Mohammad Tehranipoor · Cliff Wang
Editors

Introduction to Hardware Security and Trust

Springer

Introduction to Hardware Security and Trust

Mohammad Tehranipoor • Cliff Wang
Editors

Introduction to Hardware Security and Trust

Springer

Editors
Mohammad Tehranipoor
ECE Department
University of Connecticut
371 Fairfield Way, Unit 2157
Storrs, CT 06269
USA
tehrani@engr.uconn.edu

Cliff Wang
Computing and Information Science Division
US Army Research Office
PO Box 12211
Research Triangle Park
NC 27709-2211
USA
cliff.wang@us.army.mil

ISBN 978-1-4419-8079-3 e-ISBN 978-1-4419-8080-9
DOI 10.1007/978-1-4419-8080-9
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2011935539

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Contents

Contributors

Swarup Bhunia Case Western Reserve University, Cleveland, Ohio, USA

Zhimin Chen ECE Department, Virginia Tech, Blacksburg, VA 24061, USA

Jordan Cote Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Srinivas Devadas Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, USA

Jia Di Computer Science and Computer Engineering Department, University of Arkansas, Fayetteville, Arkansas, USA

Yunsi Fei Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Yin Hu Worcester Polytechnic Institute, Worcester, MA, USA

Yier Jin Department of Electrical Engineering, Yale University, New Haven, CT 06520, USA

Ramesh Karri Polytechnic Institute of New York University, Brooklyn, NY, USA

Farinaz Koushanfar Electrical and Computer Engineering Department, Rice University, Houston, Texas 77215-1892, USA

Jeremy Lee DFT Engineer, Texas Instruments, Dallas, TX, USA

Eric Love Department of Electrical Engineering, Yale University, New Haven, CT 06520, USA

Chujiao Ma Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Junxia Ma University of Connecticut, Storrs, CT, USA

Ken Mai Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Mehrdad Majzoobi Electrical and Computer Engineering Department, Rice University, 6100 Main, MS380, Houston, TX 77005, USA

Yiorgos Makris Department of Electrical Engineering, Yale University, New Haven, CT 06520, USA

Juan Carlos Martinez Santos Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Currently on leave from Universidad Tecnologica de Bolivar, Cartagena, Colombia

Seetharam Narasimhan Case Western Reserve University, Cleveland, Ohio, USA

Miodrag Potkonjak Computer Science Department, University of California Los Angeles, Los Angeles, CA 90095-1596, USA

Gang Qu Electrical and Computer Engineering Department, Institution for Systems Research, University of Maryland, College Park, MD 20742, USA

Jeyavijayan Rajendran Polytechnic Institute of New York University, Brooklyn, NY, USA

Kurt Rosenfeld Google Inc., New York, USA

Ulrich Rührmair Computer Science, Technische Universitat Munchen, Munich, Germany

Patrick Schaumont ECE Department, Virginia Tech, Blacksburg, VA 24061, USA

Zhijie Shi Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Sergei Skorobogatov University of Cambridge, Computer Laboratory, JJ Thomson Avenue, Cambridge CB3 0FD, UK

Berk Sunar Worcester Polytechnic Institute, Worcester, MA, USA

Mohammad Tehranipoor UCONN Electrical and Computer Engineering, University of Connecticut, 371 Fairfield Way, Unit 2157 Storrs, CT 06269-2157, USA

Dale R. Thompson Computer Science and Computer Engineering Department, University of Arkansas, Fayetteville, Arkansas, USA

Nicholas Tuzzio UCONN Electrical and Computer Engineering, University of Connecticut, 371 Fairfield Way, Unit 2157, Storrs, CT 06269-2157, USA

Bing Wang Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA

Lin Yuan Synopsys Inc., Mountain View, CA 94043, USA

Chapter 1
Background on VLSI Testing

Junxia Ma and Mohammad Tehranipoor

1.1 Introduction

As technology feature size of devices and interconnects shrink at the rate predicted by Moore's law, gate density and design complexity on single integrated chip (IC) keep increasing in recent decades. The close to nanoscale fabrication process introduces more manufacturing errors. New failure mechanisms that are not covered by current fault models are observed in designs fabricated in new technologies and new materials. At the same time, the power and signal integrity issues that come with scaled supply voltages and higher operating frequencies increase the number of faults that violate the predefined timing margin. VLSI testing has become more and more important and challenging to verify the correctness of design and manufacturing processes. The diagram shown in Fig. 1.1 illustrates the simplified IC production flow. In the design phase, the test modules are inserted in the netlist and synthesized in the layout. Designers set timing margin carefully to account for the difference between simulation and actual operation mode, such as uncertainties introduced by process variation, temperature variation, clock jitter, etc. However, due to imperfect design and fabrication process, there are variations and defects that make the chip violate this timing margin and cause functional failure in field. Logic bugs, manufacturing error, and defective packaging process could be the source of errors. It is thus mandatory to screen out the defective parts and prevent shipping them to customers to reduce custom returns.

Nowadays, the information collected from testing is used not only to screen defective products from reaching the customers but also to provide feedback to improve the design and manufacturing process (see Fig. 1.1). In this way, VLSI testing also improves manufacturing yield level and profitability.

J. Ma (✉) · M. Tehranipoor
UCONN Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA
e-mail: junxia@engr.uconn.edu; tehrani@engr.uconn.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_1, © Springer Science+Business Media, LLC 2012

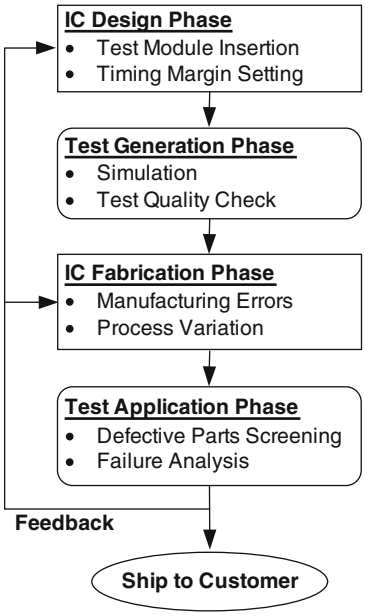


Fig. 1.1 Simplified IC design, fabrication, and test flow

1.2 Test Costs and Product Quality

Although high test quality is preferred, it always comes at the price of high test cost. Trade-offs are necessary to reach the required test quality with minimum cost [1]. In this Section, concepts of test costs, VLSI yield, and product quality are introduced. These concepts, when applied in electronic test, lead to economic arguments that justify design-for-testability (DFT) [2].

1.2.1 Test Costs

Test cost includes the cost of automatic test equipment (ATE) (initial and running cost), the cost of test development (CAD tools, test vector generation, test programming) [3], and the cost of DFT [4]. The scan design techniques can significantly reduce the cost of test generation and the Built-in self-test (BIST) method can lower the complexity and cost of ATE [5].

As shown in Fig. 1.2, the electronic industry tests chips in different levels. Wafer testing is performed during semiconductor device fabrication using automated test equipment (ATE). During this step, each device that is present on the wafer is tested for functional defects by applying special test patterns to it. The wafer is then cut into rectangular blocks, each of which is called a die. Each good die is then packaged, and all packaged devices are tested through final testing again

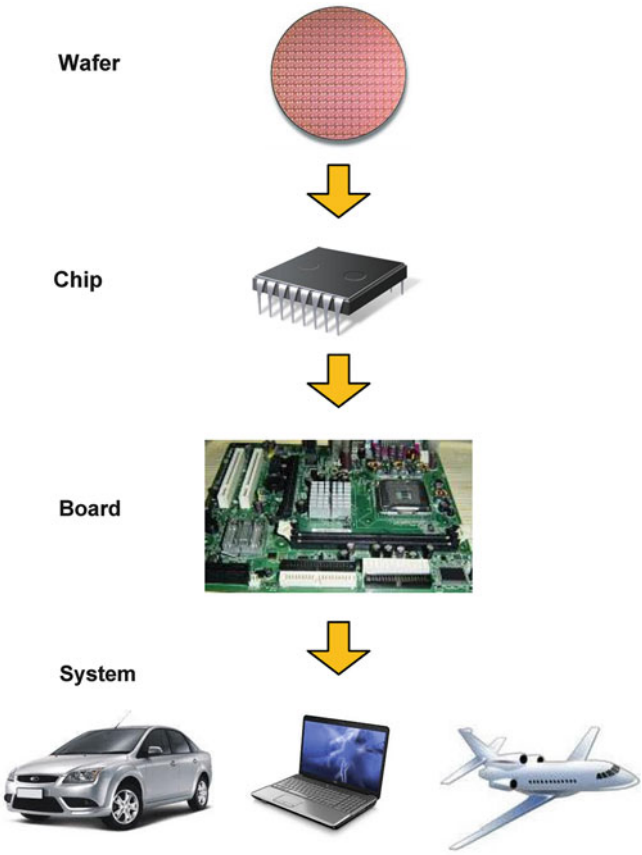


Fig. 1.2 Test levels: wafer, packaged chip, PCB, system in field

on the same or similar ATE used during wafer probing. After the chips are shipped to the customers, they will perform PCB testing and system testing again because the rule of ten holds according to experience [6]. It usually requires ten times more expensive cost than chip level to repair or replace defective ASIC at PCB level. After chips are assembled into systems, if a board fault is not caught in PCB testing, it needs ten times as much at the system level as at the board level to find the fault. Nowadays, as the systems are much more complex than year 1982 when the empirical rule was first stated in [6], the times of cost increase are much more than 10X. For airplanes, a chip fault uncaught in testing can cause thousands or millions times loss. For the same reason, VLSI testing is essential to reach "zero-defect" goal for mission critical applications.

1.2.2 Defect, Yield, and Defect Level

A manufacturing *defect* is a finite chip area with electrically malfunctioning circuitry caused by errors in the fabrication process. Defect on wafers could be caused by process variation, such as impurities in wafer material and chemicals, dust particles on masks or in the projection system, mask misalignment, incorrect temperature control, etc. Typical defects are broken (open) metal wires, missing contracts, bridging among metal lines, missing transistors, incorrect doping levels, void vias, resistive open vias, and many other phenomena that can cause the circuit to fail. A chip with no manufacturing defect is called a good chip. Fraction (or percentage) of good chips produced in a manufacturing process is called the *yield*. Yield is denoted by symbol Y. For chip area A, with fault density f, where f is the average number of faults per unit area, fault clustering parameter β, and fault coverage T, the yield equation [5] is expressed as later.

$$Y(T) = \left(1 + \frac{TAf}{\beta}\right)^{-\beta}. \tag{1.1}$$

Assuming that tests with 100% fault coverage ($T = 1.0$) remove all faulty chips, the yield $Y(1)$ is:

$$Y = Y(1) = \left(1 + \frac{Af}{\beta}\right)^{-\beta}. \tag{1.2}$$

Good test process can reject most of the defective parts. However, even it can reject all the faulty chips, it cannot improve the process yield by itself unless the diagnostic information collected during test is feedback to the design and fabrication process. There are two ways of improving the process yield [5]:

1. *Diagnosis and repair*. Defective parts are diagnosed and then repaired. Although in this way the yield is improved, it increases the cost of manufacturing.
2. *Process diagnosis and correction*. By identifying systematic defects and their root cause, the yield can be improved once the cause is eliminated during manufacturing process. Process diagnosis is preferred method of yield improvement.

A metric used to measure the effectiveness of tests and the manufactured product quality is defect level (DL), which is defined as the ratio of faulty chips among the chips that pass tests. It is measured as parts per million (ppm). For commercial VLSI chips a DL greater than 500 ppm is considered unacceptable.

There are two methods for the determination of defect level. One is from the field return data. Chips failing in the field are returned to the manufacturer. The number of returned chips normalized to one million chips shipped is the defect level. The other way is using test data. Fault coverage of tests and chip fallout rate are analyzed. A modified yield model is fitted to the fallout data to estimate the defect level, where chip fallout is the fraction of chips failing up to a vector in the test set, which is $1 - Y(T)$.

When chip tests have a fault coverage T, the defect level is given by the following equation [5]:

$$\mathrm{DL}(T) = \frac{Y(T) - Y(1)}{Y(T)} = 1 - \frac{Y(1)}{Y(T)} = 1 - \left(\frac{\beta + TAf}{\beta + Af}\right)^{\beta}, \tag{1.3}$$

where Af is the average number of faults on the chip of area A and β is the fault clustering parameter. Af and β are determined by test data analysis. This equation gives DL as a fraction that should be multiplied by 10^6 to obtain *ppm*. For zero fault coverage, $\mathrm{DL}(0) = 1 - Y(1)$, where $Y(1)$ is the process yield. For a 100% fault coverage, $\mathrm{DL}(1) = 0$.

An alternative equation relating detects level, Yield, and fault-coverage, in case of unclustered random defects is [22]:

$$\mathrm{DL}(T) = 1 - Y^{1-T}, \tag{1.4}$$

where T is the fault coverage of tests and Y is the ratio of the "devices tested good" to the "total number of devices tested or fabricated/manufactured."

1.3 Test Generation

1.3.1 Structural Test vs. Functional Test

In the past, functional patterns were used to verify if there are any errors at the output. A complete functional test will check each entry of the truth table. It is possible with small input numbers. However, as the exhaustive testing of all possible input combinations grows exponentially as the number of inputs increases, such a test will be too long and impossible for real circuits with several hundred inputs. Eldred derived tests that would observe the state of internal signals at primary outputs of a large digital system in 1959 [7]. Such tests are called structural tests because they depend on the specific structural (gate type, interconnect, netlist) of the circuits [5]. Structural test has become more attractive over the last decade because of the controllable testing time.

Structural testing is considered as white-box testing because the knowledge of the internal logic of a system is used for test generation. It makes no direct attempt to determine if the overall functionality of the circuit is correct. Instead, it checks whether the circuit has been assembled correctly from low-level circuit elements as specified in the netlist. The stipulation is that if the circuit elements are confirmed to be assembled correctly then the circuit should be functioning correctly. Functional test attempts to validate that the circuit under test functions according to its functional specification. It can be viewed as black-box test. Functional automatic test-pattern generation (ATPG), (refer to 1.3.4) programs generates complete set

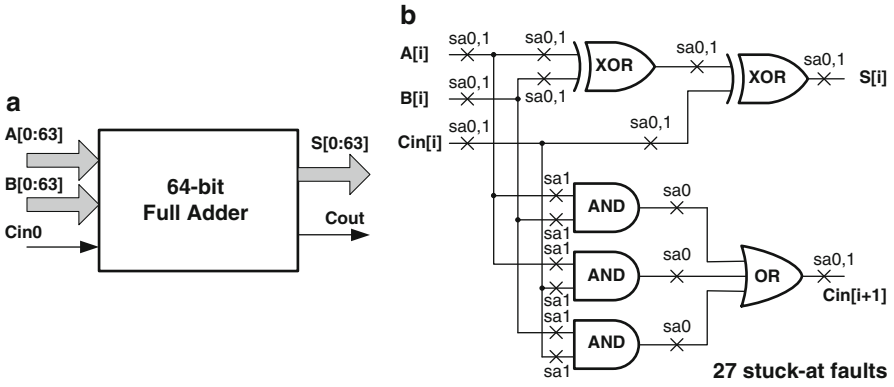


Fig. 1.3 A 64-bit ripple-carry adder: (**a**) functional test; (**b**) structural stuck-at fault test

of tests for circuit input-output combinations to completely exercise the circuit function. Figure 1.3 shows a 64-bit ripple-carry adder and the logic circuit design for one bit slice of the adder. As can be seen from Fig. 1.3a, the adder has 129 inputs and 65 outputs. Therefore, to completely test it using functional patterns, we need $2^{129} = 6.80 \times 10^{38}$ input patterns, and to verify $2^{65} = 3.69 \times 10^{19}$ output response. Using ATE whose operating frequency is 1 GHz, it would take 2.15×10^{22} years to apply all of these patterns to this adder circuit assuming that the circuit can operate at 1 GHz too. Today considering most of the circuit size much larger than this simple adder, exhaustive functional test is impractical to test them in most of the case. Nowadays small numbers of functional test patterns are found to be useful to screen sever defects. For some applications, such as microprocessors, functional testing is still a very important part. It is quite fast to apply structure test to this 64-bit adder circuit. There are totally 27 stuck-at faults for one bit adder after we discard the equivalent faults in Fig. 1.3b. For a 64-bit adder, there are $27 \times 64 = 1{,}728$ faults. It needs at most 1,728 test patterns. Using 1 GHz ATE it needs only 0.000001728 s to apply these patterns. Since this pattern set covers all possible stuck-at faults in this adder, it achieves same fault coverage as the huge functional test pattern set. Thus we can see the advantage and importance of structural testing.

1.3.2 Fault Models

There are three terms that are usually used to describe the incorrectness of an electronic system.

- *Defect:* A defect in an electronic system is the unintended difference between the implemented hardware and its intended design. Typical defects in VLSI chips are: process defects, material defects, aging defects, and package defects.

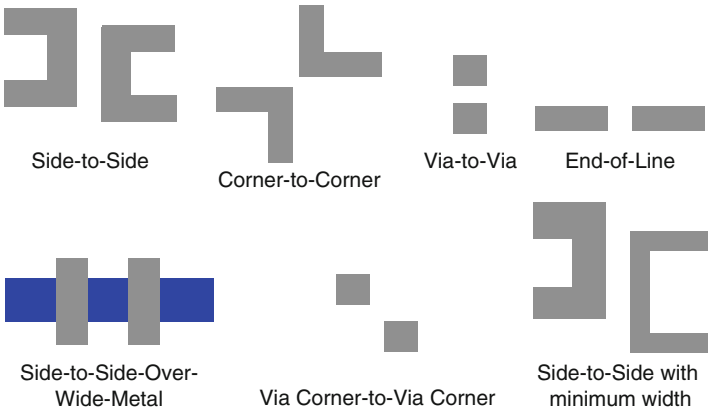


Fig. 1.4 Five Type of bridging faults [26]

- *Error:* A wrong output signal produced by a defective system is called an error. An error is an effect whose cause is some "defect."
- *Fault:* A representation of a "defect" at the abstracted function level is called a fault.

Fault model is a mathematical description of how a defect alters design behavior. A fault is said to be detected by a test pattern if, when applying the pattern to the design, any logic value observed at one or more of the circuit's primary outputs differs between the original design and the design with the fault. There are a lot of fault models developed to describe different kinds of physical defects. The most common fault modes for modern VLSI test includes: stuck-at fault, bridging fault, delay faults (transition delay fault and path delay fault), stuck-open and stuck-short faults, etc.

- *Stuck-at faults:* A signal, which is an input or an output of a logic gate or flip-flop is stuck at a 0 or 1 value, independent of the inputs to the circuit. Single stuck-at fault is widely used, i.e., two faults per line, stuck-at-1 (sa1), and stuck-at-0 (sa0). An example of stuck-at fault in the circuit is shown in Fig. 1.3.
- *Bridging faults:* Two signals are connected together when they should not be. Depending on the logic circuitry employed, this may result in a wired-OR or wired-AND logic function. As there are $O(n^2)$ potential bridging faults, they are normally restricted to signals that are physically adjacent in the design. Sketches of seven typical types of bridging faults are shown in Fig. 1.4. These types are derived from DRC and DFM rules and known bridge causing layout features [8]:
  - Type 1: Side-to-Side
  - Type 2: Corner-to-Corner
  - Type 3: Via-to-Via
  - Type 4: End-of-Line

  - Type 5: Side-to-Side Over Wide Metal
  - Type 6: Via Corner-to-Via Corner
  - Type 7: Side-to-Side with Minimum Width

- *Delay faults:* These faults make the signal propagate slower than normal, and cause the combinational delay of a circuit to exceed clock period. Specific delay faults are: transition delay faults (TDF), path delay faults (PDF), gate delay faults, line delay faults, and segment delay faults. Among them slow-to-rise and slow-to-fall PDF and TDF are the most commonly used ones. Path delay fault model targets the cumulative delay through the entire list of gates in a path, while the transition fault model targets each gate output in the design.
- *Stuck-open and Stuck-short faults:* AMOS transistor is considered as an ideal switch. Stuck-open and stuck-short faults model the switch being permanently in either the open or the shorted state. And they assume just one transistor to be stuck-open or stuck short. The effect of a stuck-open fault is a floating state at the output of the faulty logic gate. It can be detected in the similar way as detecting a stuck-at fault at the output fault on the gate's output pin. The effect of stuck-short fault is that it short-connects power line and ground line. Quiescent current (IDDQ) measurement can be used to detect such fault.

1.3.3 Testability: Controllability and Observability

Testability is represented by *controllability* and *observability* measures that approximately quantify how hard it is to set and observe internal signal of a circuit. Controllability is defined as the difficulty of setting a particular logic signal to a 0 or a 1. Observability is defined as the difficulty of observing the state of a logic signal. Testability analysis can be used for analysis of difficulty of testing internal circuit parts, and based on it to redesign or add special test hardware (test point) in the circuit to increase its testability. It can also be used as guidance for algorithms computing test patterns to avoid using hard-to-control lines. Test generation algorithms using heuristics usually apply some kind of testability measures to their heuristic operations, which greatly speed up the test generation process. Through testability analysis, estimation of fault coverage, number of untestable faults and test vector length is also possible.

Testability analysis involves circuit topological analysis without test vectors and search algorithm. It has linear complexity. Sandia controllability observability analysis program (SCOAP) is a systematic, efficient algorithm, proposed by Goldstein [9] and widely used to compute controllability and observability measures. It consists of six numerical measures for each signal (l) in the circuit. Three combinational measures are:

- CC0(l): combinational 0-controllability; it represents the difficulty of setting circuit line to logic 0.

- CC1(l): combinational 1-controllability; it represents the difficulty of setting circuit line to logic 1.
- CO(l): combinational observability; it describes the difficulty of observing a circuit line.

Similarly, there are three sequential measures, which are SC0(l) as sequential 0-controllability, SC1(l) as sequential 1-controllability, and SO(l) as sequential observability. Generally, the three combinational measures are related to the number of signals that may be manipulated to control or observe signal l. The three sequential measures are related to the number of time-frames (or clock cycles) needed to control or observe [5]. The controllability range is between 1 to infinity (∞) and observability range is from 0 to ∞. The higher the measure is, the more difficult it will be to control or observe that line.

According to Goldstein's method [9], the method to compute combinational and sequential measures is described as blow.

1. For all PIs I, set CC0(I)=CC1(I)=1 and SC0(I)=SC1(I)=0; For all other nodes N, set CC0(N)=CC1(N)=SC0(N)=SC1(N)=∞.
2. Starting from PIs to POs, use the CC0, CC1, SC0, and SC1 equations, to map logic gate and flip-flop input controllabilities into output controllabilities. Iterate until the controllability numbers stabilize in feedback loops.
3. For all POs U, set CO(U)=SO(U)=0; For all other nodes N, set CO(N)=SO(N) $=\infty$. Working from POs to PIs, use the CO and SO equations and the precomputed controllabilities to map output node observabilities of gates and flip-flops into input observabilities. For fanout stems Z with branches with branches Z1, . . . , ZN, SO(Z)=min(SO(Z1), . . . , SO(ZN)) and CO(Z)=min(CO(Z1), . . . , CO(ZN)).
4. If any node remains with CC0/SC0=∞ then that node is 0-uncontrollable. If any node remains with CC1/SC1=∞ then that node is 1-uncontrollable. If any node remains with CO=∞ or SO=∞ then that node is unobservable. These are sufficient but not necessary conditions.

For the computation of controllability for single logic gate, if a logic gate output is produced by setting only one input to a controlling value then

$$\text{output controllability} = \min\,(\text{input controllabilites}) + 1. \tag{1.5}$$

If a logic gate output can only be produced by setting all inputs to noncontrolling value, then

$$\text{output controllability} = \sum(\text{input controllabilites}) + 1. \tag{1.6}$$

If an output can be controlled by multiple input sets, such as XOR gate, then

$$\text{output conrollability} = \min\,(\text{controllabilities of input sets}) + 1. \tag{1.7}$$

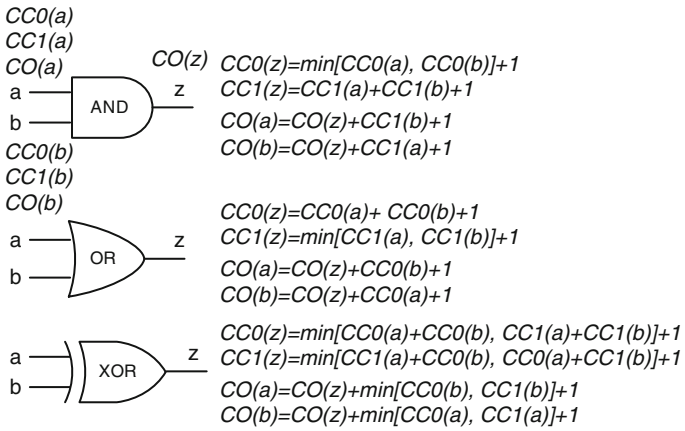


Fig. 1.5 SCOAP controllability and observability calculation

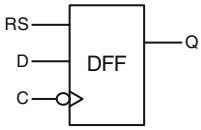


Fig. 1.6 Resettable, negative-edge-trigged D flip-flop

For a logic gate with an input signal that needs to be observed,

$$\text{input observaility} = \text{output observability} + \sum(\text{controllabilities of setting all other pins to non-controlling value}) + 1. \tag{1.8}$$

Figure 1.5 presents examples of SCOAP controllability and observability calculation using AND, OR, and XOR gates.

Figure 1.6 shows a resettable negative-edge triggered D flip-flop (DFF). The combinational controllablities CC1 or CC0 measures how many lines in the circuit must be set to make DFF output signal Q as 1 or 0, whereas sequential controllabilites SC1 or SC0 measures how many times flip-flops in the circuit must be clocked to set Q to 1 or 0. To control Q line to 1, one must to set input D to 1 and forcing a falling clock edge on C. And the reset signal line RS needs to keep as 0. Note that one needs to add 1 for the sequential measures when signals propagate from flip-flop inputs to output. Thus, CC1Q and SC1(Q) are calculated in the following way:

$$\begin{aligned} \text{CC1(Q)} &= \text{CC1(D)} + \text{CC1(C)} + \text{CC0(C)} + \text{CC0(RS)} \\ \text{SC1(Q)} &= \text{SC1(D)} + \text{SC1(C)} + \text{SC0(C)} + \text{SC0(RS)} + 1 \end{aligned} \tag{1.8}$$

There are two ways to set Q to 0, either through setting reset signal RS while holding clock C at 0 or clock a 0 through input D. Thus, CC0Q and SC0(Q) are calculated using the following equations:

$$\begin{aligned} \text{CC0(Q)} &= \min[\text{CC1(RS)}+\text{CC0(C)}, \quad \text{CC0(D)}+\text{CC1(C)}+\text{CC0(C)}+\text{CC0(RS)}] \\ \text{SC0(Q)} &= \min[\text{SC1(RS)}+\text{SC0(C)}, \quad \text{SC0(D)}+\text{SC1(C)}+\text{SC0(C)}+\text{SC0(RS)}]+1 \end{aligned} \tag{1.9}$$

The input D can be observed at Q by holding RS low and generating a falling edge on the clock line C:

$$\begin{aligned} \text{CO(D)} &= \text{CO(Q)} + \text{CC1(C)} + \text{CC0(C)} + \text{CC0(RS)} \\ \text{SO(D)} &= \text{SO(Q)} + \text{SC1(C)} + \text{SC0(C)} + \text{SC0(RS)} + 1 \end{aligned} \tag{1.10}$$

RS can be observed by setting Q to a 1 and using RS:

$$\begin{aligned} \text{CO(RS)} &= \text{CO(Q)} + \text{CC1(Q)} + \text{CC0(C)} + \text{CC1(RS)} \\ \text{SO(RS)} &= \text{SO(Q)} + \text{SC1(Q)} + \text{SC0(C)} + \text{SC1(RS)} + 1 + 1 \end{aligned} \tag{1.11}$$

There are two ways to indirectly observe the clock line C: (1) set Q to 1 and clock in a 0 from D, or (2) reset the flip-flop and clock in a 1 from D. Thus,

$$\begin{aligned} \text{CO(C)} &= \min\,[\text{CO(Q)} + \text{CC0(RS)} + \text{CC1(C)} + \text{CC0(C)} + \text{CC0(D)} + \text{CC1(Q)}, \\ &\qquad \text{CO(Q)} + \text{CC1(RS)} + \text{CC1(C)} + \text{CC0(C)} + \text{CC1(D)}] \\ \text{SO(C)} &= \min\,[\text{SO(Q)} + \text{SC0(RS)} + \text{SC1(C)} + \text{SC0(C)} + \text{SC0(D)} + \text{SC1(Q)}, \\ &\qquad \text{SO(Q)} + \text{SC1(RS)} + \text{SC1(C)} + \text{SC0(C)} + \text{SC1(D)}] + 1 \end{aligned} \tag{1.12}$$

1.3.4 Automatic Test-Pattern Generation

ATPG is an electronic design automation (EDA) method/technology used to find an input (or test) sequence that, when applied to a digital circuit, enables testers to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects. These algorithms usually operate with a fault generator program, which creates the minimal collapsed fault list so that the designer needs not to be concerned with fault generation [5]. Controllability and observability measures are used in all major ATPG algorithms. The effectiveness of ATPG is measured by the amount of modeled defects, or fault models, that are detected and the number of generated patterns. These metrics generally indicate test quality (higher with

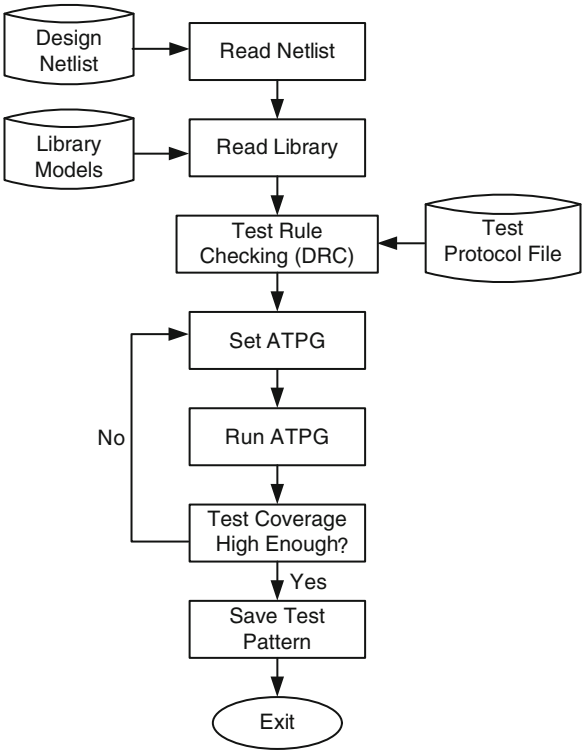


Fig. 1.7 Basic ATPG flow in EDA tool

more fault detections) and test application time (higher with more patterns). ATPG efficiency is another important consideration. It is influenced by the fault model under consideration, the type of circuit under test (full scan, synchronous sequential, or asynchronous sequential), the level of abstraction used to represent the circuit under test (gate, register-transistor, switch), and the required test quality [10].

Today, because of the circuits' size and time-to-market requirement, all the ATPG Algorithms are performed by EDA tools. Figure 1.7 illustrates the basic ATPG running flow in EDA tool. The tool first reads in the design netlist and library models, then after building the model, it checks test design rules that are specified in the test protocol file. If any violations happen in this step, the tool reports the violations rule as warning or errors depending on the severity. Using the ATPG constraints specified by the users, the tool performs ATPG analysis and generates test pattern set. If the test coverage meets the users' needs, test patterns will be saved in files with specified format. Otherwise the users can modify the ATPG settings and constrains and rerun ATPG.

To be noted, there are two metrics about coverage: *test coverage* and *fault coverage*. Test coverage is the percentage of detected faults among those detectable and gives the most meaningful measure of test pattern quality. Fault Coverage is the

percent detected of all faults. It gives no credit for undetectable faults. Usually, test coverage is used as a measure for the effectiveness of the test patterns generated by the ATPG tool.

1.4 Structured DFT Techniques Overview

1.4.1 Design for Testability

DFT techniques are widely used in nowadays integrated circuits. "DFT" is a general term applied to design methods that lead to more thorough and less costly testing. In general, DFT is achieved by employing extra hardware circuits for test purpose. The extra test circuits provide improved access to internal circuit elements. Through these test circuits, the local internal state can be controlled and/or observed more easily. It adds more controllability and observability to the internal circuits. DFT plays an important role in the development of test programs and as an interface for test application and diagnostics. With appropriate DFT rules implemented, many benefits ensue from designing a system so that failures are easy to detect and locate. DFT can bring the many benefits. Generally, integrating DFT in the development cycle can help:

- Improve fault coverage
- Reduce test generation time
- Potentially shorten test length and reduce test memory
- Reduce test application time
- Support hierarchical test
- Realize concurrent engineering
- Reduce life-cycle costs

These benefits come at the price of extra cost from pin over-ahead, more area and thus low yield, performance degradation and longer design time. However, as it reduces the overall costs of the chip, DFT is a cost-effective methodology and widely used in IC industry.

Three types of components need test in electronic systems: digital logic, memory blocks, and analog or mix-signal circuits. There are specific DFT methods for each type of components. DFT methods for digital circuits include: Ad-hoc methods and structured methods. Ad-hoc DFT method relies on good design experience and experienced designers to find the problem area. Circuit modification or test points insertion is required to improve the testability for these areas. The Ad-hoc DFT techniques are usually too labor-intensive and do not guarantee good results from ATPG. For these reasons, for large circuits it is discouraged to use Ad-hoc DFT. The common structured methods include: scan, partial scan, BIST, and boundary scan. Among them, BIST is also used for memory blocks testing. In the following, we will give a short introduction for each of these structured DFT techniques.

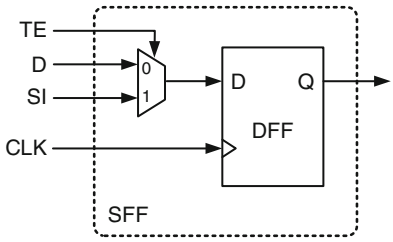


Fig. 1.8 A scan flip-flop (SFF) constructed by D-type flip-flop and multiplexer

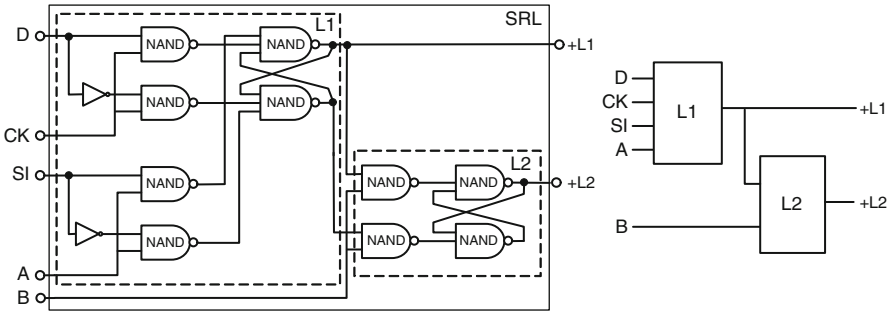


Fig. 1.9 Level-sensitive scan design (LSSD) cell

1.4.2 Scan Design: Scan Cell, Scan Chain, and Scan Test Compression

Scan is the most popular DFT technique. Scan design offers simple read/write access to all or subset of storage elements in a design. It also enables direct control of storage elements to an arbitrary value (0 or 1) and direct observation of the state of storage elements and hence the internal state of the circuit. In short, it gives the circuit enhanced controllability and observability.

1.4.2.1 Scan Cell

Scan design is realized by replacing flip-flops by scan flip-flops (SFFs) and connecting them to form one or more shift registers in the test mode. Figure 1.8 illustrates a SFF designed based on D-type flip-flop (DFF). A multiplexer is added in front of the DFF to construct a scan D-type flip-flop (SDFF). The test enable (TE) signal controls the working mode of the SDFF. When it is high, it selects the test mode and the scan-in (SI) bits are taken as the input of the DFF. When the TE signal is low, the SDFF works as in functional mode. It acts as a normal DFF and takes value D from the combination circuits as the input to the DFF.

SDFF is generally used for clock edge-trigged scan design, while level-sensitive scan design (LSSD) cell is used for level-sensitive, latch-based designs. Figure 1.9 a shows a polarity-hold shift register latch design [11] that can be used as an LSSD

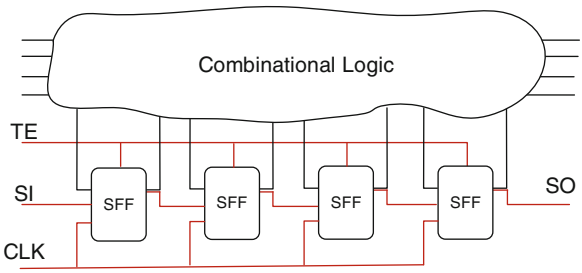


Fig. 1.10 Scan chain in a sequential circuit design

scan cell. The scan cell consists of two latches, a master two-port D-latch L1 and a slave D-latch L2. D is the normal data line and CK is the normal clock line. Line +L1 is the normal output. Lines SI, A, B, and L2 form the shift portion of the latch. SI is the shift data in and +L2 is the shift data out. A and B are the two phase, non overlapping shift clocks. The major advantage of using an LSSD scan cell is that it can be used for latch-based design. In addition, it avoids performance degradation introduced by the MUX in shift-register modification. As LSSD scan cells are level-sensitive, designs using LSSD are guaranteed to be race-free. However, this technique requires routing for the additional clocks, which increases routing complexity. It can only be used for slow test application. Normal-speed testing is impossible.

1.4.2.2 Scan Chain

Figure 1.10 shows a scan chain in a sequential circuit design. The SFFs are stitched together and consist of a scan chain. When test enable signal TE is high, the circuit works in test (shift) mode. The inputs from scan-in (SI) are shifted through the scan chain; the scan chain states can be shifted out through scan chain and observed at the scan-out (SO) pin. The test program compares the SO values with expected values to verify the chips performance.

Multiple scan chains are usually used to reduce the time to load and observe. SFFs can be distributed among any number of scan chains, each having a separate scan-in (SI) and scan-out pin. The integrity of scan chains must be tested prior to application of scan test sequences. A shift sequence 00110011... of length $n + 4$ in scan mode (TC=0) produces 00, 01, 11, and 10 transitions in all flip-flops and observes the result at scan chain output SO, where n is the number of SFFs in the longest scan chain.

1.4.2.3 Scan Test Compression

As chips becomes bigger and more complex, the growing test-data volume causes a significant increase in test cost because of much longer test time and large tester-memory requirements. For a scan-based test, the test data volume is proportional to

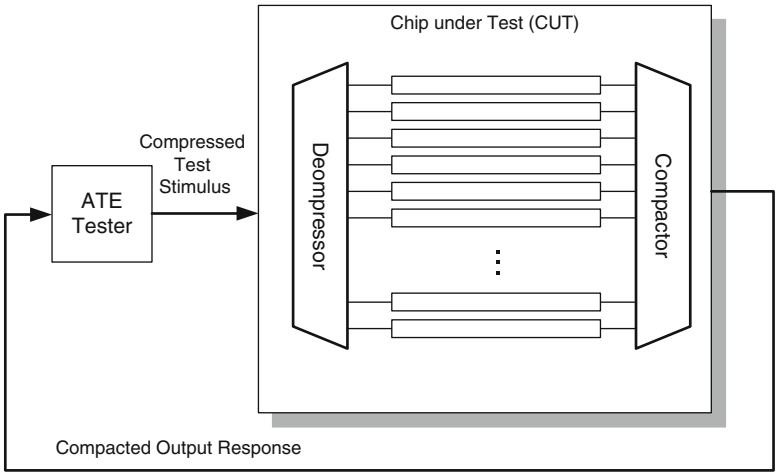


Fig. 1.11 Scan test compress

the number of test cycles. While number of test cycles and test time are related to the number of scan cells, scan chains, and scan patterns as shown in the equations later. Although theoretically increasing shift frequency reduces test time, in practice shift frequency cannot be increased too much due to power dissipation and design constraints.

$$\text{Test Cycles} \approx \frac{\text{Scan Cells} \times \text{Scan Patterns}}{\text{Scan Chains}}$$
$$\text{Test Time} \approx \frac{\text{Scan Cells} \times \text{Scan Patterns}}{\text{Scan Chains} \times \text{Shift Frequency}}. \tag{1.9}$$

As manufacturing test cost depends very strongly on the volume of test data and test time, one of the key requirements is to reduce them dramatically. Test Compression was developed to help address this problem.

There are large portion of don't-care bits left when an ATPG tool generates a test for a set of faults. Test compression takes advantage of the small number of significant values to reduce test data and test time. Generally, the idea is to modify the design to increase the number of internal scan chains and shorten maximum scan chain lengths. As illustrated in Fig. 1.11, these chains are then driven by an on-chip decompressor, usually designed to allow continuous flow decompression where the internal scan chains are loaded as the data is delivered to the decompressor [12]. Many different decompression methods can be used [13]. One common choice is a linear finite state machine, where the compressed stimuli are computed by solving linear equations. For industrial circuits with test vectors care-bits ratio ranging from 3% to 0.2%, the test compression based on this method often results in compression ratios of 30–500 times [12].

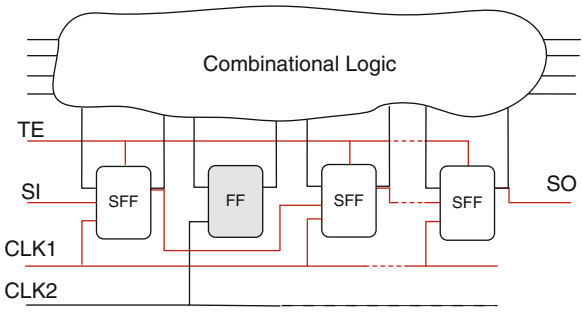


Fig. 1.12 Partial Scan design

A compactor is required to compact all the internal scan chain outputs to the output pins. As can be seen from Fig. 1.11, it is inserted between the internal scan chain outputs and the tester scan channel outputs. The compactor must be synchronized with the date decompressor, and must be capable of handling unknown (X) states, which may come from false and multicycle paths, or other unexpected reasons.

1.4.3 Partial-Scan Design

While full scan designs replace all flip-flops with SFFs, partial scan design only select a subset of flip-flops to be scanned, which provides a wide spectrum of design solutions that trade off testability for the overheads (i.e., area, power overheads) incurred by scan design.

Figure 1.12 illustrates the concept of partial scan. Being different from the full-scan design shown in Fig. 1.10, not all the flip-flops are SFFs. Two separate clocks are used for scan operation and functional operation.

Selection of flip-flops that can provide the best improvements in testability is a critical part of the partial scan design process. Most SFF selection methods are based on one or several of the following techniques: testability analysis, structural analysis, and test generation [14]. Testability-based methods analyze the testability of the circuit using SCOAP measures and improve the testability by partial scan. However, for circuits with complex structure, the fault coverage achieved may not be adequate using these techniques. Partial scan selection by structural analysis targets to remove all feedback loops from a circuit, and thus simplify the circuit structure for the test generation algorithm. The problem for such techniques is, for many circuits, it may be infeasible or unnecessary to break all feedback loops to achieve desirable fault coverage. Test generation-based methods exploit information from the test generator to drive the scan selection process. The main advantage of using test generation-based techniques is that it is possible to target specific fault detection objectives rather than simplify the circuit or improve testability of specific regions in the circuit. However, it typically results in expensive computational and storage requirements [14].

As a separate clock is used for the scan operation, the states of the non-SFFs can be frozen during the scan operation and any state can be scanned into the scan register without affecting the states of the non-SFFs. In this way, test vectors can be efficiently generated by a sequential circuit test generator. However, it poses problem in the need for multiple clock trees and tight constraint on clock skew when routing of clock signals.

1.4.4 Boundary Scan

The boundary-scan technique uses a shift-register stage to test interconnects and clusters of logic, memories, etc. The boundary-scan register consists of boundary-scan cells which are inserted adjacent to each component pin so that signals at component boundaries can be controlled and observed using scan testing principles. The boundary scan controller has also emerged as the standard mechanism on SoC designs for initiating and controlling the multiple internal memory BIST controllers. Boundary scan is now a well-known and documented IEEE standard, and some test software vendors offer automated solutions. IEEE 1149.1, also known as JTAG or boundary scan, was introduced in 1990 [15]. This standard endeavors to solve test and diagnostic problems arising from loss of physical access caused by the increasing use of high pin count and BGA devices, multilayer PCBs, and densely packed circuit board assemblies. The standard outlines predefined protocols for testing and diagnosing manufacturing faults. It also provides a means for on-board programming of nonvolatile memory devices such as Flash, or in-system programming of devices like PLDs and CPLDs.

Figure 1.13 illustrates the essential boundary-scan architecture. The block of logic circuits to be tested is connected to multiple boundary-scan cells. The cells are created along with the IC circuitry when the chip is fabricated. Each cell can monitor or stimulate one point in the circuitry. The cells are then linked serially to form a long shift register whose serial input, designated Test Data Input, and serial-output ports Test Data Output become the basic I/O of a JTAG interface. The shift register is clocked through external clock signal (TCK). In addition to the serial-in, serial-out, and clock signals, a Test Mode Select (TMS) input is provided as well as an optional Test Reset pin (TRST). The TMS, TCK, and TRST signals are applied to a finite state machine called the test access port controller. Along with external binary instructions, it controls all of the possible boundary-scan functions. To stimulate the circuit, test bits are shifted in. This is called a test vector.

The primary advantage of boundary-scan technology is the ability to observe and control data independently of the application logic. It also reduces the number of overall test points required for device access, which can help lower board fabrication costs and increase package density. Simple tests using boundary scan on testers can find manufacturing defects, such as unconnected pins, a missing device, and even a failed or dead device. In addition, boundary scan provides better diagnostics. With boundary scan, the boundary-scan cells observe device

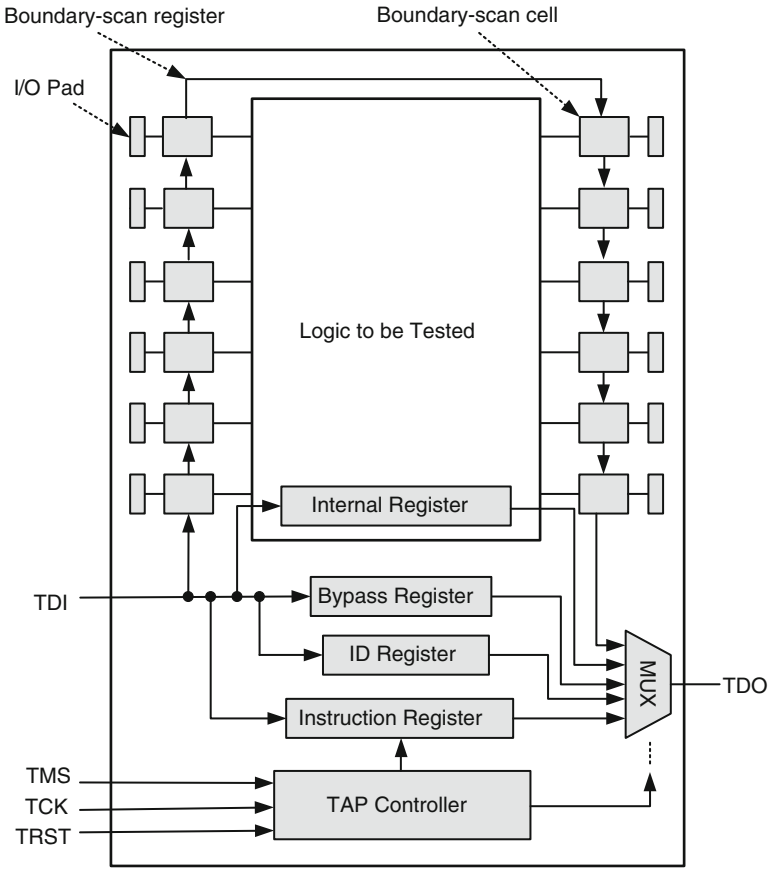


Fig. 1.13 Boundary scan architecture

responses by monitoring the input pins of the device. This enables easy isolation of various classes of test failures. Boundary scan can be used for functional testing and debugging at various levels, from IC tests to board-level tests. The technology is even useful for hardware/software integration testing and provides system level debug capability [16].

