. Unfortunately, it neither prevents placing additional features to an existing IP (already described in Sect. 13.3.2), nor making a clone which the exact copy of the original IP. *How can we then prevent an adversary placing additional features to an existing IP or making a perfect clone?* We can prevent an adversary to modify an IP by simply introducing integrity verification in the encryption process. Similarly, the obfuscated netlist (see details in Sect. 13.3.3) inherently prevents the cloning of IPs. As each IP is locked by using a set of key gates, even if the attackers copy the netlist completely, they cannot unlock it without a proper *CUK*. If we incorporate these two solutions in the design phase, we can completely eliminate IP piracy.

Note that the simulation of an SoC having these locked 3PIPs needs to be addressed, as these IPs will work properly only upon receiving a proper *CUK*. It is necessary to protect these *CUKs* from the SoC designer. Otherwise, there is no point of adding the locks into the IPs in the first place. In the following, we will present a solution of simulating a 3PIP by providing a valid *CUK* securely to the simulation tool without the interception by untrusted SoC designers.

The integrity verification of an IP is necessary to prevent IP modification by an untrusted SoC designer as we described before. We use a cryptographic hash function [13] to create an IP digest (see message digest [10]) to make it resistant against modification. Any modification, including addition or deletion of extra features, to a 3PIP will result in a different IP digest than the original one, which can easily be detected by comparison in an EDA tool.

Figure 13.15 shows our proposed flow to prevent cloning and modification of 3PIPs. The IP owners first compute an IP digest which is the hash of the entire locked netlist. An IP header is then created which contains the *CUK* for the sim-

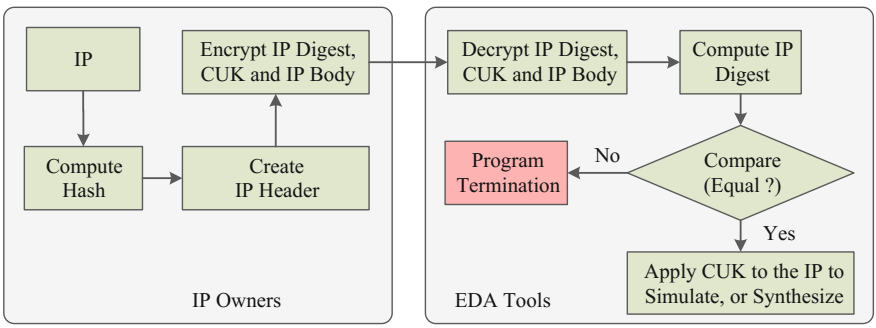
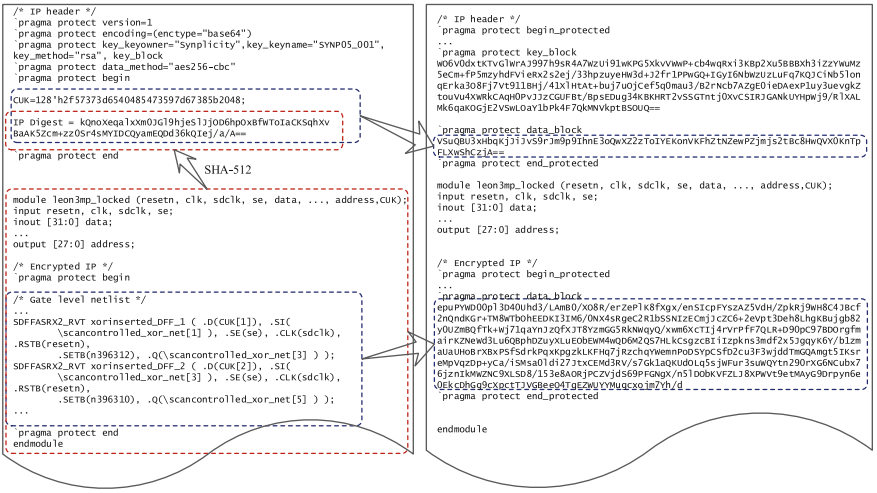


Fig. 13.15 Proposed flow to prevent IP piracy integrated into FORTIS



(a) Locked IP. (b) Locked and encrypted IP.