1.4.5 BIST Methods

Built-in Self Test, or BIST, is a DFT methodology of inserting additional hardware and software features into integrated circuits to allow them to perform self-testing, thereby reducing dependence on an external ATE and thus reducing testing cost. The concept of BIST is applicable to about any kind of circuit. BIST is also the solution

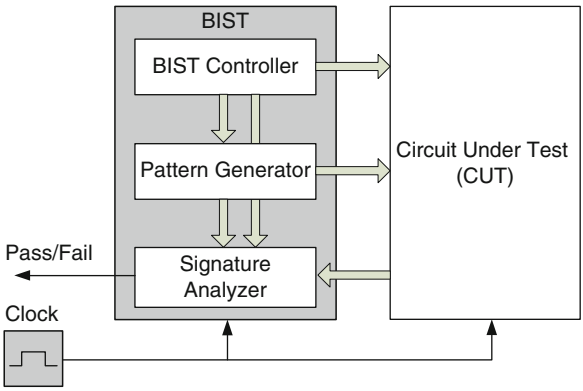


Fig. 1.14 Build-in self-test architecture

to the testing of circuits that have no direct connections to external pins, such as embedded memories used internally by the devices. Figure 1.14 shows BIST architecture. In BIST, a test pattern generator generates test patterns and a signature analyzer (SA) compares test responses. The entire process is controlled by BIST controller.

The most two common classification of BIST are the Logic BIST (LBIST) and the Memory BIST (MBIST). LBIST, which is designed for testing random logic, typically employs a pseudo-random pattern generator to generate input patterns that are applied to the device's internal scan chain, and a multiple input signature register (MISR) for obtaining the response of the device to these test input patterns. An incorrect MISR output indicates a defect in the device. MBIST is used specifically for testing memories. It typically consists of test circuits that apply a collection of write-read-write sequences for memories. Complex write-read sequences are called algorithms, such as MarchC, Walking 1/0, GalPat, and Butterfly. The cost and benefit models for MBIST and LBIST are presented in [17]. It analyzes the economics effects of BIST for logic and memory cores.

Advantages of implementing BIST include:

- Low test cost, as it reduces or eliminates the need for external electrical testing using an ATE
- Improved testability and fault coverage
- Support of concurrent testing
- Shorter test time if the BIST can be designed to test more structures in parallel
- At-speed testing

Disadvantages of implementing BIST include:

- Silicon area, pin counts and power overhead for the BIST circuits
- Performance degradation and timing issues
- Possible issues with the correctness of BIST results, as the on-chip testing hardware itself can fail

1.5 At-Speed Delay Test

At-speed delay test is widely used to test timing-related failures. It has become a common practice for nowadays semiconductor industry to include it in its test flow. In this Section, we will briefly introduce the basics of at-speed delay test, including its application, fault models used, test clock configuration, and some challenging issues when applying delay test on nanometer designs.

1.5.1 Why At-Speed Delay Test?

As technology scales, feature size of devices and interconnects shrink and silicon chip behavior becomes more sensitive to on-chip noise, process and environmental variations, and uncertainties. The defect spectrum now includes more problems such as high-impedance shorts, in-line resistance, power supply noises and crosstalk between signals, which are not always detected with the traditional stuck-at fault model. The number of defects that cause timing failure (setup/hold time violation) is on the rise. This leads to increased yield loss and escape and reduced reliability. Thus structured delay test, using transition delay fault model and path delay fault model, are widely adopted because of their low implementation cost and high test coverage. Transition fault testing models delay defects as large gate delay faults for detecting timing-related defects. These faults can affect the circuit's performance through any sensitized path passing through the fault site. However, there are many paths passing through the fault site; and TDF are usually detected through the short paths. Small delay defects can only be detected through long paths. Therefore, path delay fault testing for a number of selected critical (long) paths is becoming necessary. In addition, small delay defects may escape when testing speed is slower than functional speed. Therefore at-speed test is preferred to increase the realistic delay fault coverage. In [18], it is reported that the defects per million rates are reduced by 30–70% when at-speed testing is added to the traditional stuck-at tests.

1.5.2 Basics on At-Speed Test: Launch-off-Capture and Launch-off-Shift

The transition fault and path delay fault are the two most widely used fault models for at-speed delay test. Path delay model targets the cumulative delay through the entire list of gates in a predefined path while the transition fault model targets each gate output in the design for a slow-to-rise and slow-to-fall delay fault [5]. The transition fault model is more widely used than path delay because it tests for at-speed failures at all nets in the design and the total fault list is equal to twice the number of nets. On the other hand, there are billions of paths in a modern design to

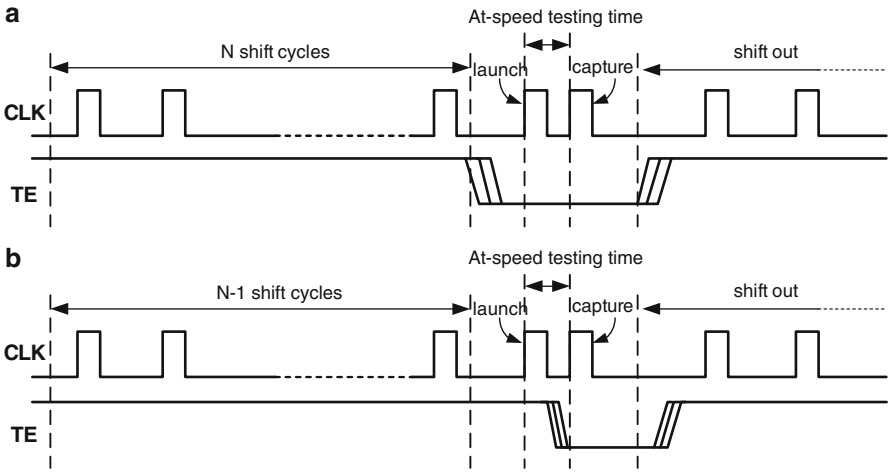


Fig. 1.15 Clock and Test Enable waveforms for LOC and LOS Tests

be tested for path delay fault leading to high analysis effort. This makes path delay fault model very cost intensive compared to transition fault model.

Compared to static testing with the stuck-at fault model, testing logic at-speed requires a test pattern with two vectors. The first vector launches a logic transition value along a path, and the second part captures the response at a specified time determined by the system clock speed. If the captured response indicates that the logic involved did not transition as expected during the cycle time, the path fails the test and is considered to contain a defect.

Scan based at-speed delay testing is implemented using launch-off-capture (LOC) (also referred as broadside [19]) and Launch-off-shift (LOS) delay tests. LOS tests are generally more effective, achieving higher fault coverage with significantly fewer test vectors, but require a fast scan enable, which is not supported by most designs. For this reason, LOC-based delay test is more attractive and used by more industry designs. Figure 1.15 shows the clock and test enable waveforms for LOC and LOS at-speed delay tests. From this figure, we can see LOS has a high requirement on the TE signal timing. An at-speed test clock is required to deliver timing for at-speed tests. There are two main sources for the at-speed test clocks. One is the external ATE and the other is on-chip clocks. As the clocking speed and accuracy requirements rise, since the complexity and cost of the tester increase, more and more designs include a phase-locked loop or other on-chip clock generating circuitry to supply internal clock source. Using these functional clocks for test purposes can provide several advantages over using the ATE clocks. First, test timing is more accurate when the test clocks exactly match the functional clocks. Secondly, the high-speed on-chip clocks reduce the ATE requirements, enabling use of a less expensive tester [18].

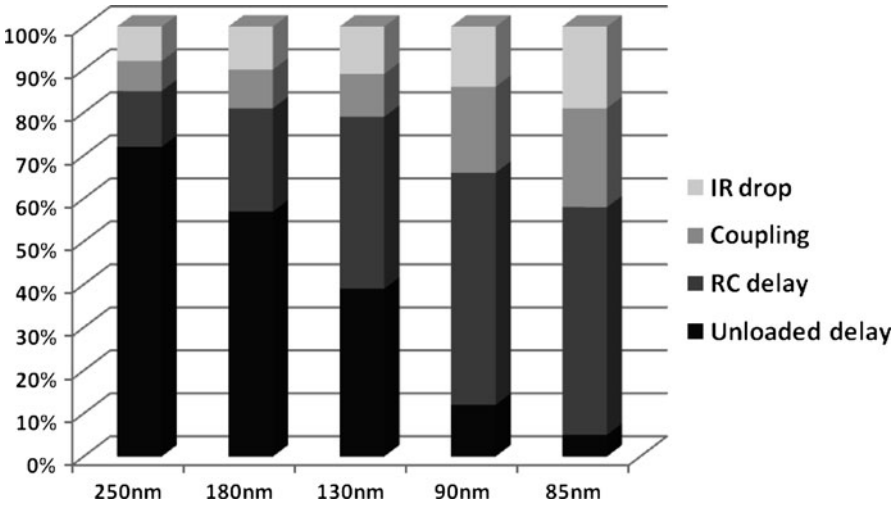


Fig. 1.16 At nanometer process nodes, parasitic effects increase [20]

1.5.3 At-Speed Delay Test Challenges

As circuit complexity and functional frequency increase, power integrity and timing integrity are becoming more and more important to circuit design and test. The test power consumption, supply voltage noise, crosstalk noise caused by signal coupling effect, and hot spots caused by nonuniform on-chip temperature will significantly impact yield and reliability. As shown in Fig. 1.16 from [20], with technology node shrinks, the percentage of delay caused by coupling effect between signal lines (crosstalk noise) and IR-drop on power and ground lines (power supply noise) is taking a larger portion. Power supply noise and crosstalk noise are becoming two important noises that impact circuits' timing integrity. The lower supply rails in today's ICs mean much less immunity from signal-integrity problems that tie directly into power integrity [21]. Supply voltages on many high-end ICs are now down to 1 V and below, leading to decreasing margins for voltage fluctuation. Simultaneous switching noise can cause ground to fluctuate, leading to difficult-to-isolate signal-integrity problems and timing issues. Power, timing, and signal integrity (SI) effects are all interdependent at 90-nm and below.

Timing failures are often the result of a combination of weak points in a design and silicon abnormalities, which reduce the noise immunity of the design and expose it to SI issues. For example, a poor power planning or missing power vias can incur on-chip power droop for some test vectors. The power droop can impact a gate(s) on a critical path and it may cause timing failure. This failure may only be excited with certain test vectors as inputs. If the corresponding test vector is not included in the test pattern set, the failure becomes an escape and cannot be reproduced during diagnosis with the current test pattern set. Current automatic test pattern

generation (ATPG) tools are not aware of the switching distribution on the layout and the pattern induced noises. There are escapes and "No Problem Found" parts returned by customers, which have passed the tests using the layout-unaware test patterns generated by ATPG tools. Thus, high-quality test patterns are imperative which can be used to capture noise-induced delay problems during production test and identify noise-related failures during diagnosis [23–25].

References

1. Druckerman H, Kusco M, Peteras S, and Shephard P III (1993) Cost trade-offs of various design for test techniques. Proc Econo Des Test Manuf, pp 45–50
2. Agrawal VD (1994) A tale of two designs: the cheapest and the most economic. J Electron Test 2–3(5): 131–135
3. Dear ID, Dislis C, Ambler AP, Dick JH (1991) Economic effects in design and test. IEEE Design Test Comput 8(4): 64–77
4. Pittman JS, Bruce WC (1984) Test logic economic considerations in a commercial VLSI chip environment. In: Proceedings of the International Test Conference, October 1984, pp 31–39
5. Bushnell ML, Agrawal VD (2000) Essentials of Electronic Testing for Digital, Memory, and Mixed-signal VLSI Circuits. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
6. Davis B (1982) The Economics of Automatic Testing. McGraw-Hill, London, United Kingdom
7. Eldred RD (1959) Test routines based on symbolic logical statements. J ACM 6(1): 33–36
8. Keim M, Tamarapalli N, Tang H, Sharma M, Rajski J, Schuermyer C, Benware B (2006) A rapid yield learning flow based on production integrated layout-aware diagnosis. In: ITC. Paper 7.1
9. Goldstein LH (1979) Controllability/observability analysis of digital circuits. IEEE Trans Circuits Syst CAS-26(9): 685–693
10. Martin G, Scheffr L, Lavagno L (2005) Electronic Design Automation for Integrated Circuits Handbook. CRC Press, West Palm Beach, FL, USA, ISBN: 0849330963
11. Eichelberger EB, Williams TW (1977) A logic design structure for LSI testability. In: Proceedings of the Design Automatic Conference, June 1977, pp 462–468
12. Rajski J, Tyszer J, Kassab M, Mukherjee N (2004) Embedded deterministic test. IEEE Trans Comput Aided Design 23(5): 776–792
13. Touba NA (2006) Survey of test vector compression techniques. IEEE Design Test Comput 23(4): 294–303
14. Boppana V, Fuchs WK (1996) Partial scan design based on state transition modeling. In: Proceedings of the International Test Conference (ITC'96), p 538
15. IEEE Standard 1149.1–2001 (2001) Standard test access port and boundary-scan architecture. IEEE Standards Board
16. Oshana R (2002) Introduction to JTAG. In: EE Times Design, 29 October 2002
17. Lu J-M, Wu C-W (2000) Cost and benefit models for logic and memory BIST. In: Proceedings of the DATE 2000, pp 710–714
18. Swanson B, Lange M (2004) At-speed testing made easy. In: EE Times, vol 3, June 2004
19. Savir J, Patil S (1994) On broad-side delay test. In: Proceedings of the IEEE 12th VLSI Test Symposium (VTS 94), pp 284–290
20. Bell G (2004) Growing challenges in nanometer timing analysis. In: EE Times, 18 October 2004
21. Maliniak D (2005) Power integrity comes home to roost at 90 nm. In: EE Times, 03 February 2005
22. Williams TW, Brown NC (1981) Defect level as a function of fault coverage. IEEE Trans Comput C-30(12): 987–988

23. Ma J, Lee J, Tehranipoor M (2009) Layout-aware pattern generation for maximizing supply noise effects on critical paths. In: Proceedings of IEEE 27th VLSI Test Symposium (VTS'09), pp 221–226
24. Ma J, Ahmed N, Tehranipoor M (2011) Low-cost diagnostic pattern generation and evaluation procedures for noise-related failures. In Proceedings of IEEE 29th VLSI Test Symposium (VTS'11), pp 309–314
25. Ma J, Lee J, Ahmed N, Girard P, Tehranipoor M (2010) Pattern grading for testing critical paths considering power supply noise and crosstalk using a layout-aware quality metric. In: Proceedings of GLSVLSI'10, pp 127–130
26. Goel SK, Devta-Prasanna N, Ward M (2009) Comparing the effectiveness of deterministic bridge fault and multiple-detect stuck fault patterns for physical bridge defects: a simulation and silicon study. In: Proceedings of International Test Conference (ITC'09), pp 1–10

Chapter 2
Hardware Implementation of Hash Functions

Zhijie Shi, Chujiao Ma, Jordan Cote, and Bing Wang

2.1 Introduction to Cryptographic Hash Functions

Hash algorithm is a type of cryptographic primitives. Hash algorithms take as input a message of arbitrary length, and produce a *hash* or *message digest* as output. This process can be denoted as:

$$h = H(M),$$

where M is the input message and h is the hash generated by the hash algorithm H. Normally, the size of the hash h is fixed by the algorithm. For a cryptographic hash function, the hash length should be large enough to prevent an attack from finding two or more messages that generate the same hash. Currently, the most commonly used hash algorithms are MD5 [1] and SHA-2 [2].

In general, the cryptographic hash algorithms should have the following properties:

1. *Preimage resistance*. Given a hash h, it should be difficult to find a message M such that $h = H(M)$. This is part of the one-way property. It is easy to compute the hash from a message, but it is extremely difficult to deduce a message from the hash.
2. *Second preimage (or 2nd-preimage) resistance*. Given a message M_1, it is difficult to find another message M_2 such that M_1 and M_2 generate the same hash. Any change in the input results in changes, often wild changes, in the hash.
3. *Collision resistance*. It should be difficult to find two messages M_1 and M_2 such that $M_1 \neq M_2$, but $H(M_1) = H(M_2)$.

A hash function that is both preimage resistant and 2nd-preimage resistant is called a one-way hash function. Note that preimage resistance does not imply 2nd-preimage

Z. Shi (✉) · C. Ma · J. Cote · B. Wang
Computer Science and Engineering Department, University of Connecticut, Storrs, CT, USA
e-mail: zshi@engr.uconn.edu; chujiao.ma@engr.uconn.edu; cote@engr.uconn.edu; bing@engr.uconn.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_2,

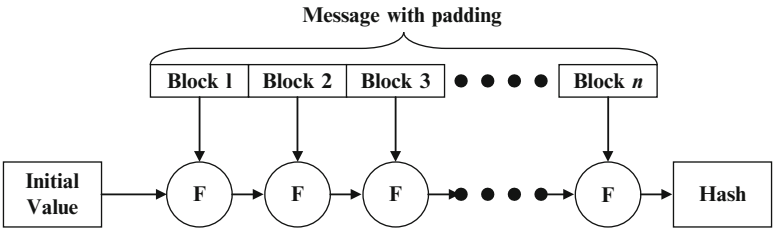


Fig. 2.1 The Merkle-Damgård model of hash functions

resistance, and vice versa. If a hash function is collision resistant (and thus also 2nd-preimage resistant), it is a collision resistant hash function (CRHF). Although collision resistance does not guarantee preimage resistance, most CRHFs in practice appear to be preimage resistant [3].

2.1.1 Construction of Hash Functions

Hash function can be constructed in several ways. The Merkle-Damgård model, illustrated in Fig. 2.1, has shown good properties over the years [4], and has been adopted in the design of many successful hash functions such as MD5 and SHA-2. In this model, the message is padded and divided into blocks of uniform length. The blocks are then processed sequentially with a compression function F. Starting with an initial hash, F repeatedly generates a new intermediate hash value from the previous one and a new message block. The output of the final compression function is the hash of the message. The Merkle-Damgård model makes it easy to manage large inputs and produce a fixed-length output. The security of this scheme relies on the security of the compression function F. It can be proven that if the compression F is collision resistant then the hash function constructed from the Merkle-Damgård model is also collision resistant [4].

MD5, SHA-2, and their variants are the most popular hash algorithms. They follow the Merkle-Damgård model and use logic operations such as AND, OR, and XOR in their compression functions. Recently, collision pairs have been found for MD5, SHA-0 and SHA-1 [5–8], making these algorithms vulnerable to attacks because they do not have the collision resistance property. Although no successful attacks have been reported for algorithms in the SHA-2 family, e.g., SHA-256 and SHA-512, these algorithms may be vulnerable to the same type of attacks because they are designed with similar principles: a Merkle-Damgård model with a compression function consisting of logic operations. Thus, it is imperative to study new compression functions and even new construction models for fast and secure hash functions. In fact, National Institute of Standards and Technology (NIST) held a public competition to select new hash algorithm, SHA-3, starting from 2007 [9].

SHA-3 is expected to be more secure than SHA-2. New hash design techniques are also adopted in many SHA-3 candidates. We will look at some of the candidates later in this chapter.

A hash function can also be constructed from a symmetric-key cipher, such as DES or AES. Several schemes have been proposed to turn a block cipher into a hash function. These scheme include Davies-Meyer, Matyas-Meyer-Oseas [10], and Miyaguchi-Preneel schemes [5, 6, 11, 12]. The Miyaguchi-Preneel scheme also has an iterative structure. It starts with an initial hash and updates the hash when a block of message is processed. The scheme can be expressed as $h' = E_{g(h)}(M_i) \oplus h \oplus M_i$, where M_i is message block i, h the hash before M_i is processed, h' the updated hash after M_i is processed, and $g()$ is a function that converts h to a key for the block cipher E.

Another notable algorithm is Whirlpool, a hash function constructed with the Miyaguchi-Preneel scheme [13]. It has been recommended by the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) project and adopted by International Organization for Standardization (ISO) as part of ISO 10118–3. The block cipher in Whirlpool is a dedicated cipher called W, which is similar to the AES algorithm despite significant differences. No successful attacks to Whirlpool have been found so far. However, the Whirlpool algorithm is much slower than other hash algorithms. For example, on Pentium III, Whirlpool's performance is 36.5 cycles/byte while MD5's is 3.7 cycles/byte and SHA-1's is 8.3 cycles/byte [14].

2.1.2 Application of Hash Functions

Hash algorithms can be used to verify data integrity. After receiving data, one can compute the hash of the received data, and compare it with the hash of the original data, which may be sent through secure channels or obtained from a trusted source. If they are the same, there is a high confidence level that the message was not modified during the transmission (because of the 2nd-preimage resistance). Nowadays, many software download sites provide the MD5 or SHA-2 hash of the software available for downloading.

Hash algorithms can also be used together with public-key algorithms to generate digital signatures. For example, Alice can sign a document by encrypting the hash of the message with her private key. The ciphertext can be used as Alice's signature. Anyone who wants to verify the signature can decrypt the ciphertext with Alice's public key, and compare the decrypted value with the hash generated from the message. This process is shown in Fig. 2.2. The left part of the figure generates the signature with Alice's private key, and the right part checks the signature with her public key.

Hash algorithms can be used for authentication as well. In this case, the hash value of the user's password, instead of the password itself, is transmitted and compared by the server. When computing the hash, the password may be concatenated with a random value generated by the server. Hence, the hashes are different every time, preventing an attacker sniffing on the network traffic

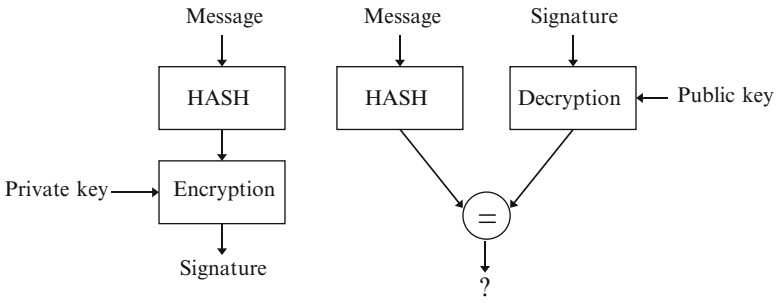


Fig. 2.2 Application of hash algorithm in digital signature

from reusing an old hash. For example, the Challenge-Handshake Authentication Protocol (CHAP) [15] uses this approach in the Point-to-Point Protocol (PPP) [16] for dial-up Internet connections.

Hash algorithms can also be used in Hash-based Message Authentication Code algorithms [17] have been used in many systems to provide both data integrity and message authentication, especially in resource constrained environments where public-key algorithms are too slow and energy consuming.

2.2 Hardware Implementation of Hash Functions

MD5 and SHA-2 (more specifically SHA-256) are commonly used hash functions today. Both of them adopt the Merkle-Damgård construction. As they have much in common, many optimization techniques can be applied to both algorithms.

2.2.1 MD5

MD5 is a common Merkle-Damgård-based hash function. A MD5 block has 512 bits, which can be divided into sixteen 32-bit words $(M_0, \ldots, M_{15})$. The internal hash state has 128 bits, which are stored as four 32-bit words and denoted by the 4-tuple (A, B, C, D), and is initially set to a predetermined initialization vector. The basic computation unit of MD5 is called a step. Each step alters the state variables (A, B, C, D), as shown in Equation 2.1, using a compression function F that takes B, C, and D as input and generates a 32-bit value as output. In the equation, $\lll$ indicates the left rotation operation and $W \lll S$ rotates bits in W to left by S. In total, MD5 has 64 steps. The first 16 steps are called round one, the next 16 round two, and so on. Rounds also have distinct compression functions, as specified in Equation 2.2, where i is the step number.

Table 2.1 A 3-stage MD5 pipeline

Steps	Computation of B
Step i	$B_i = B_{i-1} + (\text{AKM}_i + \text{F}(B_{i-1}, C, D)) \lll S_i$
Step $i+1$	$\text{AKM}_{i+1} = M_i + 1 + (\text{AK}_{i+1})$
Step $i+2$	$\text{AK}_{i+2} = (K_{i+2} + B_{i-1})$

$$\begin{aligned} A_{\text{new}} &= D \\ B_{\text{new}} &= B + (M_i + K_i + A + F(B, C, D)) \lll S_i \\ C_{\text{new}} &= B \\ D_{\text{new}} &= C \end{aligned} \tag{2.1}$$

$$F(B, C, D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & : 0 \leq i \leq 15 \\ (B \wedge C) \vee (C \wedge \neg D) & : 16 \leq i \leq 31 \\ B \oplus C \oplus D & : 32 \leq i \leq 47 \\ C \oplus (B \wedge \neg D) & : 48 \leq i \leq 63 \end{cases} \tag{2.2}$$

$$M_i = \begin{cases} M_i & : 0 \leq i \leq 15 \\ M_{5i+1(\text{mod } 16)} & : 16 \leq i \leq 31 \\ M_{3i+5(\text{mod } 16)} & : 32 \leq i \leq 47 \\ M_{7i \ (\text{mod} 16)} & : 48 \leq i \leq 63 \end{cases} . \tag{2.3}$$

As there are only 16 words in a message block, each word is used four times in the 64 steps. The message scheduler, Equation 2.3, determines which of the 16 message words M_i is used in a particular step i. For example, round one simply uses word M_i for step i, while round two uses $M_{5i+1(mod16)}$. Furthermore, $K_0, \ldots, K_{63}$ and $S_0, \ldots, S_{63}$ are predetermined constant values. After the completion of all 64 steps, the latest values (*A, B, C, D*) are added to the input values (*A, B, C, D*) of the block, using modulo 2^{32} addition. That sum is the updated hash value and serves as the input hash value for the next message block (512 bits) if there is one. The final hash is the concatenation of A, B, C, and D generated from the last block. The specification for MD5 was presented in [1].

2.2.1.1 Pipelining

The operations in each step of MD5 are not complicated. When implemented in hardware, the number of operations that can be done in parallel is limited by data dependency. The data dependency between the state variables of MD5 allows for interesting pipelining applications. In particular, since only B is updated in a step and other words are not, one may use data forwarding as described in [18]. Because some future values of A are known, the computation of B_{i+1} and B_{i+2} may start early. As shown in Table 2.1, the updated values of B can be computed in three stages. When B_i is computed in step i, AKM_{i+1} in step $i+1$ and AK_{i+2} in step $i+2$ can be computed in parallel for B_{i+1} and B_{i+2}.

Another approach to MD5 pipelining is to take advantage of the independency among rounds. For example, in a 4-stage pipeline with one stage dedicated to each

Table 2.2 FPGA implementations of MD5

Type	Block latency (ns)	Throughput (Mbps)
Iterative [21]	843	607
Pipelined (64 stage) [21]	706	725
Parallel (10x) [21]	875	5,857
Pipelined (3 stage) [18]	–	746
Iterative [19]	633	810
Pipelined (32 stage) [19]	764	21,428
Pipelined (32 stage), Loop unrolled [19]	511	32,035

round, the 16 steps in each round are computed iteratively within a stage. Therefore, the stages are much longer. One extension to this architecture is to unroll the loop for the 16 steps in each stage, requiring dedicated logic for each step rather than reusing logic for 16 steps in a pipeline stage. There are also performance gains as the results of the reduction in clock and register delay [19].

2.2.1.2 Other Optimizations

Carry save adders (CSA) are commonly used as an alternative to the full adder when there are multiple numbers to be added. A CSA is much smaller and faster than a regular adder because it does not require carry propagation. A regular adder is needed only at the end of a series of CSA additions. However, in FPGA implementations, the utility of using CSA is diminished since some platforms incorporate partial addition optimizations which allow ripple carry adders to perform just as well while using less area [20].

If the hash function is being implemented in FPGA, block RAM can be used to improve performance. Block RAM is a dedicated part of the chip which may be used to store constants. This takes the burden off the combinatorial logic blocks, freeing them for other purposes. This means that more CLBs are available to the routing algorithm, resulting in more direct routes and faster performance.

2.2.1.3 MD5 Performance

Table 2.2 lists some fast MD5 implementations. The FPGA implementations have reached extremely high throughput up to 32 Gbps. The reader should exercise caution while reviewing these numbers, as a high throughput does not imply a low latency. This is especially true for parallel block implementations. Processing blocks for a single message cannot be done in parallel since they are dependent on each other. Instead, a parallel block implementation is similar to using multiple processing units with independent input streams. Hence, the configuration is only useful when the input is parallel in nature.

Table 2.3 SHA-2 state variables, compression functions, and message scheduler

$A_{\text{new}} = T_1 + T_2$	$T_1 = H + \sum_1(E) + \text{Ch}(E, F, G) + K_t + W_t$
$B_{\text{new}} = A$	$T_2 = \sum_0(A) + \text{Maj}(A, B, C)$
$C_{\text{new}} = B$	
$D_{\text{new}} = C$	
$E_{\text{new}} = D + T_1$	$\text{Ch}(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G)$
$F_{\text{new}} = E$	$\text{Maj}(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$
$G_{\text{new}} = F$	$\sum_0(A) = (A \ggg s_1) \oplus (A \ggg s_2) \oplus (A \ggg s_3)$
$H_{\text{new}} = G$	$\sum_1(E) = (E \ggg s_4) \oplus (E \ggg s_5) \oplus (E \ggg s_6)$
$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$	
$\sigma_0(W_{t-15}) = (W_{t-15} \ggg s_7) \oplus (W_{t-15} \ggg s_8) \oplus (W_{t-15} \gg s_9)$	
$\sigma_1(W_{t-2}) = (W_{t-2} \ggg s_{10}) \oplus (W_{t-2} \ggg s_{11}) \oplus (W_{t-2} \gg s_{12})$	

Note :$\oplus$: bitwise exclusive OR, $+$: addition (modulo 2^{32} or 2^{64}), $\ggg$: rotate right, $\gg$: shift right

2.2.2 SHA-2

SHA-2 has several different modes, SHA-224, 256, 384, and 512, where the numbers denote the length of the hash. Shorthand "SHA-256" and the like refer to specific SHA-2 modes and not SHA-1, which has a 160 bit digest. SHA-2, like MD5, is a Merkle-Damgård-based hash and so the initial setup is similar. For SHA-224 and SHA-256, the message block size (512 bits) is the same as in MD5. However, the number of state variables is doubled. There are eight state variables in SHA-2. In SHA-384 and SHA-512, the word size is also doubled to 64 bits. Thus, the block size increases to 1,024 bits.

There are no rounds in SHA-2, although there are still 64 steps in SHA-256 and 80 steps for SHA-384/512. The message scheduler also involves more than simply shuffling the message chunks. Rotation and exclusive OR are used to combine words in earlier rounds. In addition, a set of new compression functions Ch, Maj, $\sum_0$, and $\sum_1$ replaces MD5's F function. Table 2.3 shows the operations in SHA-2, which are identical for all variants (although the word sizes and constants are different).

2.2.2.1 Iterative Bound Analysis

The optimization of an algorithm is limited by its data dependencies. A method called Iterative Bound Analysis (IBA) has been proposed to find the theoretical maximum performance limit [22]. This means that while advances in silicon process may increase the speed of implementations, the speed gained by refactoring an implementation from an architectural standpoint will have an inherent limit.

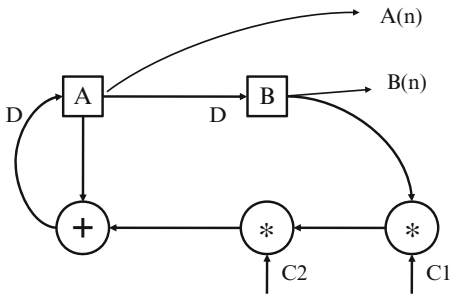


Fig. 2.3 A DFG example from [22]

To find the bound with IBA, the data flow graph (DFG) for the algorithm is first constructed. For example, Fig. 2.3 shows the DFG for the following equation:

$$A(n+1) = A(n) + B(n) * C1 * C2$$
$$B(n+1) = A(n)$$

In the equation, $C1$ and $C2$ are constants; A and B are variables updated in a loop. The DFG has two types of nodes. One type is the variables, such as A and B, and the other type is the operations that are needed for updating the variables, such as $+$ and $*$. There are also two types of delays. Algorithmic delays, such as updating A and B, are denoted by D on edges that are essential for the algorithm and cannot be removed. Functional delays are from operation nodes, such as $+$ and $*$, indicating the time needed for performing the operations. IBA assumes all the operations are atomic, i.e., functional operations cannot be split or merged.

A DFG may have loops, like the one shown in Fig. 2.3. A loop is defined as a path that begins and ends at the same node. It contains both algorithmic delays and functional delays. The sum of the functional delays on a loop is the running time of the loop. The performance bound for a loop is given by the loop's running time divided by the number of algorithmic delays in the loop. There may be multiple loops in the DFG. The iterative bound of the algorithm is the maximum of the performance bound for all loops in the DFG. It defines the best throughput that may be reached.

Two techniques are proposed in [22] to make the performance close to that of the iterative bound. One of them is called retiming, which moves the algorithmic delays (the D edges in DFG) through functional nodes. Retiming can reduce the critical path delay, which is the longest total functional delay between two adjacent algorithmic delays, by balancing the functional delays among algorithmic delays. The second technique in [22] is called unfolding, which is similar to loop unrolling. It performs multiple iterations of the loop in a single cycle. After unfolding, the DFG has more functional nodes, thus having more opportunities to reach the optimal throughput. Retiming can also be applied here to balance out the delays in an unfolded DFG.

Table 2.4 SHA-256 implementations

Type	Platform	Block latency (ns)	Throughput (Mbps)
Pipelined (4 stage), Unrolled [25]	FPGA	614	3,334
Operation reordering [20]	FPGA	433	1,184
Loop unrolled [22]	ASIC(0.13 μm)	86	5,975
Pipelined (3 stage) [26]	ASIC (0.13 μm)	∼70	>6,500

Loop unrolling is a commonly used technique in the implementation of hash algorithms. One implementation was able to increase the speed of SHA-512 by 30% with an area penalty of 48% [23]. In [24], SHA-512 achieved a speedup of 24.6% with an area penalty of 19.4 after 2x unrolling. Likewise, 4x unrolling yielded a 28.7% speed up with an area penalty of 75.6%, so returns are diminishing.

2.2.2.2 SHA-2 Performance

Table 2.4 summarizes the best performance of some SHA-2 implementations. The techniques mentioned in the previous section yield good results. However, SHA-2 is actually slower than MD5 because of the increased level of interdependency and complexity. These characteristics limit the amount of data forwarding pipelining and low-level parallel computation that can be applied. Note that we have some extremely low latency ASIC implementations for SHA-2. This is primarily due to high clock rate, e.g., approximately 1 GHz in [26, 27].

2.2.3 Optimized for Area

Sometimes a compact design is required for platforms with limited resources such as area or power. An example is RFID tags, which are very limited in both aspects. They typically have a maximum average current draw of 15 μA total [28]. As of 2009, RFID tags typically have 3,000 ∼ 4,000 gates available for cryptographic purposes [29].

The smallest implementation for SHA-256 was reported in [30], where the authors obtained a size of 8,588 gates on a 250 nm CMOS process. This implementation is capable of a throughput of 142 Mbps. The authors used a folding technique to optimize for area. This is basically the opposite of the unrolling process described earlier. In this case, a single iteration takes seven cycles to complete. While there are seven modulo addition operations in SHA-2, only one physical adder is synthesized and then reused in each step. Another technique shrinks the message scheduler, which takes up more space than the compression function. This reduction is primarily due to using only a single 32-bit register rather than sixteen 32-bit registers. These registers were used for timing purposes to obtain W_{t-2}, W_{t-7},

W_{t-15} and W_{t-16} by having the 16 registers in series so that the correct W values are always available. The area optimized version's single register holds temporary values that are written back into the memory block. The needed W values for each computation are stored in the memory rather than sequential registers.

Some applications do not require all the properties of a secure cryptographic hash function. For example, some applications may need only one-wayness, not requiring collision resistance. Therefore, some special purpose hash functions that have only the needed secure properties for particular applications can be implemented with smaller area. The downside of using a less known algorithm is that it has not been subject to as much scrutiny as popular algorithms have been. Therefore, the level of security provided by these hash functions is not well known. On the other hand, an obscure algorithm does have a unique (perhaps remote) advantage over popular ones in that someone may have indeed found an attack to a popular algorithm but has declined to publicize it.

2.3 SHA-3 Candidates

Due to security concerns of SHA-1 and recent advances in the cryptanalysis of hash algorithms, NIST held a public competition for a new hash algorithm standard, SHA-3, which is meant to replace SHA-2. Therefore, the new algorithm is expected to be more secure SHA-2. As of now, early 2011, the competition is at the final round, with five algorithms up for consideration. This section examines the constructions and properties of all five algorithms in the final round: Keccak based on sponge construction, BLAKE based on HAIFA construction, Grøstl based on the chop-Merkle-Damgård iterations, Skein based on tweakable block cipher Threefish and Unique Block Iteration (UBI), and JH based on iterative construction with generalized AES methodology [31].

2.3.1 Keccak

Keccak is a family of cryptographic hash functions based on sponge construction. Sponge construction, shown in Fig. 2.4, is a simple iterated construction with a variable-length input and arbitrary length output based on a fixed length transformation (or permutation) operating on a fixed number of bits. The fixed number of bits is the width of the permutation, or bit state b. The bit state is the sum of bit rate r and bit capacity c. The permutation in this case is called Keccak-f, the most important building block of this algorithm. There are seven Keccak-f permutations, indicated by Keccak-[b], where $b = 25 \times 2^{\ell}$ and ℓ ranges from 0 to 6 producing the values $b = 25, 50, 100, 200, 400, 800$, and 1,600. The default permutation is Keccak-f [1600] with $r = 1,024$ and $c = 576$ [32].

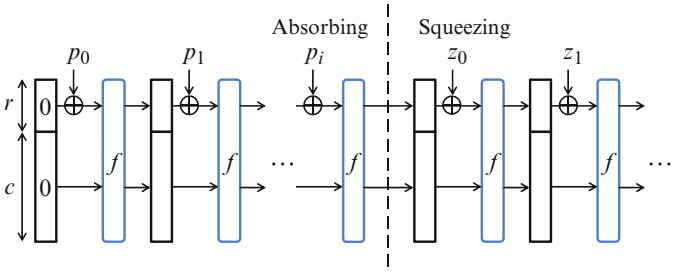


Fig. 2.4 Diagram of sponge construction

θ *step*

$$C[x] = A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] \quad \forall x \text{ in } 0 \ldots 4$$

$$D[x] = C[x-1] \oplus (C[x+1] \lll 1) \quad \forall x \text{ in } 0 \ldots 4$$

$$A[x,y] = A[x,y] \oplus D[x] \quad \forall (x,y) \text{ in } (0 \ldots 4, 0 \ldots 4)$$

ρ *and* π *step*

$$B[y, 2x+3y] = A[x,y] \lll r[x,y] \quad \forall (x,y) \text{ in } (0 \ldots 4, 0 \ldots 4)$$

χ *step*

$$A[x,y] = B[x,y] \oplus \left((\neg B[x+1,y]) \wedge B[x+2,y] \right) \quad \forall (x,y) \text{ in } (0 \ldots 4, 0 \ldots 4)$$

ι *step*

$$A[0,0] = A[0,0] \oplus RC$$

return A

Fig. 2.5 Pseudocode of Keccak-f

Before any permutation is performed, Keccak initializes state s and pads the message to make the string a multiple of r. The padded message, represented by P, is then divided into i blocks. For sponge construction, there are two phases. In the first phase (absorbing) the r-bit input message blocks are XOR-ed into the first r bits of the state, interleaved with applications of the function f (Keccak-f). This phase is finished when all message blocks are processed. In the second phase (squeezing) the first r bits of the state are returned as hash bits, interleaved with applications of the function f. This phase is finished when the desired length of hash is produced [33].

The basic block in Keccak is Keccak-f, an iterated permutation consisting of a sequence of almost identical rounds [4]. The number of rounds n_r depends on the permutation width b, and is computed as $n_r = 12 + 2l$, where $2^l = b/25$. This gives 24 rounds for Keccak-f[1600]. Each round consists of five steps illustrated in Fig. 2.5. In the figure, A denotes the complete permutation state array. $A[x, y]$ denotes a particular word in the 5×5 state array. $B[x, y]$, $C[x]$, and $D[x]$ are intermediate variables, and $C[x]$ and $D[x]$ are vectors. All the operations on the indices are done modulo 5. Rotation amounts $r[x, y]$ are constants. RC in the ι step is the round constant. All the rounds of Keccak-f perform identical operations, except for using different round constant RC (consult [34] for the computation of RC).

Keccak-f is designed such that the dependence on CPU word length is only due to fixed rotations, leading to an efficient use of CPU resources on a wide range of processors. The default bit state for Keccak-f permutation (1,600) is chosen in favor of 64-bit architectures. When implemented on 32-bit processors, the rotation on 64-bit words is the only operation that cannot be simply divided into 32-bit operations. The algorithm's symmetry allows the exploitation of SIMD instructions and pipelining in processors, which results in compact code in software, or a compact coprocessor circuit suitable for constrained environments [34].

The full Keccak algorithm design has no known security vulnerabilities because the sponge construction has been successful against generic attacks. It is simple, allows variable length output, and is flexible (such as the tradeoff between bit rate and security).

2.3.2 BLAKE

Blake is based on the HAIFA iteration mode with a compression function that uses a modified version of the stream cipher ChaCha. While Merkle-Damgård only uses the previous hash value and the current message block in the compression function, HAIFA also uses a salt and a counter that indicates how many blocks have been processed as part of the input [35].

Depending on the message size and word length, different BLAKE hash functions can be used. BLAKE-224 and BLAKE-256 operate on 32-bit words, with block size of 512 bits and salt of 128 bits. BLAKE-384 and BLAKE-512 operate on 64-bit words with block size of 1024 bits, salt of 256 bits. The numbers in the algorithm name indicate the digest length.

All four variants use the same compression function and only differ by the initial value, the message padding, and the truncation of the output. The compression function of BLAKE adopts the wide-pipe design. A large inner state is initialized, injectively updated by message-dependent rounds, then compressed to return the next chain value. The actual compression method is based on modified version of ChaCha, which is a variant of Salsa20 family of stream ciphers. In the following, BLAKE-512 will be used as the example to explain the algorithm.

In BLAKE-512, the message is first padded and divided into blocks of 1,024 bits. The intermediate hash value is then initialized. The initial values used in BLAKE are same as those used in SHA-2 (e.g., BLAKE-512 uses the values from SHA-512, BLAKE-256 uses the values from SHA-224). Then, BLAKE iteratively updates the intermediate hash value using the compression function: $h_{i+1} \leftarrow \text{compress}(h_i, m_i, s, t_i)$, where m_i is the message block, s is the salt, and t_i is the counter indicating the number of bits that have been processed so far.

The compression function of BLAKE-512 has 16 rounds. The inner state of the compression function is represented as a 4×4 matrix of words. For each round, a nonlinear function G that operates on four words is applied to columns and diagonals of the state. This process is illustrated in Fig. 2.6. First, all four columns

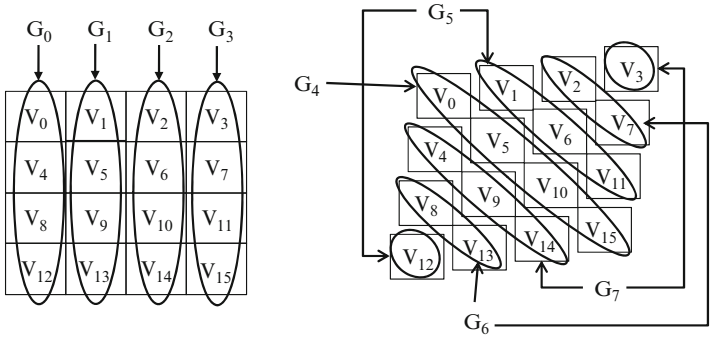


Fig. 2.6 Column step (*on the left*) and diagonal step (*on the right*) in BLAKE

Fig. 2.7 Pseudocode of G in BLAKE-512

$$
\begin{aligned}
a &\leftarrow a + b + \left(m_{\sigma_r(2i)} \oplus C_{\sigma_r(2i+1)}\right)\\
d &\leftarrow (d \oplus a) \ggg 32\\
c &\leftarrow c + d\\
b &\leftarrow (b \oplus c) \ggg 25\\
a &\leftarrow a + b + \left(m_{\sigma_r(2i+1)} \oplus C_{\sigma_r(2i)}\right)\\
d &\leftarrow (d \oplus a) \ggg 16\\
c &\leftarrow c + d\\
b &\leftarrow (b \oplus c) \ggg 11
\end{aligned}
$$

are updated independently with $G_0, \ldots,$ and G_3. This procedure is called a column step. Thereafter, the four disjoint diagonals are updated with $G_4, \ldots,$ and G_7. This procedure is called a diagonal step. Note that $G_0, \ldots,$ and G_3 can be computed in parallel because each of them updates a distinct column of the matrix. Similarly, $G_4, \ldots,$ and G_7 update distinct diagonals and thus can be parallelized as well.

The pseudocode of $G_i(a,\ b,\ c,\ d)$ is shown in Fig. 2.7, where σ_r represents a permutation of integers between 0 and 15. There are ten different permutations, which are reused when the round number is ten or greater. For example, round 10 uses σ_0 and round 11 uses σ_1.

After performing the round function, the new hash values are extracted from the state and salt:

$$
\begin{aligned}
h'0 &\leftarrow h0 \oplus s0 \oplus v0 \oplus v8\\
h'1 &\leftarrow h1 \oplus s1 \oplus v1 \oplus v9\\
&\ldots\\
h'7 &\leftarrow h7 \oplus s3 \oplus v7 \oplus v15
\end{aligned}
\quad .
$$

As the salt has only four words, each word is used twice. For example, $s0$ is used in $h'0$ and $h'4$.

Lastly, the algorithm truncates the final chaining value if needed. The truncation method slightly differs depending on the version of the algorithm [36].

There are several important design choices in BLAKE. The HAIFA iteration mode provides resistance to long-message second preimage attacks, and the local wide-pipe internal structure makes local collisions impossible. The compression algorithm is based on ChaCha, which has good performance and has been intensively analyzed. The properties of compression function also allow parallel processing. The G functions performed on the four columns and diagonals can be done in parallel, which allows convenient performance-area trade-off. The performance can be improved if more resources are available for performing multiple G functions in parallel.

2.3.3 Grøstl

Grøstl is based on the wide-pipe design and chop-Merkle-Damgård iterations. A wide-pipe design has an internal state that is larger than the final hash output. When all the message blocks are processed, the internal state is truncated to get the final hash output. This process is similar to chop-Merkle-Damgård, except Grøstl applies a final output transformation before truncating the result to the desired output length. The returned message digests can be any number of bits from 8 to 512, in 8-bit steps. The hash function processing message digest of size n is called Grøstl-n.

In Grøstl, the message is first padded to a multiple of l, the length of a message block. Then the padded message is split into l-bit message blocks m_1 to m_t, and each message block is processed sequentially. The compression function comes in two variants: for Grøstl that outputs up to 256 bits, l is defined to be 512; for larger outputs, l is 1024. An l-bit initial value is set as the initial hash value $h_0 = iv$, and then the message blocks are processed as follows:

$$h_i \leftarrow f(h_{i-1}, m_i) \quad \text{for } i = 1, \ldots, t$$

$$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h,$$

where f is the compression function and h_i is the updated state after processing message block i. The compression function consists of two permutations P and Q. The two permutations are identical except for the different round constants. Both permutations consist of the following four round transformations:

- AddRoundConstant
- SubBytes
- ShiftBytes (or ShiftBytesWide in permutations for 1,024-bit blocks)
- MixBytes

Both P and Q have the same number of rounds, which depends on the internal state size: ten rounds are recommended for $l = 512$ and fourteen rounds for $l = 1{,}024$. In each round, the round transformation is applied on a state represented by a matrix A of bytes. For the short variant, the matrix is 8 by 8, and for the large variant it is 8 by 16.