Fig. 13.16 IP header insertion for the simulation of a locked IP

ulation of an SoC and the IP digest. The IP is then encrypted (the code inside the *'pragma protect* blocks) by using a symmetric encryption method (e.g., Advanced Encryption Standard-Cipher Block Chaining (AES-CBC) [45]) recommended in encryptP1735.pl script [33]. This symmetric key is now encrypted by the public keys of different EDA vendors such that these vendors can later on decrypt them to get the IP.

We propose a new IP digest comparison flow during synthesis and simulation of SoCs. The EDA tool first needs to decrypt the encrypted portion of the IP header and the IP body. An IP digest has to be calculated from the decrypted IP by using the same hash function used before to form the IP digest. A comparison needs to be performed with the IP digest retrieved from the IP header and newly computed IP digest. If both of them are equal, then it is ensured that no modifications to the program has been made; otherwise, the program has to be terminated.

Figure 13.16 shows an example of our proposed encrypted IP. We use SHA-512 [13] to form an IP digest, which is attached to the IP header along with the *CUK*. We use Synopsys encryptP1735.pl script [33] to encrypt the IP header and IP body. Figure 13.16a shows a locked IP. The encryption is carried out in two parts - (*i*) The IP vendor encrypts the IP data (data block) using its own symmetric key which is called the data key. We use aes256-cbc as symmetric encryption algorithm to encrypt the data block. (*ii*) The IP vendor then encrypts the data key with its public key by using asymmetric encryption to create a key block. The encryption version, encode type, key owner, key name, and key method need to be mentioned. We use RSA as asymmetric encryption to generate the key block which is attached to the IP header (see Fig. 13.16b).

13.6 Conclusion

In this chapter, we have presented various cryptographic primitives from modern cryptography and used them along with obfuscation to provide forward trust for different entities involved in the SoC design and manufacturing process. We obfuscate the netlist by using a set of key gates. The obfuscated netlist works properly when it receives a correct chip unlock key during the activation. We also present a communication protocol between the fabricated chips and the SoC designers/IP owners for preventing IP overuse and IC overproduction that operates before the activation of chips by the unlock key. Our proposed modification does not have any impact on manufacturing test process. To address IP overuse, we presented a trusted authentication platform in the SoC. This TAP is trusted by the all parties involved in the SoC design. The encrypted IP with additional IP digest check prevents the SoC designer from IP piracy. As the IPs are locked by using a set of XOR/XNOR gates, even if the attackers copy the netlist completely, they cannot unlock it without the proper *CUK*, which prevents IP cloning. Finally, we have shown that the design flow for ensuring forward trust is resistant to all known attacks.

References

1. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inf Theory 22(6):644–654
2. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
3. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48(177):203–209
4. Miller VS (1985) Use of elliptic curves in cryptography. Conference on the theory and application of cryptographic techniques. Springer, Berlin, 1985, pp 417–426
5. Krawczyk H, Canetti R, Bellare M (1997) HMAC: keyed-hashing for message authentication
6. FIPS, "198-1," (2007) The keyed-hash message authentication code (HMAC). National Institute of Standards and Technology
7. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
8. Vernam GS (1919) Secret signaling system. US Patent 1,310,719
9. Katz J, Lindell Y (2014) Introduction to modern cryptography. CRC Press, Boca Raton
10. Paar C, Pelzl J, (2009) Understanding cryptography: a textbook for students and practitioners. Springer Science & Business Media, Berlin
11. Standard DE (1977) Federal information processing standards publication 46. National Bureau of Standards, US Department of Commerce
12. Pub NF (2001) 197: Advanced encryption standard (aes). Fed Inf Process Stand Publ 197:0311–441
13. NIST (2012) FIPS PUB 180-4: secure hash standard
14. NIST (2008) FIPS PUB 198-1: the keyed-hash message authentication code (HMAC)
15. Dworkin M (2004) Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality (nist sp 800-38c)
16. Yeh A (2012) Trends in the global IC design service market. DIGITIMES research
17. Tehranipoor MM, Guin U, Forte D (2015) Counterfeit integrated circuits: detection and avoidance. Springer, Berlin
18. Guin U, Huang K, DiMase D, Carulli J, Tehranipoor M, Makris Y (2014) Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proc IEEE 102(8):1207–1228