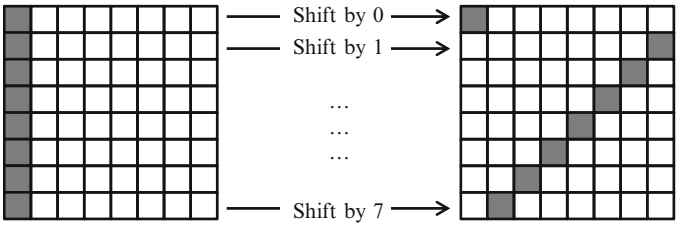
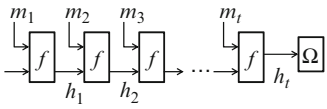


Fig. 2.8 ShiftBytes in Grøstl

Fig. 2.9 Diagram of Grøstl hashing process



The first step is AddRoundConstant, which adds (XOR) a round-dependent constant to the state matrix. The process in round i can be represented as $A \leftarrow A \oplus C_P[i]$ for permutation P and $A \leftarrow A \oplus C_Q[i]$ for permutation Q, where A is the state and $C_P[i]$ and $C_Q[i]$ are the round constants.

The second step is SubBytes, which substitutes each byte in the state matrix with its image taken from the S-Box. Grøstl uses the same S-Box as AES. If $a_{i,j}$ is the element in row i and column j of A, SubBytes transformation can be denoted as $a_{i,j} \leftarrow S(a_{i,j}), 0 \leq i < 8, 0 \leq j < v$.

The third step is ShiftBytes or ShiftBytesWide, which cyclically shift the bytes within a row to the left by the number of positions, wrapping as necessary. ShiftBytes is illustrated in Fig. 2.8. ShiftBytesWide uses the same method but each row has 16 bytes.

The last step of the transformation of the compression function is MixBytes, which multiplies each column of A by an 8×8 matrix of constants, defined as B. The transformation can be written as $A \leftarrow B \times A$.

When the last message block has been processed, the resulting h_t is then put through function Ω. The final hash is $H(m) = \Omega(h_t) = \text{trunc}_n(P(h_t) \oplus h_t)$. The function $\text{trunc}_n(x)$ discards all but the trailing n bits of x and $n < l$ [37]. The process is illustrated in Fig. 2.9.

Grøstl is very flexible. It can generate digest of many sizes, and seems secure against known attacks. The permutations are constructed with wide-pipe design, which makes all known, generic attacks on the hash function much more difficult. Therefore, it is possible to give strong statements about the resistance of Grøstl against large classes of cryptanalytic attacks. As Grøstl is based on AES, its countermeasures against side-channel attacks are well understood.

2.3.4 Skein

Skein is a family of hash functions based on tweakable block cipher Threefish and Unique Block Iteration (UBI) [38]. Tweakable block cipher allows Skein to hash configuration data along with the input text in every block and greatly improves Skein's flexibility. UBI is a chaining mode that combines an input chaining value with an arbitrary input size to a fixed output size.

Skein has three different internal state sizes, which are also the message block sizes: 256, 512, and 1,024 bits. Skein-512 is the default variant and consists of 72 rounds. Skein-1024 is an ultra-conservative and highly secure variant and consists of 80 rounds. On the other end, Skein-256 is a low-memory variant that can be implemented using about 100 bytes of RAM and consists of 72 rounds.

The UBI chaining mode combines an input chaining value with an arbitrary length input string and produces a fixed size output. Each UBI computation takes a message block and a 128-bit tweak value as input. The tweak value encodes how many bytes have been processed so far, whether this is the first and/or last block of the UBI computation, and what type of UBI computation it is. The types of UBI computations include key, configuration block, message, and output. A UBI call takes three parameters as input: the output of the previous UBI call (0 for the first call), the current content to be processed, and the type of content.

When used as a straightforward hash function, Skein consists of three UBI calls: a configuration call with the chaining value initialized to 0, then a message call that takes a message of up to $2^{96} - 1$ bytes long, and lastly an output call that truncates the chained value to produce the final hash.

- $IV = \text{UBI}\ (0, \mathit{Config}, \text{CFG})$
- $G = \text{UBI}\ (\mathit{IV}, M, \text{MSG})$
- $\mathit{Output} = \text{Truncate}\ (\text{UBI}(G, 0, \text{OUT}), o)$.

The UBI chaining mode is based on the tweakable block cipher Threefish. In Skein-512, the UBI has 72 rounds, each consisting of four simple nonlinear functions (MIX), followed by a permutation of the eight 64-bit words. The idea is that a larger number of simple rounds are more secure than fewer number of complex rounds. MIX, the basic block of Threefish, operates on two 64-bit words at a time. It uses only three mathematical operations: XOR, addition, and rotation on 64-bit words. Suppose the two words are A and B, the function of MIX can be expressed as $(A, B) = (A + B, (B <<< R) \oplus (A + B))$ [39], where R is a constant. The permutation on eight words that follows the MIX operation is the same for all rounds.

In Skein-512, a subkey is injected every four rounds, and the rotation constants R are repeated every eight rounds. The subkeys are generated from key words, tweak words, and counter value. For more details on subkey generation, please refer to Sect. 2.3.3 in the Skein submission document [38].

Skein has the option to include keys or to work in hash tree mode. To use Skein as a MAC function or any keyed hash function, a UBI call for the keys, with the

first input set to 0, is done before the configuration block. The output of the key UBI call is used as the first input of the configuration UBI call. In the optional hash tree mode, each leaf UBI call takes a message block as the input, and the output of every two UBI calls goes into a UBI call at the next level until only the root UBI remains.

Skein is designed to be simple, secure, and efficient. It is based only on three primitive operations: XOR, addition, and rotation of 64-bit words. The best attack made by the author on Threefish-512 is on 35 of the 72 rounds. Skein is quite efficient on a variety of platforms, especially on 64-bit processors. Skein-512, the default or primary variant, can be implemented in approximately 200 bytes of state. Skein-512 hashes data at 6.1 clock cycles per byte on a 64-bit CPU. This means that on a 3.1 GHz x64 Core 2 Duo CPU, Skein hashes data at 500 Mbps per core, almost twice as fast as SHA-512.

2.3.5 JH

JH is an iterative hash algorithm that produces hash values of 224, 256, 384, and 512 bits. The compression functions for all four versions are the same, and JH-512 will be used in the following discussion. For hardware implementation, the round functions of JH block cipher are identical and use techniques similar to the AES row rotations.

In JH-512, the message is first padded to a multiple of 512 bits, and then divided into blocks of four 128-bit words. For each iteration, the message block is put through the compression function F_8 to update the chaining value H_i of 1,024 bits: $H_i = F_8(H_{i-1},\ M_i)$. For the first iteration, $H_0 = F_8(H_{-1},\ M_0)$, where H_{-1} consists of two bytes representing the message digest size followed by 0s, and M_0 is set as 0. F_8 first compresses the 512-bit message block M_i with the first half of the previous chaining value H_{i-1}, feeds the result to function E_8 (to be described later), and then combines the output of E_8 with the message block. After all blocks have been processed, the n-bit hash value of the message is the last n bits of H [40].

The steps of compression function for each message block in JH-512, $H_i = F_8(H_{i-1}, M_i)$, is illustrated below, where A, and B denote two 1024-bit words:

1. $A_j = H_{i-1,j} \oplus M_i\ for 0 \le j \le 511$;
   $A_j = H_{i-1,j}\ for 512 \le j \le 1023$;
2. $B = E_8\ (A)$;
3. $H_{(i),j} = B_j\ for 0 \le j \le 511$;
   $H_{(i),j} = B_j \oplus M_i\ for 512 \le j \le 1023$;

The bijective function E_8 used in the compression function F_8 is based on d-dimensional generalized AES methodology (d is 8 in this case). The computation of $B = E_8\ (A)$ is given as follows:

1. Group the 1,024 bits in A into 2^8 4-bit elements to obtain Q_0;
2. For $r = 0$ to 34, $Q_{r+1} = R_8(Q_r, C_r^{(8)})$;

3. $Q_{36} = R_8^*(Q_{35}, C_{35}^{(8)})$;
4. De-grouping the 2^8 4-bit elements in Q_{36} to obtain B.

Each $C_r^{(8)}$ is a 2^8-bit constant. The computation of $Q = R_8(A, C_r^{(8)})$ consists of the following three steps and R_8^* only has step 1.

1. Consider A as 256 4-bit elements and replace each element with its image in an S-Box. There are two S-Boxes, and a bit in $C_r^{(8)}$ decides which S-Box to use for the corresponding 4-bit element. The results can be represented by 256 4-bit elements v_i $(0 \leq i \leq 255)$.
2. A linear transformation L is applied on each pair of 4-bit elements. The output of this step is $(w_{2i}, w_{2i+1}) = L(v_{2i}, v_{2i+1})$ for $0 \leq i \leq 127$. The linear transformation L implements a (4, 2, 3) Maximum Distance Separable code over GF (2^4). Each bit in (w_{2i}, w_{2i+1}) is an XOR of a set of bits from (v_{2i}, v_{2i+1}).
3. A permutation is done on the 256 4-bit elements generated in Step 2. $(Q_0, Q_1, \ldots, Q_{2^8-1}) = P_8(w_0, w_1, \ldots, w_{2^8-1})$.

For step 3, P_8 is a composition of three permutations: $P_8 = \phi_8 \circ P_8' \circ \pi_8$. All three functions are permutation on 2^8 elements. And the details of the three permutations are described as follows:

$$
\begin{aligned}
&B = \pi_8(A):\\
&B_{4i+0} = a_{4i+0} \; for \; i = 0 \; to \; 2^6 - 1;\\
&B_{4i+1} = a_{4i+1} \; for \; i = 0 \; to \; 2^6 - 1;\\
&B_{4i+2} = a_{4i+2} \; for \; i = 0 \; to \; 2^6 - 1;\\
&B_{4i+3} = a_{4i+3} \; for \; i = 0 \; to \; 2^6 - 1;\\
&B = P_8'(A):\\
&\quad b_i = a_{2i} \; for \; i = 0 \; to 2^7 - 1;\\
&b_{i+2^7} = a_{2i+1} \; for \; i = 0 \; to \; 2^7 - 1;\\
&B = \phi_8(A):\\
&\quad b_i = a_i \; for \; i = 0 \; to \; 2^7 - 1;\\
&b_{2i+0} = a_{2i+1} \; for \; i = 2^6 \; to \; 2^7 - 1;\\
&b_{2i+1} = a_{2i+0} \; for \; i = 2^6 \; to \; 2^7 - 1;
\end{aligned}
$$

The compression function of JH algorithm is illustrated in Fig. 2.10. The message block is then XOR-ed with the first half of the previous hash value, the result of which is put through function E, which consist of a number of rounds that include S-Box, linear transformation L, and permutation P_d. The message is then XOR-ed with the second half of the output of E, which produces the updated hash value.

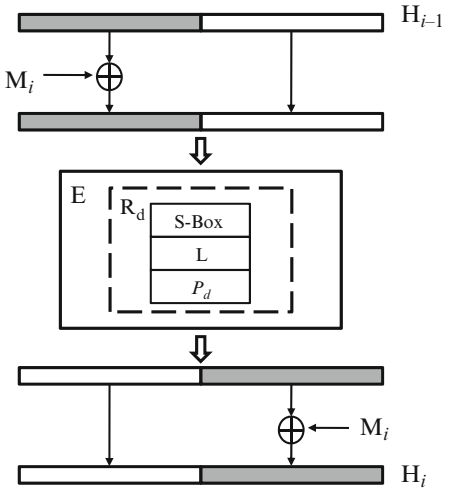


Fig. 2.10 Diagram of the compression function in JH

In JH-512, each message block has 64 bytes and passes through the 35.5 round compression function that involves 9216 4×4 bit S-Boxes. The large number of active S-Boxes ensures that JH is strong against differential attacks. As the key of the block cipher is constant, no extra variables are introduced into the middle of the compression function. Therefore, it is much easier to analyze the security with respect to differential attacks. As the block cipher outputs do not need to be truncated, the structure is also efficient.

2.3.6 Performance

Before we compare the performance of the candidate algorithms, we first give an overview of major operations in each algorithm. Table 2.5 summarizes the major operations in five algorithms in the final round of the SHA-3 competition. Most operations used in the five algorithms are simple. When implemented in hardware, fixed permutations and rotations with fixed amounts can be done with careful wire routing. S-Box can be implemented with logic operations. The more expensive operations include matrix multiplication in Grøstl and addition in BLAKE and Skein.

In addition to security, performance is one of the main criteria for evaluating the SHA-3 candidates. However, it is very challenging to compare all SHA-3 candidates due to various choices of technologies and multiple optimization goals. Most performance evaluation in literature is done on one to three algorithms at a time and cannot be adequately compared with each other due to difference in implementation and measurements.

Table 2.5 Major operations in five SHA-3 algorithms

Algorithm	Block size	No. of round	Major operations in a round
Keccak-1600	1,024 bits	24	$\oplus$, $\lll$, $\neg$, $\wedge$ on 64-bit words
BLAKE-512	1,024 bits	16	$+$, $\oplus$, $\ggg$ on 64-bit words
Grøstl-512	1,024 bits	16	$\oplus$, 8-bit S-Box, ShiftRow, matrix multiplication. Mainly 8-bit operations
Skein-512	512 bits	72	$+$, $\lll$, $\oplus$ operations on 64-bit words. 1 permutation of eight 64-bit words
JH-512	512 bits	35	$\oplus$, 4-bit S-Box, and a permutation of 256 4-bit elements

When implemented with hardware, fixed permutations, $\lll$, and $\ggg$ can be done with wire routing.

Table 2.6 Performance of five final SHA-3 candidates on FPGA [41]

Algorithm	Area (slices)	Max. frequency (MHz)	Throughput (Mbps)	Throughput/area (Mbps/slice)
SHA-2–256	656	125	985	0.966
SHA-2–512	1,213	110	1,264	0.713
Keccak-256	1,117	189	6,263	3.17
Keccak-512	1,117	189	8,518	4.32
BLAKE-32	1,118	118	1,169	0.707
BLAKE-64	1,718	91	1,299	0.449
Grøstl-256	2,391	101	3,242	1.257
Grøstl-512	4,845	123	3,619	0.799
Skein-512	1,786	84	1,945	0.706
JH	1,291	250	1,941	1.1

The FGPA implementations of the algorithms are compared in [41]. The results are summarized in Table 2.6. To fairly analyze the designs and variants, all designs were implemented in slice logic on a Virtex-5 FPGA. Message padding for all designs were included as part of the hardware, and the test was for message digest of size 224, 256, 384, and 512. The efficiency of the architecture is also compared in terms of throughput per unit area. According to the results in [41], Keccak has higher throughput while requiring smaller area.

Reference [42] compared the ASIC implementations of round-two candidates. The results are summarized in Table 2.7. The implementations are optimized for maximum peak throughput, with consideration for reasonable area. If the throughput can only be increased a few percent further at the cost of increase in area of several dozen percent, the slightly lower throughput with more area-efficient implementation is considered. The important implementation decisions in [42] for each algorithm are listed later.

- Keccak: One Kekkak-f round per cycle.
- BLAKE: Four G functions are implemented in parallel; two pipeline registers; additional cycle for chaining; carry-save adders.

Table 2.7 Comparison of five SHA-3 candidates with SHA-256 in [42][a]

Algorithm	Latency (cycles)	Area (GE)	Clock frequency (MHz)	Throughput (Gbps)
BLAKE-32	22	38,877	144.15	3.355
JH-256	39	51,212	259.54	3.407
Keccak-256	25	56,713	267.09	11.624
Grøstl-256	22	53,680	202.47	4.712
Skein-256	10	47,678	64.75	1.658
Skein-512	10	76,250	43.49	2.227
SHA-256	66	19,515	211.37	1.640

[a]Use the UMC 0.18 μm FSA0A_C standard-cell library

Table 2.8 The best performance of five SHA-3 candidates and SHA-256

Algorithm	Technology	Area	Frequency	Throughput (Gbps)
Keccak-1600[43]	ASIC (0.13 μm)	48 kGE	526 MHz	22
Keccak-1600[43]	FPGA Stratix III	4,684 ALUT	206 MHz	8.5
BLAKE-512 [44]	ASIC (90 nm)	79 kGE	532 MHz	18.8
BLAKE-512 [45]	FPGA Virtex 5	108 slices	358 MHz	0.3
Grøstl-256 [46]	ASIC (0.18 μm)	59 kGE		6.3
Grøstl-512 [47]	FPGA Virtex 5	19k slices	84 MHz	6.1
Skein-512 [39]	ASIC (32 nm)	61 kGE	1.13 GHz	58
Skein-512 [48]	FPGA Virtex 5			0.82
JH-256 [42]	ASIC(0.18 μm)	51 kGE	260 MHz	3.4
JH-512 [41]	FPGA Virtex 5	1.7k slices	144 MHz	1.9
SHA-256 [26]	ASIC (0.13 μm)		∼1 GHz	>6.5

- Grøstl: Shared P/Q permutation; S-Boxes and MixBytes separated; S-Boxes with one pipeline register.
- Skein: Eight Threefish rounds unrolled; generic adders.
- JH: 320 S-Boxes (one cycle per R8 round); combinational S-Boxes.
- SHA-2: No unrolling or quasi-pipelining; generic adders.

Table 2.8 shows the best performance of the five algorithms reported in literature, in terms of throughput. Skein has the highest throughput for ASIC, followed by Keccak. Keccak also has the highest throughput for FPGA implementations, followed by Grøstl. BLAKE and Skein have good performance in ASIC implementations, but poor performance on FPGA. The slow adders on FPGA may cause the performance degradation.

As the final SHA-3 algorithm has not been decided yet, the algorithm may still be tweaked and improved. For example, Keccak went through minor revision during each round of the competition. BLAKE went through minor revision during round 2 of the competition. The performance of BLAKE shown here is for the version before revision. There are no changes in the other three algorithms so far. On the other hand, as the competition is already in the final round, no major changes are expected from all candidates. Therefore, the impact of potential changes on the implementation should be small.

From Table 2.8, we can also see that four out of five SHA-3 candidates in the final rounds have an implementation that has higher throughput than SHA-256. Therefore, SHA-3 is expected to be more secure and, at the same time, have higher throughput than SHA-2.

References

1. Rivest R (1992) The MD5 message-digest algorithm. In: The Internet Engineering Task Force (ITEF) Internet Draft, no. RFC-1321, April 1992
2. National Institute of Standards and Technology (1994) Secure hash standard. In: Federal Information Processing Standards Publication 180–1, April 1994
3. Menezes A, Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography, 1st edn. CRC Press, West Palm Beach, FL, USA
4. Damgard I (1990) A design principle for hash functions. In: Proceedings of Cryptology, Crypto '89, vol 435, pp 416–427
5. Wang X, Feng D, Lai X, Yu H (2004) Collisions for hash functions: MD4, MD5, HAVAL-128 and RIPEMD. http://eprint.iacr.org/2004/199.pdf. Accessed August 2004
6. Wang X, Yu H, Yin YL (2005) Efficient collision search attacks on SHA-0. In: Advances in Cryptology – CRYPTO'05, vol 3621, pp 1–16
7. Wang X, Yin YL, Yu H (2005) Finding collisions in the full SHA-1. In: Advances in Cryptology – CRYPTO'05, vol 3621, pp 17–36
8. Wang X, Hongbo Y (2005) How to break MD5 and other hash functions. In: Advances in Cryptology EUROCRYPT 2005, pp 19–35
9. National Institute of Standard and Technology (2007) Cryptographic hash algorithm competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html. Accessed November 2007
10. Matyas SM, Meyer CH, Oseas J (1985) Generating strong one-way functions with cryptographic algorithm. IBM Tech Disclosure Bull 27(10A): 5658–5659
11. Preneel B, Govaerts R, Vandewalle J (1989) Cryptographically secure hash functions: an overview. In: ESAT Internal Report, K. U. Leuven
12. Miyaguchi S, Iwata M, Ohta K (1989) New 128-bit hash function. In: Proceedings 4th International Joint Workshop on Computer Communications, pp 279–288
13. Barreto PSLM, Rijmen V (2000) The Whirpool hash function. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html. Accessed November 2000
14. Nakajima J, Matsui M (2002) Performance analysis and parallel implementation of dedicated hash functions. In: Proceedings of EUROCRYPT 2002, Lecture Notes in Computer Science, vol 2332, pp 165–180
15. Lloyd B et al. (1992) PPP authentication protocols. In: The Internet Engineering Task Force (ITEF) Internet Draft, RFC-1334, October 1992
16. Simpson W (1994) The point-to-point protocol. In: The Internet Engineering Task Force (ITEF) Internet Draft, RFC-1661, July 1994
17. National Institute of Standards and Technology (2002) The keyed-hash message authentication code (HMAC). In: FIPS PUB, vol 198
18. Hoang AT, Yamazaki K, Oyanagi S (2008) Multi-stage pipelining MD5 implementations on FPGA with data forwarding. In: 16th International Symposium on Field-Programmable Custom Computing Machines, pp 271–272, April 2008
19. Wang Y, Zhao Q, Jiang L, Yi S (2010) Ultra high throughput implementations for MD5 hash algorithm on FPGA. In: High Performance Computing and Applications, pp 433–441
20. Chaves R, Kuzmanov G, Sousa L, Vassiliadis S (2006) Improving SHA-2 hardware implementations. In: Cryptographic Hardware and Embedded Systems-CHES 2006, pp 298–310

21. Jarvinen K, Tommiska M, Skytta J (2005) Hardware implementation analysis of the MD5 hash algorithm. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, vol 9, p 298a
22. Lee YK, Chan H, Verbauwhede I (2007) Iteration bound analysis and throughput optimum architecture of SHA-256 (384,512) for hardware implementations. In: Proceedings of the 8th international conference on Information security applications, vol 256, pp 102–114
23. Lien R, Grembowski T, Gaj K (2004) A 1 Gbit/s partially unrolled architecture of hash functions SHA-1 and SHA-512. In: Topics in Cryptologyâ CT-RSA 2004, pp 1995–1995
24. Crowe F, Daly A, Kerins T, Marnane W (2005) Single-chip FPGA implementation of a cryptographic co-processor. In: Proceedings. 2004 IEEE International Conference on Field-Programmable Technology (IEEE Cat. No.04EX921), pp 279–285
25. Athanasiou G, Gregoriades A, Panagiotou L, Goutis C, Michail H (2010) High throughput hardware/software co-design approach for SHA-256 hashing cryptographic module in IPSec/IPv6. Global J Comput Sci Technol 10(4): 54–59
26. Dadda L, Macchetti M, Owen J (2004) An ASIC design for a high speed implementation of the hash function SHA-256 (384, 512). In: ACM Great Lakes Symposium on VLSI, pp 421–425
27. Dadda L, Macchetti M, Owen J (2004) The design of a high speed ASIC unit for the hash function SHA-256 (384, 512). In: Proceedings Design, Automation and Test in Europe Conference and Exhibition, vol 256, pp 70–75
28. Feldhofer M, Wolkerstorfer J (2007) Strong crypto for RFID tags – a comparison of low-power hardware implementations. In: 2007 IEEE International Symposium on Circuits and Systems, pp 1839–1842, May 2007
29. Peris-Lopez P, Hernandez-Castro J, Tapiador J, Ribagorda A (2009) Advances in ultra-lightweight cryptography for low-cost RFID tags: Gossamer protocol. Inform Security Appl 56–68
30. Kim M, Ryou J, Jun S (2009) Efficient hardware architecture of SHA-256 algorithm for trusted mobile computing. Architecture. Springer Verlag, Berlin, Heidelberg, New York, pp 240–252
31. Perlner R, Chang S, Kelsey J, Nandi M, Paul S, Regenscheid A (2009) Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition. September 2009
32. Bertoni G, Daemen J, Peeters M, Assche GV (2009) Keccak specifications Version 2. http://keccak.noekeon.org/Keccak-specifications-2.pdf. Accessed July 2011
33. Morawiecki P, Srebrny M (2010) A SAT-based Preimage Analysis of Reduced KECCAK Hash Functions. Santa Barbara, CA, 23–24 August 2010
34. Bertoni G, Daemen J, Peeters M, Assche GV (2010) Keccak sponge function family main document. http://keccak.noekeon.org/Keccak-main-2.1.pdf. Accessed June 2010
35. Biham E, Dunkelman O (2006) A framework for iterative hash functions: HAIFA. In: Second NIST Cryptographic Hash Workshop
36. Henzen L, Meier W, Raphael C-W, Phan, Aumasson J-P (2009) SHA3 Proposal BLAKE. 7 May 2009
37. Knudsen LR, Matusiewicz K, Mendel F, Rechberger C, Schlaffer M, Søren S, Gauravaram TP (2008) Grøstl – a SHA-3 Candidate
38. Lucks S, Schneier B, Whiting D, Bellare M, Kohno T, Callas J, Ferguson JWN (2008) The Skein Hash Function Family
39. Sheikh F, Mathew SK, Walker RKJ (2010) A Skein-512 hardware implementation. http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/presentations/WALKER_skein-intel-hwd-slides.pdf. Accessed August 2010
40. Wu H (2009) The Hash Function JH. http://www3.ntu.edu.sg/home/wuhj/research/jh/. Accessed July 2011
41. Hanley N, Hamilton M, Lu L, Byrne A, O'Neill M, William P, Baldwin MB (2010) FPGA Implementations of the Round Two SHA-3 Candidates, August 2010
42. Feldhofer M, Kirschbaum M, Plos T, Schmidt J-M, Tillich ASS (2010) Uniform evaluation of hardware implementations of the round-two SHA-3 candidates. In: The Second SHA-3 Candidate Conference

43. Bertoni G, Daemen J, Peeters M, Assche GV (2010) The Keccak sponge function family: hardware performance. http://keccak.noekeon.org/hw_performance.html. Accessed November 2010
44. Henzen L, Aumasson J-P, Meier W, Phan R VLSI Characterization of the Cryptographic Hash Function BLAKE. http://www.131002.net/data/papers/HAMP10.pdf. Accessed July 2011
45. Beuchat J-L, Okamoto E, Yamazaki T (2010) Compact Implementations of BLAKE-32 and BLAKE-64 on FPGA
46. Grøstl – a SHA-3 candidate. http://www.groestl.info/implementations.html. Accessed July 2011
47. Baldwin B, Byrne A, Hamilton M et al. (2009) FPGA Implementations of SHA-3 Candidates: CubeHash, Grøstl, LANE, Shabal and Spectral Hash. http://eprint.iacr.org/2009/342.pdf. Accessed July 2011
48. Long M (2009) Implementing Skein Hash Function on Xilinx Virtex-5 FPGA. http://www.schneier.com/skein_fpga.pdf. Accessed February 2009

Chapter 3
RSA: Implementation and Security

Nicholas Tuzzio and Mohammad Tehranipoor

3.1 Introduction

The need for secure communication methods between remote parties has existed for as long as remote parties have wished to communicate. The rise of the digital computer in the beginning of the twentieth century brought with it a new paradigm in secure communications – computer-based encryption algorithms. Encryption is a process by which some private set of data, often called a plaintext, is deterministically converted in a seemingly random set of data called a ciphertext. This ciphertext is then later converted deterministically back into the plaintext through a process called decryption. Encryption and decryption can be thought of as the "locking" and "unlocking" of the plaintext. Imagine a plaintext as a piece of paper with sensitive information written on it. By placing this piece of paper in an unbreakable box with an unpickable lock, we have "encrypted" this data in such a way that only the person or persons with the right key can "decrypt" the data. There are two main ways in which two parties A and B could transmit data with this ideal box/lock combination. First, party A could give party B a copy of their unpickable lock's key during some secure, personal meeting between the two parties. At a later date, party A could "encrypt" the data using their box and lock and then send the box to party B, who, having obtained the key to this lock at the prior meeting would be able to "decrypt" the data upon receiving it. In this example, both parties could continue sending data back and forth to each other while using the same key. This type of encryption is called "symmetric" encryption – both parties use the same key.

There are some concerns involved with symmetric encryption – for example, it is vital that the key is considered "secret," and is never obtained by any parties besides those who will be sending or receiving the messages. There is another way

N. Tuzzio · M. Tehranipoor (✉)
UCONN Electrical and Computer Engineering, University of Connecticut, 371 Fairfield Way, Unit 2157, Storrs, CT 06269-2157, USA
e-mail: nicholas.tuzzio@engr.uconn.edu; tehrani@engr.uconn.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_3,

in which parties A and B could transmit data securely. In this second way, the two parties A and B both have their own lock and their own key. However, the key is not exchanged between the two parties. Instead, parties A and B swap locks with each other. Whenever party A wishes to send party B a secure message, party Aq can place their data in their unbreakable box and lock the box uses party B's key. By doing this, party A has ensured that only party B will be able to retrieve that data – not even party A can reopen that box. party B can go through the same process using party A's lock to send party A data. In this scenario, the two parties are using different keys to perform the encryption, and this type of encryption is called "asymmetric" encryption. The RSA algorithm is an asymmetric encryption algorithm which allows secure communications between parties. For a party to use RSA to secure communications, they must generate two keys – a public key and a private key. In the above box/lock analogy, the public key takes the place of the locks which are exchanged between the parties, and the private key takes the place of the key which is used to open the lock. If party A wishes to retrieve some private information from party B, party A must first send party B a copy of their public key. party B can then use that public key to encrypt the data and then sends the encrypted data back to party A, who can then decrypt the data using their private key. It is worth noting that, as in symmetric encryption, the private key must still be kept a secret from all parties who should not be privy to the encrypted information, and if a party's private key is compromised they must distribute a new public key to keep communications secure. However, the need for an initial secure key exchange has been reduced (not removed, which we will show later).

The encryption algorithm RSA is well-known and widely used, because of its simple mathematical description and proven security. Because of its widespread adoption, implementations of RSA exist in a variety of hardware- and software-related forms. It is worth noting that while a mathematical description of an algorithm can be provably secure, actual implementations of algorithms cannot be considered to be perfect, and for this reason it is worth analyzing the ways that algorithms are implemented to ensure that the security that comes with the use of an algorithm is not lost during the implementation stage. In this chapter, we will analyze the RSA algorithm and its security in relation to hardware implementation. First, the RSA algorithm will be described in detail, with the benefits and downfalls of the algorithm analyzed with respect to security. Second, the basics of hardware implementations of RSA will be considered. Third, security considerations and concerns related to implementing RSA in hardware will be examined. Finally, the long-term future of the RSA algorithm in light of all of these security concerns will be explored.

3.2 Algorithm Description and Analysis

The RSA algorithm was first publicly described in the late 1970s by Rivest et al. The algorithm was quickly patented with the help of MIT, and the algorithm was released into the public domain by the patent holders two weeks before the patent

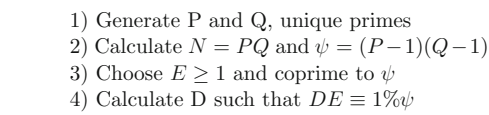


Fig. 3.1 RSA key generation algorithm [6]

was due to expire in 2000. As the algorithm was released into the public domain, hardware and software implementations of the algorithm have been developed in vast quantities. Considering both the longevity and ubiquity of RSA in regards to modern cryptography, it is worth examining the algorithm so that a full understand of the strengths and weaknesses of RSA are known to all who intend to use it.

As stated previously, RSA requires each party involved in the communications to create two keys – a public key for other parties to use for encryption, and a private key for the receiving party to decrypt that information. The public key is distributed while the private key is kept secret. RSA's security hinges on the fact that it is computationally infeasible to compute the private key from information contained in the public key. To understand why this is, we must first examine the RSA key generation algorithm. The first step involved in this algorithm is the generation of two large, unique, random prime numbers. These two primes, which we will refer to as P and Q, are multiplied to create an even larger number, which we will refer to as N. The two primes P and Q are also used to create another number, referred to as ψ, which is computable as $(P-1)(Q-1)$. We use this number ψ to find an integer E which is subject to the requirements that E be greater than one and less than ψ, while also ensuring that E is coprime to ψ (that is, two numbers share no divisors other than one). The numbers ψ and E are used to find another number, D, which is the integer which satisfies the equation $DE \equiv 1\%\psi$. Once the numbers N, E, and D have been computed, the key generation portion of the algorithm will have been completed.The public key is made up of the two numbers N and E, while the private key will be made up of the two numbers N and D (Fig. 3.1).

The encryption and decryption steps of the RSA algorithm use the public and private keys, respectively. To encrypt a plaintext message P, we must first convert it into an integer M. The encrypted version of this integer is computed by using the values N and E from the public key to perform the calculation $C = M^E \% N$, with C being the ciphertext version of the message P. To decrypt this ciphertext, we need the values N and D from the private key to perform the calculation $M = C^D \% N$, and then whatever techniques that were used to convert the plaintext into an integer can be reversed to obtain the original plaintext. One of the most interesting things about the RSA algorithm is its relatively simple mathematical description. The key generation steps of RSA can be described by a single multiplication to obtain N, the fulfillment of two conditionals statements to obtain E, and the fulfillment of one additional conditional statement to obtain D. The encryption and decryption steps can be described by a single mathematical operation – exponentiation, using different values for encryption and decryption. Due to this simple description, RSA may seem like an easy algorithm to implement and use. One must examine the steps of RSA a bit more closely to truly understand the requirements and difficulties facing any implementation of the algorithm.

The first step in the key generation process is perhaps the most difficult and most important step in using RSA. In this step, we generate two unique, large, random prime numbers. These numbers are then used to calculate the modulus, N, which is present in the public and private keys. The security that RSA provides is created through the fact that it is very difficult to calculate the two large random primes even with knowledge of their product. In theory, a person who obtains a public key has all of the information that they should need to calculate the private key – all that is required to calculate the private key would be for the interested party to factor the modulus N from the public key into its two composite primes. However, the "integer factorization problem" renders this infeasible. Currently, no algorithm exists which can factor large integers into their composite primes in polynomial time; all known algorithms run in exponential or subexponential time. This does not mean that large integers cannot be factored; they simply take an exponentially increasing amount of time to factor with respect to the size of the number being factored. Thus, RSA is not inherently uncrackable. To use RSA securely, the modulus must be chosen such that it would be unrealistic for any attacker to factor the modulus in some period of time. The definition of "secure usage of RSA" has changed over time because of increases in computing power available to attackers; in 2010, a standard modulus size is the 1024-bit product of two 512-bit prime numbers. However, 1024-bit moduli may not be the standard for long as a 768-bit modulus was factored using around 2,000 h of computation time on an economy CPU in 2009 and 1024-bit moduli would be the next major factoring milestone. Currently, 2048-bit and 4096-bit moduli are also seen in common usages of RSA and will not be realistically factorizable for some time.

The difficulty in factoring the public modulus N into its component primes P and Q is what creates the security in RSA – the problem of finding the private exponent D from the public modulus N is as difficult as finding the component primes P and Q from the public modulus N. However, this security comes with a cost – there are strict requirements on which P and Q can be chosen to form the modulus N. The integers P and Q must be large, unique, prime, and random. Each of these requirements is itself a benefit to the security of the modulus and a cost to the difficulty of the algorithm. P and Q must both be large numbers – to generate a 1024-bit modulus; we must multiply two 512-bit numbers. While generating, storing, and performing operations on numbers this size can be difficult from a resource perspective in hardware or software, the size of the modulus is proportional to the security of the encryption. The two numbers must also be unique from each other. Were P and Q chosen such that they were equal, an attacker could potentially recognize the modulus as a perfect square and be able to factor it in significantly less time. The primality of the two numbers is perhaps the most important criteria and the most difficult to fulfill. If the integers P and Q are not prime, that means that the modulus will have more prime factors than just P and Q, and for a given modulus size these prime factors will be smaller than they would have been if P and Q were prime. Increasing the number of prime factors, as well as decreasing the size of the prime factors, will make factoring the modulus much easier. However, the generation of prime numbers is not simple. A random number b has approximately

a $\frac{2}{\ln b}$ chance of being prime. For numbers around the size of 2^{512}, about 1 in every 200 numbers will be prime. It is not the generation of prime numbers that is the problem – given enough time with a random number generator, we will eventually obtain a prime number. The difficulty lays in the verification of the prime number. The most basic way to check if an integer P is prime is to check if it is divisible by every integer from 1 to $\sqrt{P}$. However, polynomial time algorithms do exist for primality testing. Often, when random primes are generated in software, they are checked for primality for some amount of time by a nondeterministic primality algorithm and once they are verified as prime with some high probability they are run through a deterministic algorithm for a final verification.

Without question, the generation of the random primes to create the public modulus is a difficult and important step of RSA. The quality of the results obtained in this step directly impact the security of the encryption offered by the algorithm. From an algorithmic standpoint, however, this is not the only step that is difficult to perform in actual implementations of the algorithm. Primality tests and random number generation are both difficult mathematical problems that have been performed on a variety of software and hardware platforms and there exist a large number of ways to do either operation. Other operations required by the algorithm may not be as fundamentally difficult to perform, but they still have their own costs and requirements. For example, the two steps which occur immediately after the primes P and Q have been generated are to obtain the integers N and ψ, which are calculated as PQ and $(P-1)(Q-1)$, respectively. Multiplication is an operation which is trivial to write into an algorithm, and nontrivial to compute using digital technology. In hardware and software, many different systems and routines implement multiplication with a variety of time and resource requirements. Multiplication does not just occur in this step of the RSA process, either. In the encryption and decryption algorithms of RSA, modular exponentiation is required. Exponentiation can be thought of as repeated multiplication in the same way that multiplication can be thought of as repeated addition. The time and resource requirements of any system which will be creating RSA keys or performing RSA encryption and decryption will be directly fimpacted by the choice of multiplication systems incorporated into their design, and later we will examine some of the time, resource, and security requirements of some basic hardware multiplier design choices.

We have analyzed some of the more important aspects of RSA key generation, encryption, and decryption. While the algorithms themselves are very important, the understandings mentioned here on random prime generation and multiplier design are important initial design considerations for any RSA implementations. Additionally, it is easier to understand the type of security offered by RSA when considering the implications of the requirements of the algorithm. RSA is not a perfect encryption algorithm – given enough resources, any data encrypted by RSA can be decrypted. It is the difficulty involved with that process that is the heart of RSA's security.

3.3 Hardware Implemention Introduction

RSA is very simple to understand and explain from an algorithmic standpoint. Ignoring the process of finding random prime numbers, basic RSA usage requires only a system which is capable of performing modular exponentiation on very large numbers. Exponentiation itself can be thought of as a repeated multiplication operation, and it is for this reason that we will discuss the hardware multiplier as the core of any hardware implementation of RSA. There are multiple ways to design a hardware multiplier and to insert it into a system, so we will examine the ways in which several different multiplier styles could be used in an RSA system.

Multiplication itself can be thought of as a process of repeated addition. The process of hardware addition is a subject which has been researched intensively. From the simple full-adder we have constructed ripple-carry adders, carry-lookahead adders, and overall a large suite of hardware constructs for addition which encompass a large range of time and area requirements (Fig. 3.2).

The construction and choice of these adders is just as complex a topic as the construction and choice of multipliers, so we will only consider a generic

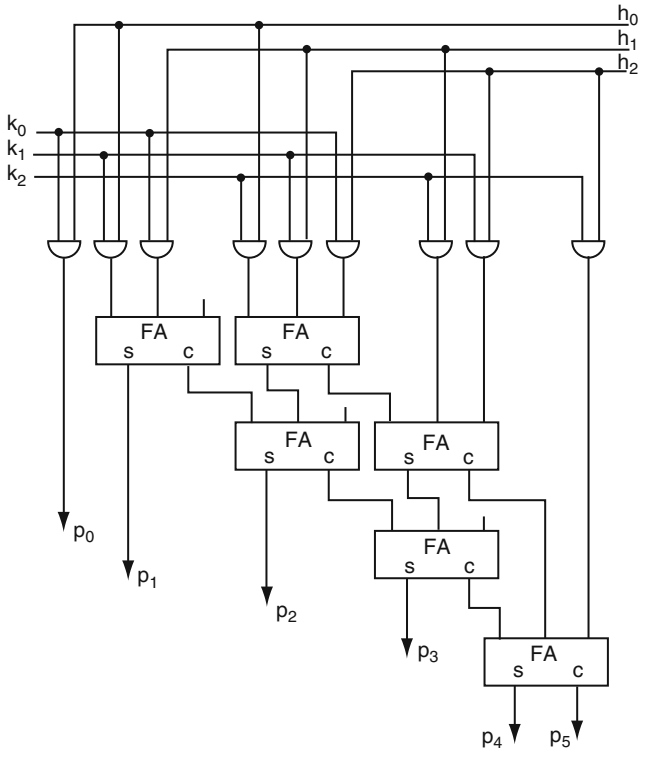


Fig. 3.2 A 3-bit combinational multiplier [2]

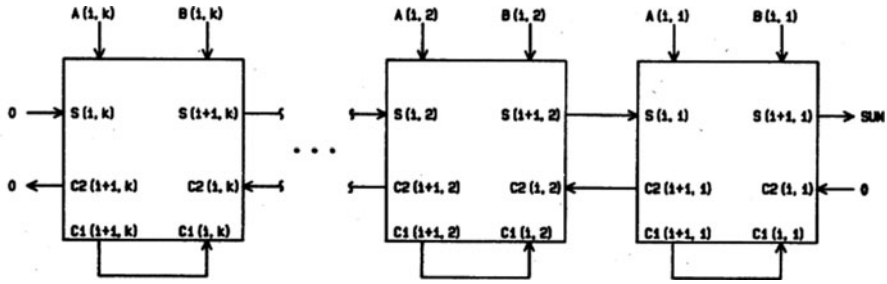


Fig. 3.3 Connections of 5-3 compressors in a bit-serial multiplier [7]

combinational adder in our systems. Addition of two n-bit numbers requires an n-bit adder, which produces an n-bit result with carry-out (or an $(n + 1)$-bit result). We can multiply two n-bit numbers by using n n-bit adders, or alternatively by using n^2 full adders.The combinational multiplier is made up of an n-by-n matrix of full adders. While there are several ways that the adders in the circuit can be connected together to result in a multiplication circuit, the general mathematical concept is the same: each row or column of the multiplier will compute one of n partial products – intermediate stages in the multiplication. Each partial product is similar to one of the values that are summed in a traditional long-hand method of paper multiplication. This process is much easier to understand in binary than in decimal format – each row or column i of the multiplier will compute the value AB_i and provide this value as a carry-input to the row or column i+1 that is computing the partial product AB_{i+1}. The least significant bit of every row or column i will be sent the output as the ith output of the combinational multiplier, and each output bit of the final row or column will be used as upper-bits of the product.

This simple combinational multiplier has quadratic area requirements (requiring n^2 full adders to multiply two n-bit numbers) but finds the product of the two numbers in constant time. The ability to find the product of the two numbers in constant time comes at a cost – in hardware, the number of full adders through which each bit of the product has to travel is great. This results in incredibly long in-circuit paths, meaning that the "constant" time required to calculate the product could be infeasibly long for a particular circuit. In the end, even this "constant" time is a rather slow way to compute the product of two numbers, and it comes at the cost of quadratic area requirements. The combinational multiplier is rarely used in practice and is quite obviously inferior to the serial and bit-serial methods introduced next (Fig. 3.3).

The combinatorial multiplier discussed previously can be thought of as a full-parallel multiplier; that is, both the multiplicand and the multiplier are provided as inputs in full to the combinatorial circuit. The combinatorial multiplier which can multiply two n-bit operands requires $2n$ inputs and $2n + 1$ outputs, but also does not require any sequential circuitry. The bit-serial multiplier is a hardware multiplier which takes advantage of sequential hardware structures and a clock signal to

multiply two numbers using only two inputs and one output [1]. The two inputs are provided to the circuit one bit at a time, typically in order of least significant bit to most significant bit, in time with a clock signal. The bit-serial multiplier functions as a complicated shifting accumulator circuit.

During each clock cycle of the multiplication process, a partial product of the multiplication (an intermediate step) is added to an overall running sum, and the least significant bit of that running sum is shifted out of the multiplier as the circuit's output. An example bit-serial multiplier could be constructed out of cells which consist of several edge-triggered flip-flops and a 5-3 compressor module (a circuit which compresses five equally weighted inputs into three weighted outputs). An n-bit multiplier would require n of these cells, and could multiply two n-bit operands and produce the output serially in $2n + 1$ clock cycles. The main benefit of the bit-serial multiplier over the combinatorial multiplier is a great reduction in area – while the combinatorial multiplier increases quadratically in area and linearly in pin count with respect to rising operand lengths, the bit-serial multiplier described here only areas linearly in area with respect to operand lengths and requires only three pins regardless of the operand size.

In the bit-serial multiplier, the accumulator is an idea – there is no dedicated accumulator block and the product of the numbers is constantly moving and shifting through the various cells of the multiplier. There exists a multiplier between the combinatorial and serial which can be designed using a dedicated traditional accumulator. In this design, one of the operands is provided in parallel and one of the operands is provided in serial. The parallel operand must be held constant for the entire multiplication process. The serial operand is provided in order from least significant bit to most significant bit. During each cycle of the multiplication process, the incoming serial bit is examined. If the incoming bit is a one, then the value of the parallel operand is added to an initially zeroed accumulator. If the incoming bit is a zero, no addition is performed. After the addition has been performed or ignored, the least significant bit of the accumulator's value is shifted out of the multiplier as an output bit. This process then repeats until all n input bits have been loaded into the circuit, and then until all $2n + 1$ bits of the product have been shifted out of the accumulator. This design has some benefits relative to the other two multiplier designs previously discussed.

First, this hybrid parallel/serial design has a much less complicated design than the other two multipliers. All that is required for this multiplier is a comparator which determines whether an addition should occur, and an n-bit adder which uses its own output as one of its two inputs. This design can be made to use a linear amount of area with respect to the operand length (like the bit-serial multiplier) but also requires a linear increase in pin count with respect to the operand length (like the combinatorial multiplier). This hybrid design works best when one operand of the multiplication will be constant – when we know that we will always want to multiply some number by the same amount. In this scenario, the constant operand could be hard-wired as the parallel input. Knowing that one operand will be constant may allow some reductions in area to occur as well, as some of the multiplier's internal connections will be optimizable during the design process (Fig. 3.4).

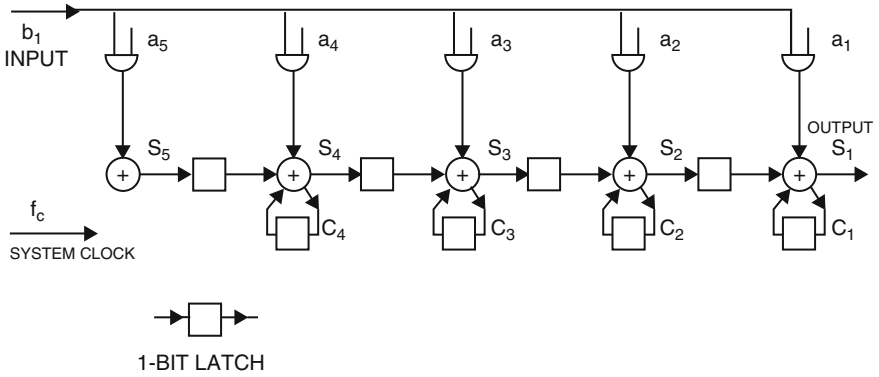


Fig. 3.4 Connections in a serial-parallel multiplier [3]

Of course, these three theoretical designs are only three of the most basic hardware multipliers possible. In an actual integrated circuit, the hardware multiplier used will have optimizations heaped on top of optimizations designed for some optimized variant of the above. Additionally, there are mathematical techniques like Montgomery reduction and Karatsuba multiplication which can reduce the number of additions required to compute a multiplication by some factor. These techniques and differences between real and theoretical multipliers greatly affect the time, area, and complexity requirements of a circuit, but are mostly irrelevant when discussing the security concerns of the hardware multiplier in the RSA system.

As previously stated, the modular exponentiation operation is the most central operation used by an RSA system, and so an understanding of this operation's implementation is important to anyone who would wish to implement RSA. Modular exponentiation itself can be reduced to a process of repeated multiplications, and so the necessary hardware multiplier in an RSA system makes a good basic unit for security analysis. We will use the three previously discussed multiplier designs as examples while discussing some thoughts on the security of an RSA system in hardware.

3.4 Security Analysis

RSA is a complicated cryptographic algorithm and any system which produces RSA public/private key pairs or calculates RSA encryptions/decryptions in hardware will be a complicated system. We have examined, in depth, the algorithms which produce RSA keys and encrypted messages using those keys. We have also examined, and again in depth, the design choices which can be made related to the most basic and important unit in a hardware implementation of RSA, the hardware multiplier. By taking into consideration the requirements of the algorithm and the constraints of

the functional units used to implement the algorithm, we can begin to examine the security implications and considerations that need to be made to create a truly secure hardware implementation of RSA.

However, to be able to discuss the hardware security of a hardware implementation of a cryptographic algorithm, we must examine what it means for such an implementation to be secure. First, a hardware implementation of a cryptographic algorithm must produce the correct output given the proper input. This may seem obvious but in a system as complicated as an RSA encryption/decryption module, small errors in the design or fabrication process can create small errors which could potentially alter the output, rendering the encrypted plaintext undecipherable. Any design choices which can potentially alter the security of the algorithm itself, such as the length of the private and public keys, must be chosen in such a way that hardware requirements are met while also providing security to those who use the system. The system should also be robust to known types of attacks, whether they have been used in practice or are largely theoretical. Keeping these general ideas in mind, let us begin to analyze some of RSA's unique security requirements.

Let us assume we have a hardware system which can perform RSA key generation, encryption, and decryption. This system would implement the algorithms described at the beginning of this chapter. However, from a security standpoint, the implementation of these algorithms is incredibly nontrivial. The first step of the first algorithm discussed – the public/private key pair generation algorithm – is to generate two large, random, unique prime numbers. Each one of the words "large," "random," "unique," and "prime" creates its own implementational issues. First, the random numbers generated need to be suitably large. The security of RSA encryption comes from the difficulty involved in factoring large semiprime numbers into their prime components. The difficulty of factoring semiprimes increases exponentially with linear increases in the length of the prime factors (for now, at least, based on current algorithms). Integer factorization is a field of constant research and progress, and as such larger and larger integer factorizations have been done over time. For this reason, suitably large keys with suitably large prime factors must be used, or else it will be trivial for an attacker to crack an encrypted message. As of 2010, private key lengths of 2048 or 4096 bits are commonly used and considered to be secure. Secondly, random number generation is a difficult process, and in RSA it is important to get it right – any deficiencies discovered in an implementation of the RSA key generation system's random number generator could potentially result in an alteration of the possible key space. For example, consider an attacker who wishes to decrypt an RSA-encrypted message without the private key (we will consider this the generic idea of an "attack" on an RSA system). If this attacker were to discover that there were a deficiency in the random number generation unit of the system – perhaps the random number generator produces a biased output, or does not produce numbers from its full range – then that attacker would be able to use the knowledge of this deficiency to greatly reduce the keyspace that the private key could possibly come from. The reduction of this keyspace results in a possible reduction in the amount of time it would take to crack an encrypted message. Additionally, these two prime numbers must also be unique. If the two large random primes are identical, their product will be easily recognizable as a

perfect square and thus the public key will be easily factorizable. This is true to a degree even if the numbers are merely close to each other; also, if the numbers are too far apart from each other, resulting in one being much larger than the other, the public key is susceptible to some other types of attacks. Finally, these two numbers must be prime – this is possibly the most difficult requirement of the four, because primality testing is a very difficult process. The primality of the two numbers is of utter importance because the product of two nonprimes will have more than two prime factors, greatly reducing the complexity of the factorization for equivalent public key lengths. It is telling that even the first step of the first algorithm needed to perform RSA operations is incredibly complicated and rife with security requirements and implications.

Another security concern for an RSA system, besides the design of the large random unique prime number generator, is the choice of the encryption exponent E. Recall that E is a number which is coprime with the totient ψ and greater than one. The encryption exponent is part of the public key, along with the modulus N, and is used to encrypt a plaintext message M into a ciphertext C by performing the operation $C = M^E \% N$. There are attacks which relate to the choice of E; specifically, these attacks take advantage of a choice of E which is too small, and given certain choices of E it becomes much easier for an attacker to convert a ciphertext back into a plaintext. For this reason and others, most RSA systems use a common encryption exponent – the Fermat Prime $2^{16} + 1$, or 65537 [5]. It is worth noting that, outside of a process which deficiently generates the random prime numbers that form the modulus or a choice of an inappropriately small encryption exponent, there are practically no direct attacks which can be made on an RSA system from an algorithmic standpoint. RSA, proposed in 1979, remains to this day an algorithmically strong encryption process. Unfortunately, algorithmic security is not the only quality an encryption system must provide – the implementation itself must be secure. New types of attacks have been proposed that focus on specific implementations of RSA and other encryption algorithms, and new types of defenses to defeat those general attacks have been proposed.

Consider the RSA system which we have been discussing. Let us also consider an attacker who is able to intercept messages encrypted by this system. If the RSA system has been designed correctly – it meets the security requirements for the algorithm described earlier, and more – it should be infeasibly difficult for the attacker to factor the public key used to encrypt the message, which is what it would take to decrypt the message. However, while the algorithm may be suitably secure, this does not mean that the hardware implementation itself is secure. Let us now consider an attacker who has access to the physical system itself, as well as the output to the system. Various researchers have proposed a set of hardware attacks called "side-channel" attacks. These attacks do not directly attack the algorithm, but rather focus on the actual physical implementation of the algorithm. A side-channel attack is one which takes advantage of unintended leaks of information through nontraditional channels. The two most common channels in which attackers commonly look for leaked information are in the timing activity of the hardware and the power usage of the hardware. The techniques used to analyze these two channels are similar in concept and do not even always require physical access to the system.

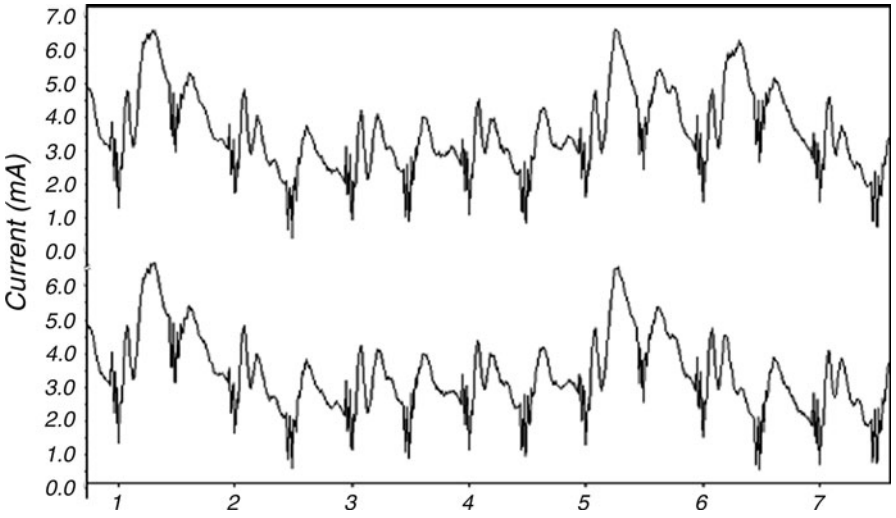


Fig. 3.5 Differential power analysis – the differences in cycle 6 can be used to obtain operational information [4]

To see how a side-channel attack might work, consider a hardware system designed to compute RSA encryption. This system would receive a plaintext M as an input and use an RSA public key to convert it into a ciphertext C by use of the formula $C = M^E \% N$. An attacker who wished to intercept and decrypt the ciphertext C would have to factor the modulus N, from the public key, in order to calculate the private key and decrypt the message. If the system has been designed well, this should be very difficult to do without additional information. However, consider that the attacker knows this about the system computing this modular exponentiation: that the modular exponentiator uses, at its core, the simple hybrid serial/parallel multiplier described previously. This system would compute the ciphertext by first squaring the plaintext, reducing it by the modulus N, and then repeating that process on the product $E - 1$ times. Perhaps the system has been designed such that the previous product is the operand which is provided in parallel, and the plaintext is the operand which is provided in serial. This would mean that, to compute $C = M^E \% N$, the plaintext M is going to be passed serially to the multiplier e times. In the hybrid multiplier previously discussed, each bit of the serial operand is examined to determine whether or not the parallel operand is going to be added to the accumulator. When the current bit of the serial operand is one, an addition is performed, and no addition is performed when the current bit is zero. This difference is irrelevant from an algorithmic perspective, but the decision to perform or not perform an addition has ramifications on the timing and power usage of the system. For example, perhaps performing the addition requires a longer amount of time than not performing it (Fig. 3.5). Or, perhaps the system uses more power during the clock cycles where it performs the addition than the clock

cycles where it does not. In either of these situations, an attacker who could measure and distinguish these timing or power differences would be measuring differences that are directly related to information which is supposed to be secret. Clearly, this is an incredibly simple example – only tiny changes to this hybrid multiplier would be required to make it more secure (for example, performing an addition no matter what but discarding the result). However, it gives an idea of an entire class of attacks which some never consider. The fact that side-channel information could leak private information to an attacker means that systems designers need to think about security outside of the high-level algorithmic realm.

Of course, the hybrid serial/parallel multiplier isn't the only hardware structure which may leak information, and the encryption operation isn't the only operation in RSA where private information could be leaked. The combinatorial multiplier, with its large area requirements, make it susceptible to power analysis. Given a fine enough resolution on measurements of power usage in certain areas of an integrated circuit (most likely through temperature measurement), one may be able to tell that entire rows or columns of the combinatorial multiplier were not being used in a particular multiplication operation, giving away information about the operands used during that multiplication operation. This type of measurement could be defeated in several ways – perhaps the full adders which make up the combinatorial multiplier could be distributed throughout the integrated circuit, making power analysis of the multiplier more difficult at the cost of decreased circuit speed. However, reverse engineering of the integrated circuit could result in these power measurements becoming viable yet again. The bit-serial multiplier previously discussed uses a large number of flip-flop memory elements in its construction, some of which hold their values throughout operation and some of which change often. Each cell of the bit-serial multiplier has at least two flip-flops which do not change throughout the cells operation, and perhaps information about the values in these flip-flops could be inferred from information related to the timing of a particular cell during a multiplication. These types of attacks are general in nature, because they apply to specific implementations of the algorithms. Two different systems which compute RSA encryption may be susceptible to entirely different sets of attacks because of design decisions made by the designers. Additionally, it is not only the encryption operation which is susceptible to these attacks – again, they are general attacks that depend on the system. Perhaps the multiplier that an RSA system uses to compute the modulus from the two random primes is of a different design than the multiplier used for modular exponentiation during encryption. If the multiplier that computes the modulus is prone to attack, the attacker may be able to easily discover the private prime numbers which make up the modulus, giving the attacker the private key and thus allowing access to all messages encrypted with that public key.

The security of a system which implements the RSA algorithm depends on design decisions involving both the algorithm and the implementation of the algorithm. Different implementations of the algorithm will be susceptible to different types of attacks, with side-channel attacks being an especially powerful tool when the appropriate design decisions have not been made. Side-channel attacks don't always

have to require physical access to the system, either – an especially powerful RSA system might require so much power that the power information to be analyzed might be able to be pulled from the electrical circuit the system is powered by. Or, perhaps the amount of time that a system takes to compute certain values and transmit them over a network could be analyzed to glean insight into the operation of the system. Unfortunately, those who would wish to intercept encrypted messages will always be looking for new ways in which they can take advantage of the design choices of those who wish to encrypt messages.

3.5 Conclusion

Since 1979, RSA has been an incredible tool for those who wish to engage in private communications and store data securely. Today, public-key cryptography and RSA in particular are operations which nearly every computer can perform natively. RSA is a simple algorithm to describe from a mathematical perspective, but underneath those simple algorithms lay a technical complexity which makes implementing RSA a daunting task. Additionally, the constant efforts of those who would flaunt the privacy of others means that new ways of attacking RSA are being developed constantly. Some of these efforts result in small changes to the way the algorithm is used – for example, the recommendation of the use of $2^{16+}1$ as the encryption exponent. Others, like the potential of leaking information over side channels, may require large changes that greatly alter the complexity, area, or power requirements of the system. Defeating those who would attempt to break through the security provided by RSA requires constant thought, preparation, and analysis, so that researchers can ensure the privacy of communications throughout the world.

References

1. Chen I, Willoner R (1979) An 0(n) parallel multiplier with bit-sequential input and output. IEEE Trans Comput C-28(10): 721–727
2. de Mori R (1969) Suggestion for an i.c. fast parallel multiplier. Electr Lett 5(3): 50–51
3. Gnanasekaran R (1985) A fast serial-parallel binary multiplier. IEEE Trans Comput C-34(8): 741–744
4. Kocher P, Jaffe J, Jun B (1998) Differential power analysis. Technical report, Cryptography Research
5. Menezes A, van Oorschot P, Vanstone S (1997) Handbook of Applied Cryptography. CRC Press, West Palm Beach, FL, USA
6. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21: 120–126
7. Strader NR, Rhyne VT (1982) A canonical bit-sequential multiplier. IEEE Trans Comput C-31(8): 791–795

Chapter 4
Security Based on Physical Unclonability and Disorder

Ulrich Rührmair, Srinivas Devadas, and Farinaz Koushanfar

4.1 Introduction

As the number of networked smart objects, programs, and data is constantly increasing, there is an equally growing demand to ensure the security and reliability of these units. As they are pervasive in our daily lives, this issue has become a significant societal challenge. One central task lies in realizing secure and reliable identification, authentication, and integrity checking of these systems.

Traditional security methods based on secret digital keys often do not provide adequate solutions for this purpose. One major point of vulnerability relates to their hardware implementations and key storage: A whole host of attacks for extracting, estimating, or cloning secret keys that are stored digitally in nonvolatile memory have been developed and reported over the past several years. The situation is especially problematic for embedded and mobile low power devices with a small form factor, where the adversaries can often gain full and direct access to the device.For many FPGA-based reconfigurable devices, which are increasingly growing in market share, the permanent storage of secret keys can be a problem: Integrating secure nonvolatile memory (NVM) on FPGAs incurs additional costs and fabrication overhead and, thus, it is often not included. Therefore, keys have to either be stored in external memory, where they are highly vulnerable,

U. Rührmair (✉)
Computer Science, Technische Universität München, Munich, Germany
e-mail: ruehrmair@in.tum.de

S. Devadas
Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, USA
e-mail: devadas@mit.edu

F. Koushanfar
Electrical and Computer Engineering Department, Rice University, Houston, Texas 77215-1892, USA
e-mail: farinaz@rice.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_4,

or an additional back-upbattery to power on-chip volatile storage must be used, which increases cost and system complexity. We refer interested readers to Chap. 6 of this book for a full discussion of FPGA vulnerabilities and security.

Over recent years, an alternative security approach has therefore emerged, which is based on the inherent, hard-to-forge and unique disorder of physical objects. It constitutes a promising alternative which can address the standing challenges of classical security that were described earlier. Two major classes of disorder-based security systems that have been proposed are *Unique Objects (UNOs)* and *Physical Unclonable Functions (PUFs)*. A Unique Object is a physical system that, upon measurement by an external apparatus, exhibits a small, fixed set of inimitable analog properties that are not similar to any other objects. It shall be impossible to intentionally fabricate a second object with the same properties, even if the properties and exact structure of the original object are known. Such properties can be referred to as the "fingerprint" of a unique object for obvious reasons. We discuss several media that exhibit such unique disorder, including paper, fibers, magnetic disks, radiowave scatterers, and optical tokens.

PUFs are the second important class of disordered systems that can be employed for reliable identification, authentication, key storage, and other security tasks. The term and acronym "PUF" for denomination of this class first appeared in [1]. In a nutshell, a PUF is a disordered physical system S that, when interrogated by a *challenge (or input, stimulus)* denoted by C_i, generates a unique device *response (or output)* denoted by R_{C_i}. This response shall depend on the applied challenge and on the specific disorder and device structure of the PUF. The unclonability requirement in the PUF definition is that it should be intractable for an adversary with physical access to create a physical or software clone of a PUF.

Both the challenge-response pairs of PUFs and the fingerprints of Unique Objects have the purpose of uniquely identifying any device with high probability. In order to realize this in practice, we need stable repeated measurements, and must be able to cope with noise and varying operational conditions. In such scenarios, error correcting codes may be used to ensure the desired operability and robustness of the system [2–7]. Other options are averaging or calibrating the device's operational conditions [8, 9].

Two important metrics that are typically applied to categorize the uniqueness and robustness of PUF responses and UNO fingerprints are *inter-* and *intra-device* distances. Inter-device distance is often quantified as the average Hamming distance between the responses to the same challenge obtained from two different PUFs/UNOs, or the average distance between the fingerprints of two unique objects measured in the same conditions. Intra-device distance is the average Hamming distance between the responses to the same challenge applied at different times and environmental conditions to the same PUF/UNO, or the average distance between the repeatedly measured fingerprint(s) of a unique object. Ideal PUFs and UNOs should lead to large inter-device and small intra-device distances. Another key requirement for PUFs and unique objects is the entropy of the resulting responses or fingerprints. The entropy quantifies the number of independent IDs that can be generated by the same device architecture.

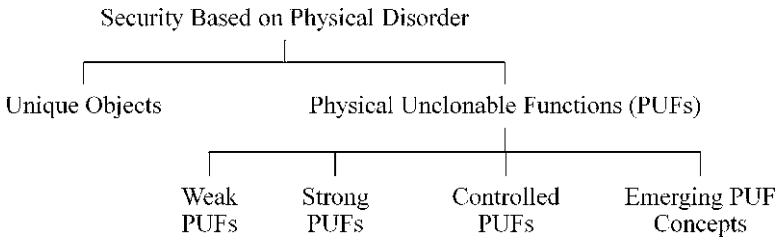


Fig. 4.1 Organization of the chapter

Despite the similarities between UNOs and PUFs, there are several important differences between them that distinguish these two security primitives (and their subclasses) from each other. This chapter provides a conceptual categorization and summary of the field of physical disorder based cryptography and security, also termed *physical cryptography* in [10]. Whenever applicable, the concepts are interleaved with examples from the contemporary literature and implementation details. A number of surveys on the PUF subject are already existent, for example, [11] and a recent book with several chapters dedicated to PUFs [12]. We will cite these review articles whenever applicable, and emphasize that the concepts in this chapter are complementary to the contemporary literature in this area.

4.1.1 Organization of this Chapter

Figure 4.1 gives an overview of the classes of physical disorder-based security tokens discussed in this chapter. Each of the discussed subjects are shown as a branch in the chart. The next section reviews UNOs including paper-based (fiber-based) fingerprints, magnetic signatures, and RF-based Certificates of Authenticity. Section 4.3 discusses the weak PUF class including Physically Obfuscated Keys, SRAM-PUFs, and butterfly PUFs. Strong PUFs are the subject of Sect. 4.4. Examples of PUF structures that can provide building blocks for Strong PUFs include optical PUFs, arbiter PUFs, XOR arbiter PUFs, and analog cellular arrays. Emerging PUF designs and research challenges are presented in Sect. 4.6. We conclude the chapter in Sect. 4.8.

4.2 Unique Objects

Extracting an objects's fingerprint based on its random physical disorder has been exploited for more than three decades. In absence of an established common term, we call this class of structures "Unique Objects (UNOs)".

A Unique Object is a physical entity that exhibits a *small, fixed set of unique analog properties* (the "fingerprint") upon being measured by an external equipment.

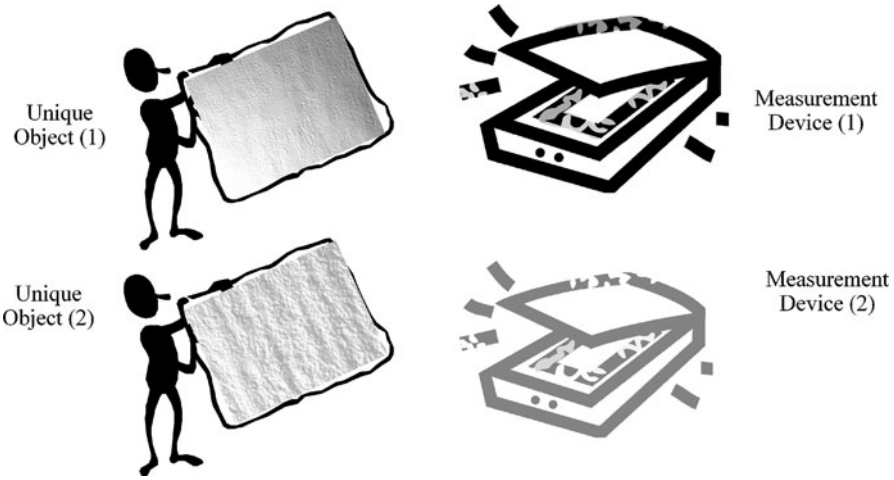


Fig. 4.2 Two Unique Objects (based on paper structures in this example), and two fingerprint measurement devices. The cloning of a Unique Object should be prohibitively costly, while it should be possible to mass-manufacture large numbers of measurement devices that can characterize the fingerprints at the desired level of accuracy

It should be possible to measure the fingerprint quickly and preferably by an inexpensive device. The "fingerprint" should be specific to the object such that it is practically infeasible to find or build another instance of the object with the same specs, even if the object's fingerprint and its detailed structure are known (see also [13]).

More precisely, a Unique Object and its fingerprint should meet the following properties:

1. *Disorder.* The fingerprint should be based on the unique disorder of the physical object.
2. *Operability.* The fingerprint should be adequately stable over time, and must be robust to aging, environmental conditions, and repeated measurements. It must be possible to fabricate other instances of the measurement equipment with similar characterization capability. The measurement and characterization cost and time should be practically and economically viable.
3. *Unclonability.* It should be prohibitively expensive or impractical for any entity (including the manufacturer) to produce another object that presents the same unique fingerprint characteristics when queried by the measurement device.

Figure 4.2 demonstrates the scenario. It is assumed that each Unique Object in the figure has an unclonable fingerprint that is specific to it. Also, it is assumed that both measurement equipment are able to characterize the object's fingerprint at the desired level of resolution and accuracy. In other words, the UNO shall be the unique and unclonable part of the system, while the measurement device can be mass-produced with the same functionality.

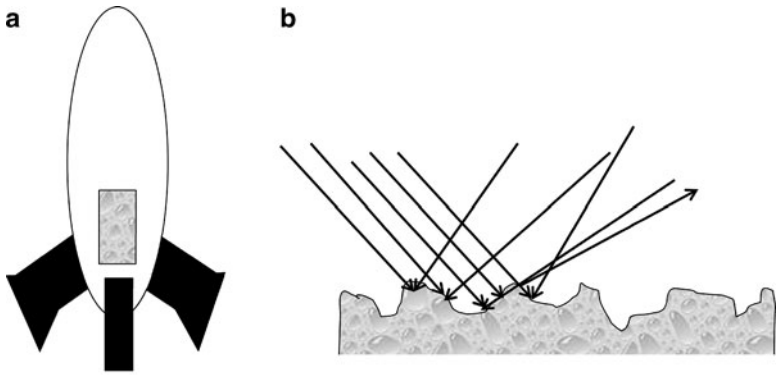


Fig. 4.3 (**a**) A thin, random layer of light scattering particles sprayed on the missiles. (**b**) Illuminating the surface from different angles would generate different inference patterns

4.2.1 History and Examples of Unique Objects

Using biometrics for fingerprinting dates back to the nineteenth century. Although human hand fingerprints and other biometric entities are closely related to Unique Objects, a discussion of biometrics fingerprinting is outside the scope of this chapter. We refer the interested readers to comprehensive books on the subject [14, 15].

4.2.1.1 Sprayed Random Surfaces

Perhaps the earliest reported usage of Unique Objects for security was proposed by Bauder during the cold war for use in nuclear weapons treaties [16, 17]. To tag the nuclear missiles unforgeably, a thin, random layer of light scattering particles was sprayed onto the missiles. The layer was illuminated from various angles, and images of the resulting interference patterns were recorded by an inspector. On later inspections, the interference patterns could be measured anew, and compared to the record of the inspector. An example is shown in Fig. 4.3.

The scheme was assumed secure even if an adversary would know the (relatively few) illumination angles and the resulting patterns used by the inspector, and even if he had access to the unique layer for a long time period and could investigate its structure. Even under these circumstances, it was presumed infeasible to produce a second layer which generated the same speckle pattern. Of course, if an adversary knows all illumination angles and the resulting patterns used by the inspector, this system cannot be used for remote authentication, since an adversary can merely replay back the digitized responses/images upon receiving a challenge. Furthermore, the scheme can only be used by an inspector who carries trusted measurement equipment and uses it on the Unique Object directly, which was presumably the usage model of the system during the cold war.

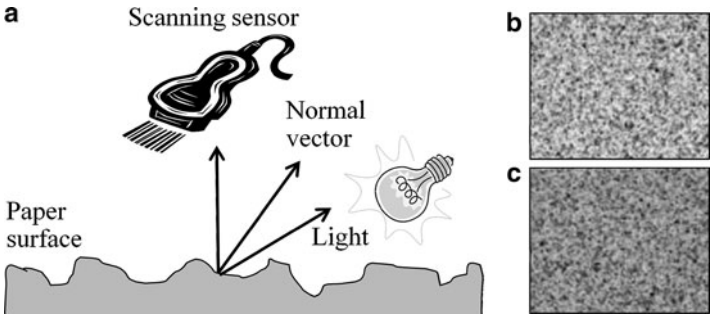


Fig. 4.4 (**a**) Scanner produces different images of the paper surface based on the page orientation. The light seen at the sensor depends on the angle between the surface normal and the light source; (**b**) A region of the document can be scanned from top-to-bottom; and (**c**) The same document region can be scanned from left-to-right. The 3D texture can be estimated by combining (**b**) and (**c**) (Figure inspired by studies in [28])

4.2.1.2 Fiber-Based Random Surfaces

Other early implementations of Unique Objects were based on randomly distributed fibers in solid-state objects, for example, paper fibers in banknotes [18], or metallic fibers in a thin substrate on bank cards measured by a magnetic reader [19]. A seminal reference that discusses Unique Objects from a more fundamental cryptographic perspective is [20], which was later extended by [21]. Reference [20] is, to our knowledge, the first academic source that suggests the combined use of Unique Objects and digital signatures to create offline verifiable labels.

Several seeds laid in this early work were followed up in later research. Firstly, surface irregularities in paper or other materials were further investigated by [22–28]. The authors of [23, 24] create unforgeable postal stamps and document authenticators from paper irregularities and digital signatures; [22] shows that the paper surface fingerprints are robust to soaking and a number of other transformations;

Reference [26] investigates the use of surface-based labels in developing countries and rural regions; [27] deals with database and error correcting algorithms for surface-based identification; and [28] offers a detailed treatment centering around inexpensive measurement methods for paper surfaces by commodity scanners as demonstrated in Fig. 4.4. The complex reflective behavior of surfaces has even led to commercially available security systems [29, 30].

Secondly, the use of randomly disordered fibers contained in a fixing matrix was described in [31–34]. Reference [32, 33] use light conducting optical fibers, and measure the individual light transfer via these fibers into various spatial segments of the matrix. Each instance is created as a collection of fibers randomly positioned in an object by a transparent gluing material that permanently fixes the fibers' positions. Readout of the random structure of a fiber-based note is performed using the following fact: if one end of a fiber is illuminated, the other end will also be lit as shown in Fig. 4.5.

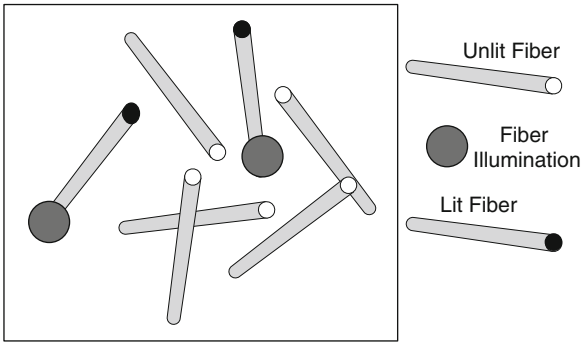


Fig. 4.5 Examples of randomly placed, fixed-length fibers. The square demonstrates the embedding substrate. Three fibers lit by spot illumination light as described in [33]

Reference [31] employs randomly distributed metal particles, and measures their scattering behavior by a near-field read-out. Reference [34] pours ultra-violet fibers into a paper mixture and measures the optical response.

4.2.1.3 Unique Objects and Digital Rights Management

An observation that further propelled the field was that common data carriers such as (again) paper, but also ICs, CDs, and DVDs can have unique features. Sometimes their unique properties arise just in the very process of writing or imprinting data onto them. One early reference in this context is [35]. There, the unique irregularities in CD-imprinting (such as individual height, shape, and length of the bumps) are used to secure the CD's digital content that is stored in exactly these bumps. Conceptually, the same suggestion is made in [36] and [37], yet at a much greater level of scientific detail. For more information on optical media (CD) fingerprints, we refer interested readers to [12]. The irregularities in letters printed on paper have been suggested to secure paper documents in [38]. Finally, several methods for uniquely identifying and authenticating chips will be described in the remainder of this chapter.

4.2.1.4 Other Implementations of Unique Objects

Other studies proposed novel classes of unique structures, and can be best categorized according to the employed read-out technique. A number of Unique Objects with radio wave read-out in the (relative) far field were suggested in the commercial sector [39–44]. For most of them, doubts have been raised with respect to their unclonability [13]. Another radio wave approach measures the unique RF signals created by higher harmonic oscillations [45]. Unique Objects with magnetic read-out have been investigated in [46]. Alternative optical concepts that dig deeper into physics and utilize more complex effects and structures have been suggested in [47]. They include photonic crystals and resonant energy transfer between

optically active particles. Surprisingly, there is little work on Unique Objects with electrical read-out, even though this would promise particularly simple and inexpensive measurement. One recent source is [10], where the unique current–voltage characteristics of imperfect diodes are exploited. Finally, even DNA-based approaches have been suggested [48], and made it to the marketplace some time ago [49].

Finally, the question of error correction of the measured unique signals is treated en passant in most of the earlier publications, including [27, 28, 32, 34]. References solely dedicated to error correction include [50–52].

4.2.2 Protocols and Applications of Unique Objects

The natural application of Unique Objects is to label items of value (such as commercial products, bank cards, banknotes, passports, access cards, etc.) in an unforgeable manner. Proving authenticity is particularly useful as losses due to counterfeiting of digital goods and physical objects amount to worldwide losses in a three-digit billion dollar range [53]. Two basic approaches can be applied.

1. In one classic and straightforward approach, the Unique Object is physically attached to the item it protects, or consists of a measurable unique characteristic of the protected item itself. The Unique Object's fingerprint is stored in a central database. When authentication is needed, the object's fingerprint is measured and compared to the stored value in the database. The requirements for this protocol include existence of a central database and an authenticated online connection to the database.
2. An alternative approach has been pioneered, to our knowledge, in [20], and has been termed Certificate of Authenticity (COA) in [31]. Again, the Unique Object is physically attached to the protected item (or is a measurable unique characteristic of the protected item itself). In addition to the Unique Object, complementary information is stored directly on the item, for example via a printed barcode. The information includes a numerical encoding of the fingerprint, error-correcting codes [52], item-related information I (such as the origin of the item), and most importantly, a digital signature of the fingerprint and I. In order to verify the validity of the label/the item, a verification device does the following: It reads the complementary information from the item, and verifies the validity of the digital signature by use of a public verification key stored in the device. Secondly, it measures the fingerprint of the label/the item by a trusted measurement apparatus, and verifies if it matches the fingerprint given and signed in the complementary information.

The advantage of the approach (2) is that it does not need a connection to a database and that it can work offline. Neither the label nor the testing apparatus needs to contain secret information of any sort. This leads to the strong asset that neither

tampering with a label nor with any of the widespread testing apparatuses can lead to secret key extraction and a global failure of the system through one extracted key. The measurement apparatus has to be trusted to work correctly.

Variants and combinations of the two basic protocols given earlier have been proposed in Unique Objects literature, e.g., [35–37].

4.2.3 Security Features

4.2.3.1 No Secret Information in Hardware

The most striking feature of Unique Objects is that they contain no piece of information that must remain secret, and which would have to be protected by costly and laborious means. Their security rests not on the assumption that some key or some other information about their structure remains secret; rather, it is built on the hypothesis that it is infeasible to build a clone of the object even if all its internal details are known. It is based directly on the limitations of current nano-scale fabrication methods.

Furthermore, a COA can even be verified for validity without possession of any secret keys; any verifying party merely must hold a public key to check the digital signature contained in the COA. This allows the widespread distribution of labels and testing apparatuses, without risking a global security break through secret key compromise in either the labels or apparatuses, which is significant. The only secret key that is required can be stored at the authority that creates the signatures, where it can usually be protected much better. The authenticated communication required in classic protocol (1) above can be established by typical cryptographic methods. At the same time, parties using the system must rely on the integrity of the measurement apparatus. This implies that remote authentication to a central authority by an untrusted terminal is not possible, and therefore limits applicability of Unique Objects.

4.2.3.2 Structural Sensitivity as a Security Benchmark

One critical measure for the security of Unique Objects is their structural sensitivity: How much are the output signal and the unique measured fingerprints of the object affected if we change its inner structure slightly, by a factor of δ, say? This parameter determines the level of exactness that an adversary has to reproduce the Unique Object in order to remain undetected. It can be employed as a benchmark to rank competing candidates of Unique Objects.

4.2.3.3 Attacks on Unique Objects

The main attack on Unique Objects is refabrication or cloning. It is not necessary to rebuild the original with perfect precision; merely, a second structure needs to be fabricated that generates the same measured fingerprint as the original from the view of the measurement apparatus. This structure could in principle have a totally different size, lengthscale, or appearance; it might even be a smart, reactive system that artificially synthesizes the correct response. Note that purely numerical modeling attacks such as the ones executed on PUFs [54] are pointless and not applicable to Unique Objects. Such attacks can help a fraudster to numerically predict the (numerical) response of a PUF to a randomly chosen challenge. But in case of UNOs, the attacker is assumed to know these responses anyway; his task lies in fabricating a clone that produces the same analog response upon measurement with an external apparatus that he/she cannot influence. This is foremost a physical manufacturing challenge, not a question of modeling.

4.2.3.4 Quantum Systems vs. Unique Objects

Quantum systems, such as polarized photons, were among the first systems whose inherent physical features have been suggested for security systems [55, 56]. It is long known that the laws of quantum physics forbid the cloning of a quantum system with an unknown state, for example of a photon with an unknown polarization. Could a polarized photon hence be interpreted as a specific object according to our definition, with its unique property being the polarization angle? This is not the case: One condition of the Unique Object definition is that the adversary knows the unique properties of a Unique Object. But once the polarization of the photon is known, many photons with the same polarization state can be generated. Unique Objects thus relate on a different type of unclonability than quantum systems.

4.3 Weak Physical Unclonable Functions (Weak PUFs)

One class of Physical Unclonable Functions based on inherent device variations are Weak PUFs. They exploit the disordered, unique, internal structure of the underlying fabric as a nonvolatile memory for storing the secret keys. In an ideal case, the volatile keys generated by Weak PUFs upon power-up cannot be determined by external and invasive attacks due to construction or tamper-proof properties of the pertinent structure. Weak PUFs are also known under the name of Physically Obfuscated Keys (POKs) [2].

The term *Weak PUF* was coined in [57] to refer to PUFs with a limited number of challenge-response pairs (CRPs) in contrast to Strong PUFs that contain many CRPs. The following specification has it in greater detail.

1. *Challenge-Response Pairs.* A Weak PUF can be interrogated by one (or a very small number of) fixed challenge(s) C_i, upon which it generates response(s) R_{C_i} that depends on its internal physical disorder.
2. *Key Derivation.* The response(s) R_{C_i} from a Weak PUF is (are) exploited by the device for deriving a standard digital key that can be used for security applications.
3. *Practicality and operability.* The generated response R_{C_i} should be sufficiently stable and robust to environmental conditions andmultiple readings.

4.3.0.5 Weak PUFs vs. UNOs

It is important and necessary to differentiate Weak PUFs from Unique Objects: Applications of Unique Objects require an adversarial model where Eve has time to inspect all features of the Unique Object, and will often know its internal structure and unique fingerprint. Furthermore, these unique properties are measured by an external apparatus. The exact opposite holds for Weak PUFs: Their responses are measured internally, and the derived key is kept secret in the embedding hardware.

4.3.1 History and Implementation Examples

4.3.1.1 ICID PUFs

ICID is the first proposed and designed circuit structure for generating a Weak PUF (or *random chip ID*) based on process variations [58]. They devised an array of addressable MOSFETs (shown in Fig. 4.6), with common gate and source and sequentially selected drains driving a resistive load. Because of device threshold voltage mismatches (resulting from process variation) the drain currents are randomly different. Therefore, at each die, a unique sequence of random voltages would be generated at the load. ICID exploits these unique sequences of random but repeatable voltages to construct unique identification. In 0.35 μm technology, the authors reported about 10% false positive and false negative results for repeating random bits on their test circuits. Identification capability can be improved by increasing the bit length.

4.3.1.2 Physically Obfuscated Keys

Under the name of a Physically Obfuscated Key (POK), Gassend proposed a type of Weak PUF that was built from the first integrated Strong PUF (see Fig. 4.8 in Sect. 4.3.2 for the architecture, and see Sect. 4.4 for Strong PUF) [2]. The POK/Weak PUF would only utilize one (or a small subset) of all possible challenges

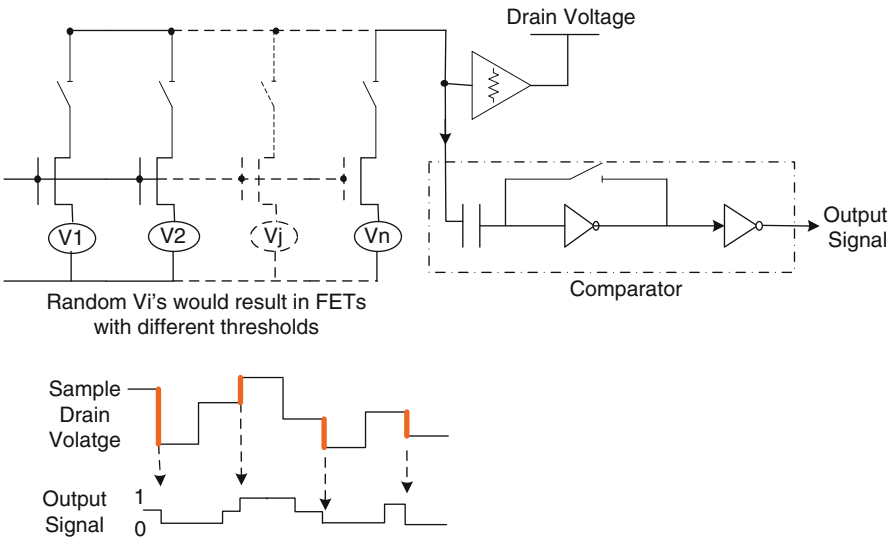


Fig. 4.6 Array of ICID transistors producing a sequential random voltage proposed in [58]

for a Strong PUF. This allows using them exactly as a digital key that is more resistant to physical attack, because it extracts its information from a complex physical system.

4.3.1.3 SRAM-Based PUFs

A commonly used candidate structure for a Weak PUF exploits the positive feedback loop in an SRAM or an SRAM-like structure. If a write operation is used, the cross-coupled device starts transitioning to the inserted value, and the transition is sped up by the positive feedback loop in the structure. When no write operation is in place and the system is not in any of the states (initial power up), the inherent small transistor threshold mismatches and thermal and shot noise trigger the positive feedback loop so the state would be in one of its two possible stable points (0 or 1). The effects of common mode process variations including lithography, and common mode noise (e.g., substrate temperature and supply fluctuations) is similar on the differential device structure and does not strongly impact the transition.

The idea of fingerprinting of semiconductor integrated circuits using SRAM was originally suggested in a 2002 patent, but no experimental implementation data were included [59]. The work in [60] constructed a custom-built array of SRAM-like cells that generated random values based on threshold mismatches in 0.13 μm technology (Fig. 4.7). Their experiments have shown a close to uniform distribution of the bits (close to Normal distribution of Hamming distances) and more than 95% bit stability. The work in [61] showed that initialization of SRAM can produce a

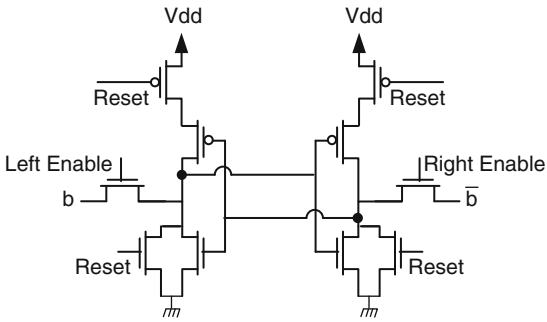


Fig. 4.7 The positive feedback loop created by the cross-coupled NOR (NAND) gates is used for storing a 0 or a 1 in SRAM-like memory structures

physical fingerprint for each chip. They have also shown that the fingerprints can pass the standard NIST randomness tests for runs [61]. The authors in [57] also exploit the initial state of the SRAMs in an FPGA to extract IDs based on differential device mismatches. They coined the term *intrinsic PUF* to refer to structures that do not need additional circuitry for embedding the PUF.

Since not all FPGAs have SRAMs built in them, the work in [5] proposed *butterfly PUFs* based on reconfiguring the FPGA cells to construct two back-to-back NAND gates (in positive feedback mode) similar to the SRAM structure. Note that the Butterfly PUF cannot be considered intrinsic, since it should be custom configured the same way as any other logic circuitry can be made on a reconfigurable fabric.

4.3.1.4 Coating PUFs

Another construction of a Weak PUF is a *coating PUF* that provides a practical implementation of read-proof hardware. A read-proof hardware device has the property that once constructed, no outside entity can read (extract) information on the data stored in the device. The authors in [62] introduced coating PUFs as form of a protective coating that can be sprayed on the IC and cover its surface. The coating is composed of a matrix material doped with random dielectric particles (i.e., different kinds of particles of random shape, size, and location with a relative dielectric constant differing from the coating matrix's dielectric constant). The top metal layer of the IC contains an array of sensors that are used to measure the local capacitance values of the coating.

One central property of a Coating PUF is their purported tamper sensitivity: It is assumed that any tampering with the coating (such as invasive penetration, or removal from the covered IC) strongly and irrecoverably changes its properties. Reference [62] has positively evaluated the resilience of coating PUFs against some optical and invasive attacks.

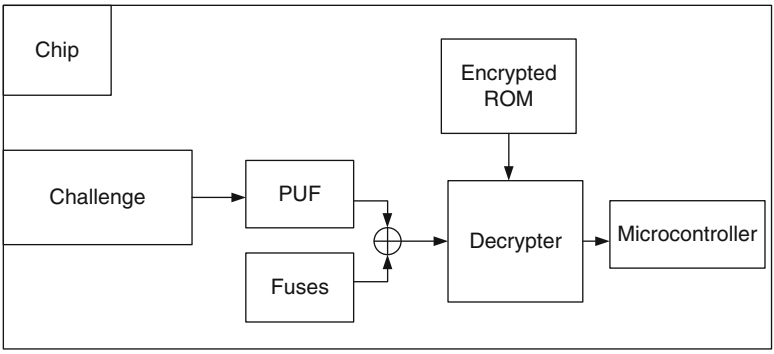


Fig. 4.8 A POK built by using a Strong PUF proposed in [2]

4.3.1.5 Resistive PUFs

Another instance of silicon-based PUFs are based on power distribution and resistance variation of chips that have appeared in recent literature [63, 64].

4.3.2 Protocols, Applications, and Security

4.3.2.1 Secret Key Generation and Storage

Weak PUFs provide a method for secret key generation and storage based on random disordered physical medium fluctuations. Therefore, any security protocol that leverages the storage of a secret key can utilize a Weak PUF in its flow. To our knowledge, the earliest security protocols and IP protection applications based on Weak PUFs/POKs were presented in [2, 65]. Other protocols and applications including metering and RFID protection based on Weak PUFs were presented in [5, 57, 62, 66, 67].

4.3.2.2 IP Protection Application

Weak PUFs were proposed for protecting hardware IP and ICs against piracy. A proposed system for protecting programmable hardware IP against piracy is shown in Fig. 4.8 taken from [2]. Assume that the design is a microcontroller with a compression algorithm stored in ROM. A Strong PUF is hardwired with other functions on the chip to generate a k-bit key K that is the same for all chips to mitigate the cost. The challenges to the PUF are also hardwired to be fixed. That PUF response is combined with the contents of burned on-chip fuses through an exclusive-or operation to produce K. A decoder uses K to decrypt the ROM content.

By selecting the fuse bits one can generate the same decrypting key K on all chips. The response never leaves the chip during the decryption operation. Even if the state of all the fuses are discovered, the key would remain secret.

4.3.2.3 Secure Processor

Suh [65] describes how a Weak PUF can be embedded in a secure processor which then can be used for applications such as certified execution and software licensing. In one design, the weak PUF is used to generate a seed for a public/private key pair. The seed and private key are never exposed and the public key is published and certified by a certification authority. In another, the seed is used as a symmetric key to encrypt a secondary symmetric key that is known to the user of the processor. Again, the seed remains unknown, and is only used to encrypt a given secondary key and decrypt the secondary key for internal use in secure execution.

4.3.2.4 Active IC Metering

Another usage of Weak PUFs was for active IC metering that protects the hardware against foundry piracy (overbuilding) [66]. Here, the functional specification of the design in the finite state machine (FSM) domain was modified. The alteration was such that an exponential number of states were added to the design with a low overhead. Hiding a state in the large state-space of the FSM was later shown to be an instance of a provably obfuscatable general output multipoint function. It was shown that the transitions from the hidden state cannot be found by having access to the layout and even access to the register's contents that store the state. Upon fabrication at the foundry, based on the Weak PUF's response, the design would be in one of the hidden obfuscatable states that is called a locked state. This locked state can be read out by everybody, but the passkeys to the functional (unlocked) state can only be provided by the original designer who has access to the modified FSM.

4.3.2.5 Security Analysis

Weak PUFs are commonly attributed three advantages:

1. They are harder to read-out than standard digital keys that are stored permanently in nonvolatile memory (NVM) since keys only exist when the chip is powered.
2. They can possess some natural tamper sensitivity, meaning that any tampering with the device, or even with the hardware system that embeds the PUF, alters the physical features of the device and the key derived from them.
3. They save on costs, because they avoid the process steps necessary to include NVM in hardware systems.

Some of these assets must be analyzed further. Let us start with advantage (1): Weak PUFs clearly avoid the *long-term* presence of *digital* keys in NVM. But the security of a Weak PUF-based hardware still depends on the secrecy of a single digital key derived from the Weak PUF, which is present for at least a short period after its derivation from the PUF's responses. This creates a single digital point of failure for the system. Weak PUFs furthermore cannot alleviate the permanent presence of secret information in the hardware in general: If an adversary knows the disorder or fabrication mismatches that determine the responses of the Weak PUF, he may simulate and derive these responses.

Further, Weak PUF-based hardware may suffer from similar weak spots as other systems built on standard binary keys. Side channel or emanation analysis may be possible; and since the device will apply some standard cryptoprimitives to K, its security will thus depend on the same unproven computational assumptions as any classical system built on digital keys.

Regarding the above asset (3), it must be stressed that error correcting information is vital for Weak PUFs; any single bit flips make the system not applicable any more. This necessitates the use of accompanying error correcting information, which must be stored in NVM of some form. To the asset of Weak PUFs, this storage can be external and/or public; further, it need not be implemented in the hardware that contains the Weak PUF.

4.4 Strong Physical Unclonable Functions (Strong PUFs)

Immediately after the introduction of Weak PUFs or POKs, a second class of PUFs was put forward [1, 68–70]. They have later often been referred to as Strong PUFs, for example in [57, 71, 72].

In a nutshell, a Strong PUF is a disordered physical system with a very complex input–output behavior that depends on its disorder. The system must allow very many possible inputs or challenges, and must react with outputs or responses that are a function of the applied challenge and of the specific disorder present in the system. The input/output behavior should be so complex that it cannot be imitated numerically or by any other device.