19. Guin U, DiMase D, Tehranipoor M (2014) Counterfeit integrated circuits: detection, avoidance, and the challenges ahead. J Electron Test 30(1):9–23
20. Tehranipoor M, Salmani H, Zhang X (2014) Integrated circuit authentication: hardware trojans and counterfeit detection. Springer, Berlin
21. Castillo E, Meyer-Baese U, García A, Parrilla L, Lloris A (2007) "IPP@HDL: efficient intellectual property protection scheme for IP cores. IEEE Trans Very Large Scale Integr Syst 15(5):578–591. http://dx.doi.org/10.1109/TVLSI.2007.896914
22. Kahng AB, Lach J, Mangione-Smith WH, Mantik S, Markov IL, Potkonjak M, Tucker P, Wang H, Wolfe G (2006) Constraint-based watermarking techniques for design IP protection. Trans Comput-Aided Des Integr Circuits Syst 20(10):1236–1252. http://dx.doi.org/10.1109/43.952740
23. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(10):1493–1502
24. Tehranipoor M, Wang C (2012) Introduction to hardware security and trust. Springer, Berlin
25. Koushanfar F, Qu G (2001) Hardware metering. In: Proceedings of the IEEE-ACM design automation conference, pp 490–493
26. Roy J, Koushanfar F, Markov I (2008) EPIC: ending piracy of integrated circuits. In: Proceedings of the conference on design, automation and test in Europe, pp 1069–1074
27. Contreras G, Rahman T, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: Proceedings of the international symposium on fault and defect tolerance in VLSI systems
28. Rahman MT, Forte D, Shi Q, Contreras GK, Tehranipoor M (2014) CSST: preventing distribution of unlicensed and rejected ICS by untrusted foundry and assembly. In (2014) IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT). IEEE, pp 46–51
29. Guin U, Shi Q, Forte D, Tehranipoor M (2016) FORTIS: a comprehensive solution for establishing forward trust for protecting IPs and ICs. ACM Trans Des Autom Electron Syst (TODAES)
30. Guin U (2016) Establishment of trust and integrity in modern supply chain from design to resign
31. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Security analysis of logic obfuscation. In: 2012 49th ACM/EDAC/IEEE design automation conference (DAC), pp 83–89
32. DASC (2014) 1735–2014 - IEEE approved draft recommended practice for encryption and management of electronic design intellectual property (IP)
33. Synopsys (2014) Synopsys FPGA synthesis synplify pro for lattice: user guide
34. Bushnell M, Agrawal V. (2000) Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits. Springer, Berlin
35. Synopsys (2015) Compression for highest test quality and lowest test cost. https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Test/Pages/dftmax-ultra-ds.aspx
36. Synopsys (2015) High quality, low cost test. https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Test/Pages/DFTMAX.aspx
37. Nagaraj P (2015) Choosing the right scan compression architecture for your design. Technical report
38. Synopsys (2015) DFT compiler, DFTMAX^{TM}, and DFTMAX^{TM} ultra user guide
39. IEEE Standards Association and others (2001) 1149.1–2001 - IEEE standard test access port and boundary scan architecture. IEEE
40. Jeong DS, Thomas R, Katiyar R, Scott J, Kohlstedt H, Petraru A, Hwang CS (2012) Emerging memories: resistive switching mechanisms and current status. Rep Prog Phys 75(7):076502
41. Miyamoto A, Homma N, Aoki T, Satoh A (2011) Systematic design of RSA processors based on high-radix montgomery multipliers. IEEE Trans Very Large Scale Integr (VLSI) Syst 19(7):1136–1146
42. Holcomb DE, Burleson WP, Fu K (2007) Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In: Proceedings of the conference on RFID security
43. Sunar B, Martin W, Stinson D (2007) A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans Comput 56(1):109–119

44. Kaliski B (2003) Twirl and RSA key size. http://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm
45. Dworkin M (2001) NIST special publication 800-38A: recommendation for block cipher modes of operation