More specifically, a Strong PUF is a disordered physical system S with the following features:

1. *Challenge-Response Pairs.* The Strong PUF can be interrogated by challenges C_i, upon which it generates a response R_{C_i} that depends on its internal physical disorder and the incident challenge. The number of CRPs must be very large; often (but not always) it is exponential with respect to some system parameter, for example with respect to the number of components used for building the PUF.
2. *Practicality and operability.* The CRPs should be sufficiently stable and robust to environmental conditions and multiple readings.

3. *Access mode.* Any entity that has access to the Strong PUF can apply multiple challenges to it and can read out the corresponding responses. There is no protected, controlled or restricted access to the PUF's challenges and responses.
4. *Security.* Without physically possessing a Strong PUF, neither an adversary nor the PUF's manufacturer can correctly predict the response to a randomly chosen challenge with a high probability. This shall hold even if both parties had access to the Strong PUF at an earlier time for a significant period, and could make any reasonable physical measurements on the PUF, including (but not limited to) determination of many CRPs.

The definition earlier is more qualitative than quantitative in order to remain intuitive; a more formal and thorough definition can be found in [72].

Unique vs. Weak vs. Strong

While aUnique Object must always possess an external and a Weak PUF always an internal measurement equipment, this is left open for Strong PUFs; both variants are possible and have been realized in practice (see [68, 69] for an optical PUF with an external and [1, 73] for an electrical PUF with an internal measurement apparatus). Unique Objects require a trusted measurement apparatus whereas Strong PUFs once "bootstrapped" (cf. Sect. 4.4.2) can be remotely authenticated with an untrusted measurement apparatus. Another difference between Strong PUFs and Unique Objects lies in the exact adversarial model and the relevant security properties: While the adversary's aim in the case of Unique Objects lies in physically fabricating a clone device with the same properties, his goal in the case of Strong PUFs is to learn how to predict the input/output behavior of the Strong PUF. The latter is a mixture of numerical assumptions and physical hypotheses. This fact does not exclude that the same structure can be used as a Unique Object and as a Strong PUF under different read-out schemes, for example; but not every Unique Object is a Strong PUF and vice versa.

Weak PUFs possess only a small number of fixed challenges, whereas Strong PUFs have a very large number of challenges. In Weak PUFs, the responses remain secret and internal. To the contrary, Strong PUFs allow free querying of their responses.

4.4.1 History and Examples of Strong PUFs

4.4.1.1 Optical PUF

The first implementation of a Strong PUF has been suggested in [68,69], albeit under the different name of a physical one-way function. It consists of a transparent plastic token which contains a large number of randomly distributed glass spheres as shown in Fig. 4.9. We call this implementation an *optical PUF*. An individual, unclonable

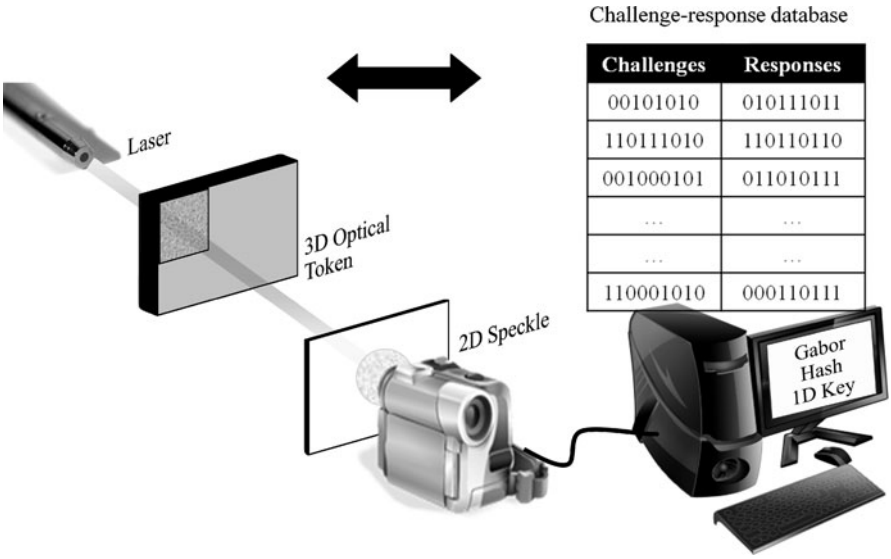


Fig. 4.9 A 3D inhomogeneous transparent plastic token being optically challenged (illuminated under different angles and points of incidents) and produces an output in form of an interference pattern. The output is hashed to produce a 2D image, which is in turn filtered by a multiscale Gabor transform to produce a 1D key as proposed in [69]

token is illuminated under different angles and points of incidence (which are regarded as the challenges of the system), and produces an interference pattern, which is considered the response of the system. We draw the reader's attention to the similarity in Figs. 4.3 and 4.9. The main difference is a usage one: optical PUFs are assumed to have a large number of challenges, and a secret set of challenge-response pairs is stored in a central database. Thus, optical PUFs can be remotely authenticated.

This construction is presumably secure (no attacks are known to this date), but the measurement apparatus is external and relatively large, potentially leading to practicality issues and stability problems when the token is measured by different apparatuses at different locations.

4.4.1.2 Arbiter PUF

Almost simultaneously to optical PUFs, the first integrated electrical Strong PUFs including "Arbiter PUFs" were put forward in [1, 73]. Reference [1] is also the first publication that uses the term PUF as a common abbreviation for the expressions Physical Random Function and Physical Unclonable Function. Unlike optical PUFs, silicon PUFs do *not* require external measurement equipment. They are based on the runtime delay variations in electrical circuits.

In one implementation, an electrical signal is split into two parallel signals, which race against each other through a sequence of k electrical components, for example, k multiplexers. This architecture is shown in Fig. 4.10. As shown in the figure, the

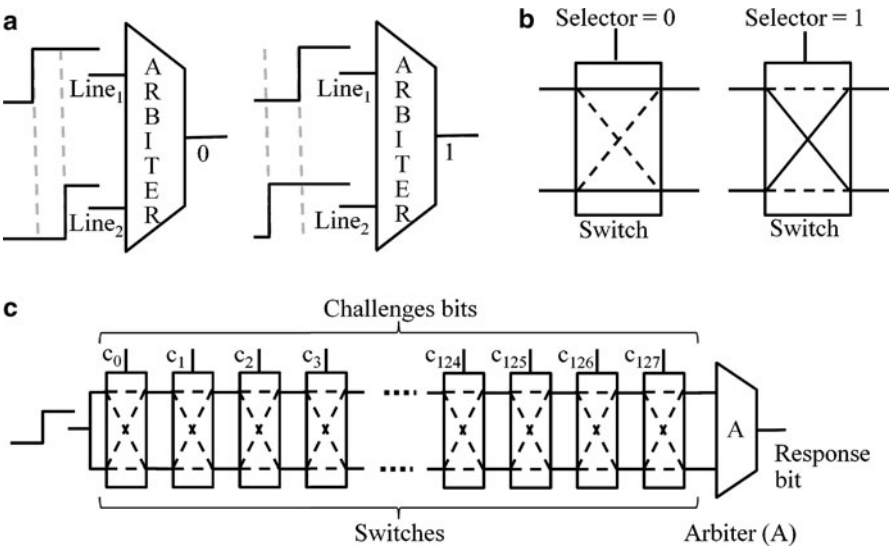


Fig. 4.10 (**a**) Demonstration of an arbiter's operation: the relative time of signal arrival at Line_1 and Line_2 would determine the value of the output bit; (**b**) Demonstration of a selector's operation: the selector bit would decide if the *top* and *bottom* lines continue in the same order, or they switch places; (**c**) An arbiter PUF with 128 challenge bits $c_0, \ldots, c_{127}$ applied as the selectors to the switches. The switch selectors dynamically configure two parallel paths with random delay differences that would form the response generated by the arbiter [74])

challenges are applied to the selectors of the multiplexers. The exact signal paths are determined by these challenge bits $b_1, \ldots, b_k$ applied at the multiplexers. At the end of the k components, an arbiter element decides which of the two signals arrived first and correspondingly outputs a zero or a one, which is regarded as the system's response.

It was clear from the beginning that these first electrical candidates were prone to modeling attacks as mentioned in [1]. Attacks using machine learning algorithms have been carried out, see Sect. 4.4.2.2. In these attacks, the adversary collects many challenge-response pairs (CRPs), and uses them to derive the runtime delays occurring in the subcomponents of the electrical circuit. Once they are known, simple simulation and prediction of the PUF becomes possible, breaking its security. One reason why these attacks worked so well lies in the fact that plain Arbiter PUFs have relatively simple linear models, in which the delay of each of the two signals can be approximated as the linear sum of the delays in the subcomponents. This makes standard machine learning algorithms applicable to the problem.

4.4.1.3 Variants of the Arbiter PUF

The earlier issues naturally led to the introduction of nonlinear electrical PUFs, for example, XOR arbiter PUFs, Lightweight Secure PUFs, and Feedforward Arbiter

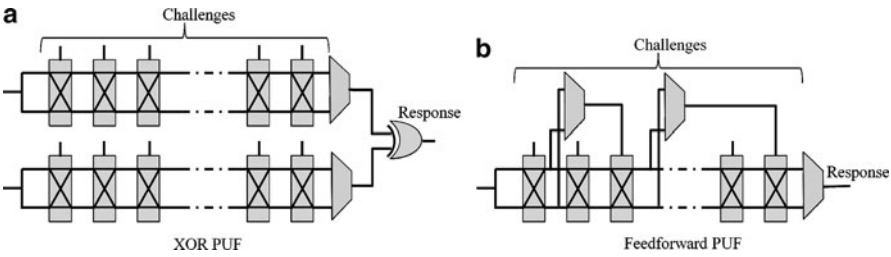


Fig. 4.11 (**a**) An arbiter PUF wtih added XORing of two arbiter outputs; (**b**) Feedforward PUF

PUFs [11, 74–76]. In an XOR arbiter PUF, multiple arbiter outputs are XOR'ed to form a response. In Fig. 4.11a, an example is shown where two arbiter outputs are XOR'ed. In the Feedforward Arbiter PUF, the output of intermediate multiplexer(s) on the signal paths are input to so called Feedforward arbiter(s). The Feedforward arbiter output is then fed to the input of another multiplexer forward on the signal path. In Fig. 4.11b, an example of a Feedforward arbiter structure is shown. All of the aforementioned structures employ the basic Arbiter PUF architecture, but refine its architecture by introducing additional, nonlinearities. These structures showed a significantly higher resilience against machine learning attacks, but still could be attacked up to a certain level of size and complexity [54, 77].

Arbiter PUFs and their variants have been shown to have small and stable integrated electrical implementations and have been commercialized [78].

4.4.1.4 Legacy PUFs

Reference [80] has proposed using the ICs' timing path signatures that are unique for each state-of-the-art CMOS chip (because of process variations) as a PUF. The work in [81, 82] has shown that all ICs that are fabricated in new CMOS process nodes that contain nontrivial process variations have a unique signature that can be extracted using noninvasive methods by the structural side channel tests such as IDDT, IDDQ, or delay tests. They have shown a unified gate-level characterization of the signatures for all side-channels that could be used as a compact representation. It was shown that statistical signal processing methods can be adopted for ensuring rapid and robust characterization [83–87]. The interesting aspect of this line of work is that the signatures are intrinsic to all legacy ICs, and there is no need for insertion of additional circuits or structures by the manufacturer or other parties who are interested in verifying the chip's authenticity by its specific signature. Therefore, it can be readily used for digital rights management of integrated circuits in the supply chain and for anti-counterfeiting protection.

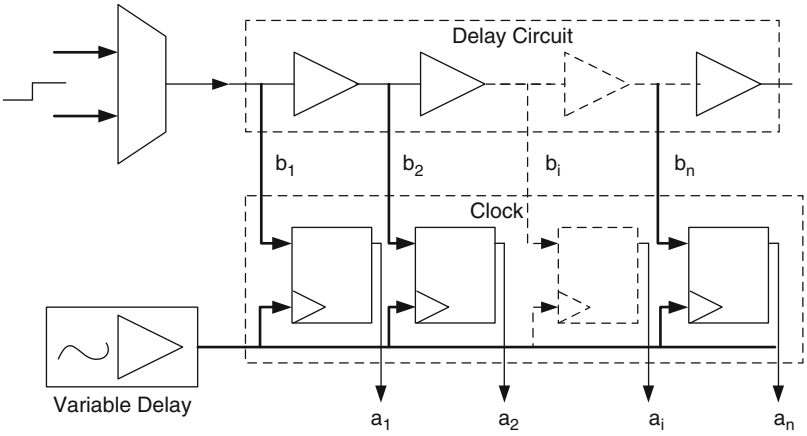


Fig. 4.12 The glitch PUF architecture samples the glitches on the path and the arrival of a glitch compared to a clock signal generates the response bits in the FFs [79]

4.4.1.5 Analog PUF Family

New, recent suggestions for Strong PUFs have tried to exploit the analog characteristics of electrical signals, such as in analog cellular arrays [88]. The system suggested in [88] imitates optical wave propagation in an electrical cellular nonlinear network, transferring the known complexity of optical PUFs into electrical circuits. Another nonlinear electrical suggestion is [79] that is based on the nonlinear propagation of glitches on a logic path. Figure 4.12 demonstrates the architecture of the glitch PUF system, where the glitches based on the delay difference between the signal path and the clock signal are stored in the response FFs. Finally, integrated optical PUFs have been proposed [89], but their security seems suspect if merely linear scattering media are used (see appendix of [90]).

4.4.2 Protocols, Applications, and Security

4.4.2.1 Protocols and Applications

The archetypical application of Strong PUFs is the identification and authentication of hardware systems (or other security tokens such as credit cards) [1, 68, 69]. The corresponding protocols are usually run between a central authority (CA) and a hardware/token carrying a Strong PUF S. One assumes that the CA had earlier access to S, and could establish a large, secret list of challenge-response-pairs (CRPs) of S using a trusted external measurement apparatus in the case, for example, of an optical PUF. This step is usually referred to as *bootstrapping*. Whenever the hardware, possibly at a remote location, wants to identify itself to

the CA at some later point in time, the CA selects some CRPs at random from this list, and sends the challenges contained in these CRPs to the hardware in the clear. The hardware applies these challenges to S, and sends the obtained responses to the CA, also in the clear. If these responses *closely* match the prerecorded responses in the CRP-list, the CA believes the identity of the hardware. Note that each CRP can only be used once, whence the CRP-list shrinks over time, and needs to be large. As noted earlier, an exact match is not required, and a certain level of noise in the responses can be tolerated.

Another application that has been mentioned is key exchange or key establishment based on Strong PUFs [69]; a formal protocol has been given in [89]. However, it has been shown in [91] that such key exchange protocols can suffer from problems regarding their forward secrecy and their repeated use for session key exchange. Reference [91] also proposed a new type of PUF that can fix this issue, called erasable PUFs. We note that this new type of erasable PUFs is different than the earlier FPGA PUFs that could be configured and erased for each authentication session [77, 92].

The earlier protocols give Strong PUFs broad cryptographic applicability. They can be employed for any application which requires the above cryptographic tasks, often without storing explicit digital keys in the hardware containing the PUF.

4.4.2.2 Security Features and Attacks

Attacks on Strong PUFs will either try to build a physical clone, i.e., a second physical system that behaves indistinguishably from the original PUF, or a digital clone, i.e., a computer algorithm that imitates the PUF's challenge-response behavior.

It has been rightfully stressed in early publications on Strong PUFs [68, 69] that they can avoid the classical, well-known number-theoretic assumptions in cryptographic protocols. But is the security of Strong PUFs entirely free of computational assumptions, and can it merely be built on their internal entropy and randomness? It is known that the maximal amount of randomness or entropy in a physical system is polynomially bounded in the size of the system [71, 72, 93]. This implies that the overall number of possible challenges of many PUFs is larger than their entropy. In particular, this observation necessarily holds for any PUFs with an exponential number of challenges.

An adversary therefore often merely needs to gather a small subset of all CRPs of a Strong PUF to obtain (at least indirect) knowledge about all CRP-relevant information/entropy contained in the PUF. Once he has gathered such a subset, it is merely a computational assumption that he cannot derive an internal PUF model from it which allows PUF prediction. For example, he could set up a system of (in-)equations from the CRP subset, whose variables describe the inner PUF structure. If he can solve this system of (in-)equations efficiently, he can break the PUF. The hypothesis that he will not be able to do so is just another type of unproven computational assumption.

This perhaps surprising observation is not just a theoretical concern. The modeling attacks presented in [2, 54, 71, 75, 77, 94–96] are practical, and prove the basic feasibility and effectiveness of such attacks. They also exhibit that such attacks reach their limits when the involved computations become too complex; for example, the authors of [54] could not attack XOR arbiter PUFs with $k > 6$ XORs because the complexity grew exponentially in k.

In other words, the security of many Strong PUFs is dependent on the underlying computational assumptions. In favor of Strong PUFs, it must be said that these assumptions are independent of the classical number-theoretic assumptions such as the factoring or discrete logarithm function, and that Strong PUFs can help to establish an independent basis for cryptography and security. Furthermore, they have other security advantages, as discussed in the remainder of this section. The only subtype of Strong PUFs whose security is strictly independent of computational assumptions are SHIC PUFs [10,97,98]. The price they pay for this feature is an intrinsically slow read-out speed and a comparably large area consumption.

This brings us to another central point related to Strong PUF security. Strong PUFs avoid the use of explicit digital keys in hardware. But do they avoid the presence of secret information in hardware in general? Once the internal configuration of a Strong PUF has become known, an adversary will almost always be able to predict and hence break the PUF. To illustrate our point, consider the Arbiter PUF and its variants: Once the internal runtime delays have become known, the structure can be fully predicted and broken. Therefore Strong PUFs, just like classical cryptosystems, often depend on the assumption that some internal information remains secret. In their favor, this information is arguably hidden better than if stored explicitly in the form of a digital key. F will usually be known to the adversary and efficiently computable.

4.4.2.3 Security Benchmarks

Natural security benchmarks for Strong PUFs must evaluate the complexity of their challenge-response behavior and their resilience against modeling attacks. To this end, various measures have been proposed: (1) Theoretical analysis of the overall internal entropy of the PUF [69]. (2) Theoretical analysis of the entropy/information-theoretic independence of the CRPs [99–101]. (3) Empirical, statistical analysis of large CRP sets by statistical tools and compression algorithms [95,102,103]. (4) Empirical analysis by assessment of machine learning curves over instances of increasing size and complexity [95, 103].

Let us briefly discuss these approaches. One downside of (1) is that it usually does not consider the CRP-relevant entropy, but the general entropy of the system, which is often very much larger. (2) is a suitable measure. On the downside, it can be difficult to derive theoretically, and does not take into account computational aspects. (3) and (4) are easy to apply and generic tools, but do not provide definite security guarantees. (3) does not require an generic model of the PUF (such as the linear additive delay model for arbiter PUFs), while method (4) needs such a model before it can be applied.

4.5 Controlled Physical Unclonable Functions

4.5.1 Specification of Controlled PUFs

Let us start by specifying the notion of a Controlled PUF: A Controlled Physical Unclonable Function (CPUF) is a PUF that has been bound with an algorithm in such a way that it can only be accessed through a specific Application Programming Interface (API).

The main problem with (uncontrolled) Strong PUFs is that anybody can query the PUF for the response to any challenge. To engage in cryptography with a PUF device, a user who knows a CRP has to use the fact that only he and the device know the response to the user's challenge. But to exploit that fact, the user has to tell the device his challenge so that it can get the response. The challenge has to be told in the clear because there is no key yet. Thus a man in the middle can hear the challenge, get the response from the PUF device and use it to spoof the PUF device.

Clearly, the problem in this attack is that the adversary can freely query the PUF to get the response to the user's challenge. By using a CPUF in which access to the PUF is restricted by a control algorithm, this attack can be prevented. The API through which the PUF is accessed should prevent the man-in-the-middle attack we have described without imposing unnecessary limitations on applications.

4.5.2 History and Implementation

CPUFs can perform all operations that a Strong PUF can perform. While the details of various CPUF APIs are beyond the scope of this paper, useful APIs have been developed [70, 104] that satisfy the following properties:

1. *Access Control.* Anybody who knows a CRP that nobody else knows, can interact with the CPUF device to obtain an arbitrary number of other CRPs that nobody else knows. Thus users are not limited to using a small number of digital outputs from the PUF. Moreover, if one of these new CRPs was revealed to an adversary, transactions that use the other CRPs are not compromised. This is analogous to key management schemes that use session keys derived from a master key.
2. *Secret Sharing.* Anybody can use a CRP that only they know to establish a shared secret with the PUF device. Having a shared secret with the PUF device enables a wide variety of standard cryptographic primitives to be used.
3. *Control Algorithm.* The control algorithm is deterministic. Because hardware random number generators are sensitive and prone to attack, being able to avoid them is advantageous.
4. *Cryptographic Primitive.* The only cryptographic primitive that needs to be built into the control algorithm is a collision resistant hash function. All other cryptographic primitives can be updated during the lifetime of the CPUF device.

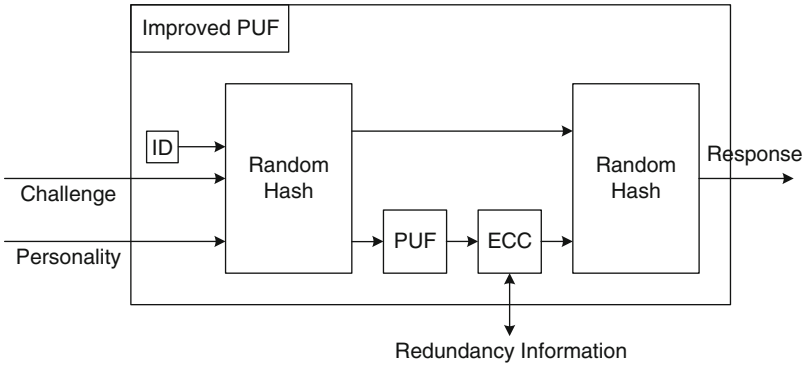


Fig. 4.13 An example architecture for a controlled PUF proposed in [70]

By selecting an appropriate API, a CPUF device can be resistant to protocol attacks. With careful design, Optical and Silicon PUFs can be made in such a way that the chip containing the control logic is physically embedded within the PUF: the chip can be embedded within the bubble-containing medium of an Optical PUF, or the delay wires of a Silicon PUF can form a cage on the top chip layer. This embedding should make probing of the control logic considerably more difficult, as an invasive attacker will have to access the wires to be probed without changing the response of the surrounding PUF medium.

The PUF and its control logic have complementary roles. The PUF protects the control logic from invasive attacks, while the control logic protects the PUF from protocol attacks. This synergy makes a CPUF far more secure than either the PUF or the control logic taken independently. Figure 4.13 demonstrates an example architecture of how a controlled PUF can be used for improving a PUF. A random hash function is placed before the PUF to prevent the adversary from doing a PUF chosen challenge attack. So a model-building adversary is prevented from selecting challenges that allow him to extract the PUF parameters. To ensure response consistency, an Error Correcting Code (ECC) is used. An output random hash function is used to decorrelate the response from the actual physical measurements, and therefore rendering a model-building adversary's task even harder.

4.5.3 Protocols, Applications, and Security

Because there is no algorithmic way to tie together all the keys produced by a device, the device will have to take an active part in protocols like certificate verification, that would not usually need any device involvement. This limitation is offset by a decreased vulnerability to invasive attacks.

There are many applications for which CPUFs can be used, and we give two examples here. Other applications can be imagined by studying the literature on secure coprocessors, in particular [105]. We note that the general applications for which this technology can be used include all the applications today in which there is a single symmetric key on a chip.

A bank could use certified execution to authenticate messages from PUF smartcards. This guarantees that the message the bank receives originated from the smartcard. It does not, however authenticate the bearer of the smartcard. Some other means such as a PIN number or biometrics must be used by the smartcard to determine if its bearer is allowed to use it. If the privacy of the smartcard's message is a requirement then the message can also be encrypted.

A second application is for computers that implement private storage [106–112]. A program wishing to store encrypted data in untrusted memory uses an encryption key which depends uniquely on the PUF and its program hash. This requires a CPUF in order to accomplish the unique dependency. This idea is implemented in the AEGIS processor [112, 113].

Physically obfuscated keys generated from *Weak* PUFs seem to increase the difficulty of an invasive attack, but they still have a single digital point of failure. When the device is in use, the single physically obfuscated master key is present on it in digital form. If an adversary can get that key he has totally broken the device's security. CPUFs exploit the parameterizability of the complex physical system like *Strong* PUFs do. For each input to the physical system, a different key is produced. Thus the complexity of the physical system is exploited to the utmost.

As noted previously one difficulty with Weak PUFs is that their output is noisy. For use in cryptography, we need error-correction which does not compromise the security is required. For Weak PUFs only one response has to be made noise-free, for CPUFs many responses have to potentially be corrected. We need to store an error correcting syndrome with each challenge-response pair. Secure and robust error correction has been considered for Weak PUFs (see [7]) but these schemes need to be efficiently generalized to CPUFs.

4.6 Emerging PUF Concepts

There are a number of new concepts that have emerged in the area of PUFs, and the pace of innovation is rapid. We mention interesting new concepts proposed in the past couple of years in this section, and address ongoing research challenges in Sect. 4.7.

4.6.1 PUFs with Secret Models

In classical identification schemes based on Strong PUFs, the verifier must possess a large list of CRPs that have been premeasured in a secure bootstrapping phase

[1,68]. The challenges sent to the prover are chosen randomly from this list, and the responses obtained from the prover are verified for correctness against this list. As the list must suffice for the lifetime of the device, it must be large, which imposes uncomfortable storage requirements on the verifier.

It has been independently observed by [77, 114, 115] that such storage requirements may be lifted if the verifier instead stores a secret model for the PUF, by which he can simulate and predict arbitrary responses of the PUF. Such secret models can furthermore allow the offline verification of a PUF's identity, i.e., they can enable identification protocols that are run without an online connection to a trusted authority holding a CRP-list. The underlying PUF primitive could be called *Secret Model PUF* or *SM PUF*, for short.

Secret Model PUFs are a very useful concept that leads to improved practicality features and new protocols. They do not lift two important constraints of Strong PUFs, though: First, the model itself must be kept secret, similar to a secret key. They therefore require the authenticating entity to store a symmetric key to decrypt the secret model stored in encrypted form on the PUF device. Second, SM PUFs still contain some secret information, namely the information that was used to set up the secret model (for example the internal runtime delays). These two requirements are only overcome by the concepts proposed in the next Sects. 4.6.2 and 4.6.3.

4.6.2 Timed Authentication

For certain implementations of Strong PUFs, the real-time interval in which the PUF generates its responses may be noticeably shorter than the time that any numerical model or purported clone would require to the same end.

In a PUF-related context, this observation has first been stated in [77]. They noted that for certain FPGA-based PUFs, only the authentic hardware would be able to generate the response in a minimum number of cycles, and that a model built based on the device characteristics would likely be slower in finding the response to a given challenge (compared to the original device). They proposed an authentication protocol that exploits this unique property of the original FPGA device: A time-bound set by the protocol for obtaining the correct response after applying a random challenge ensured that only the authentic device could respond. This scheme has been referred to as Timed Authentication (TA) in [77].

Reference [77] suggests an "asymmetry" in the timed computational capabilities of the authentic device compared to other entities. This asymmetry was elaborated on for thwarting the modeling attacks. However, the proposed protocol is a symmetric key like scheme, since it requires a secret list of CRPs. It was noted that asymmetry can lift the feature that the internal configuration of the PUF-hardware must remain secret. Think of the optical PUF introduced in Sect. 4.4.1 as an example: Even if the position of all internal scattering elements/bubbles was known to an adversary, he would still find it hard to simulate the complex input–output behavior of the scattering medium in real-time. The same holds for the FPGA-based implementation of TA discussed in [77].

4.6.3 PUFs with Public Models

Section 4.6.1 told us that a secret model for a Strong PUF can replace the CRP list. Section 4.6.2 described that certain Strong PUFs operate faster than any adversarial model and emulation. Both concepts can be combined to enable PUFs with simulation models that can be made public (and hence can be simulated by everyone), but which still operate faster than any clone or model (including the public model, of course). The manufacturer or some other entity can tie the model to the respective PUF, by, for example, signing the model, or keeping it in a trusted public register. This allows everyone to simulate the responses of the PUF with some time overhead. Only the party holding the PUF can determine the PUF's responses fast, i.e., within a certain time bound, by a physical measurement on the PUF. This allows public key like functionalities and protocols. Hardware systems based on such a concept have the further intriguing advantage that they can eradicate the presence of any form of secret information in the cryptographic hardware, while still being usable in typical digital network applications such as remote identification and message authentication.

4.6.3.1 History

The concept of PUFs with public models has been introduced and developed independently in several lines of research. Under the name of a Public PUF (PPUF), this primitive has been introduced in [13, 117, 118], building on a hardware concept that had been published earlier [77, 81, 119, 120]. Protocols and applications of PPUFs have since been developed [117, 118, 121]. Under the name of a SIMPL system, the same concept was put forward completely independently in [122, 123]. Implementations, protocols and applications of SIMPLs have been elaborated on in [87, 90, 124–126, 128]. In another line of research, the concept of PUFs with public models has been made explicit with implementation results on FPGAs under the name Time-Bounded Authentication (TBA) in [9, 92]; this builds on the concept of TA treated in the last section [77].

4.6.4 Quantum Readout PUFs

Reference [129] proposed modifying the challenge-response mechanism of a PUF with quantum states, called a *Quantum Readout PUF* [130]. The properties of the quantum states prevent an adversary from intercepting the challenges and responses without modifying them. Thus, there is no need for a trusted location for bootstrapping. However, no proof-of-concept implementation or practical architecture for this structure has been proposed to date. Finally, interfacing the quantum readout device to the regular PUF is likely a challenge.

4.6.5 SHIC PUFs

A final recent concept are PUFs with Super-High Information Content, abbreviated *SHIC PUFs*[1] References [10, 97, 98]. SHIC PUFs are Strong PUFs whose large number of CRPs are pairwise independent in an information-theoretic sense. Unlike other Strong PUFs, this allows them to become independent of computational assumptions in their security. The price they pay is a relatively large area consumption and slow read-out speed on the order of 10^2 to 10^4 bits per second. SHIC PUFs are unlikely to be used in low-cost commercial applications in the near future, since there are other, more favorable solutions to this end. But they represent an intriguing theoretical tool, since they are a variant of Strong PUFs with information-theoretic security. Furthermore, investigating their optimal implementation is rewarding from a technological perspective, since it relates to fundamental technological questions such as "How much random information can we store and reliably extract from a solid-state system?", and "How can we make the speed in which information is released from a solid-state system inherently *slow*?".

4.7 Future Research Topics

4.7.1 Open Public PUF Questions

The main open questions related to PUFs with Public Models concern their hardware realization:

- How can it be guaranteed that the model requires more time to simulate than the PUF device requires to return a response?
- How can it be guaranteed that a well-equipped adversary for sure takes longer than the PUF device, while any poorly equipped honest party can simulate the response in feasible time in the course of a communication protocol?
- Can the model be close enough to the PUF so that an adversary finds it difficult to physically clone the PUF, but loose enough to allow for variation due to environmental conditions of PUF responses?

While there have been many recent proposals for timed authentication, we are not aware of any implementation that definitively settles the above questions. This leaves strong potential for future research. If a workable, small and inexpensive implementation of PPUFs, SIMPL systems or TBA systems is found eventually, or if one of the existing implementations is shown to possess all necessary properties, this would have a massive impact on the way we perform cryptography and construct security hardware.

[1] SHIC PUFs are to be pronounced as *"chique PUFs"* according to [10].

4.7.2 Efficient Hardware Implementations: Overhead vs. Security

Recent work has discussed how it could be possible to safeguard PUFs against reverse-engineering and modeling attacks [10, 54, 77, 88, 97, 98]. However, most methods that aim at protecting against such attacks add strongly to the power, size, delay, instability, or cost overhead of the system. Also techniques for ensuring the tamper-proof properties, such as inaccessibility of the Weak PUF, would require addition of tamper-proof circuitry and material to the devices. One major future research topic is how the security of Strong PUFs and/or the tamper sensitivity of Strong PUFs and Weak PUFs can be realized with a minimal hardware overhead. These future research questions naturally relate to circuit design and, concerning tamper sensitivity, also to the material sciences.

4.7.3 Error Correction and Practical Operability

A suite of security applications of PUFs, such as secret key generation by Weak PUFs, require full and error-free reconstruction of the keys. However, environmental conditions and aging may affect the measured responses in strong ways. Methods for compensation of such effects, such as circuit reliability enhancement techniques, error correction and secure sketches, are being developed [2, 4, 5, 7, 8, 131]. Further development of methods that ensure robustness of PUFs with a limited amount of leaked information is of great interest. One key challenge is that the maximum number of unpredictable bits should be known at the design time. If the unpredictability exceeds the bound set at the time of design, the error correction method would not be able to compensate for the errors. Therefore, careful experimental studies for each new PUF structure are needed for characterizing the performance under different temperature, voltage, and/or other environmental and operational conditions, constituting a future area of active and fruitful interplay between hardware analysis and error correction techniques.

4.7.4 IC Metering and Counterfeit Detection

A counterfeit product is an illegal forgery or imitation of an original design. Because of the dominance of the contract foundry model, IP sharing/reuse, and outsourcing, the electronic products are increasingly vulnerable to piracy attack and counterfeiting. *IC metering* is a set of security protocols that enable the design house (authentic IP owner) to achieve postfabrication control over their ICs [66, 119, 132, 133]. In *passive IC metering*, the IP rights owner is able to identify and monitor the devices [119, 132]. Passive metering can be directly enabled by certain types of PUFs. In *active IP metering*, in addition to identification and monitoring, the IP rights

holder can actively control, enable/disable, and authenticate a device [66]. We refer the interested readers to Chap. 8 of this book for a comprehensive survey of this topic. Addressing piracy attacks is notoriously hard since the adversaries are often financially strong, technologically advanced and informed of the design details. A set of open research questions have to do with developing security methods, PUF architectures, and controlled PUF protocols that can directly address the piracy attack models and counterfeiting.

4.7.5 Attacks and Vulnerability Analysis

To date, a number of attacks and countermeasures for PUFs are reported, see for example the detailed discussions in Sect. 4.4.2. However, PUFs have yet to undergo more refined cryptanalysis and evaluation of physical and side-channel attacks by a large community of researchers, similar to the way many traditional cryptographic primitives and protocols have been analyzed and attacked. For PUFs to be widely accepted, this seems to be a central future task that needs to be performed.

4.7.6 Formalization and Security Proofs

One relatively untouched area within physical cryptography and PUFs are the foundations of these fields. Formal definitions and security proofs for PUF-based protocols are just about to develop. For example, [71, 72] provide a thorough discussion of existing PUF definitions. Reference [72] give new formal definitions for Strong PUFs that lead to a first reductionist security proof for a Strong PUF-based identification scheme. This type of work will likely prove essential for sound future development of the field, and will represent one of the major upcoming research topics within the area.

4.7.7 New Protocols and Applications

Up to now, PUFs and UNOs have mainly been used for authentication and identification purposes, and have mainly been seen as a security tool. But recently, a fundamental result indicated that PUFs possess a strong cryptographic potential: Oblivious transfer (and all protocols that can be derived from it) can be realized by Strong PUFs [134]. Protocol design and optimization will thus be active future research topics.

4.8 Conclusion

Security and protection based on random physical media and objects is a fast-growing field that has recently enjoyed considerable research interest. Ensuring authenticity, security, protection, and integrity of data, hardware and software intellectual property, computers, networks, identities, and cyber-physical systems is a standing challenge. Traditional digital methods for these tasks often rely on digital labels or digitally stored secret keys that are vulnerable to forging, cloning, and other attacks. As discussed extensively in the previous sections, the unique and unclonable character of disordered physical structures can be exploited to address many of the vulnerabilities of these traditional concepts.

This chapter presented a new classification for the area of physical disorder-based cryptography and security. We dealt with disorder-based identification, authentication, and other security methods. We then focused on four new classes of security devices based on physical disorder: Unique Objects, Weak Physical Unclonable Functions (Weak PUFs), Strong PUFs, and Controlled PUFs. Alongside with defining each class and discussing the history and relevant work, we described existing hardware implementations of these novel security primitives. We discussed emerging concepts in the area, including Timed Authentication and Public PUFs and SIMPL systems. We complemented the chapter by a treatment of future research challenges, which could prove helpful as a guideline to graduate students or anyone who wants to conduct research in the area.

Acknowledgement The authors would like to thank Prof. Wayne P. Burleson for his valuable comments and suggestions. The authors would also like to thank Azalia Mirhoseini for her help with some of the figures. Ulrich Rührmair and Farinaz Koushanfar have equally contributed to this book chapter.

References

1. Gassend B, Clarke D, van Dijk M, Devadas S (2002) Silicon physical random functions. In: Computer and Communication Security Conference
2. Gassend B (2003) Physical random functions, Master's thesis, Massachusetts Institute of Technology
3. Suh G, O'Donnell C, Devadas S (2007) AEGIS: a Single-Chip secure processor. IEEE Design Test Comput 24(6): 570–580
4. Yin C, Qu G (2010) LISA: maximizing RO PUF's secret extraction. In: Hardware-Oriented Security and Trust (HOST), pp 100–105
5. Kumar S, Guajardo J, Maes R, Schrijen G-J, Tuyls P (2008) Extended abstract: the butterfly PUF protecting IP on every FPGA. In: Hardware-Oriented Security and Trust (HOST), pp 67–70
6. Maes R, Tuyls P, Verbauwhede I (2009) Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs. In: Cryptographic Hardware and Embedded Systems (CHES), pp 332–347

7. Yu M-DM, Devadas S (2010) Secure and robust error correction for physical unclonable functions. In: IEEE Design Test Comput 27: 48–65
8. Majzoobi M, Koushanfar F, Devadas S (2010) FPGA PUF using programmable delay lines.In: IEEE Workshop on Information Forensics and Security, in press
9. Majzoobi M, Koushanfar F (2011) Time-bounded authentication of FPGAs. In: Under Revision for IEEE Transactions on Information Forensics and Security (TIFS)
10. Rührmair U, Jaeger C, Hilgers C, Algasinger M, Csaba G, Stutzmann M (2010) Security applications of diodes with unique current–voltage characteristics. In: Financial Cryptography and Data Security (FC), pp 328–335
11. Suh G, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference (DAC), pp 9–14
12. Sadeghi A, Naccache D (eds) (2010) Towards Hardware-Intrinsic Security: Foundations and Practice. Springer, Berlin, Heidelberg, New York
13. Kirovski D (2010) Anti-counterfeiting: mixing the physical and the digital world. In: Sadeghi A-R, Naccache D (eds) Towards Hardware-Intrinsic Security. Springer, Berlin, Heidelberg, New York, pp 223–233
14. Li S, Jain A (eds) (2009) Encyclopedia of Biometrics. Springer, USA
15. Maltoni D, Maio D, Jain A, Prabhakar S (2009) Handbook of Fingerprint Recognition. Springer, London
16. Kirovski D (2008) Personal communication, Dagstuhl, Germany
17. Graybeal S, McFate P (1989) Getting out of the STARTing block. Scient Am (USA) 261(6): 64–65
18. Bauder D (1983) An anti-counterfeiting concept for currency systems. Research report PTK-11990. Sandia National Laboratories, Albuquerque, N.M
19. Brosow J, Furugard E (1980) Method and a system for verifying authenticity safe against forgery. US Patent 4,218,674
20. Simmons G (1984) A system for verifying user identity and authorization at the point-of sale or access. Cryptologia 8(1): 1–21
21. ——, (1991) Identification of data, devices, documents and individuals. In: IEEE International Carnahan Conference on Security Technology, pp 197–218
22. Buchanan J, Cowburn R, Jausovec A, Petit D, Seem P, Xiong G, Atkinson D, Fenton K, Allwood D, Bryan M (2005) Forgery:fingerprintingdocuments and packaging. Nature 436(7050): 475
23. Smith J, Sutherland A (1999) Microstructure based indicia. Proc Automatic Identification Adv Technol AutoID 99: 79–83
24. Métois E, Yarin P, Salzman N, Smith J (2002) FiberFingerprint identification. In: Workshop on Automatic Identification, pp 147–154
25. Seem P, Buchanan J, Cowburn R (2009) Impact of surface roughness on laser surface authentication signatures under linear and rotational displacements. Optic Lett 34(20): 3175–3177
26. Sharma A, Subramanian L, Brewer E (2008) Secure rural supply chain management using low cost paper watermarking. In: ACM SIGCOMM workshop on Networked systems for developing regions, pp 19–24
27. Beekhof F, Voloshynovskiy S, Koval O, Villan R, Pun T (2008) Secure surface identification codes. In: Proceedings of SPIE, vol 6819, p 68190D
28. Clarkson W, Weyrich T, Finkelstein A, Heninger N, Halderman J, Felten E (2009) Fingerprinting blank paper using commodity scanners. In: IEEE Symposium on Security and Privacy, pp 301–314
29. The ProteXXion System,Bayer AG, http://www.research.bayer.com/edition-19/protexxion.aspx and http://www.research.bayer.com/edition-19/19_Protexxion_en.pdfx
30. Ingeniatechnology, http://www.ingeniatechnology.com/
31. DeJean G, Kirovski D (2007) RF-DNA: Radio-frequency certificates of authenticity. Cryptographic Hardware and Embedded Systems (CHES), pp 346–363

32. Kirovski D (2004) Toward an automated verification of certificates of authenticity. In: ACM Electronic Commerce (EC), pp 160–169
33. Chen Y, Mihçak M, Kirovski D (2005) Certifying authenticity via fiber-infused paper. ACM SIGecom Exchanges 5(3): 29–37
34. Bulens P, Standaert F, Quisquater J (2010) How to strongly link data and its medium: the paper case. IET Information Security 4(3): 125–136
35. Kariakin Y (1995) Authentication of articles. Patent writing, WO/1997/024699, available from http://www.wipo.int/pctdb/en/wo.jsp?wo=1997024699
36. Hammouri G, Dana A, Sunar, B (2009) CDs have fingerprints too. Cryptographic Hardware and Embedded Systems (CHES), pp 348–362
37. Vijaywargi D, Lewis D, Kirovski D (2009) Optical DNA. Financial Cryptography and Data Security (FC), pp 222–229
38. Zhu B, Wu J, Kankanhalli M (2003) Print signatures for document authentication. In: Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS). ACM, New York, pp 145–154
39. Collins J (2004) RFID fibers for secure applications. RFID J 26
40. RF SAW Inc. http://www.rfsaw.com/tech.html
41. Creo Inc. http://www.creo.com
42. Inkode Inc. http://www.inkode.com
43. Microtag Temed Ltd. http://www.microtag-temed.com/
44. CrossID Inc., Firewall Protection for Paper Documents. http://www.rfidjournal.com/article/articleview/790/1/44
45. Loibl C (2009) Entwurf und Untersuchung berührungslos abfragbarer einzigartiger Objekte. Master's thesis, Fachgebiet Höchstfrequenztechnik, Technische Universität München
46. MagnePrint. http://www.magneprint.com/
47. Rührmair U, Stutzmann M, Lugli P, Jirauschek C, Müller K, Langhuth H, Csaba G, Biebl E, Finley J (2009) Method and system for security purposes. European Patent Application Nr. EP 09 157 041.6
48. Clelland C, Risca V, Bancroft C (1999) Hiding messages in DNA microdots. Nature 399(6736): 533–534
49. November AG. http://www.november.de/archiv/pressemitteilungen/pressemitteilung/article/sichere-medikamente-dank-dna-codes-der-identif-gmbh%.html
50. Kirovski D (2005) A point-set compression heuristic for fiber-based certificates of authenticity. In: Data Compression Conference (DCC), pp 103–112
51. ——, (2004) Point compression for certificates of authenticity. In: Data Compression Conference (DCC), p 545
52. Dodis Y, Reyzin L, Smith A (2004) Fuzzy extractors: how to generate strong keys from biometrics and other noisy data. In: Advances in cryptology-Eurocrypt. Springer, Berlin, Heidelberg, New York, pp 523–540
53. Alliance for Gray Market and Counterfeit Abatement (AGMA), http://www.agmaglobal.org/
54. Rührmair U, Sehnke F, Sölter J, Dror G, Devadas S, Schmidhuber J (2010) Modeling attacks on physical unclonable functions. In: ACM Conference on Computer and Communications Security (CCS), pp 237–249
55. Bennett C, Brassard G, Breidbart S, Wiesner S (1983) Quantum cryptography, or unforgeable subway tokens. In: Advances in Cryptology–Proceedings of Crypto, vol 82, pp 267–275
56. Bennett C, Brassard G et al. (1984) Quantum cryptography: public key distribution and coin tossing. In: International Conference on Computers, Systems and Signal Processing, vol 175. Bangalore, India
57. Guajardo J, Kumar S, Schrijen G, Tuyls P (2007) FPGA intrinsic PUFs and their use for IP protection. In: Cryptographic Hardware and Embedded Systems (CHES), pp 63–80
58. Lofstrom K, Daasch WR, Taylor D (2000) Ic identification circuit using device mismatch. In: ISSCC, pp 372–373
59. Layman P, Chaudhry S, Norman J, Thomson J (2002) Electronic fingerprinting of semiconductor integrated circuits. US Patent 6,738,294

60. Su Y, Holleman J, Otis B (2007) A 1.6pJ/bit 96 (percent) stable chip ID generating circuit using process variations. In: IEEE International Solid-State Circuits Conference (ISSCC), pp 200–201
61. Holcomb D, Burleson W, 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
62. Tuyls P, Schrijen G-J, Skoric B, van Geloven J, Verhaegh N, Wolters R (2006) Read-proof hardware from protective coatings. In: Cryptographic Hardware and Embedded Systems (CHES), pp 369–383
63. Helinski R, Acharyya D, Plusquellic J (2009) A physical unclonable function defined using power distribution system equivalent resistance variations. In: Design Automation Conference (DAC), pp 676–681
64. ——, (2010) Quality metric evaluation of a physical unclonable function derived from an IC's power distribution system. In: Design Automation Conference, ser. DAC, pp 240–243
65. Suh GE (2005) AEGIS: a Single-Chip Secure Processor. Ph.D. dissertation, Massachusetts Institute of Technology
66. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX Security Symposium, pp 291–306
67. Holcomb D, Burleson W, Fu K (2009) Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Trans Comput 58(9): 1198–1210
68. Pappu R (2001) Physical one-way functions. Ph.D. dissertation, Massachusetts Institute of Technology
69. Pappu R, Recht B, Taylor J, Gershenfeld N (2002) Physical one-way functions. Science 297: 2026–2030
70. Gassend B, Clarke D, van Dijk M, Devadas S (2002) Controlled physical random functions. In: Annual Computer Security Applications Conference
71. Rührmair U, Sehnke F, Sölter J (2009) On the foundations of physical unclonable functions. Cryptology ePrint Archive, International Association for Cryptologic Research, Tech. Rep.
72. Rührmair U, Busch H, Katzenbeisser S (2010) Strong PUFs: models, constructions, and security proofs. In: Sadeghi A-R, Naccache D (eds) Towards Hardware-Intrinsic Security. Springer, Berlin, Heidelberg, New York, pp 79–96
73. Gassend B, Clarke D, van Dijk M, Devadas S (2003) Delay-based circuit authentication and applications. In: Symposium on Applied Computing (SAC)
74. Lee J-W, Lim D, Gassend B, Suh GE, van Dijk M, Devadas S (2004) A technique to build a secret key in integrated circuits with identification and authentication applications. In: IEEE VLSI Circuits Symposium, New-York
75. Lim D (2004) Extracting Secret Keys from Integrated Circuits. Master's thesis, Massachusetts Institute of Technology, Cambridge, USA
76. Gassend B, Lim D, Clarke D, van Dijk M, Devadas S (2004) Identification and authentication of integrated circuits. Concurrency and Computation: Practice and Experience 16(11): 1077–1098
77. Majzoobi M, Koushanfar F, Potkonjak M (2009) Techniques for design and implementation of secure reconfigurable pufs. ACM Trans Reconfig Technol Syst (TRETS) 2(1): 1–33
78. Devadas S, Suh E, Paral S, Sowell R, Ziola T, Khandelwal V (2008) Design and implementation of PUF-based unclonable RFID ICs for anti-counterfeiting and security applications. In: Proceedings of 2008 IEEE International Conference on RFID (RFID 2008), pp 58–64
79. Suzuki D, Shimizu K (2010) The Glitch PUF: a new delay-PUF architecture exploiting glitch shapes. Cryptographic Hardware and Embedded Systems (CHES), pp 366–382
80. Devadas S, Gassend B (2010) Authentication of integrated circuits. US Patent 7,840,803, application in 2002
81. 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
82. Potkonjak M, Koushanfar F (2009) Identification of integrated circuits. US Patent Application 12/463,984; Publication Number: US 2010/0287604 A1

83. Koushanfar F, Boufounos P, Shamsi D (2008) Post-silicon timing characterization by compressed sensing. In: International Conference on Computer-Aided Design (ICCAD), pp 185–189
84. Shamsi D, Boufounos P, Koushanfar F (2008) Noninvasive leakage power tomography of integrated circuits by compressive sensing. In: International Symposium on Low Power Electronic Designs (ISLPED), pp 341–346
85. Nelson M, Nahapetian A, Koushanfar F, Potkonjak M (2009) Svd-based ghost circuitry detection. In: Information Hiding (IH), pp 221–234
86. Wei S, Meguerdichian S, Potkonjak M (2010) Gate-level characterization: foundations and hardware security applications. In: Design Automation Conference (DAC), pp 222–227
87. Koushanfar F, Mirhoseini A (2011) A unified framework for multimodal submodular integrated circuits trojan detection. In: IEEE Transactions on Information Forensic and Security (TIFS)
88. Csaba G, Ju X, Ma Z, Chen Q, Porod W, Schmidhuber J, Schlichtmann U, Lugli P, Ruhrmair U (2010) Application of mismatched cellular nonlinear networks for physical cryptography. In: International Workshop on Cellular Nanoscale Networks and their Applications (CNNA). IEEE, pp 1–6
89. Tuyls P, Škorić B (2007) Strong authentication with physical unclonable functions. In: Security, Privacy, and Trust in Modern Data Management, pp 133–148
90. Rührmair U (2011) SIMPL systems, or: can we construct cryptographic hardware without secret key information? In: International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), ser. Lecture Notes in Computer Science, vol 6543. Springer, Berlin, Heidelberg, New York
91. Rührmair U, Jaeger C,Algasinger M (2011) An attack on PUF-based session key exchange and a hardware-based countermeasure. Financial Cryptography and Data Security (FC) to appear
92. Majzoobi M, Nably AE, Koushanfar F (2010) FPGA time-bounded authentication. In: Information Hiding Conference (IH), pp 1–15
93. Bekenstein J (2005) How does the entropy/information bound work? Found Phys 35(11): 1805–1823
94. Oztürk E, Hammouri G, Sunar B (2008) Towards robust low cost authentication for pervasive devices. In: Pervasive Computing and Communications (PerCom), pp 170–178
95. Majzoobi M, Koushanfar F, Potkonjak M (2008) Testing techniques for hardware security. In: International Test Conference (ITC), pp 1–10
96. ——, (2008) Lightweight secure PUF. In: International Conference on Computer Aided Design (ICCAD), pp 670–673
97. Rührmair U, Jaeger C, Bator M,Stutzmann M, Lugli P, Csaba G Applications of high-capacity crossbar memories in cryptography, In IEEE Transactions on Nanotechnology, no. 99, p 1
98. Jaeger C, Algasinger M, Rührmair U, Csaba G, Stutzmann M (2010) Random pn-junctions for physical cryptography. Appl Phys Lett 96: 172103
99. Tuyls P, Skoric B, Stallinga S, Akkermans AHM, Ophey W (2005) Information-theoretic security analysis of physical uncloneable functions. In: Financial Cryptography and Data Security (FC), pp 141–155
100. Škorić B (2008) On the entropy of keys derived from laser speckle; statistical properties of Gabor-transformed speckle. J Optics A Pure Appl Optic 10(5): 055304
101. Skoric B, Maubach S, Kevenaar T, Tuyls P (2009) Information-theoretic analysis of capacitive physical unclonable functions. J Appl Phys 100(2): 024902
102. Kim I, Maiti A, Nazhandali L, Schaumont P, Vivekraja V, Zhang H (2010) From statistics to circuits: foundations for future physical unclonable functions. Towards Hardware-Intrinsic Security, pp 55–78
103. Sehnke F,Schmidhuber J, Rührmair U (2010) Security benchmarks for strong physical unclonable functions, in submission
104. Gassend B, van Dijk M, Clarke D, Torlak E, Devadas S, Tuyls P (2008) Controlled physical random functions and applications. ACM Trans Inform Syst Secur (TISSEC), 10(4): 1–22

105. Yee BS (1994) Using secure coprocessors. Ph.D. dissertation, Carnegie Mellon University
106. Carroll A, Juarez M, Polk J, Leininger T (2002) Microsoft palladium: a business overview. In: Microsoft Content Security Business Unit, http://www.microsoft.com/presspass/features/2002/jul02/0724palladiumwp.asp
107. Alves T, Felton D (2004) Trustzone: Integrated Hardware and Software Security, ARM white paper
108. Microsoft, Next-Generation Secure Computing Base. http://www.microsoft.com/resources/ngscb/defaul.mspx
109. Group TC (2004) Tcg specification architecture overview revision 1.2. http://www.trustedcomputinggroup.com/home
110. Lie D, Thekkath C, Mitchell M, Lincoln P, Boneh D, Mitchell J, Horowitz M (2000) Architectural support for copy and tamper resistant software. In: International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX), pp 168–177
111. Lie D (2003) Architectural support for copy and tamper-resistant software. Ph.D. dissertation, Stanford University, Menlo Park, CA, USA
112. Suh GE, Clarke D, Gassend B, van Dijk M, Devadas S (2003) AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: International Conference on Supercomputing (MIT-CSAIL-CSG-Memo-474 is an updated version)
113. Suh GE, O'Donnell CW, Sachdev I, Devadas S (2005) Design and implementation of the AEGIS single-chip secure processor using physical random functions. In: International Symposium on Computer Architecture (ISCA)
114. Devadas S (2008) Non-networked rfid puf authentication. US Patent Application 12/623,045
115. Oztiirk E, Hammouri G, Sunar B (2008) Towards robust low cost authentication for pervasive devices. In: International Conference on Pervasive Computing and Communications (PerCom), pp 170–178
116. Beckmann N, Potkonjak M (2009) Hardware-based public-key cryptography with public physically unclonable functions. In: Information Hiding. Springer, Berlin, Heidelberg, New York, pp 206–220
117. Potkonjak M (2009) Secure authentication. US Patent Application 12/464,387; Publication Number: US 2010/0293612 A1
118. ——, (2009) Digital signatures. US Patent Application 12/464,384; Publication Number: US 2010/0293384 A1
119. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. In: International Workshop on Information Hiding (IHW), pp 81–95
120. Koushanfar F, Potkonjak M (2007) Cad-based security, cryptography, and digital rights management. In: Design Automation Conference (DAC), pp 268–269
121. Potkonjak M, Meguerdichian S, Wong J (2010) Trusted sensors and remote sensing. In: IEEE Sensors, pp 1–4
122. Rührmair U, Stutzmann M, Csaba G, Schlichtmann U, Lugli P (2009) Method for security purposes. European Patent Filings EP 09003764.9, EP 09003763.1, EP 09157043.2
123. Rührmair U (2009) SIMPL Systems: on a public key variant of physical unclonable functions. Cryptology ePrint Archive, International Association for Cryptologic Research, Tech. Rep.
124. Rührmair U, Chen Q, Stutzmann M, Lugli P, Schlichtmann U, Csaba G (2009) Towards electrical, integrated implementations of simpl systems, cryptology ePrint archive. International Association for Cryptologic Research, Tech. Rep.
125. Chen Q, Csaba G, Ju X, Natarajan S, Lugli P, Stutzmann M, Schlichtmann U, Ruhrmair U (2009/2010) Analog circuits for physical cryptography. In: 12th International Symposium on Integrated Circuits (ISIC'09), IEEE. Singapore, 14–16 December 2009 pp 121–124
126. Rührmair U, Chen Q, Stutzmann M, Lugli P, Schlichtmann U, Csaba G (2010) Towards electrical, integrated implementations of simpl systems, In: Workshop in Information Security Theory and Practice (WISTP), pp 277–292
127. Chen Q, Csaba G, Lugli P, Schlichtmann U, Stutzmann M, Rührmair U (2011) Circuit-based approaches to SIMPL systems. J Circ Syst Comput 20: 107–123

128. Rührmair U (2011) SIMPL Systemsas a Cryptographic and Security Primitive, In To be submitted to IEEE Trans. on Information Forensics and Security (TIFS)
129. Škorić B (2010) Quantum readout of physical unclonable functions. In: Progress in Cryptology–AFRICACRYPT 2010, pp 369–386
130. Ékoric B (2010) Quantum readout of physical unclonable functions. In: Progress in Cryptology (AFRICACRYPT), ser. Bernstein D, Lange T (eds) Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York, vol 6055, pp 369–386
131. Bösch C, Guajardo J, Sadeghi A, Shokrollahi J, Tuyls P (2008) Efficient helper data key extractor on FPGAs. In: Cryptographic Hardware and Embedded Systems (CHES), pp 81–197
132. Koushanfar F, Qu G (2001) Hardware metering. In: Design Automation Conference (DAC), ser. DAC, pp 490–493
133. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: ICCAD
134. Rührmair U (2010) Oblivious transfer based on physical unclonable functions (extended abstract). In: Acquisti A, Smith SW, Sadeghi A-R (eds) TRUST, ser. Lecture Notes in Computer Science, vol 6101. Springer, Berlin, Heidelberg, New York, pp 430–440

Chapter 5
Hardware Metering: A Survey

Farinaz Koushanfar

5.1 Introduction

As the integrated circuits scale in feature sizes and exponentially grow in functionality, they also become increasingly complex. The sophisticated chips in design and fabrication today require costly, intricate, and complex processes that can only be performed in state-of-art fabrication facilities. Building or maintaining such facilities for the present CMOS technology is reported to be more than 3 Billion dollars and growing, described as the most expensive factories ever built by humankind [1, 2]. Given this increasingly glowing cost and complexity of foundries and their processes, the semiconductor business model has largely shifted to a contract foundry business model (a.k.a. horizontal business model) over the past two decades. For example, Texas Instruments (TI) and Advanced Micro Devices (AMD), two chip making giants that have traditionally used their in-house facilities for fabricating their chips, have both recently announced outsourcing most of their sub-45 nm fabrication to major contract foundries worldwide.

In the horizontal business model, the relationship between the designer and the foundry is *asymmetric*: the designed IP is transparent to the manufacturers who can reproduce (overbuild) the ICs with a negligible overhead because of the ready availability of the masks; but the details of the fabrication process, quantity, and possible modifications to the original designer's chip blueprint (in form of layout files such as OASIS format) are clandestine to the design house. The existing business model and contract agreements are insufficient for full protection of the designer IP rights [3, 4]. The IP owners disclose the details of their IP and also pay for building costly masks based on their designs; they trust the foundry not to pirate their designs and not to overbuild more ICs. The mask's ready availability

F. Koushanfar (✉)
Electrical and Computer Engineering Department, Rice University, Houston, Texas 77251-1892, USA
e-mail: farinaz@rice.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_5, © Springer Science+Business Media, LLC 2012

at the foundry, relative low cost of silicon, and lack of designer's (IP rights owner's) control over the foundry fabrication further ease piracy. The internal of the manufactured ICs are opaque because of the design's multiple layers and its complexity; interface circuitry that only allows limited external access to certain internal chip components; and the packaging. What exacerbates the problem is that the ICs are exploited for multiple applications with strict anticloning and security requirements, including but not limited to bank cards, security devices, and weapons. Given the criticality of applications and designer's huge losses to piracy, preventing IC theft, piracy, and overbuilding by the contract foundries have become increasingly crucial from the government, industry, business, and consumer point of views.

IC metering is a set of security protocols that enable the design house to achieve postfabrication control over their ICs. The term "hardware metering" was first coined in 2001 [5, 6] to refer to the first passive method for uniquely tagging each IC's functionality while maintaining the same input/output behavior and synthesis flow. Around the same time and independently, methods for unclonable identification of ICs including physical unclonable functions (PUFs) have been in development [7,8]. Since then, multiple newer methods for giving a design house (IP rights over) control over their designs have been proposed for metering. Note that the levels of postfabrication protection and control are dependent on the attack model, the desired protection level, and the security assumptions for the IC supply chain.

The focus of this chapter is on hardware metering. It is important to emphasize the difference between hardware watermarking and hardware metering. While metering attempts to uniquely tag each chip produced from a certain design by active or passive methods to facilitate tracing the chips, watermarking's objective is to uniquely tag the design itself so that all the chips produced by the same design/mask would carry the same watermark. Therefore, watermarking is not a good countermeasure against overbuilding, since it cannot distinguish among the different chips fabricated by similar masks. Hardware metering, provides a way to uniquely fingerprint or tag each chip and/or each chip's functionality, so it is possible to distinguish between the different chips manufactured by the same mask. A full coverage of watermarking is outside the scope of this chapter. For a more comprehensive coverage, we refer the interested readers to an excellent book [9], and several important papers on this topic [10–15].

The remainder of the chapter is organized in the following way. Section 5.2 outlines a new taxonomy for the work in metering, beyond the traditional well-known passive and active classifications. Section 5.3 covers the work in passive metering including nonfunctional reproducible metering, nonfunctional unclonable metering, and functional metering. In Sect. 5.4, we outline and discuss the research in active metering which covers both the methods based on combinational and sequential locking and control embedded within the design, and also the methods that require external cryptography module(s) in addition to the combinational/sequential locking. Lastly, Sect. 5.5 concludes our metering discussions.

5.2 Taxonomy and Models

As mentioned earlier, metering has been classified into two categories, passive and active [45]. Passive metering provides a way for unique identification of a chip, or for specifically tagging an IC's functionality so that it can be passively monitored. Rudimentary passive metering methods have been used for many decades, by physically indenting a serial number on each device, or by storing the identifiers in the permanent memory. We call the former method as *indented serial numbers*, while the second method is called the *digitally stored serial numbers*. Both methods can be classified as *nonfunctional identification* methods, as the unique ID is separate from the chip's functionality. Since serial numbers (both indented and digital) can be easily copied and placed on new chips, we subclassify both methods to the *reproducible nonfunctional identification* category.

Because of vulnerability of the serial numbers and digital identification numbers to cloning and removal attacks, about a decade ago, the ICID approach introduced methods for generating unclonable IDs based on the inherent random process variations of the silicon [7]. As the randomness is existing in the process and cannot be controlled or cloned, we refer to this class of identification as *unclonable identification* or *intrinsic fingerprint extraction*. Unclonable identifiers (IDs) are a form of Physical Unclonable Functions (PUFs) that were comprehensively discussed and classified in Chap. 7 of this book. According to the classification provided in the referred chapter *weak PUFs* (a class of PUFs that is able to generate secret keys) can be used as the unclonable IDs for IC metering. We subclassify such chip identification methods based on the inherent process variation as *unclonable nonfunctional identification*.

Shortly after the introduction of ICID, a fundamentally new way of passive metering was introduced [5, 6]. In this method, the identifiers were linked to the chip's internal functional details during the synthesis step, such that each chip's function would get a unique signature. We refer to this type of passive metering as *functional metering*. Both unclonable and reproducible identifiers can be interfaced to the chip's functionality to make a unique signature for it. Note that the functionality would remain unchanged from the input/output standpoint, and only a set of internal transactions would be unique to each chip.

Most passive metering methods described so far, rely on an added component, or changing the design for holding the identifiers, or presynthesis modifications for functional metering. A passive metering method that can uniquely identify each chip with addition of components or modifications to the design is called *extrinsic*. In contrast, an *intrinsic* passive metering methods do not need any added components or design modifications. The big advantage of intrinsic identification methods is that since they do not rely on an added component, they can be readily used on existing legacy designs. The intrinsic identification may be based on digital or analog values that are caused by the random physical disorder of the phenomena, in this case the inherent silicon process variations.

An example for an intrinsic digital identification is a weak PUF based on SRAM, where the existing SRAM memory cells inside the FPGA are used as unclonable IDs [16]. An example for an intrinsic analog ID that is applicable to both ASIC and FPGA, is a nondestructive method for extracting the existing variation signatures of the embedded circuits (caused by the inherent process variations) [17, 18]. The extracted signatures can be used as a fingerprint for each device. While the focus of the earlier work was on timing signatures [17], the more recent work has shown that the signatures from the other side-channels, such as IDDQ and IDDT measurements can be also used for intrinsic chip identification [18]. A unified framework by gate-level translation of the process variation components was formed and therefore, a presentation of the chip's unique signatures in the gate-level domain became possible.

Active metering, in addition to unique identification or remote passive monitoring of a device, provides an active way for the designers to enable, control, and disable the device. The term active metering was first coined in [19] where the authors introduced the first known method for actively controlling the ASIC chips.

Active metering enriched the realm of functional metering, by hiding states and transitions in the design that can only be accessed by the original designer. Very recent research results on active metering has shown that the original method introduced in [19] can be constructed to be provably secure by showing a transformation of the provably obfuscatable family of generalized point functions [20]. As the states and transitions used for controlling (also called locking and unlocking) of the chips are integrated within the functional specification of the design, we refer to this type of metering as *internal active hardware metering*.

Since the introduction of the original active hardware metering in [19], a number of other interesting methods for this purpose have been proposed. Aside from the internal active hardware metering methods that only design modifications (sequential or combinational) for lock embedding [21], other methods of active metering based on inclusion of external cryptography circuits were introduced [22–25]. We refer to this type of active metering as *external active hardware metering*. Both internal and external active hardware metering exploit a random identifier in a digital form that may or may not be unclonable. For example, as the digital random identifier, burned fuses can be used. Note that burned fuses are reproducible at the foundry, and therefore, they cannot be used as a countermeasure for the foundry piracy attack. However, they may not be reproducible by average customers who may not have access to the fuses without depackaging and invasively probing the chips.

Figure 5.1 demonstrates a summary of the taxonomy described in this section. Notice on the chart the distinction between the analog and the digital identification for both reproducible and unclonable IDs. Although both analog and digital identification are possible, the existing methods for passive metering based on functional identification and active metering (internal and external) all use the digital identifiers. This is because the digital IDs can be readily integrated within the logical flow. In the remainder of the chapter, we focus on detailed description of the mechanisms and structures that have been proposed for passive and active metering.

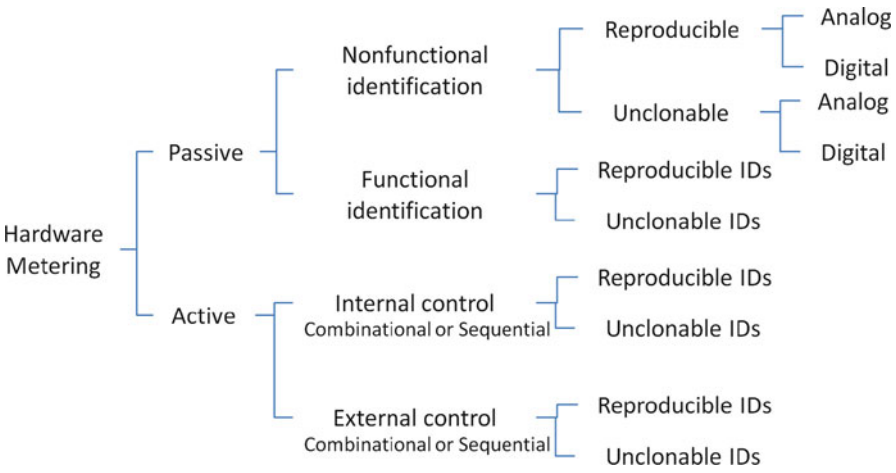


Fig. 5.1 A Taxonomy of metering methods

5.3 Passive IC Metering

In this section, we separately review the work in nonfunctional and functional metering and fingerprinting of circuits.

5.3.1 Passive Metering by Nonfunctional Identification

5.3.1.1 Reproducible Identifiers

There is no clear record of when the IC companies have started to indent IDs on the packages, or to have a separate piece of ID set aside for storing a digital identifier for their devices. Also, there is no clear public record of when/if the IC companies have used the digital IDs to monitor their devices in hands of the users. Also, burn-in fuses are used at the design houses for carving identifiers on the chips for identification purposes.

Perhaps the best known (and controversial) incident is the Intel Pentium III processors that were publicly announced to a include a unique identifier, called the *Processor Serial Number (PSN)*. The PSN could be used to monitor the user activities via the networks. Although Intel made a utility that would give the control over enabling/disabling the PSN to the device owners, it was demonstrated that rogue web sites were able to access even the disabled PSN. In 1999, several consumer privacy groups jointly filed a complaint against Intel with the Federal Trade Commission. After much debate over the privacy concerns, Intel quietly decided that the next generation of its processor, starting from the Williamette family, would not include the PSN. Pentium-III processors were not recalled from the market and they still have the PSN installed [26, 27].

As mentioned earlier, the drawback of indented IDs, digitally stored IDs, and burn-in fuses is that they can be tampered with, removed, or reproduced. As an example, a plausible and documented attack is repackaging of the older technology as a new one. Digitally stored IDs and serial numbers can be often read by noninvasive probing, and can be rewritten with a small effort. Also none of the available reproducible identification methods are able to withstand the foundry attack where the products are overbuilt. The foundry can simply reproduce an ID by writing to the memory, or by reproducing the numbers on the package.

5.3.1.2 Unclonable Identifiers

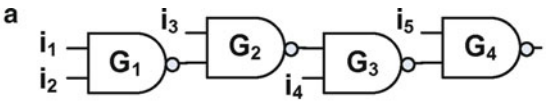
To overcome the limitations of the digital ID storage, methods for exploiting the random process variations in silicon to generate a random unique identifying (fingerprint) have been proposed. If exploiting the random process variations is done by introducing a new circuit structure, this is an instance of an extrinsic identification. If no new circuitry is introduced and only the existing on chip components are used for identification, the method would be an instance of an intrinsic identification. Note that the extrinsic/intrinsic distinction also holds for FPGAs: if the contents of the FPGA (without reconfiguring the cells) is used then the identifiers are intrinsic. Any use of reconfiguration of the devices would render the identification method an extrinsic one.

Extrinsic methods. The earliest known instance of this class of work was ICID [7], where an array of addressable transistors with common gate and source with sequentially selected drains was driving a resistive load. The transistors incurred threshold mismatches as a result of process variations. Therefore, the drain currents were randomly different, generating a sequence of unique random numbers on the transistors. Several work in more sophisticated identification and authentication circuit methods has followed this early work, including the physical unclonable functions that were later introduced [8, 17]. Chapter 7 of this book has a thorough discussion on PUFs and on other unique objects based on the random physical disorder, including a systematic and new classification of the various work in this area.

In this chapter, we limit the discussion on the other methods that can be used for passive and active metering. It is also possible to make a distinction between analog and digital extrinsic identifiers. For example, ICID generates random bits that are in digital form, and can be potentially readily included in a digital computation setting. It is plausible to design circuitry whose analog output depends on the underlying silicon process variations. As for many applications it is desired to have digital outputs that can be readily integrated within the remainder of the digital design, most of the extrinsic ID extraction circuitry have a digital response.

Intrinsic methods. The work in [17] has also proposed using the ICs' timing path signatures that are unique for each state-of-the-art CMOS chip (because of

a



b

NAND2	
Input	Leakage (nA)
00	37.84
01	100.3
10	95.7
11	454.5

c

Gate	Scaling factor	
	IC 1	IC 2
G_1	0.5	2.4
G_2	1.3	0.6
G_3	2.1	4
G_4	3	0.9

d

Input vector	Leakage (nA)	
	IC 1	IC 2
00011	1391	2055
10101	2082	1063
01110	1243	2150
11001	1841	1905

Fig. 5.2 (**a**) A design consisting of four NAND2 gates, (**b**) leakage current vs. input for NAND2, (**c**) scaling factors of gates on two ICs, and (**d**) total leakages of ICs for different input vectors (source [18])

process variations) as a PUF. The interesting studies and new results in [18,28] have shown that unique identification is possible for almost all ICs designed in state-of-the-art technology, as long as external test vectors can be applied and structural (side channel) tests such as leakage, timing, and dynamic power measurements can be performed. This feature, allows identification of all legacy ICs designed in technologies that contain manufacturing variability, without adding any components to the chip. The requirement is that test measurement equipment for performing the structural tests is available and can be used.

The proposed method in this latter work was based on the gate-level characterization of each IC that could be structurally tested. As the characterization is done nondestructively, there is no need for an additional circuitry. The extracted characteristics were continuous values, and multiple ways for discretizing and using the characteristics as unique chip IDs were suggested. Such nondestructive identification of legacy ICs can provide many opportunities for creation of new IC metering and security protocols. The other interesting note about using the structural tests is that for a given number of gates and their interconnections, there are many more test vectors, allowing one to generate an exponentially larger space of challenge-response pairs for one circuit.

Perhaps the best way to review the hidden characteristic extraction approach is by showing an example (figure from [18]). In Fig. 5.2a a small circuit consisting of four 2-input NAND gates. Figure 5.2b shows the static (leakage) current of a nominal NAND2 for different possible inputs. However, because of process variations, the leakage current greatly varies from one chip to another. Figure 5.2c shows scaling factors for the four gates in two example chips, and finally, Fig. 5.2d demonstrates the total leakage current for the two chips, respectively. Therefore, measurements from the total leakage over the different inputs are linearly decomposed to their gate-level components. Similar methods can be used for decomposing the timing measured for multiple test vectors and paths, and for separating the total dynamic current measured at the gate level.

Such multimodal gate level characterization methods have been further improved, not only for IC fingerprinting applications but also for accurate post-silicon characterization [29, 30] and for Trojan (malware) detection [31–35]. Another example of intrinsic identification methods are SRAM PUFs that are covered in detail in Chap. 7.

Aside from the work focusing on logic-level characteristic extraction of chips, later research in this area has shown that other forms of circuit parasitics, including the unique resistances on each chip is able to generate a unique response to the measurements [36, 37].

5.3.2 Passive Functional Metering

A notable progress in the field was the advent of methods for unique functional identification of chips. The first such known method was based on making the control path of each chip unique, so that each chip would have a specific internal control sequence. Despite the internal differences, the input and output behavior of all the chips coming from a single design and mask are the same. The challenge is fabricating the different chips from the same mask and the same design layout files. The work in [5, 6] proposed designing chips that have a single datapath that can be controlled by multiple versions of the same control path specifications. A small part of the chip is retained programmable, so the control path would be programmed into the chip post-silicon.

Subsequently, a new design methodology for realizing multiple control paths for one data path was suggested [5, 6]. For example, one solution was to permute the subsets of variables that are assigned to a particular register. To achieve multiplicity, during the logic synthesis step, redundant equivalent states are created for a selected set of states. The selection is based on the existing constraints on the concurrent states that should be stored in separate variables, with the goal of keeping the register overheads very low. Each copy of the variable would obtain a different state assignment, and any permutation of the duplicate assignments could be used for the equivalent states. As the state assignment is done by graph coloring, creation of redundant states would correspond to adding a vertex to the graph and replicating all the edges of the node to be duplicated for the new vertex. The state assignment for the modified graph can be solved by using the conventional graph coloring tools. Programmable read logic to the registers enables selecting the correct permutation of the variables for each unique copy of the control sequence.

5.3.2.1 Analysis of Passive Functional Metering

The passive metering protocol for detection of the unauthorized chips is to monitor and evaluate the chips while they are in use. Before testing an authorized chip, the programmable part is loaded with a specific permutation of the control path. Now,

if more than one copy of a single permutation is detected, a counterfeit component is flagged. This protocol would work well if many of the chips are online and can be queried for their permutation version of the internal control structure. One way to realize online querying is by XORing the states of the FFs to generate a checksum of the states, or by performing other variants of integrity checking on the states.

One interesting scenario for passive metering is where the chips are returned unprogrammed to the designer who would enter the controller specifications before testing the chips. The IP rights owner would ensure that each of the chips are uniquely programmed and that the foundry is not involved in the programming step. However, this approach by itself does not strongly deter the attackers, because an adversary with access to one unlocked chip can replicate the programmable memory's content from one chip and then use the information to configure and enable other chips. To avoid such direct replication attacks, the idea of integrating the programmable part with the unclonable IDs coming from the chip was also suggested. At the time of writing the first passive hardware metering paper in 2000, the only known unclonable identifiers where the ICIDs [7]. Therefore, the data for the programmable part could not be replicated on other chips, naturally defending against the overbuilding attacks.

The evaluation results in [5, 6] demonstrate that it is possible to obtain multiple permutations and selection of the internal ordering of the control sequences with a very low overhead. An obvious drawback of the presented passive metering approach is the overhead of adding the programmable part to ASICs, as this would require extra mask steps, incurring an additional cost. Two probabilistic analysis were presented for the original passive metering method: (1) the first set of analysis answers the question of how many experiments should be conducted before one can conclude the absence of unauthorized parts with a certain level of confidence; and (2) the second set of analysis aims at estimating the number of unauthorized copies made, in case duplicate chips are detected on the market. Because these two analysis methods are generalizable to many other metering and unique identification scenarios, in what follows we present the details of the analysis.

1. Assume that the design house demands the foundry to fabricate n copies, but the foundry indeed fabricates N chips where $N >> n$. If the company makes $k-1$ copies of each, the total number of available ICs from the design would be: $N = kn$. Note that it is proven that the foundry has the best chance of not getting detected by fabricating equal number of copies of each chip. If we draw l from the N objects consisting of k copies of distinct designs, the probability of no duplicate would be:

$$\text{Prob}[n,k,l] = \left[1 - \frac{k-1}{N-1}\right].\left[1 - \frac{2(k-1)}{N-2}\right] \ldots \left[1 - \frac{(l-1)(k-1)}{N-l-l}\right], \quad (5.1)$$

that is upper bounded by:

$$\text{Prob}[n,k,l] \leq \left[1 - \frac{p}{n}\right].\left[1 - \frac{2p}{n}\right] \ldots \left[1 - \frac{(l-1).p}{n}\right], \quad (5.2)$$

where $p = 1 - \frac{1}{k}$. As can be seen earlier, as the value of k increases, the probability Prob$[n, k, l]$ of not finding unauthorized parts after l random tests (without replacement) decreases. The probability Prob$[n, k, l]$ decreases as the number of tests l, increases. In essence, the quantity $1 - \text{Prob}[n, k, l]$ measures the foundry's honesty and it increases as l increases. For a designer to obtain a desired level of confidence α, one need to find the smallest l such that $(1 - \text{Prob}[n, k, l]) \geq \alpha$. Since finding an exact closed form formula for (5.1) is challenging, the solution is often found by numerical estimations or by using approximations in case of large n.

2. Assuming that k is uniformly distributed, one can immediately find the probability that the first unauthorized copy is found at the $l + 1$-th test as:

$$\text{Prob}[n, k, l + 1] = \text{Prob}[n, k, l].\frac{l.(l-1).(k-1)}{N-1}. \tag{5.3}$$

The expected number of tests to find the first unauthorized copy would be:

$$\sum_{k=1}^{\inf} \sum_{l=1}^{n(k-1)+1} l.\text{Prob}[n, k, l], \tag{5.4}$$

and if the first failure occurs at l then the expectation for k is:

$$E[k] = \sum_{k=1}^{\inf} k.\text{Prob}[n, k, l]. \tag{5.5}$$

5.4 Active IC Metering

Active hardware metering not only uniquely and unclonably identifies each chip but also provides an active mechanism to control, monitor, lock, or unlock the ICs post fabrication. To ensure irreproducibility, active metering requires a form of unclonable digital IC identifier such as a weak PUF [38]. One of the first presented applications of metering was for designer's IC enabling. Figure 5.3 demonstrates the global flow of the first known active hardware metering approach for enabling that was described in [19]. Similar IC enabling flows were later adopted for both internal and external active integrated circuits metering. There are typically two main entities involved: (1) a design house (a.k.a designer) that holds the IP rights for the manufactured ICs, and (2) 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 identify the best places to insert a lock. The subsequent design phases (e.g., RTL, synthesis, mapping, layout, and pin placement) take their routine courses. The foundry would receive the blueprint of the chip in form of OASIS

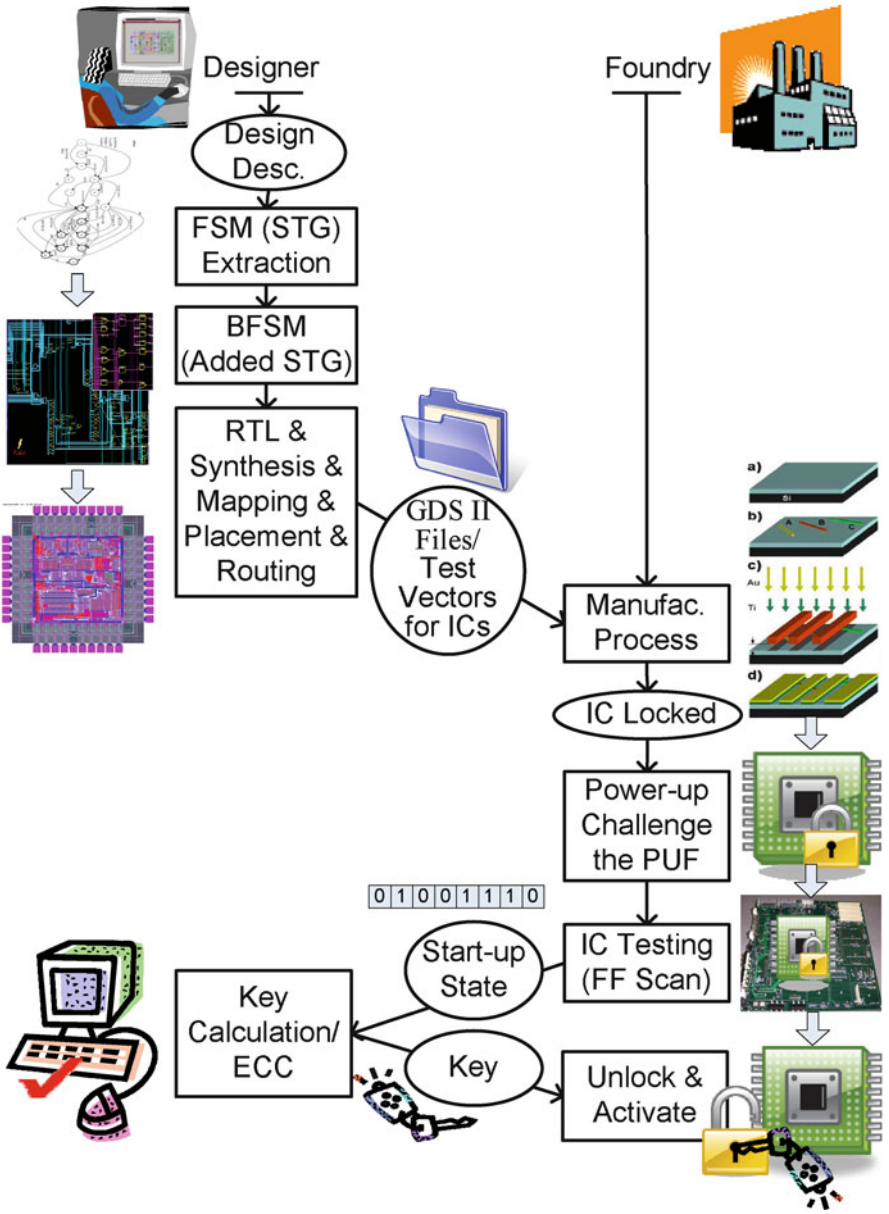


Fig. 5.3 The global flow of the IC enabling by active metering

files (or GDS-II) along with other required information for fabricating the chips including the test vectors. 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. Each IC typically contains an unclonable digital identifying unit, such as a weak PUF.

Building a mask is a costly and complex process, involving multiple fine steps that should be closely controlled [1, 2]. Once the foundry lithographs a mask, multiple ICs would 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 scans the unique identifier information out of each IC and sends the content back to the design house. The design house that uses the designer-specific knowledge or an asymmetric cryptography protocol, is the only entity who could compute the unlocking sequence for each locked chip. Additionally, the designer could compute the error correcting code (ECC) to correct for any further changes to the unclonable digital identifiers. The ECC is very important since a few of PUF response bits may be unstable and change at a later time because of noise, environmental conditions (e.g., temperature), or circuit instability. The key for unlocking the chip and the ECC would then be sent back to the fab.

The work in [19, 21] have also discussed methods such that the designer's asymmetric information about parts of the design could be utilized for other control purposes, including but not limited to online enabling/disabling and continuous authentication.

5.4.1 Internal (Integrated) Active IC Metering

The first set of introduced methods for metering were internal [19]. The active IC control mechanism in this class of work leverages: (1) the functional description of the design, and (2) unique and unclonable IC identifiers. The locks are embedded within the structure of the common computation model in hardware design, in form of a finite state machine (FSM). The designer exploits the high level design description to form the designs behavioral model in the FSM format. FSM is typically represented by the State Transition Graph (STG) where the vertices on the graph correspond to the states in the FSM, and the transitions between the FSM states are represented by the directed edges incident to the vertices. In the remainder of this chapter, we use the terms FSM and STG interchangeably. Let us describe the approach in [19]. We use the term *original FSM* to refer to the design's finite state machine before modifications (with $|S|$ states). Therefore, the original FSM could be implemented using $K = log|S|$ FFs.

Now assume that we modify the original FSM by augmenting to its states and transitions. We call the modified design a *boosted finite state machine (BFSM)*. To build a BFSM with $|S'| + |S|$ states, we would require $K" = log\{|S'| + |S|\}$ FFs. Additional edges are also introduced to the BFSM to ensure the reachability of its states. Observe that for a linear growth in the number of FFs denoted by $K' = K" - K$, the number of states exponentially increases. Indeed, by adding a number of FFs and tolerating the overhead of this addition, it is possible to set $S' >> S$ so that the number of new states are exponentially many more than $|S|$.

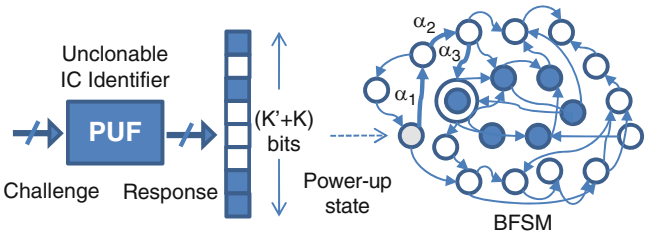


Fig. 5.4 The PUF response is fed to the FFs storing the states of the BSFM. The original states are shown in *dark*, and the added states are demonstrated in *white* color on the STG that represents the BFSM

The IC also contains a PUF unit that generates random bits based on the unclonable process variations of the silicon that are unique on 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.

Upon the IC's power up, the initial values of the design's FFs (i.e., *power-up state*) is determined by the unique response from the PUF on each chip. This is shown in the Fig. 5.4. 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 [8]. The number of added FFs can be set such that the value $2^{K"} >> 2^{K}$. In other words, the values $K"$ 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.

Because there are exponentially many 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 transitions from this nonfunctional initial power-up state to the functional *reset state* of the original FSM shown by double circle on the example.

The IP rights owners who have 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 find a path on the graph and use the input values corresponding to the path transition (from the STG description) so the states transition to the reset state. However, there is only one combination from exponentially many possibilities for the input of 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 are what define the designer's secret. The passkey for unlocking the chip is the sequence of inputs that can traverse the state of the BFSM (describing the control component

of the chip) from the initial random power-up state to the original 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. 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.

The recent results in [20] provide the first comprehensive set of proofs and a secure construction of the outlined internal active metering. The author shows 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 [39]. Even though heuristic methods for FSM obfuscation were proposed earlier, e.g., [40, 41], no provable security for such a constructions was available. The significance of the proposed construction and security proofs for the obfuscated FSM goes beyond hardware metering and extends to most previous work in information hiding and obfuscation of sequential circuits [40,41]. A detailed description and security proofs for this work is outside the scope of this paper. The method has been shown to be resilient against a spectrum of proposed attacks [20].

Another internal hardware metering method based on FSM modifications was proposed in [21]. The FSM modifications however, were drastically different than those described in [19]. Here, only a few states are added to the original FSM. Indeed, a few states of the FSM are selected for replication. A replicated state q_i' (i.e., a copy of an original state q_i) is an added state to the graph such that all the incident edges from/to the neighbors of q_i are copied to the state q_i'. Different passkeys are associated with the original edges and the replicated edges. The locking is performed such that only one of q_i or q_i' are selected for each chip by the random unique IDs of that chip. The ID can be scanned out from the chip. The provided input for edge transitions to and from the selected state is the one passkey that can perform unlocking.

An example for this operation is demonstrated in Fig. 5.5. One the figure, the overall schematic for an FSM with a lock on the replicated state (q_3 on the example) is demonstrated. Three replications of the state q_3, denoted by q_3', q_3'', and q_3''' are shown for the sake of the example. The response from the PUF determines the transition edge to one of the redundant states (either q_3 or one of its replicates). The response is also XOR'd with the passkey (computed by the original designer) to provide the transition out of the redundant state. Without the passkey, the transition will not take place or will be incorrect with a very high probability. Note that in real settings, the key length should be properly selected to thwart the brute-force attacks and guarantee security by point function. The significance of this new structure is that the locks are embedded within the internal states that are also visited during the IC's normal operation. Therefore, the locks and keys are continually accessed during the IC's normal operation, creating an opportunity for continuous authentication and self-checking.

It is interesting to note that FSM-based hardware metering method can be also used in the context of third party IP integration, where each of the IP cores on a chip

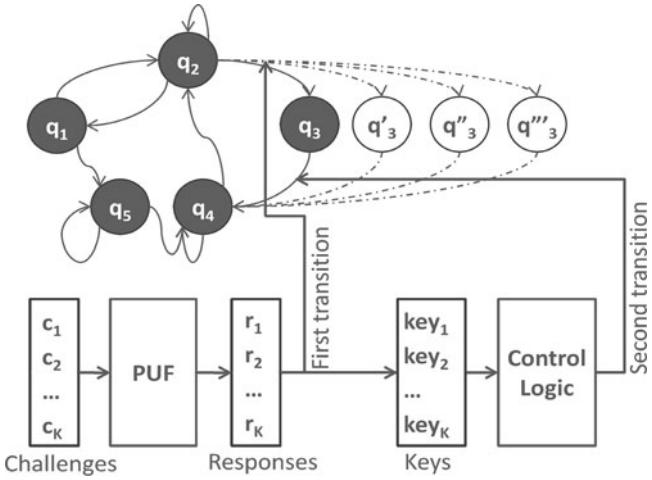


Fig. 5.5 Locking and unlocking by replicating a few states. The PUF response is used for the first transition to one of the redundant states (q_3, q_3', q_3'', q_3'''). The passkey provides the transition from the pertinent redundant state to the next state

can be enabled, disabled, or otherwise controlled [42]. In this approach, a designer or an integrator is the reuser of existing third party IP cores. Two other entities involved in this design and reuse protocol model are the fabrication plant and an authorized system verifier, referred to as a *certificate authority (CA)*. This latter entity a trusted third party who ensures a trust among the hardware IP providers, the reusers, and the fab.

Let us consider a scenario where multiple third party core blocks are to be protected. Each IP block contains a lock in its FSM structure. The resuer includes also two new modules in the design, the unclonable identification circuitry, and an embedded control module. The embedded control module interacts with and controls the various locks inside each third-party IP block. The blueprint of the design is sent to the CA before sending to fabrication. The CA certifies the IP core providers and the reuser. The post-silicon chips are tested for their unclonable IDs that are sent back to the CA. The CA contacts the third party IP providers and the reuser to get the passkeys for the IP block cores and the embedded control module. Note that similar third party IP protection models can be applied to cases where external active hardware metering is used for locking each core.

5.4.2 External Active IC Metering

External active IC metering methods lock every IC by asymmetric cryptographic techniques that require a specific external key. The use of asymmetric cryptography

for external IC metering was first proposed by EPIC [23]. As EPIC has been a basis for most of the subsequent work in external active metering, we discuss this methodology in detail.

To support Public Key Cryptography (PKC), the IP rights holder must generate a pair of master keys (MKs) (public and private) that will remain unaltered. The private master key (MK-Pri) embodies IP rights for a given design and is never transmitted. Each fabricated chip produces its own random public and private key pairs upon start-up. Also embedded in the register transfer level (RTL) are the public master key (MK-Pub) and the minimal circuitry to support the EPIC's combinational locking mechanism.

EPIC implements combinational locking in the chip's main modules by adding XOR gates on a number of selected noncritical wires, with added control signals connected to the common key (CK) register. When the correct CK is present, the circuit is equivalent to the original; otherwise, the chip's behavior is changed, as if stray inverters were located on selected wires. EPIC produces the CK at random to prevent it from being stolen earlier. After modifying the placed design, the designer securely communicates the CK to the IP rights holder and erases all other copies. Routing and other physical optimizations then proceed as normal, followed by manufacturing. Upon manufacturing, each IC will be uniquely locked because of the interface with the random and unclonable IDs generated by the IC.

While activating a chip, the foundry must have a secure link with the designer (IP rights holder) and must send the RCK-Pub for the IC to be activated. EPIC's protocol uses the foundry's private key to authenticate the transmission. Extensions to this protocol may send a time stamp, serial number, or other identifying sequences. In response, the designer (IP rights holder) transmits the input key (IK) that shows the CK encrypted with the PCK-Pub and signed by the MK-Pri afterward. The ordering of encryption and signing of the CK for generating the IK is crucial so that entities other than the designer (IP rights holder) cannot produce IKs, even if the CK is compromised. Using the RCK-Pub to encrypt the messages makes statistical attacks against the MK-Pri more complex. The designer can use the foundry's public key to additionally encrypt the resulting IK so that only the manufacturer can receive it. The IC decrypts the IK using the RCK-Pri and MK-Pub that authenticate it as being sent by the designer. Upon decryption, the CK is generated that unlocks the chip and facilitates testing. After the testing step, the IC can be sold.

EPIC is shown to be resilient against a number of proposed attacks, as described in [25]. Note that an early version of EPIC was evaluated by other researchers in terms of security and overhead [43]. They found that EPIC is vulnerable if the IK is calculated from the CK, MK-Pri, and RCK-Pub in the wrong order; the CK must first be encrypted with the PCK-Pub and then the resultant ciphertext must be signed by the MK-Pri that is a standard protocol for public-key communication with nonrepudiation. On the other hand, if the IK is computed properly, no successful logic-level attacks against EPIC are known. These issues were discussed and fully addressed in the latest version of EPIC [25].

Reference [24] presented an external IC locking method built upon secret sharing. The chip and the design plant share a secret key that is interfaced with the combinational logic on the circuit and is used for locking and controlling of the buses that can be used to connect and interface the multiple cores on one chip.

Reference [22] proposed an active IC metering approach that shared many of the ideas, protocols, and methods developed by EPIC. The work presented different design choices and an example on how to implement the protection scheme in resource-constrained environment. The low overhead approach was targeted to both device authentication and user authentication.

Reference [44] introduced another combinational locking method that like [5, 6] utilizes a small programmable part within the chip, that is referred to by *reconfigurable logic barriers*. However, unlike the earlier work that used the programmable parts for passive metering, the reconfigurable logic barriers in [44] were used for active metering. Reference [44] decomposes the IP functionality $F(x)$ into F_{fixed} and F_{reconfig}. The idea is to protect the design by not disclosing the F_{reconfig} to the fabrication plant. The withheld F_{reconfig} partition is programmed into the reconfigurable locations during the activation stage using a secure key distribution. The suggestion combinational locking method is a mix of XORs (similar to [23]) and k-input lookup tables (LUTs). A selection heuristic was proposed to optimize the lock placement. The heuristic is aimed at making a barrier for the logic transition, such that the signals would not traverse to the output without passing through the locks. Except for heuristic methods, no security guarantee was provided by this work.