Index

A
Active hardware metering, 163, 164, 178, 179, 183
Active hardware obfuscation, 51
Area utilization, 116
Asymmetric ciphers, 317, 321
Attack analyses and countermeasures, 126
Attack resiliency, 236
Automatic test pattern generation (ATPG), 39

B
Backward trust, 325
Benchmarks, 29
Benes network, 116, 117, 119, 120, 124, 131
BEOL, 243, 244, 246, 259, 260
Best-possible obfuscation, 222, 224, 229–231, 237
BFSM graph topology, 176
Biometric-based Key Generation, 123
BISA, 265–268, 271–279, 282, 285–287
Blackhole FSM, 175
Board level, 105
Boolean satisfiability, 78
Boosted Finite State Machine (BFSM), 164, 167, 168, 183
Boundary scan, 48
Built-in-self-authentication, 265, 266, 271
Built in self test (BIST), 49

C
Chapter organization, 106
Chip editor, 20
Chip level, 104
Choice of optimal set of nodes, 199
Circuit camouflaging, 89–94
Circuit partition attack, 94–97, 101
Clone, 137
Comparison b/w 3D/2.5D Split Fabrication and Logic Locking, 297
Computing k-Security, 251
Configurable CMOS, 90, 91, 98
Counterfeit, 137
Counterfeit integrated circuit, 189
Countermeasures, 179
Countermeasures and attack coverage, 128
Cryptographic algorithms, 135
Cryptographic primitives, 317

D
3D/2.5D IC authentication, 310
3D/2.5D IC-Based Obfuscation, 292
Design flow against IP piracy, 301
Design-for-test (DfT), 137, 139, 143, 145, 151
Design granularities, 301
Design modification, 109
Determination of effectiveness, 212
3Dictest/2.5D IC testing, 309
Differential scan-based attack, 143
Digital signature, 323
2.5D Integration, 292–295, 298, 309
3D Integration, 243–245, 257, 291, 294, 296, 311, 312

E
Effect of environmental variations, 56
Encryption, 4, 5, 12–15, 22, 23
Equivalence of circuits, 36
Error correcting code (ECC), 164, 165, 175, 180

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9

F
Fabrication cost, 300
Fault coverage, 38
Fault modeling, 37
FEOL, 243, 244, 246, 259
Fingerprinting, 89
Finite state machine (FSM), 162, 163, 166–171, 173, 175, 178–180, 183
Finite state machines functional equivalence, 224
Flow, 164
Forward trust, 324, 325, 327, 329, 330, 344
Functional testing, 37

G
Gate level, 15
Goals of the obfuscation technique, 193
Greatest Common Divisor (GCD), 227

H
Hardware metering methodology, 167
Hardware obfuscation, 6, 14, 21, 24, 28, 29, 33–35, 37, 50–52, 59, 62, 64, 65, 71
Hardware obfuscation benchmarks, 29
Hardware trojan, 190–192, 205, 213, 217, 246–249, 260, 263, 265, 266, 268, 269, 271, 272, 277, 294, 299, 301
Hill climbing attack, 84

I
IC overproduction, 317, 320, 329–331, 334, 335, 340, 341, 344
Instruction set Obfuscation, 19
Integrated Circuit, 7
Intellectual property (IP), 3, 4, 6, 13
Intrinsic and nonintrinsic PUFs, 58
IP digest, 328, 342–344
IP infringement, 190, 217
IP overuse, 317, 320, 328, 329, 331, 334, 337, 341
IP piracy, 4, 12, 16, 83, 86, 189, 190, 194, 263, 265, 267, 276, 277, 285, 287, 293, 302, 317, 327, 329, 341, 342, 344
ISCAS-89 benchmark circuits, 203
Isolated gates, 92
Iterative automatic synthesis, 178

J
Joint Test Action Group (JTAG), 145, 149, 151, 152

K
Key based, 22
Key based classification, 21
Key-guessing attacks, 74, 77
Key-Locked OBfuscation (KLOB), 222, 230, 231, 237
Key management, 121
Key propagation attacks, 77
Key transfer, 340
K-security, 245, 246, 249, 251, 252, 258, 270, 279, 283, 285

L
Layout level, 17
Layout obfuscation, 299
Locking SIB, 152
Logic encryption, 81
Logic locking, 71