As mentioned earlier, the advantage of such programmable parts is keeping a part of the design to the IP rights holder. The drawback is the added process and mask overhead incurred for implementing the programmable components within ASIC. For more technical details, we refer the interested readers to the published papers on the external active metering subject, including [22–25, 44].

5.5 Conclusions

This chapter provided a comprehensive overview of hardware integrated circuits (IC) protection by metering. IC metering refers to mechanisms, methods and protocols that enable tracking of the chips post-silicon. IC metering is motivated by the increased rate of outsourcing of leading-edge chip fabrication to offshore foundries that creates opportunities for overbuilding and piracy. IC metering helps the designers to identify and/or track their designs postfabrication. In our new taxonomy, hardware metering was divided into two categories: passive and active. Passive metering either assigns an identifier to the chip (which maybe reproducible or unclonable) or it assigns a signature to the internals of an IC, while the chip maintains its functionality and integrity. One interesting aspect of passive metering is that it is possible to uniquely monitor and track legacy ICs without the need for added circuitry by exploiting the inherent variations in silicon. In active metering, not only the ICs are uniquely identified but also parts of the chip's functionality

can be only accessed, locked (disabled), or unclocked (enabled) by the designer. We discussed both internal and external active hardware metering. Overall, the hope is that by limiting the opportunities for piracy and theft of ICs using post-silicon control mechanisms and protocols, hardware metering would be able to directly and significantly improve the business and practice of semiconductor design and manufacturing.

References

1. Mouli C, Carriker W (2007) Future fab: How software is helping intel go nano–and beyond. IEEE Spectrum 44(3): 38-43
2. Santo B (2007) Plans for next-gen chips imperiled. IEEE Spectrum 44(8): 12–14
3. Defense Science Board (DSB) study on high performance microchip supply (2005) http://www.acq.osd.mil/dsb/reports/ADA435563.pdf
4. Managing the risks of counterfeiting in the information technology industry (2005) White paper by KPMG and the alliance for gray market and counterfeit abatement (AGMA)
5. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. In: International Workshop on Information Hiding (IHW), Springer, Berllin, Heidelberg, New York, pp 81–95
6. Koushanfar F, Qu G (2001) Hardware metering. In: Design Automation Conference (DAC), Design Automation Conference (DAC), pp 490–493
7. Lofstrom K, Daasch WR, Taylor D (2000) *IC* identification circuit using device mismatch. In: International Solid-State Circuits Conference (ISSCC), pp 372–373
8. Gassend B, Clarke D, van Dijk M, Devadas S (2002) Silicon physical random functions. In: CCS, pp 148–160
9. Qu G, Potkonjak M (2003) In: Intellectual Property Protection in VLSI Design. Springer, Kluwer Publishing, New York, ISBN 1-4020-7320-8
10. Kirovski D, Potkonjak M (1999) Localized watermarking: methodology and application to operation scheduling. In: The International Conference on Computer-Aided Design (ICCAD), pp 596–599
11. Lach J, Mangione-Smith W, Potkonjak M (1998) Signature hiding techniques for FPGA intellectual property protection. In: The International Conference on Computer-Aided Design (ICCAD), pp 186–189
12. Torunoglu I, Charbon E (2000) Watermarking-based copyright protection of sequential functions. IEEE J Solid-State Circ (JSSC) 35(3): 434–440
13. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans Comput Aided Des Integr Circ Syst (TCAD) 20(9): 1101–1117
14. Koushanfar F, Hong I, Potkonjak M (2005) Behavioral synthesis techniques for intellectual property protection. ACM Trans Des Autom Electron Syst (TODAES) 10(3): 523–545
15. 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
16. Holcomb D, Burleson W, Fu K (2009) Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Transactions on Computers 58(9): 1198–1210
17. Devadas S, Gassend B (2002) Authentication of integrated circuits. Application in US Patent 7,840,803, 2010
18. Alkabani Y, Koushanfar F, Kiyavash N, Potkonjak M (2008) Trusted integrated circuits: a nondestructive hidden characteristics extraction approach. In: Information Hiding conference (IH). Springer, Berlin, Heidelberg, New York, pp 102–117
19. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX Security Symposium, pp 291–306

20. Koushanfar F (2011) Provably Secure Active IC Metering Techniques for Piracy Avoidance and Digital Rights Management. In: IEEE Transactions on Information Forensics and Security (TIFS), to appear
21. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: The International Conference on Computer-Aided Design (ICCAD), pp 674–677
22. 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
23. Roy J, Koushanfar F, Markov I (2008) EPIC: Ending piracy of integrated circuits. In: Design Automation and Test in Europe (DATE), pp 1069–1074
24. Roy J, Koushanfar F, Markov I (2008) Protecting bus-based hardware IP by secret sharing. In Design Automation Conference (DAC), pp 846–851
25. Roy J, Koushanfar F, Markov I (2010) Ending piracy of integrated circuits. IEEE Comput 43: 30–38
26. Pentium III wikipedia page, http://en.wikipedia.org/wiki/pentium_iii
27. Pentium III serial numbers, http://www.pcmech.com/article/pentium-iii-serial-numbers/
28. Potkonjak M, Koushanfar F (2009) Identification of integrated circuits. US Patent Application 12/463,984; Publication Number: US 2010/0287604 A1, May
29. Koushanfar F, Boufounos P, Shamsi D (2008) Post-silicon timing characterization by compressed sensing. In: The International Conference on Computer-Aided Design (ICCAD), pp 185–189
30. Shamsi D, Boufounos P, Koushanfar F (2008) Noninvasive leakage power tomography of integrated circuits by compressive sensing. In: International Symposium on Low Power Electronic Design (ISLPED), pp 341–346
31. Potkonjak M, Nahapetian A, Nelson M, Massey T (2009) Hardware Trojan horse detection using gate-level characterization. In: Design Automation Conference (DAC), pp 688–693
32. Nelson M, Nahapetian A, Koushanfar F, Potkonjak M (2009) SVD-based ghost circuitry detection. In: Information Hiding (IH), pp 221–234
33. Alkabani Y, Koushanfar F (2009) Consistency-based characterization for IC Trojan detection. In: International Conference on Computer Aided Designs (ICCAD), pp 123–127
34. Wei S, Meguerdichian S, Potkonjak M (2010) Gate-level characterization: foundations and hardware security applications. In: Design Automation Conference (DAC), pp. 222–227
35. Koushanfar F, Mirhoseini A (2011) A unified submodular framework for multimodal IC Trojan detection. In: IEEE Transactions on Information Forensics and Security (TIFS): 6(1): pp. 162–174
36. Helinski R, Acharyya D, Plusquellic J (2010) Quality metric evaluation of a physical unclonable function derived from an ICs power distribution system. In: Design Automation Conference (DAC), pp. 240–243
37. Helinski R, Acharyya D, Plusquellic J (2010) Quality metric evaluation of a physical unclonable function derived from an IC's power distribution system. In: Design Automation Conference, (DAC), pp 240–243
38. Rührmair U, Devadas S, Koushanfar F (2011) Book chapter in introduction to hardware security and trust. In: Tehranipoor M, Wang C (eds.) Chapter 7: Security based on Physical Unclonability and Disorder. Springer, Berlin, Heidelberg, New York
39. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K (2001) On the (im)possibility of obfuscating programs. In: Advances in Cryptology (CRYPTO), pp 1–18
40. Yuan L, Qu G (2004) Information hiding in finite state machine. In: Information Hiding Conference (IH), Springer, Berlin, Heidelberg, New york, pp 340–354
41. Chakraborty R, Bhunia S (2008) Hardware protection and authentication through netlist level obfuscation. In: The International Conference on Computer-Aided Design (ICCAD), pp 674–677
42. Alkabani Y, Koushanfar F (2007) 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

43. Maes R, Schellekens D, Tuyls P, Verbauwhede I (2009) Analysis and design of active *IC* metering schemes. In: International Symposium on Hardware-Oriented Security and Trust (HOST), pp 74–81
44. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. IEEE Design & Test of Computers 27: 66–75
45. Koushanfar F (2011) Integrated circuits metering for piracy protection and digital rights management: an overview, Great Lake Symposium on VLSI (GLS-VLSI), pp. 449–454

Chapter 6
Secure Hardware IPs by Digital Watermark

Gang Qu and Lin Yuan

6.1 Introduction

The purpose of this chapter is to introduce the basic concepts and approach to secure the hardware design intellectual properties (IPs) from unauthorized use. We borrow the idea of watermarking from the well-established field of multimedia data protection, where digital watermarks are embedded into the object and can be extracted when necessary to establish ownership. However, watermarking design IP is a much more challenging task.

6.1.1 Design Reuse and Design IP

The ever-increasing logic density has resulted in more transistors on silicon than designer's ability to design them meaningfully. This creates the *design productivity gap* between what can be built and what can be designed. To close this gap, we need a significant shift in design methodology. At the heart of this shift is the principle of design reuse, which is one of the most significant design technology innovations in the modern very large scale integration (VLSI) design era.

In this new design method, large previously designed blocks, such as bus controllers, CPUs, and memory subsystems, will be integrated into an application specific integrated circuit (ASIC) architecture to deliver routine functions in a

G. Qu (✉)
Electrical and Computer Engineering Department, Institution for Systems Research, University of Maryland, College Park, MD 20742, USA
e-mail: gangqu@umd.edu

L. Yuan
Synopsys Inc., Mountain View, CA 94043, USA
e-mail: yuanl@synopsys.com

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_6,

predictable manner. Designers can then focus on what they do best to design new blocks, representing their true design innovation, based on the system requirements. This not only makes it possible to have the new products on market in a timely and cost effective fashion, the newly developed blocks will also be tested, documented, and deposited as new design IPs in the internal IP library for future reuse.

6.1.2 What is a Design IP?

According to the *Reuse Methodology Manual*, design intellectual property is a design unit that can be reasonably viewed as a stand-alone subcomponent of a complete SOC design. Such unit can be physical design blocks such as microprocessor, on-chip memory, PCI interface macro, or fully routed netlist. It can also be abstract algorithm, technique, or methodology that can make the design better.

Design IPs can be categorized into three groups based on their performance and flexibility. *Hard IPs* include graphical design system II (GDSII) file with test lists and high level model, custom physical layout, fully placed and routed netlist for a given technology library. These IPs have predictable and normally optimized performance, but they are inflexible. *Soft IPs* are delivered in the form of synthesizable hardware description language (HDL) codes like Verilog or VHDL programs. They offer great design flexibility despite the less predictable performance. Examples of *firm IPs* are placed register transfer level (RTL) subblocks and fully placed netlist for a generic technology library.

6.1.3 Why Secure Design IP?

In reuse-based design, IPs are developed not only for use but also for reuse. Hardware IP vendors have been working to make their IPs more flexible so they can be reused in other designs to make more profits. Industry leaders have established design standards and guidelines to facilitate IP reuse. Design engineers are forced to cooperate and share their data, expertise, and experience. Design details (including the RTL HDL source codes) are encouraged to be documented and made public to make reuse more convenient. But at the same time, these efforts have made IP piracy and infringement easier than ever. Since early 1990s, litigation cases in semiconductor sector related to IP infringement have been increasingly rapidly, causing an estimated annual revenue loss of billions of dollars.

One of the most dangerous threats of IP piracy comes in the form of *reverse engineering*. Reverse engineering is the process where an object is taken apart and analyzed in order to improve or duplicate the object. It is a very common practice in software and hardware industry with intentions of identifying bugs in own products and enhancing their performance of or studying competitor's products to stay in pace in terms of technology. For IC products, it is possible to grind away layer by layer of the chip and take pictures with an electron microscope of the layout of each

layer. It might be hard to figure out all the details in the IC, fabricating identical duplicate of the product is not when the layout of each layer is revealed. The cost and skills required for such reverse engineering are much less than those to develop the product and it gives attackers incentive for IP piracy.

6.1.4 What are Available to Secure IPs?

In practice, most design IPs are secured by deterrent and protection mechanisms. Commonly used deterrents include patents, copyrights, contracts, trade marks, and trade secrets. They do not directly prevent IP piracy, but rather discourage the misuse of IPs because the infringer, once being caught, may face lawsuits and severe penalty to recover the financial loss of the IP owner. However, it is the IP owner's task to identify and report IP infringement and infringer.

Protection mechanisms use means such as encryption, dedicated hardware, or chemicals to prevent unauthorized access to the IP. Standard encryption can be applied to protect design IPs (including design data) despite the rather expensive decryption process. For example, several Field Programmable Gate Array (FPGA) design tools provide users the encryption choice. Altera's encryption software enables users to encrypt their design and any attempt to compile an encrypted IP will fail if the core is not decrypted properly. Xilinx adds a decryption unit on its latest Virtex-II and Virtex-II Pro FPGA board. Its design tool allows user to select up to six plaintext messages and uses them as the key to encrypt the FPGA configuration bitstream file. The decryption unit will decrypt it before configuring the FPGA board. Chemicals have also been integrated in the chip for passive protection, mainly for military devices. When the surface of the die is scratched and exposed to the atmosphere, it will damage the exposed silicon and thus prevent reverse engineering.

Finally, there are numerous detection mechanisms proposed by researchers both from industry and academia on literally every phase of the design procedure. Naturally, efforts from industry are on identifying and tracing legal IP usage for the purpose of checking, updating, and royalty reporting among others. Research from academia concentrates on the protection and deterrent of high-tech IP piracy such as reverse engineering. This interesting distinction between industry and academia determines the fact that people will view the same question in different ways and provide different answers within different but relevant context. They are complementary to each other and will collaborate to reach the goal of securing hardware IP.

Law enforcement is the only mans to get back the revenue lost to IP piracy. Encryption and other protection mechanisms make IP piracy more difficult and more expensive. But only detection schemes such as digital watermark can enable IP owner to identify the occurrence of IP piracy, which is the first step toward legal fight (such as lawsuits) again IP infringements. In the rest of this chapter, we focus on one of the most interesting and popular detection schemes, the constraint-based watermarking approach.

6.2 Constraint-Based Watermarking for Design IP Protection

Digital data watermarking hides data (or signature) into digital media such as text, image, audio, video, and other multimedia data for the purpose of identification, annotation, and copyright. Traditional digital watermarking techniques take advantage of the limitation of human visual and auditory system and embed the signature to the original digital data as minute errors. This actually changes the original data and cannot be directly used for the protection of hardware design IPs because the value of design IP relies on its correct functionality. Consider a portion of the truth table of a 6-input 2-output function shown in Fig. 6.1. If we change the output value on the first row from 11 to 01, for example, and implement the circuit, when input combination is 101010, the circuit will output 01 instead of the desired 11. In that case, the IP malfunctions and loses its value.

Here we see the first challenge of watermarking design IP: how to embed signature into design IP without changing its functionality. The constraint-based watermarking technique translates the to-be-embedded signatures into a set of additional design constraints during the design and implementation of the IP in order to uniquely encode the signature into the IP. Considering IP invention as solving a hard problem where the requirements (such as functionality) for the IP serve as the constraints that any solution to the problem needs to meet, this basic idea can be depicted in Fig. 6.2. The figure also shows the concept of cutting solution space behind this approach.

To hide a signature, the designer first creates another set of constraints using his secret key; these constraints are selected in such a way that they do not conflict with the constraints in the original problem. Then the original and additional constraints are combined to form a more constrained stego-problem. The stego-problem, instead of the original problem, is solved to obtain a stego-solution. Using information hiding terminologies, we refer to the original problem as cover-constraints, and the signature as embedded-constraints. A stego-solution will satisfy both constraints.

1	0	1	0	1	0	***1***	***1***
1	0	1	0	1	1	***0***	***1***
1	0	1	1	0	0	***0***	***1***
1	0	1	1	0	1	***1***	***0***
1	0	1	1	1	0	***1***	***1***

Fig. 6.1 Part of the truth table of a 6-input (the *left* 6 columns) 2-output (the *right* 2 columns) function

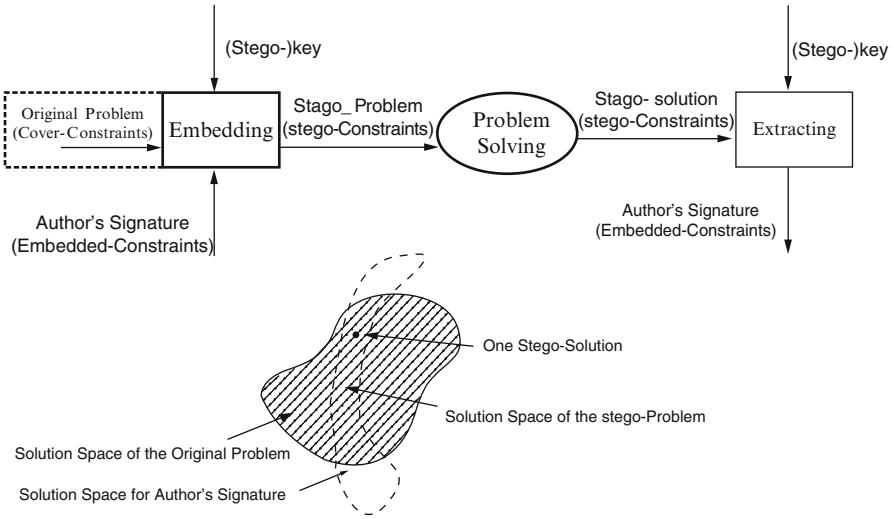


Fig. 6.2 Basic concept of the constraint-based watermarking technique

To claim the authorship, the designer has to demonstrate that the stego-solution carries information based on his signature. The stego-solution unnecessarily satisfies a set of additional constraints that look random, but the designer can regenerate them using his signature together with his secret key. Cryptography functions such as one-way hash and stream cipher will be used to generate the embedded-constraints. There are several approaches proposed to detect the embedded watermark.

We use the shaded area in Fig. 6.2 to represent the solution space of the original problem, and the region with dashed outline to represent the solution candidates that satisfy the embedded constraints based on the designer's signature. The intersection of these two regions contains all the stego-solutions that meet both the original constraints and these embedded-constraints. Let N and n be the number of solutions for the original problem and the stego-problem, respectively, and assume it is equally likely to find any solution in the solution space. Then for a reported stego-solution, the probability that it is obtained by solving the original problem is $1/N$ and the chance it is obtained from solving the stego-problem is $1/n$. When $N >> n$, this odds can be very high. For most of the problems in VLSI design, the size of solution space can be gigantic (for example, in the order of 10^{200}). If there is an effective way to convert the designer's signature into embedded-constraints and keep n "small" (say in the order of one million) then the probability that a given stego-solution is obtained by solving the stego-problem (and thus carries the designer's signature) is 10^{194} higher than solving the original problem. This probabilistic proof of the authorship is the foundation of the constraint-based watermarking techniques.

6.2.1 Example: Watermarking the Boolean Expression Minimization Problem

Consider the following Boolean function in the sum of minterm format

$$f(a,b,c,d) = a'bc'd' + a'bc'd + a'bcd + abc'd. \tag{6.1}$$

Our goal is to rewrite $f(a,b,c,d)$ in the sum of product format with the minimal number of literals. It is easy to verify that the unique answer has nine literals as in

$$f(a,b,c,d) = a'bc' + a'bd + bc'd. \tag{6.2}$$

In this case, the size of the solution size to the original problem $N = 1$. That is, everyone will have the same solution if he solves the problem correctly. So there is no room for us to embed watermark.

Now assume that there are two don't-care conditions: $a'b'c'd'$ and $abcd$. When we treat both conditions as outputting 0, the function remains the same as (6.1) with (6.2) as the only optimal solution. However, if we want 1 as output in both cases, the function becomes

$$f(a,b,c,d) = a'bc'd' + a'bc'd + a'bcd + abc'd + a'b'c'd' + abcd \tag{6.3}$$

and can be uniquely simplified to an expression with only FIVE literals

$$f(a,b,c,d) = a'c'd' + bd. \tag{6.4}$$

We can hide one bit of information behind each of these don't-care conditions as follows: to hide a "1," we force the function to output "1" on the given don't-care condition by adding the corresponding minterm to (6.1); to hide a "0," we simply remove this don't care and consider the output to be "0" on the given don't-care condition. For example, the original function (6.1) carries information "00"; expression (6.3) carries information "11." If "01" is the data we want to embed, we will modify (6.1) to

$$f(a,b,c,d) = a'bc'd' + a'bc'd + a'bcd + abc'd + abcd \tag{6.5}$$

and the unique five-literal simplification becomes

$$f(a,b,c,d) = a'bc' + bd. \tag{6.6}$$

Similarly, when "10" is hidden, the function and it simplified version will change to

$$f(a,b,c,d) = a'bc'd' + a'bc'd + a'bcd + abc'd + a'b'c'd'. \tag{6.7}$$

$$f(a,b,c,d) = a'c'd' + a'bd + bc'd. \tag{6.8}$$

Expressions in (6.2), (6.4), (6.6), and (6.8) are all functionally equivalent to the Boolean function (6.1) with two don't-care conditions: $a'b'c'd'$ and $abcd$. If we treat the don't-care conditions randomly, we may obtain any of these solutions. But when we specify the output for the don't-care conditions based on the data we want to hide, the solution becomes unique and thus can be used to establish authorship.

6.2.2 Context and Requirements for Constraint-Based Watermarking

As summarized in by Kahng et al. in one of the first works, a generic constraint-based watermarking procedure consists of the following components:

- An optimization problem with known difficult complexity. By difficult, we mean that either achieving an acceptable solution or enumerating enough acceptable solutions, is prohibitively expensive. The solution space of the optimization problem should be large enough to accommodate the digital watermark.
- A well-defined interpretation of the solutions of the optimization problem as intellectual property.
- Existing algorithms and/or off-the-shelf software that solve the optimization problem. Typically, the "black box" software model is appropriate, and is moreover compatible with defining the watermarking procedure by composition with pre- and post-processing stages.
- Protection requirements that are largely similar to the well-understand protection requirements for currency watermarking.

We embed additional constraints to the original problem as watermark, it is crucial to guarantee that the stego-solution meets all the requirements of the original problem. That is, the correct functionality of the IP needs to be maintained. In addition, an effective watermark must satisfy the following requirements:

- *High credibility*: the watermark should be readily detectable for the proof of authorship. The probability of coincidence (that is, one reach the stego-solution from solving the original problem) should be low.
- *Low overhead*: the degradation of the software or design by embedding the watermark should be minimized. For instance, in the earlier function simplification example, the optimal solution to the original problem requires only five literals. Embedding signatures "00" or "10" will result in nine-literal solutions, causing a very high overhead.
- *Resilience*: the watermark should be difficult or impossible to remove without the complete knowledge of the software or design. In the earlier example, once we decide to assign a specific output for a don't-care condition, others cannot distinguish whether this specific output comes from the original design requirement or is simply a requirement of the signature.
- *Transparency*: the addition of the watermark to software and designs should be transparent so that it can be used for existing design tools.

- *Perceptual invisibility*: the watermark must be very difficult to detect. In general, exposing the watermark to the public may make it easier to remove or alter the watermark.
- *Part protection*: ideally, a good watermark should be distributed all over the software or design in order to protect all parts of it.

6.3 Watermarking with Don't-Care Conditions

The input–output relationship, known as the truth table, defines the functionality of a system or a component of the system (also called module or function block). It is the most detailed design specification at function level. We can take advantage of the don't-care conditions inherently existing in the design specification to generate watermarking constraints and embed information. Figure 6.3 depicts an encoding scheme to embed a bitstream into a module by assigning specific values to its don't-care conditions.

```
Input: the n don't-care conditions of a module with m-bit
     output, a bit-stream {… b_i … b_1 b_0} to be embedded.
Output: list of selected don't-care conditions and the m-
     bit output value assigned to each of them.
Encoding Scheme:
1. store the n don't-care conditions in a cyclic list L;
2. i = 0;      //start with bit b_0 of the bit-stream
3. j = 0;      //start with the top don't-care in L
4. do
5. {    s = ⌊log_2 n⌋;
6.      (d)_10 = (b_{s+i-1}…b_{i+1}b_i)_2;
7.      i = s+i;
8.      add the pair of the (d+j)^th (mod n) don't care and
     its m-bit output b_{m+i-1}…b_{i+1}b_i to the output list;
9.      i = m + i;
10.     delete the (d+j)^th (mod n) don't care from L;
11.     n = n - 1;
12.     j = (d+j) mod n;
13.  } while (L is non-empty and the bit-stream has not
     been embedded)
```

Fig. 6.3 Pseudo-code for watermarking by don't-care manipulation

Table 6.1 Original truth table of a radix-4 to binary encoder

Input	Output
1000	00
0100	01
0010	10
0001	11

There are two steps to assign don't-care conditions values: choose don't-care conditions and decide the output values for each of the selected don't-care conditions. For each step, we may face multiple choices and this presents the opportunity to embed information. The *do-while* loop from line 4 to line 13 shows the procedure of selecting a don't-care condition and assigning it a specific value based on the information to be embedded. First, because there are n don't-care conditions, we are able to choose one of them based on the next $s = \lfloor log_2\ n \rfloor$ bits from the bit-stream, $b_{s+i-1}...b_{i+1}b_i$ (lines 5–7). Second, we need to assign this don't-care condition an m-bit output value as required by the module to be implemented. We use the next m bits from the bit-stream $b_{m+i-1}...b_{i+1}b_i$ (lines 8–9) as the output. Next, we update the don't-care condition list L by removing the selected don't-care condition and using the next don't-care condition in list L as the first candidate to embed more information.

We now illustrate this scheme by the example of the design of an encoder that converts radix-4 numbers into binary. As shown in Table 6.1, the output of this module will be 00, 01, 10, and 11 if the input bits are 1000, 0100, 0010, and 0001, respectively. The output for the other 12 input combinations, where there are at least two 1's in the input, are undefined. In another word, these are the don't-care conditions. Let a, b, c, d be the four input signals and X, Y be the output signals. It is easy to verify that the simplest implementation (in terms of the number of literals) will be

$$X = c + d \tag{6.9}$$

$$Y = b + d \tag{6.10}$$

Let $b_{14}b_{13}...b_1b_0 = 100010010000010$ be the 15-bit information needs to be embedded into this module. It consists of the 7-bit ASCII codes for letters "D" and "A" (stand for Design Automation) and a parity bit b_0.

There are 12 don't-care conditions in the list L,

$$L = \{0000, 0011, 0101, 0110, 0111, 1001, 1010, 1011, 1100, 1101, 1110, 1111\}$$

$\lfloor log_2 12 \rfloor = 3$. So we use the first three bits $b_2b_1b_0 = 010$ from the given bit-stream (lines 5 and 6). This gives the binary 2 and we thus select the third (0-offset) don't-care condition 0101 from the list L. Next we assign a specific output value for this input combination. The value is chosen to be the next two bits $b_4b_3 = 00$ (line 8). We add the input–output pair of (0101, 00) to the truth table and delete 0101 from the list of don't-care conditions (line 10). Now there are only 11 don't cares

Table 6.2 Watermarked truth table of the same encoder

Input	Output
1000	00
0100	01
0010	10
0001	11
0101	00
1011	00
1101	10

left and we restart from the next don't-care condition 0110 in L (lines 11 and 12). $\lfloor log_2 11 \rfloor = 3$, so we embed the next three bits $b_7 b_6 b_5 = 100$, which is the binary 4. So we select the fifth don't care 1011, counting from 0110, and assign it the output value of $00 = b_9 b_8$. Similarly, we then select 1101 from the remaining 10 don't-care conditions based on $b_{12} b_{11} b_{10}$ and give it output $10 = b_{14} b_{13}$. As a result, the 15-bit information has been encoded into the selection of these three don't-care conditions and their respective output values. Table 6.2 gives the watermarked truth table that we will use instead of the original truth table (Table 6.1) to implement this radix-4 to binary converter. In such implementation, when 0101, 1011, and 1101 are given as input, the output will be 00, 00, and 10, respectively. On the other hand, the optimal implementation in (6.9) and (6.10) will output 11 for each of these three input combinations. However, the cost of implementing the design in Table 6.2 will be higher. One of the most efficient implementations is

$$X = c + a'b' \tag{6.11}$$

$$Y = bc + b'c'd. \tag{6.12}$$

It requires eight literals, twice as many as the optimal implementation in (6.9) and (6.10). This is the overhead we pay to embed information into the design. It is pretty expensive in this case because we have added three new rows to the original truth table that contains only four rows. For complex designs with large truth table, the overhead will become low. However, such high overhead will be a piece of strong evidence to the existence of the hidden information because such high cost implementation is very unlikely to happen. This indicates the tradeoff between high credibility and low overhead in the design of any watermarking schemes. More detailed and formal discussion on this can be found in the reference listed at the end of this chapter.

6.4 Watermarking HDL Source Codes by Duplicating Modules

The earlier discussion on watermarking relies on the fact that there are don't-care conditions during the design and implementation phase, which we can control to embed information. In this section, we use a design example to illustrate how to

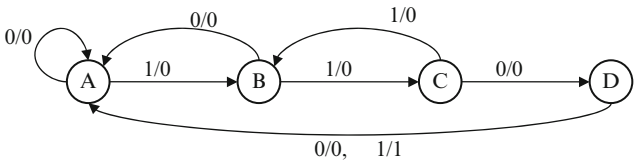


Fig. 6.4 The state transition graph for the "1101" pattern detector

perform watermarking when the original design specification does not have any don't-care conditions. Furthermore, we demonstrate how to do this at the HDL (hardware description language) source code level.

6.4.1 Example: A 4-Bit Pattern Detector

Consider the design of a "1101" pattern detector that receives one bit input data during each clock cycle. It sets output to be "1" whenever it detects a consecutive input pattern of "1101"; otherwise, the output is always "0." Figure 6.4 shows the state transition graph of the pattern detector. There are four states: A: when the current input sequence has no chance to match the pattern (for example, if the last two input bits are "00"); B: when the first bit "1" in the pattern is matched; C: when the first two bits "11" in the pattern are matched; D: when the first three bits, "110," are matched. Next we show how the Fig. 6.4 models the behavior of the pattern detector.

State A is the initial state, when an input "1" is received, we find the find bit in the pattern "1101" and move to state B, when the input is "0," we remain in state A because there is no bit matched. These two moves are represented by the two arcs A $\rightarrow$ B and A $\rightarrow$ A, where the label on the arc indicates the received input and output. From state B, an input "1" will give us "11," a step closer to finding the "1101" pattern and we move to state C, otherwise, the input "10" will not match the beginning of "1101" and we need to restart from the initial state A. Similarly, we have arcs C $\rightarrow$ D and C $\rightarrow$ B when the input is "0" and "1," respectively. Finally, at state D, when we receipt a "1," we have detected the pattern "1101," we output "1" and restart the pattern detection process. When we receipt "0," the latest 4-bit input will be "1100," it does not match any part of'1101' and we need to restart from state A.

6.4.2 A Verilog Implementation of the State Transition Graph

In Verilog HDL code, a digital system is described as a set of modules. Each module represents a logical unit (or logical function) that outputs specific signals based on

the given input signals. Modules can communicate with each other by wires and module instantiations (similar to function calls in C or C++). HDL source code is a popular and convenient way for designers to specify the design specification and there are powerful commercial compilers (called *synthesis tools*) to generate gate-level netlist from given HDL source codes.

Figure 6.5 below defines a module `detector0` to implement the pattern detector shown in Fig. 6.4 as a finite state machine. The key part is the block of codes starting from the statement of `case (currentState)`. This block describes the next state and output of the finite state machine (or the state transition graph). For example, when the current state is `b00`, next state will be either `b00` or `b01` based on the input is 0 or 1, but the output will be 0 regardless of the input value.

6.4.3 Verilog Code Watermarking by Module Duplication

For this pattern detector, from each current state and each different input value, the next state and output are all defined. That is, there is no don't-care condition. Will we still be able to hide any information behind this pattern detector? The answer is yes. When we don't have any option in the input–output pair in the definition of the module's functionality, we can explore different ways to implement it and control how these modules are used. Specifically, in Verilog HDL code, the top-level module or modules in high hierarchies instantiate, normally multiple times, basic function modules at lower hierarchy. Therefore, we can make duplicates of certain modules and then selectively instantiate either the original module or its "duplicate" to encode the watermark information. In another word, we create options for higher hierarchical modules to select from when they need to instantiate other modules.

It is trivial to make duplicates of a module, for example, by changing the names of the module, input, or output. However, as most synthesis tools are developed to optimize design, they will remove the duplicates they can detect to reduce design cost (such as hardware and power consumption). A duplicate needs not only to perform the same required function as the original one, it should also be hard for synthesis tools to identify that it is essentially equivalent to some other module. When there are don't-care conditions in the original module, we can assign different output values for the don't-care conditions every time we need to build a duplicate. This guarantees that the synthesis tool will not delete the duplicates during the optimization phase.

In the absence of don't cares, we can achieve this by implementing the module in different ways. For example, Fig. 6.6 shows the Verilog code that uses a shift register to implement the same pattern detector as its finite state machine implementation in Fig. 6.5. It stores the more recent four bits of input in a 4-bit shift register and compares it with the "1101" pattern. Apparently this detector1 module is functionally equivalent to the detector0 module in Fig. 6.5, but synthesis tools will not be able to identify their equivalence.

```
module detector0 (clk, reset, dataIn, out);
      input clk, reset, dataIn;
      output out;
      reg out;
      reg [1:0] currentState, nextState;
      always @(dataIn or currentState) begin
            case (currentState)
            2'b00: begin
                        nextState = (dataIn == 1) ? 2'b01:2'b00;
                        out = 0;end
            2'b01: begin
                        nextState = (dataIn == 1) ? 2'b10:2'b00;
                        out = 0;end
            2'b10: begin
                        nextState = (dataIn == 0) ? 2'b11:2'b10;
                        out = 0;end
            2'b11: begin
                        nextState = 2'b00;
                        out = (dataIn == 1);end
            endcase
      end
      always@(posedge clk) begin
            if(~eset) begin
                  currentState <= 2'b00;
                  out <= 0;
            end
            else currentState <= nextState;
      end
endmodule
```

Fig. 6.5 Finite state machine implementation of the pattern detector

With multiple modules implementing the same function in the same Verilog code, we can embed information behind the selection of module for instantiation. Figure 6.7 shows this process with a simple encoding scheme: instantiating detector0 to embed a bit 0 and instantiation of detector1 indicates the embedding of a bit 1.

```
module detector1 (clk,reset,dataIn,out);
      input dataIn,clk,reset;
      output out;
      reg out;
      reg [3:0] pattern;
      always@(posedge clk) begin
            if( reset) begin
                  pattern = 0;
                  out = 0;end
            else begin
                  pattern[0]= pattern[1];
                  pattern[1]= pattern[2];
                  pattern[2]= pattern[3];
                  pattern[3]= dataIn;
                  if(pattern==4'b1101) out=1;
                  else out=0;
            end
      end
endmodule
```

Fig. 6.6 Shift register implementation of the pattern detector

```
module P;
      reg clk, reset;
      reg data1,data2,data3;
      wire out1, out2, out3;
      . . .
      detector0 d1(clk, reset, data1, out1);// signature bit 0
      . . .
      detector1 d2(clk, reset, data2, out2);// signature bit 1
      . . .
      detector0 d3(clk, reset, data3, out3);// signature bit 0
      . . .
endmodule
```

Fig. 6.7 Instantiation of the pattern detectors to encode information

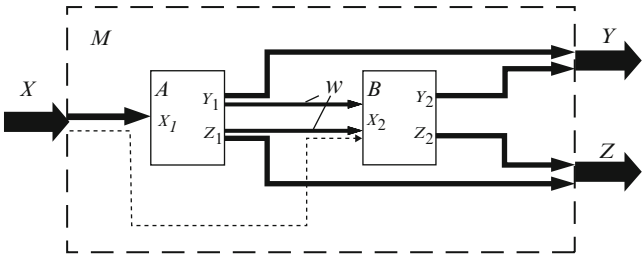


Fig. 6.8 The idea of module splitting

6.4.4 Watermarking by Module Splitting

The earlier module duplication method is limited to modules that will be instantiated multiple times. The duplicated implementation will also incur design overhead and may not be suitable for large modules. Figure 6.8 depicts the concept of the module splitting watermarking technique. It basically splits a large module into several smaller modules.

Two modules: $A(X1, Y1, Z1)$ and $B(X2, Y2, Z2)$, are used to implement a single module $M(X, Y, Z)$, where X is the set of inputs, Y the set of outputs and, Z are optional test outputs. Module splitting is performed as follows:

- First, the watermarking module A takes input $X1 \subset X$ and produces (1) part of the functional outputs in $Y1 \subset Y$, (2) part of the optional test outputs in $Z1 \subset Z$, and (3) the intermediate watermarking outputs W. W is defined according to our signature on specific input pattern of $X1$.
- Next, the correction module B takes inputs $X2 \subset W \cup X$ and produces the rest of the required outputs in $Y2$ and $Z2$. That is, $Y2 = Y - Y1$ and $Z2 = Z - Z1$.

This module splitting method ensures that the implemented module is functionally correct because the two modules A and B combined to generate signals Y and Z, same as the signals generated by M. To verify the signature, one only needs to feed module A the input pattern that we define our watermarking signal W, which will be observable from A's output. To make the watermark robust against both synthesis tools and attackers, we use watermarking signal W from A and as few as possible inputs from X as input for module B. In such way, the watermarking signal W becomes part of the design Otherwise, they will most likely be removed immediately by optimization tools. The strength of the watermark relies on the rarity of implementing module M by constraining the intermediate signal W. Although, it is secure as watermark is integrated into the design, we mention that this method may considerably increase design complexity particularly for the second module and will be vulnerable to attacks.

6.4.5 Performance Evaluation of the Proposed Techniques

We apply the proposed techniques to benchmark Verilog circuits and demonstrate that they meet the watermarking objectives. Verilog designs include circuits such as controllers, adders, multipliers, comparators, DSP cores, ALUs (from SCU-RTL and ISCAS benchmarks, and a MP3 decoder. The MP3 design and SCU-RTL benchmarks are original designs while the RT level ISCAS Verilog codes are obtained from netlists by reverse engineering.

The SCU-RTL and the MP3 benchmarks are perfect examples for the module duplication technique because of the multiple single module instantiations or function calls. However, they are not good for the module splitting method because the modules are very small. It is also possible to take advantages of the don't-care conditions if the original detailed functional description of each module is available.

For the ISCAS benchmarks, because they are reversed engineered, we cannot identify the original don't-care conditions and they have only a few modules, almost all of which are instantiated only once. Consequently, we can only use module splitting technique to protect these moderate sized modules with known functionality.

We optimize each original design by Synopsys' design analyzer and then map them to the CLASS library. After that, we collect the following design metrics: area, power, and delay through the design analyzer report. Next, we apply the proposed Verilog watermarking techniques to these designs and repeat the earlier design process for the watermarked design. As we have described, SCU-RTL benchmark is watermarked by module duplication, and ISCAS circuits by module splitting.

After optimization, we can clearly identify from the schematic view in the Synopsys design analyzer window. This insures that our watermarks survive the synthesis tools. Figure 6.6 gives the gate-level views of ISCAS 74181 circuit (a 4-bit ALU) before and after watermarking, where a 9-letter message (UMD TERPS) in ASCII is embedded by splitting the CLA module, which has three inputs and four outputs, into two modules. We document these two modules in the same way as other original modules. To test the watermark's resilience at both the Verilog code level and the gate level, we showed a group of Verilog designers the watermarked Verilog codes with documentation and Fig. 6.9. None of them could tell which one was the original.

Table 6.3 reports the design overhead on SCU-RTL benchmarks by module duplication. As expected, there is little area overhead due to the duplicated modules. However, the average area overhead is about 4.76% (and this percentage is mainly caused by the small FIR Design). The watermarked design does not introduce any additional delay and consumes only 1% more energy on an average than the original design.

Table 6.4 reports the results of watermarking ISCAS benchmarks by module splitting. In this technique, we enforce the watermark into design's functionality. In

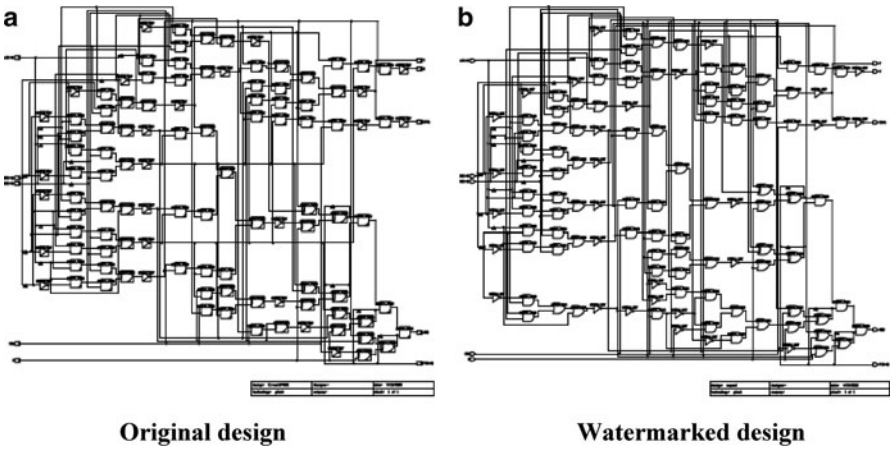


Fig. 6.9 Gate-level view of circuit 74181

Table 6.3 Watermarking SCU-RTL & MP3 Verilog benchmark circuits

Benchmark circuits		Original	Watermarked	Overhead
FIR (2,264 gates, 16 bits embedded)	Area (λ^2)	4,083	4,557	11.6%
	Power (μW)	34.49	35.33	2.4%
	Delay (ns)	48.7	48.7	0%
IIR (15,790 gates, 15 bits embedded)	Area (λ^2)	16,419	16,431	0.07%
	Power (μW)	35.33	35.06	−0.76%
	Delay (ns)	49.15	49.15	0%
IDCT (17,341 gates, 16 bits embedded)	Area (λ^2)	20,755	21,271	2.5%
	Power (μW)	23.31	23.5	0.8%
	Delay (ns)	49.2	49.2	0%
MP3 (>20,000 gates, 20 bits embedded)	Area (λ^2)	16,955	17,297	4.9%
	Power (μW)	67.49	70.82	2.0%
	Delay (ns)	49.15	49.15	0%

general, this should cause design overhead. For example, we see that both average area and power overhead are slightly over 5%. Interestingly, the circuit delay may decrease after watermarking. This might be possible, for example, if we split a module that has a signal on the critical path, this signal may be generated by the simpler watermarking module and thus reduce the delay.

From Tables 6.3 and 6.4, we can see that large design overhead often occurs for small designs (FIR, 74181, and C432). Although it is premature to claim, given the limited set of experiments, we anticipate that all design overhead will decrease for large designs and eventually become negligible for real life designs.

Table 6.4 Watermarking on ISCAS benchmark circuits

Benchmark circuits		Original	Watermarked	Overhead
74181 (61 gates, 56 bits embedded)	Area (λ^2)	86	94	9.3%
	Power (μW)	102.41	111.84	9.2%
	Delay (ns)	9.26	10.38	12.1%
C432 (160 gates, 56 bits embedded)	Area (λ^2)	176	192	9.1%
	Power (μW)	230.87	249.89	8.2%
	Delay (ns)	20.44	19.63	−4.0%
C499 (202 gates, 56 bits embedded)	Area (λ^2)	400	410	2.5%
	Power (μW)	14.75	11.71	2.5%
	Delay (ns)	14.75	11.71	−20.6%
c1908 (880 gates, 56 bits embedded)	Area (λ^2)	574	598	4.1%
	Power (μW)	581.43	612.47	5.3%
	Delay (ns)	21.82	22.54	3.3%
C7552 (61 gates, 56 bits embedded)	Area (λ^2)	4,489	4,525	0.8%
	Power (μW)	5778.1	5808.5	0.5%
	Delay (ns)	65.57	65.57	0%

6.5 Conclusions

In this chapter, we briefly discuss the intellectual properties in VLSI design and the need to protect them with a short survey on the current results and practice. We describe the basic concepts of constraint-based watermarking approach to hide information into the design for the purpose of future authentication. We illustrate this with the example of how to use don't-care conditions in the design. We further discuss two methods to perform watermarking when don't-care conditions do not present and demonstrate them on Verilog HDL codes.

To learn more, please refer to [1] for a practical guide on reuse based design methodology; [2–6] for the requirements and current industry standards on IP protection; [7–9] for constraint-based watermarking; and [10–13] for the benchmark circuits used in the experiments of this chapter.

References

1. Keating M, Bricaud P (1999) Reuse Methodology Manual, for System-On-a-Chip Designs, 2nd edn. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
2. Virtual Socket Interface Alliance (2001) Intellectual Property Protection White Paper: Schemes, Alternatives and Discussion Version 1.1, January 2001
3. Virtual Socket Interface Alliance (2000) Virtual Component Identification Physical Tagging Standard (IPP 1 1.0), September 2000
4. Virtex™-II Platform FPGA Data Sheet, http://www.xilinx.com
5. Altera Corporation. San Jose, California. http://www.altera.com/

6. Yip KW, Ng TS (2000) Partial-encryption technique for intellectual property protection of FPGA-based products. IEEE Trans Consumer Electron 46(1): 183–190
7. Kahng AB et al. (2001) Constraint-based watermarking techniques for design IP protection. IEEE Trans Comput Aided Design Integr Circuits Syst 20(10): 1236–1252
8. Qu G, Potkonjak M (2003) Intellectual Property Protection in VLSI Designs: Theory and Practice. Kluwer Academic Publishers, Dordrecht (Hingham, MA), ISBN 1–4020–7320–8
9. Yuan L, Pari P, Qu G (2004) Soft IP protection: watermarking HDL source codes. In: 6th Information Hiding Workshop (IHW'04), pp. 224–238, LNCS vol. 3200. Springer-Verlag, Berlin, Heidelberg, New York, May 2004
10. International Technology Roadmap for Semiconductors. http://public.itrs.net/Files/2001ITRS/
11. http://www.eecs.umich.edu/ ∼jhayes/iscas
12. http://www.ece.cmu.edu/ ∼ee545/s02/10/fpga 1813.txt
13. http://www.engr.scu.edu/mourad/benchmark/RTL-Bench.html

Chapter 7
Physical Attacks and Tamper Resistance

Sergei Skorobogatov

Many semiconductor chips used in a wide range of applications require protection against physical attacks or tamper resistance. These attacks assume that a direct access to the chip is possible with either establishing electrical connections to signal wires or at least doing some measurements. The importance of protection against physical attacks is dictated by the amount of valuable and sensitive information stored on the chip. This could be secret data or company secrets and intellectual property (IP), electronic money for service access, or banking smartcards. The security in chips serves to deter prospective attackers from performing unauthorized access and benefiting from it. There are many areas that rely on tamper resistance of silicon chips. One of the first was car industry with theft protection and car alarms. Then in the early 1990s service providers such as PayTV, satellite TV, and utility companies realized that their service can be stolen if the access and payment cards are not properly protected. From the late 1990s home entertainment companies realized that their game consoles became the target of dishonest users who wanted to run illegal copies of the games. These days many device manufacturers from computer peripherals and mobile phones to printers and computers are worried about possible IP theft by third parties – either competitors or subcontractors. All the above challenges force hardware engineers to find secure solutions – either better protected off-the-shelf chips or their own custom chips. As in most cases it is impractical to block direct access to the device and its components, protection against physical attacks became the essential part of the system design. These days we have a continuous battle between the manufacturers who invent new security solutions learning their lessons from previous mistakes and the hacker community which is constantly trying to break the protection in various devices. Both sides are also constantly improving their knowledge and experience. In this endless war, the front line shifts forward and backward regularly. Deep down, the

S. Skorobogatov (✉)
University of Cambridge, Computer Laboratory, JJ Thomson Avenue, Cambridge CB3 0FD, UK
e-mail: Sergei.Skorobogatov@cl.cam.ac.uk

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_7, © Springer Science+Business Media, LLC 2012

problem concerns both economics and law. On the one hand, when dishonest people try to steal property, there will be a demand to increase security. On the other, reverse engineering was always part of technological progress, helping to design compatible products and improve existing ones. The dividing line between legal (reverse engineering) and illegal (piracy) is difficult.