M
Malicious CAD Tool, 215
Man-in-the-middle attack, 127, 128, 130
Mealy machine, 165, 166
Message authentication codes (MACs), 323
Metrics, 24, 25, 28
Metrics of Hardware Obfuscation, 52
Min-Cut algorithm, 304
Mitigating the circuit, 97
Modification by input logic-cone, 194
Modification Kernel Function, 195
Moore machine, 165
Multiple-key effect, 119
Multiplexer-based circuit, 98
Multiplexer-based obfuscation, 98, 99

N
Netlist encryption, 327, 328
Netlist obfuscation, 327, 328
Network configuration, 117
Nonvolatile Memory (NVM), 52–55, 58, 63

O
Obfuscation considerations, 108
Obfuscation for Intellectual, 4
Obfuscation performance evaluation, 124
OBISA, 271, 277–285, 287
One-time programmable memory, 53
One-way random functions, 80
Output-guessing attacks, 74
Overproduction, 136, 146

P
P1735, 328, 341
Partitioning, 249
PCB obfuscation, 20, 108
Permutation based Obfuscation, 107
Permutation network, 106–108, 112–114, 116, 121, 123, 127, 131
Physical Unclonable Function (PUF), 34, 49, 51, 54–61, 63–65, 162–165, 168, 170, 175, 179, 180, 183
Point function, 163, 172, 183
Post-test activation, 84
Potential attacks, 126
Probing attack, 105, 126–128, 130
Protocol, 73
Provable obfuscation of the locks, 173
Proximity attack, 246
PUF operation, 55

R
Read only memory (ROM), 53
Redesign attack, 267, 275–277, 283
Register Transfer Level (RTL), 189, 217
Reinstalling attack, 128–130
Relevance of k-Security, 249
Removal attack, 177
Replay attack, 180
Reprogrammable memory, 53
Resizing attack, 273, 275
Resynthesis, 222, 226, 227, 229–234, 236, 237
Retiming, 222, 226, 227, 229–234, 236, 237
Reverse engineering, 4, 11, 12, 20, 72, 74, 83, 86, 89, 90, 97, 105, 107–109, 125, 127, 128, 130, 131, 179
RTL level, 14

S
Satisfiability (SAT) Problem, 35
Scan based, 44
Scan-based attack, 139
Scan-based observability attack, 142
Scan chain, 137, 138, 140, 142–144, 146, 148–150, 152, 154
Scan compression, 139
Secure layout, 246, 252, 255, 257, 259
Secure partitioning, 246, 252–254
Secure placement algorithm, 306
Secure Split Test (SST), 85, 146, 152
Security properties, 74
Segment Insertion Bit (SIB), 146, 152
Software obfuscation, 6, 15, 23, 24, 28, 29
Software obfuscation metrics, 28
Split Fabrication, 295–301, 311
Split manufacturing, 246, 249, 252, 256–260, 263–265, 268, 270, 271, 276, 277, 279, 283, 287
State space obfuscation, 191, 205, 217
State transition graph (STG), 191, 193, 207
Storage mechanisms, 54
Structural testing, 37
Structural transformation, 222–224, 229, 230, 236, 237
Stuttering, 222, 226, 228, 229, 231–234, 236, 237
Sub-graph isomorphism, 248, 251
Symmetric ciphers, 317, 318, 320–322
  block ciphers, 320
  stream ciphers, 319
System-level obfuscation, 196
System-on-Chip (SoC), 189

T
Tampering, 12
Taxonomy, 14
Test compression, 334
Testing and security, 44
Threat model, 83, 246
Thwarting output guessing attacks, 75
Tier protection, 300
Trade-off b/w BEOL security and computational, 279
Trade-Off b/w Cutsize and HD, 305
Trojan insertion effect, 208
Trojan potency, 209
True Random Number Generator (TRNG), 34, 62, 63

U
Untargeted Trojan, 268, 270
Untrusted test facility, 72, 85

V
VLSI verification and testing, 34
Volatile and nonvolatile memories, 52
Volatile memory, 52

W
Watermarking, 89
Wire lifting, 268, 279, 281, 283–287