7.1 Attack Scenarios or Why Devices are Attacked

Attacks can be used for different purposes depending on the goal. Sometimes copying a profitable on-the-market product can give easy money. Larger manufacturers could consider stealing IP from the device and mixing it with their own IP to disguise the theft. Others could try to steal secrets from the device either to produce a competitive product or to steal service. Product designers should first think about the possible motives for attacking their devices and then concentrate on the protection mechanisms. The following attack scenarios should be considered during the system design.

1. *Theft of Service* could happen when electronic devices are used to provide access to some information or service. For example, cable and satellite TV companies control the channels a viewer can see. If a pirate can bypass security or simulate the device, the service provider will lose. As the pirates normally work in a large community, any success is distributed among all members of the group, so it incurs huge losses to the service provider.
2. *Cloning and Overbuilding* are one of the most widely used attack scenarios. Cloning is used by a large variety of attackers from individuals, who want cheaper electronic gadgets, to large companies interested in increasing their sales without large investment in design. For example, dishonest competitors may try to clone existing products to reduce development costs. Of course they will have to spend some effort to disguise the fact of piracy, but compared to honest development cost this is negligible. Normally, cloning requires reverse engineering of the device to some extent. Overbuilding takes place when a contract manufacturer builds more than the requested quantity of electronic devices. The extra devices can be then sold on the market. The design could also be sold to third parties.
3. *IP Piracy* is always a big concern for many developers from private individuals to large corporations. This involves extraction of information, passwords, and cryptographic keys. This can later be used to design better product with lower investment or to read encrypted sensitive information and trade secrets.
4. *Denial of Service* can be used by a competitor to damage a vendor's product. This could happen when the device firmware is updated over a network. If the competitor manages to reverse engineer the device and work out the update protocol, he could launch a malicious update code and then switch off all the devices or even damage them by uploading bad code. For example, it is possible to permanently damage a field-programmable gate array (FPGA)

device by uploading a bad configuration file. Also, modern microcontrollers and smartcards have Flash memory for the program code. If an erase command is issued for all memory blocks then the device will stop operating for good. The developer should design firmware update features very carefully to make sure they cannot be used without proper authentication.

7.2 Levels of Tamper Resistance

It is not an easy task to estimate the protection level of a semiconductor chip as so many factors should be taken into consideration from the chip's package and die layout to memory structure, memory type, programming and access interfaces, security fuses or secret key location, protection mechanisms, and other security features such as glitch detection, power supply voltage monitors, protection meshes, tamper resistance, etc. There is no straightforward way to evaluate the hardware security of a semiconductor device; what normally has to be done is to apply different attack methods and observe the result. The more attacks tested, the more confidence in the result. In order to estimate the level of security protection several tamper protection levels were introduced by IBM [1]. Their classification suggests six security levels starting from a zero level corresponding to the system without any security protection to a high level for the virtually unbreakable system. There might, of course, be all sorts of intermediate levels which can be used to compare the devices with each other.

1. *Level ZERO.* No special security features are used in the system. All parts have free access and can be easily investigated. Example: microcontroller or FPGA with external memory.
2. *Level LOW.* Some security features are used but they can be relatively easy defeated with minimum tools required such as soldering iron and low cost analog oscilloscope. Attack takes time but does not involve more than $1,000 of equipment. Example: microcontroller with unprotected internal memory but proprietary programming algorithm.
3. *Level MODL.* Security used protects against most low cost attacks. More expensive tools are required as well as some special knowledge. Total equipment cost does not exceed $10,000. Examples: microcontrollers sensitive to power analysis and power glitches.
4. *Level MOD.* Special tools and equipments are required for successful attack as well as some special skills and knowledge. Total equipment cost is up to $100,000. Examples: microcontrollers with protection against ultraviolet (UV) light attacks and old smartcard chips.
5. *Level MODH.* Special attention is paid to design of the security protection. Equipment is available but is expensive to buy and operate. Total equipment cost is up to $1,000,000. Special skills and knowledge are required to utilise the equipment for an attack. A group of skilled attackers may be required

with complementary skills to work on the attack sequence. Examples: modern smartcard chips with advanced security protection, complex application-specific integrated circuits (ASICs), and secure FPGAs.

6. *Level HIGH.* All known attacks are defeated and some research by a team of specialists is necessary to find a new attack. Highly specialised equipment is necessary, some of which might have to be designed and built. Total cost of the attack is over a million dollars. The success of the attack is uncertain. Only large organizations like semiconductor manufacturers or government funded laboratories could afford carrying such attacks. Examples: secure cryptographic modules in certification authority applications.

For applications or devices that include cryptography, U.S. and Canadian federal government agencies are required to use a cryptographic products that has been FIPS 140 (Federal Information Processing Standards) validated [2] or Common Criteria validated [3]. Most Common Criteria protection profiles rely on FIPS validation for cryptographic security. Within the FIPS 140–2 (or 140–1) validations, there are four possible security levels for which a product may receive validation.

Security Level 1 provides the lowest level of security. It specifies basic security requirements for a cryptographic module.

Security Level 2 improves the physical security of a Level 1 cryptographic module by adding the requirement for tamper evident coatings or seals, or for pick-resistant locks.

Security Level 3 requires enhanced physical security, attempting to prevent the intruder from gaining access to critical security parameters held within the module.

Security Level 4 provides the highest level of security. The physical security provides an envelope of protection around the cryptographic module to detect a penetration into the device from any direction.

The security level of a particular device does not last forever. It is possible that a low cost attack will be found in the future when the attack tools become cheaper or available as second-hand. In addition, technological progress opens doors to less expensive attacks reducing the protection level of some products.

7.3 Attack Categories

There are several ways how semiconductor chips could be physically attacked.

Side-channel attacks allow the attacker to monitor the analog characteristics of supply and interface connections and any electromagnetic radiation by the device during normal operation.

Software attacks use the normal communication interface of the device and exploit security vulnerabilities found in the protocols, cryptographic algorithms, or their implementation.

Fault generation uses abnormal environmental conditions to generate malfunctions in the device that provide additional access.

Microprobing can be used to access the chip surface directly, so we can observe, manipulate, and interfere with the device.

Reverse engineering is used to understand the inner structure of the device and learn or emulate its functionality. It requires the use of the same technology available to semiconductor manufacturers and gives similar capabilities to the attacker.

All microprobing and reverse engineering techniques are invasive attacks. They require hours or weeks in specialized laboratory and in the process they destroy the packaging. The other three are noninvasive attacks. The attacked device is not physically harmed during these attacks. The fault attack could also be semi-invasive. It means that the access to the chip's die is required but the attack is not penetrative and the fault is generated with intensive light pulse, radiation, local heating, or other means.

Noninvasive attacks are particularly dangerous in some applications for two reasons. Firstly, the owner of the device might not notice that the secret keys or data have been stolen, therefore it is unlikely that the validity of the compromised keys will be revoked before they are abused. Secondly, noninvasive attacks often scale well, as the necessary equipment can usually be reproduced and updated at low cost.

The design of most noninvasive attacks requires detailed knowledge of both the chip and software. On the other hand, invasive microprobing attacks require very little initial knowledge and usually work with a similar set of techniques on a wide range of products. Attacks therefore often start with invasive reverse engineering, the results of which then help to develop cheaper and faster noninvasive attacks. Semi-invasive attacks can be used to learn the device functionality and test its security circuits. As these attacks do not require establishing any physical contact to the internal chip layers, expensive equipment such as laser cutters and focused-ion beam machines are not required. The attacker could succeed using a simple off-the-shelf microscope with a photoflash or laser pointer attached to it.

Attacks can be reversible when the device can be put back into the initial state, or irreversible with permanent changes done to the device. For example, power analysis and microprobing could give the attacker a result without harming the device itself. Certainly, microprobing will leave tamper evidence but usually that does not affect further device operation. On the contrary, fault injection and UV light attacks could very likely put the device into the state where the internal registers or memory contents are changed and cannot be restored. In addition, UV attacks leave tamper evidence as they require direct access to the chip surface.

7.3.1 Noninvasive Attacks

A noninvasive attack does not require any initial preparations of the device under test. The attacker can either tap the wires to the device or plug it into a test circuit for the analysis. Once found, these attacks could be easily scaled and their reproduction

does not involve very much cost. In addition, no tamper evidence is left after they are applied. Therefore they are considered to be the most serious threat to the hardware security of any device. At the same time it usually takes a lot of time and effort to find an attack on any particular device. This often involves reverse engineering the device in the sense of either disassembling its software or understanding its hardware layout. Quite common electrical engineering tools can be used for non-invasive attacks. These involve integrated circuit (IC) soldering/desoldering station, digital multimeter, universal chip programmer, prototyping boards, regulated power supply, oscilloscope, logic analyzer, and signal generator.

Noninvasive attacks can be either passive or active. Passive attacks, also called side-channel attacks, do not involve any interaction with the attacked device but, usually, observation of its signals and electromagnetic emissions. Examples of such attacks are power analysis and timing attacks. Active attacks, like brute force and glitch attacks, involve playing with the signals applied to the device including the power supply line.

The most widely used noninvasive attacks include playing around with the supply voltage and clock signal. Under-voltage and over-voltage attacks could be used to disable protection circuit or force a processor to do the wrong operation. For these reasons, some security chips have a voltage detection circuit, but this circuit cannot react to fast transients. Power and clock transients can also be used in some processors to affect the decoding and execution of individual instructions.

Another possible attack uses current analysis. We can measure with an oscilloscope the fluctuations in the current consumed by the device. Drivers on the address and data bus often consist of up to a dozen parallel inverters per bit, each driving a large capacitive load. They cause a significant power-supply short circuit during any transition.

Another possible threat to secure devices is data remanence. This is the capability of volatile memory to retain information for some time after power is disconnected. Static random-access memory (SRAM) storing the same key for a long period of time can reveal it on next power on. Another possibility is to "freeze" the memory by applying low temperature. In this case, SRAM can retain information for enough time to get access to the memory chip and read its contents. Data remanence can take place in nonvolatile memories as well; the residual charge left on a floating gate transistor may be detected. For example, it could affect a threshold level or time-switching characteristics.

The next possible way of attacking a device is playing around with its interface signals and access protocols. Also, if a security protocol is wrongly implemented, that leaves a hole for the attacker to exploit. Some microcontrollers and smartcards have a factory-test interface that provides access to on-chip memory and allows the manufacturer to test the device. If an attacker can find a way of exploiting this interface, he can easily extract the information stored inside the chip. Normally information on test circuits is kept secret by the manufacturer, but an attacker can try applying different voltages and logic levels to the pins in the hope that it will put it into test mode. This sometimes works for microcontrollers but in smartcards such test circuitry is usually destroyed after use. Also, embedded software developers

sometimes implement functions that allow downloading from internal memory for test and update purposes. That must be done in a way that prevents any access to the code without proper authentication, or so that the code can be sent out in encrypted form only.

7.3.2 Invasive Attacks

These attacks require direct access to the internal components of the device. If it is a security module or a USB dongle then it has to be opened to get access to the internal memory chips. In the case of a smartcard or a microcontroller, the packaging should be removed followed by focused ion beam (FIB) or laser depassivation to get access to the internal wires buried deep under the passivation layer of the chip. Such attacks normally require a well-equipped and knowledgeable attacker to succeed. Meanwhile, invasive attacks are becoming constantly more demanding and expensive, as feature sizes shrink and device complexity increases.

Some operations such as depackaging and chemical etching can still be performed by almost anyone with a small investment and minimal knowledge. There are also some attacks, for example optical reading of an old Mask ROM (read-only memory), or reverse engineering of a chip built with old technology and two metal layers, where gaining the access to the chip surface is enough to succeed. Despite the greater complexity of invasive attacks, some of them can be done without expensive laboratory equipment. Low-budget attackers are likely to get a cheap solution on the second-hand market for semiconductor test equipment. With patience and skill, it should not be too difficult to assemble all the required tools for under $10,000 by buying a second-hand microscope and using self-designed micropositioners.

Invasive attacks start with the removal of the chip package. Once the chip is opened it is possible to perform probing or modifying attacks. The most important tool for invasive attacks is a microprobing workstation. Its major component is a special optical microscope with a long working distance objective lens. Micropositioners are installed on a stable platform around the chip test socket and allow the movement of probe arms, with submicron precision, over a chip surface. A probing needle with an elastic hair at the end is installed on each arm and allows electrical contact to on-chip bus lines without damaging them.

On the depackaged chip, the top-layer aluminum interconnect lines are still covered by a passivation layer (usually silicon oxide or nitride), which protects the chip from the environment and ion migration. This passivation layer must be removed before the probes can establish contact. The most convenient depassivation technique is the use of a laser cutter. Carefully dosed laser flashes remove patches of the passivation layer. The resulting hole in the passivation layer can be made so small that only a single bus line is exposed. This prevents accidental contacts with neighboring lines and the hole also stabilizes the position of the probe, making it less sensitive to vibration and temperature changes. It is not usually practical to read the information stored on a security processor directly out of each single memory

cell, except for ROM. The stored data has to be accessed via the memory bus where all data is available at a single location. Microprobing is used to observe the entire bus and record the values in memory as they are accessed.

In order to read all memory cells without the help of the device software, we have to abuse a central processor unit (CPU) component such as an address counter to access memory for us. The program counter is already incremented automatically during every instruction cycle and used to read the next address, which makes it perfectly suited to serve us as an address sequence generator. We only have to prevent the processor from executing jump, call, or return instructions, which would disturb the program counter in its normal read sequence. Tiny modifications of the instruction decoder or program counter circuit, which can easily be performed by opening the right metal interconnect with a laser, often have the desired effect.

Another approach to understanding how a device work is to reverse engineer it. The first step is to create a map of the chip. It could be done by using an optical microscope with a digital camera to produce several meter large mosaics of high-resolution photographs of the chip surface. Basic architecture structures, such as data and address bus lines, can be identified quite quickly by studying connectivity patterns and by tracing metal lines that cross clearly visible module boundaries like ROM, SRAM, electrically erasable programmable ROM (EEPROM), arithmetic logic unit (ALU), and instruction decoder. All processing modules are usually connected to the main bus via easily recognizable latches and bus drivers. The attacker obviously has to be familiar with complementary metal-oxide-semiconductor (CMOS) IC design techniques and microcontroller architectures, but the necessary knowledge is easily available from numerous textbooks.

Most currently available microcontrollers and smartcard chips have feature sizes of 0.13–0.35 μm and two to seven metal layers. Chips down to 0.25 μm can be reverse-engineered and observed with manual and optical techniques, but require some specific deprocessing operations to remove one metal layer after another. For the latest generations of microcontrollers with more metal layers and features below the wavelength of visible light, it may be necessary to use more expensive tools such as scanning electron microscopes (SEM).

The most common tool used for failure analysis and to apply any modifications to the chip structure is a FIB machine. It consists of a vacuum chamber with a particle gun, comparable to a SEM. With a FIB machine the attacker can cut the metal and polysilicon interconnections and build new ones with a deep submicron precision. Using laser interferometer stages, a FIB operator can navigate blindly on a chip surface. Chips can also be polished from the rear side down to a thickness of just a few tens of micrometers. Using laser interferometer navigation or infrared (IR) imaging, it is then possible to locate individual transistors and contact them through the silicon substrate by FIB editing a suitable hole. This rear-access technique has probably not yet been used by pirates so far, but the technique is about to become much more commonly available and, therefore, has to be taken into account by designers of new secure chips. FIBs are primarily used by attackers today to simplify manual probing of deep metal and polysilicon lines. A hole is drilled to the signal line of interest and then filled with platinum to bring the signal to the surface,

where a several micrometer large probing pad is created to allow easy access. Modern FIB workstations cost less than a million dollars and are available in over a hundred organizations including universities. Some old FIB models are available on a second-hand market at a price of less than $100,000.

7.3.3 Semi-Invasive Attacks

There is a large gap between previously discussed noninvasive and invasive types of attack and many attacks fall into this gap, being not so expensive as classical penetrative invasive attacks but as easily repeatable as noninvasive attacks. Therefore, a new class of attack called semi-invasive was recently defined and introduced [4]. Like invasive attacks, they require depackaging the chip in order to get access to its surface. However, the passivation layer of the chip remains intact, as semi-invasive methods do not require creating contacts to the internal lines. This is because microprobing is not used for this attack technology and thus such expensive tools as FIBs are not required. Instead, less expensive laser microscopes can be used which can be made from a standard optical microscope by attaching a laser pointer to it.

Semi-invasive attacks are not entirely new. UV light has been used to disable security fuses in EPROM and one-time programmable (OTP) microcontrollers for many years. Modern microcontrollers are less susceptible to this attack as they were designed to withstand it. Advanced imaging techniques can be considered as semi-invasive as well. This includes various kinds of microscopy such as IR, laser scanning, and thermal imaging. Some of them can be applied from the rear side of the chip which is very useful for modern chips with multiple-metal-layer design. Some of these techniques allow observation of the state of each individual transistor inside the chip.

One of the main contributions to semi-invasive attacks is optical fault injection which can be used to modify the contents of SRAM and change the state of any individual transistor inside the chip. That gives almost unlimited capabilities to the attacker in getting control over the chip operation and abusing the protection mechanism.

Compared to noninvasive attacks, semi-invasive attacks are harder to implement as they require decapsulation of the chip. However, very much less expensive equipment is needed than for invasive attacks. These attacks can be performed in a reasonably short period of time. Also they are scalable to a certain extent, and the skills and knowledge required to perform them can be easily and quickly acquired. Some of these attacks, such as an exhaustive search for a security fuse, can be automated. If compared to invasive attacks, the semi-invasive kind do not normally require precise positioning for success because they are normally applied to a whole transistor or even a group of transistors rather than to a single wire inside the chip.

7.4 Breaking the Security with Noninvasive Attacks

7.4.1 Side-Channel Attacks

Some security-related operations a semiconductor chip performs can take a different time to complete depending on the values of the input data and the secret key. Careful timing measurement and analysis may allow recovery of the system's secret key. This idea was first published in the scientific literature in 1996 [5]. Then later these attacks were successfully performed on an actual smartcard implementation of the RSA signature [6].

To conduct the attack one needs to collect a set of messages, together with their processing time, e.g., question-answer delay. Many cryptographic algorithms were found to be vulnerable to timing attacks. The main reason why this happens is in the software implementation of each algorithm. That includes performance optimization to bypass unnecessary branching and conditional operations, cache memory usage, nonfixed time processor instructions such as multiplication and division, and a wide variety of other causes. As a result, performance characteristics typically depend on both the encryption key and the input data.

To prevent such attacks the techniques used for blinding signatures can be used [7]. The general idea is to prevent the attacker knowing the input to the modular exponentiation operation by mixing the input with a chosen random value.

Timing attacks can be applied to chips whose security protection is based on passwords, or to access control systems that use cards or keys with fixed serial numbers, for example, Dallas iButton products. The common mistake in such systems is the way the serial number of the entered key is verified against the database. Very often the system checks each byte of the key against one entry in the database and stops as soon as an incorrect byte is found. Then it switches to the next entry in the database until it reaches the end. So the attacker can easily measure the time between the input of the last key and the request for another key and figure out how many coincidences were found. With a relatively small number of attempts, he will be able to find one of the matching keys.

To prevent these attacks, the designer should carefully calculate the number of CPU cycles that take place when the password is compared and make sure they are the same for correct and incorrect passwords. For example, in the Motorola 68HC08 microcontrollers family the internal ROM bootloader allows access to the internal memory only if the correct eight-byte password was entered first. To achieve that, extra NOP commands were added to the program making the processing time equal for both correct and incorrect bytes of the password. That gives good protection against timing attacks. In the early versions of the Texas Instruments MSP430 microcontrollers such compensation was not made. As a result it was possible to guess the access password to the user Flash memory [8].

Some microcontrollers have an internal resistor-capacitor (RC) generator mode of operation in which the CPU running frequency depends upon the power supply voltage and the die temperature. This makes timing analysis more difficult as the

attacker has to stabilize the device temperature and reduce any fluctuations and noise on the power supply line. Some smartcards have an internally randomized clock signal to make measurements of the time delays useless for the attack.

A computing device's power consumption depends on its current activity. The consumption depends on changes of state of its components, rather than on the states themselves, because of the nature of CMOS transistors. When an input voltage is applied to a CMOS inverter, a transient short-circuit is induced. The rise of the current during this transient is much higher than the static dissipation caused by parasitic leakage current. Using a resistor in the power supply line, these current fluctuations can be measured. In electromagnetic analysis (EMA) a small coil placed close to the chip is used to acquire electromagnetic emission. As both the power analysis and the EMA were covered in previous chapters they will not be discussed in details here.

7.4.1.1 Brute Force Attacks

'Brute force' has different meanings for cryptography and semiconductor hardware. In cryptography, a brute force attack would be defined as the methodical application of a large set of trials for a key to the system. This is usually done with a computer or an array of FPGAs delivering patterns at high speed and looking for success.

One example could be the password protection scheme used in microcontrollers, such as the Texas Instruments MSP430 family. The password itself is 32 bytes (256 bits) long which is more than enough to withstand direct brute force attack. But the password is allocated at the same memory addresses as the CPU interrupt vectors. That, firstly, reduces the area of search as the vectors always point to even addresses within memory. Secondly, when the software gets updated, only a small part of the password is changed because most of the interrupt subroutines pointed to by the vectors are very likely to stay at the same addresses. As a result, if the attacker knows one of the previous passwords he could easily do a systematic search and find the correct password in a reasonable amount of time.

Brute force can be also applied to a hardware design implemented into an ASIC or a FPGA. In this case the attacker tries to apply all possible logic combinations to the input of the device while observing all its outputs. This kind of attack could be also called black-box analysis because the attacker does not have to know anything about the design of the device under test. He only tries to understand the function of the device by trying all possible combinations of signals. This approach works well only for relatively small logic devices. Another problem the attacker will face is that designs implemented in FPGAs or ASICs have flip-flops, so the output will probably be function of both the previous state and the input. But the search space can be significantly reduced if the signals are observed and analyzed beforehand. For example, clock inputs, data buses, and some control signals could be easily identified, significantly reducing the area of search.

Another possible brute force attack, applicable to many semiconductor chips, is applying an external high voltage signal (normally twice the power supply) to the

chip's pins to find out whether one of them has any transaction like entering into a factory test or programming mode. In fact, such pins can be easily found with a digital multimeter because they do not have a protection diode to the power supply line. Once sensitivity to a high voltage is found for any pin, the attacker can try a systematic search on possible combinations of logic signals applied to other pins to figure out which of them are used for the test/programming mode and exploit this opportunity.

The attack could be also applied to the device communication protocol in order to find any hidden functions embedded by the software developer for testing and upgrade purposes. Chip manufacturers very often embed hardware test interfaces for postproduction testing of their semiconductor devices. If the security protection for these interfaces is not properly designed, the attacker can exploit it to get access to the on-chip memory. In smartcards such test interfaces are normally located outside the chip circuit and physically removed after the test operation, eliminating any possibility of use by outsiders.

7.4.1.2 Fault Injection Attacks

Glitch attacks are fast changes in the signals supplied to the device and designed to affect its normal operation. Usually glitches are inserted in power supply and clock signals. Every transistor and its connection paths act like an RC element with a characteristic time delay. The maximum usable clock frequency of a processor is determined by the maximum delay among its elements. Similarly, every flip-flop has a characteristic time window (of a few picoseconds) during which it samples its input voltage and changes its output accordingly. This window can be anywhere inside the specified setup cycle of the flip-flop, but is quite fixed for an individual device at a given voltage and temperature. So if we apply a clock glitch (a clock pulse much shorter than normal) or a power glitch (a rapid transient in supply voltage) this will affect only some transistors in the chip and cause one or more flip-flops to adopt the wrong state. By varying the parameters, the CPU can be made to execute a number of completely different wrong instructions, sometimes including instructions that are not even supported by the microcode. Although we do not know in advance which glitch will cause which wrong instruction in which chip, it can be fairly simple to conduct a systematic search. For example, the bootloader in the Motorola MC68HC05B microcontroller checks the bit0 of first EEPROM address and grants external access to the chip's memory only if it is "1", otherwise it goes into endless loop. If the chip supply voltage is temporarily reduced, the CPU fetches an FFh value from the EEPROM rather than the actual value and this corresponds to the unsecured state of the fuse. Similarly, if the clock source is disturbed by short circuiting the external crystal resonator, multiple clock glitches are produced That way there is a good chance of escaping from the endless loop into the memory access code.

Applying clock glitches to some microcontrollers could be difficult. For example, the Texas Instruments MSP430 microcontroller family operates from an internal RC

generator in bootloader mode and it is difficult to synchronize to the internal clock and estimate the exact time of the attack. Some smartcards benefit from having randomly inserted delays in the CPU instruction flow, which makes applying the attacks even more difficult. Using power analysis could help, but requires very sophisticated and expensive equipment to extract the reference signal in real time.

Power supply voltage fluctuations can shift the threshold level of the transistors. As a result some flip-flops will sample their input at different time or the state of the security fuse will be read incorrectly. This is usually achieved by either increasing the power supply voltage or dropping it for a short period of time, normally from one to ten clock cycles. Power glitches can be applied to a microcontroller with any programming interface as they could affect both the CPU operation and the hardware security circuit. In general, they are harder to find and exploit than clock glitches because in addition to the timing parameters, the amplitude and rising/falling times are variables.

A glitch could be an external electric field transient or an electro-magnetic pulse. One approach is using a miniature inductor consisting of several hundred turns of fine wire around the tip of a microprobe needle. A current injected into this coil will create a magnetic field, and the needle will concentrate the field lines [9].

7.4.1.3 Data Remanence

Security processors typically store secret key material in SRAM, from which power is removed if the device is tampered with. It is widely known that, at temperatures below −20° C, the contents of SRAM can be "frozen"; therefore, many devices treat temperatures below this threshold as tampering events. However, it was showed that the conventional wisdom no longer holds and that data remanence can be a problem even at higher temperatures [10].

Security engineers are interested in the period of time for which an SRAM device will retain data once the power has been removed. The reason for this is as follows. Many products perform cryptographic and other security-related computations using secret keys or other variables that the equipment's operator must not be able to read out or alter. The usual solution is for the secret data to be kept in volatile memory inside a tamper-sensing enclosure. On detection of a tampering event, the volatile memory chips are powered down or even shorted to ground. If the data retention time exceeds the time required by an opponent to open the device and power up the memory, then the protection mechanisms can be defeated.

Data remanence affects not only SRAM but other memory types as well, like DRAM, UV EPROM, EEPROM and Flash [11]. As a result, some information still can be extracted from memory that has been erased. This could create many problems for secure devices which assume that all the sensitive information is gone once the memory is erased.

Unlike SRAM which has only two stable logic states, EPROM, EEPROM and Flash cells actually store analog values in the form of a charge on the floating gate of a metal-oxide-semiconductor (MOS) transistor. The floating-gate charge shifts

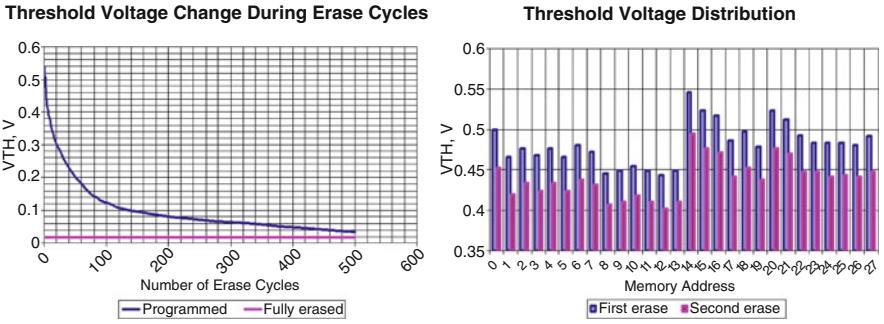


Fig. 7.1 Change of the threshold voltage during erase cycles for programmed and erased cells; threshold voltage change during second erase cycle for different memory cells

the threshold voltage of the cell transistor and this is detected with a sense amplifier when the cell is read. The maximum charge the floating gate can accumulate varies from one technology to another and normally is between 10^3 and 10^5 electrons. The amount of trapped charge can be detected by measuring the gate-induced drain leakage current of the cell or its effect can be observed indirectly by measuring the threshold voltage of the cell. In older devices, which had the reference voltage for the sense amplifier tied to the device supply voltage, it was often possible to do this by varying the device supply voltage. For example, all information can be successfully extracted from the Microchip PIC16F84A microcontroller after it has been erased several times (Fig. 7.1). In newer devices, it is necessary to change the parameters of the reference cell used in the read process, either by re-wiring portions of the cell circuitry or by using undocumented test modes built into the device by chip manufacturers.

The changes in the cell threshold voltage caused by write/erase cycles are particularly apparent in virgin and freshly-programmed cells. It is possible to differentiate between programmed-and-erased and never-programmed cells, especially if the cells have only been programmed and erased once, since virgin cell characteristics will differ from the erased cell characteristics. The changes become less noticeable after ten program/erase cycles.

7.5 Breaking the Security with Invasive Attacks

Invasive attacks start with partial or full removal of the chip package in order to expose the silicon die. There are several methods, depending upon the package type and the requirements for further analysis. For microcontrollers and smartcards, partial decapsulation is normally used, so that the device can be placed in a standard programmer unit and tested. Some devices cannot be decapsulated and still maintain their electrical integrity. In this case the chip die has to be bonded to a chip carrier

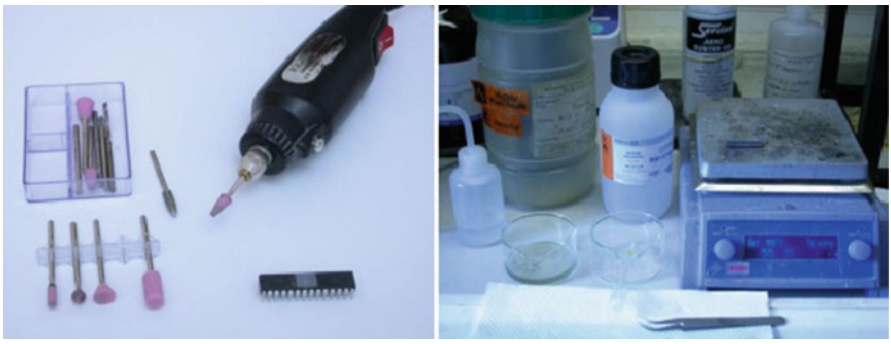


Fig. 7.2 Chip decapsulation tools

using a bonding machine which connects to the bonding pads on the die with thin aluminum or gold wire. Such bonding machines are available from different manufacturers and can be bought second-hand for less than $10,000. The contacts to the die can be also established using microprobing needles on a probing station.

It is a common opinion that decapsulation is a complicated process which requires a lot of experience. In fact it is not and anyone capable of carrying out chemical or biological work in the context of a standard high-school program can do this. All the necessary experience could be obtained by decapsulating a dozen different samples. Some precautions should be taken as the acids used in this process are very corrosive and dangerous; ideally, the work should be performed in a fume cupboard to prevent inhalation of the fumes from acids and solvents. Eyes should be protected with safety goggles and appropriate acid-resistant gloves should be worn as the acid will cause severe burns if it accidentally comes into contact with the skin. Protective clothing should be worn as well.

The process of manual decapsulation usually starts with milling a hole in the package so that the acid will affect only the desired area above the chip die (Fig. 7.2). The tools necessary for this operation are available from any do it yourself (DIY) shop for less than $20. The commonly used etching agent for plastic packages is fuming nitric acid (>95%), which is a solution of nitrogen dioxide in concentrated nitric acid. It is very strong nitrifying and oxidizing agent; it causes plastic to carbonize, and it also affects copper and silver in the chip carrier island and pins. Sometime a mixture of fuming nitric acid and concentrated sulphuric acid is used. This speeds up the reaction with some types of packages and also prevents the silver used in bonding pads and chip carrier from reacting. The acid is normally applied in small portions with a pipette into a premilled hole in a chip preheated to 50–70° C (Fig. 7.2). After 10–30 seconds the chip is sprayed with dry acetone from a washing bottle to remove the reaction products. This process has to be repeated several times until the die is sufficiently exposed. To speed up the process, the chip can be placed in a sand bath and the acid can be preheated in a glass beaker.

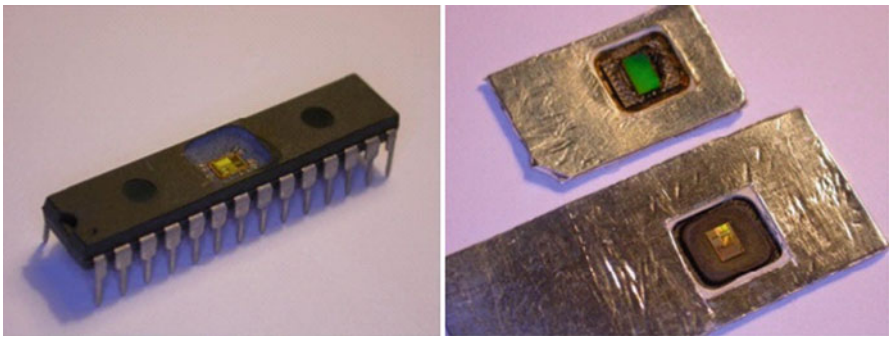


Fig. 7.3 Decapsulated chips

The acid residues can be removed from the etched plastic and from the chip surface by ultrasonic treatment. For that the chip is placed into a beaker with acetone and then put in an ultrasonic bath for 1–3 minutes. After washing the chip with acetone and drying it in an air jet, we have a clean and fully operational chip (Fig. 7.3).

A very similar approach can be used for decapsulating chips from the rear side. The only obstacle is the copper plate under the chip die which reacts slowly with the fuming nitric acid. That could create problems if the automatic decapsulator is used because the surrounding plastic will be etched away before this copper plate and the chip leads are very likely to be damaged. However, access to the rear side of the die can be established without using chemical etching. The chip package can be milled down to the copper plate which is then removed mechanically. The residues of the glue used to attach the die to the plate can be removed with solvents or by scraping it off with a wooden toothpick stick.

The same partial decapsulation technique can be used for smartcards as well (Fig. 7.3), although not all of them would maintain electrical integrity. Very often the chip has to be decapsulated completely and then bonded onto a chip carrier.

7.5.1 Deprocessing

The opposite process to chip fabrication is called deprocessing. A standard CMOS chip has many layers. The deepest doping layers inside the substrate form the transistors. The gate oxide layer isolates the gate from the active area of the transistors. The polysilicon layer on top of it forms the gates and interconnections. The interlayer oxide isolates conducting layers. Metal layers, usually made of aluminum, form the signal wires, and they are connected with other layers through "via" plugs. Finally, a passivation layer made out of silicon oxide or nitride protects the whole structure from moisture and air which could harm the die. In plastic packages the passivation layer is covered with a polymer layer, usually polyimide, to protect against sharp grains in the compound during the package formation.

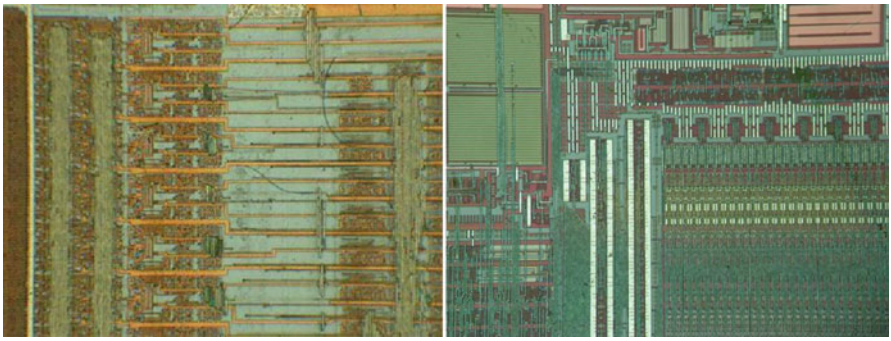


Fig. 7.4 Chemically etched chip (PIC16F77) and mechanically polished chip (ATtiny45)

There are two main applications of deprocessing. One is to remove the passivation layer, exposing the top metal layer for microprobing attacks. Another is to gain access to the deep layers and observe the internal structure of the chip.

Three basic deprocessing methods are used: wet chemical etching, plasma etching, also known as dry etching, and mechanical polishing [12]. In chemical etching each layer is removed by specific chemicals. Its downside is its isotropic nature, i.e., uniformity in all directions. That produces unwanted undercutting. As a result, narrow metal lines will have a tendency to lift off the surface. Isotropic etching also leads to etching through holes such as vias, resulting in unwanted etching of underlaying metallization (Fig. 7.4). Plasma etching uses radicals created from gas inside a special chamber. They react with the material on the sample surface to form volatile products which are pumped out of the chamber. As the ions are accelerated in an electric field they usually hit the surface of the sample perpendicularly. The removal of material is strongly anisotropic (directional). Only the surfaces hit by the ions are removed, sides perpendicular to their paths are not touched. Mechanical polishing is performed with the use of abrasive materials. The process is time-consuming and requires special machines to maintain the planarity of the surface. From the inspection perspective, the advantages of using polishing over wet and dry etching techniques is the ability to remove layer by layer and view features in the area of interest within the same plane (Fig. 7.4). It is especially useful on multilayer interconnect processes fabricated with advanced planarization techniques.

7.5.2 Reverse Engineering

This is a technique aimed at understanding the structure of a semiconductor device and its functions. In case of an ASIC or a custom IC, that means extracting information about the location of all the transistors and interconnections. In order

to succeed, a general knowledge of IC and VLSI design is required. All the layers formed during chip fabrication are removed one-by-one in reverse order and photographed to determine the internal structure of the chip. In the end, by processing all the acquired information, a standard netlist file can be created and used to simulate the device. This is a tedious and time-consuming process, but there are some companies which do such work as a standard service [13].

When it comes to reverse engineering smartcards and microcontrollers, both structural and program-code reverse engineering are required to understand how the device works. First, the security protection needs to be understood by partial reverse engineering of the chip area associated with it. Thus if memory bus encryption was used, the hardware responsible for this should be reverse engineered. Then, finally, the internal memory contents have to be extracted and disassembled to understand device functions.

A slightly different approach is required for reverse engineering FPGAs. Even if the security protection is defeated and the attacker manages to extract the configuration bitstream file from the device, he will have to spend a substantial amount of time and effort to convert it into the logic equations and primitive blocks for further simulation and analysis.

The most important tool for reverse engineering silicon chips down to 0.18 μm feature size is an optical microscope with a digital camera to produce mosaics of high-resolution photographs of the chip surface. Not every microscope would do. As light cannot pass through the chip, the microscope should have reflected light illumination. The image should be sharp and without geometric distortion and color aberration, otherwise it will not be possible to stick all the images together. The most important parameters of the microscope are resolution and magnification. The resolution of a microscope mainly depends upon its objective lenses and is defined as the smallest distance between two points on a specimen that can still be distinguished as two separate entities. Resolution is a somewhat subjective value in microscopy because at high magnification an image may appear nonsharp but still be resolved to the maximum ability of the objective.

Layout reconstruction requires the images of all the layers inside the chip to be combined. The images are normally taken automatically using a motorized stage to move the sample and special software to combine all the images together [14]. Normally, for semiconductor chips fabricated with 0.13 μm or smaller technology, images are created using a SEM which has a resolution better than 10 nm.

7.5.3 Microprobing

The most important tool for invasive attacks is a microprobing station (Fig. 7.5). It consists of five elements: a microscope, stage, device test socket, micromanipulators and probe tips. The microscope must have long working distance objectives – sufficient enough to accommodate six to eight probe tips (Fig. 7.5) between the

Fig. 7.5 Probing station and probing needles

sample and the objective lens. It should also have enough depth of focus to follow the probe tip movement. Usually the microscope has several objectives to accommodate different magnification and depths of focus. Lower magnification with greater focus depth is used for coarse location of the probe tip and higher magnification for placing the tip on a conductor wire or a test point. The chip is normally placed in a test socket that provides all the necessary signals and is controlled by a computer.

On a stable platform around the test socket, several micropositioners are installed. They allow us to move a probe tip with submicron precision. The probe tip can be either passive or active. The passive tip, for example Picoprobe T-4, can be used for both eavesdropping and injecting signals, but as it is normally connected directly to an oscilloscope, it has low impedance and high capacitance. As a result it cannot be used for probing internal signals on the chip, except for the bus lines which are usually buffered. Another application for the passive tips is making connections to the bonding pads on a fully decapsulated chip. The active tip, for example Picoprobe 12 C, has a FET amplifier close to the end of the tip. Active tips offer high bandwidth (1 GHz for Picoprobe 28) with low loading capacitance (0.02 pF for Picoprobe 18 B) and high input resistance ($>100\,\mathrm{G}\Omega$ for Picoprobe 18 B). The probe tips are made out of a tungsten wire which is sharpened to $<0.1\,\mu\mathrm{m}$ at the end for probing small features.

Modern probing stations benefit from full automatic control over the microscope, stage and micropositioners. For simple applications, a manually controlled probing station is enough and can be bought second-hand for less than $10,000. Passive probe tips are very cheap (less than $5 each) but active probes are quite expensive – over $60 for the tip plus over $1,000 for the tip holder with amplifier. However, they can easily be built from a $2 operational amplifier and a passive tip (Fig. 7.5).

7.6 Breaking the Security with Semi-Invasive Attacks

7.6.1 UV Attacks

These are among the oldest attacks used on microcontrollers since their release in the mid 70's. UV attacks were often considered invasive attacks before. But as most of them require only decapsulation of the chip, they certainly belong to the class of semi-invasive attacks. These attacks can be applied to many OTP and UV EPROM microcontrollers as their protection is designed to withstand low cost noninvasive attacks only.

The attack can be divided into two stages – finding the fuse and resetting it to the unprotected state with a UV light. As the security fuse is normally designed such that it cannot be erased earlier than the program memory, the UV light cannot be applied to the whole chip. Either the memory must be protected with opaque material, or the UV light can be applied to the fuse selectively by using a microscope or a UV laser.

As well as EPROM memory, many floating-gate memory devices are also susceptible to UV attacks [15]. Meantime, chip designers have more freedom in choosing different protections against such attacks. As the EEPROM and Flash cells can change their state in both directions, the obvious thing to do is to use an erased state of the cell to indicate the alarm state and a programmed state to correspond to disabled security. Minimal changes to the control logic will do the job. This is widely used in Flash microcontrollers from many manufacturers.

7.6.2 Advanced Imaging Techniques

Visual observation under a microscope is the first step in semiconductor analysis. As feature sizes of transistors shrink each year, structures on the chip surface become more and more difficult to observe. Down to 0.8 μm technology, it was possible to identify all the major elements of microcontrollers – ROM, EEPROM, SRAM, CPU and even instruction decoder and registers within the CPU. On chips built using 0.5 μm or 0.35 μm processes, one can hardly distinguish ROM, Flash and SRAM, whereas on chips with 0.25 μm or smaller transistors, almost nothing can be seen. This is caused not only by the small feature sizes, but most of all by multiple metal layers covering the chip surface (up to eight on modern 0.13 μm chips). In addition, planarization technology often involves filling blank spaces on metal layers with metal pads which block the optical path as well.

One approach is to use IR light, either reflected or transmitted, and observe the chip from its rear side. Silicon is almost transparent to photons with wavelengths >1100 nm. However, highly doped silicon wafers used in some modern chips, are less transparent to IR light and more intensive light source or an IR camera with higher sensitivity is required. Backside imaging is widely used in failure analysis

tasks, from locating the failures in transistors or interconnections to navigation during a FIB work. There are special microscopes designed for such applications, for example the BEAMS V-2000 from Hypervision. Needless to say, such systems cost a tremendous amount of money and can only be afforded by relatively large companies. Yet, low budget laboratories could use NIR extended microscopes with IR-sensitive video cameras.

Laser radiation can ionize an IC's semiconductor regions if its photon energy exceeds the semiconductor band gap (>1.1 eV or <1100 nm). Laser radiation with 1.06 μm wavelength (1.17 eV photon energy) has a penetration depth of about 700 μm and provides good spatial ionization uniformity for silicon devices. In active photon probing, a scanned photon beam interacts with an IC. Photons with energies greater than the band gap of silicon generate electron-hole pairs in the semiconductor. Photons with lower energies can still interact with p-n junctions, but with only a heating effect taking place, which is significantly weaker than the photovoltaic effect.

There are different scanning techniques used for photon probing in failure analysis [12]. As a photon source they normally use a laser scanning microscope. Although such microscopes have a big advantage of fast scanning – about one frame per second, their price is too high for small research labs. Usually, less expensive but much slower approach of a stationary laser source and a sample moved on an X-Y motorized stage can be used.

There are two major laser scanning techniques which can be used for hardware security analysis. One is called optical beam induced current (OBIC) and is applied to an unbiased chip to find the active doped areas on its surface [16]. Another, called light-induced voltage alteration (LIVA), applied to a chip under operation [17]. In OBIC, photocurrents are used directly to produce the image. For that, the analyzed chip is arranged with its power supply pin connected to a current amplifier and the values are registered on the computer via an acquisition board. The active areas can be seen as they produce higher current, but most of the chip is covered with metal layers which the laser cannot penetrate, so these areas do not produce any current (Fig. 7.6). In LIVA, images are produced by monitoring the voltage changes of the constant current power supply as the optical beam is scanned across the IC surface. It can be seen that memory cells have different states: where the cell holds a "1" the top is brighter, and where it is a "0" the bottom is (Fig. 7.6).

7.6.3 Optical Fault Injection

It is a new class of attacks on secure microcontrollers and smartcards [18]. Illumination of a target transistor causes it to conduct, thereby inducing a transient fault. Such attacks are practical. They do not even require expensive laser equipment and can be carried them out using a $5 laser pointer. For example, this can be used to set or reset any individual bit of SRAM in a microcontroller. Unless suitable countermeasures are taken, optical fault injection may also be used to induce errors

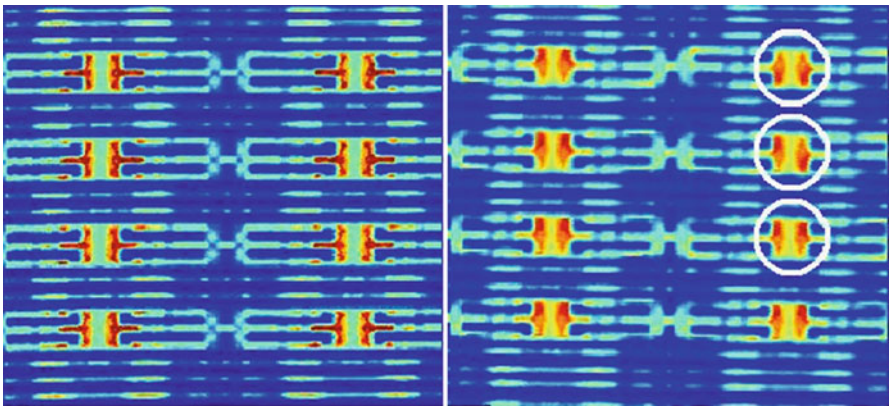


Fig. 7.6 Laser scan of unpowered and powered-up SRAM in PIC16F84 microcontroller

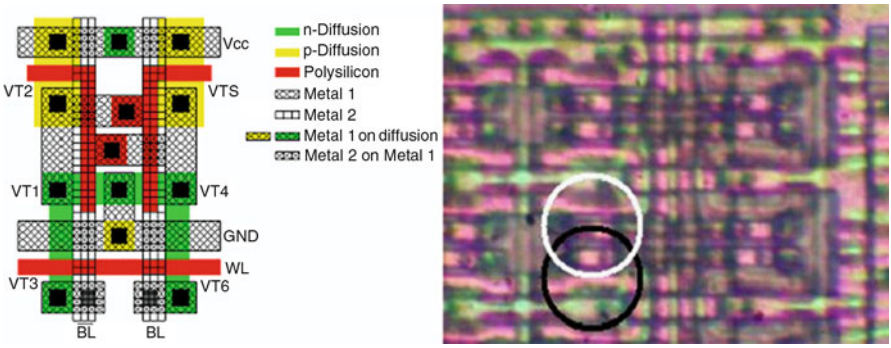


Fig. 7.7 Layout of SRAM cell and SRAM area in PIC16F84 microcontroller

in cryptographic computations or protocols, and to disrupt the processor's control flow. It thus provides a powerful extension of existing glitching and fault analysis techniques. This vulnerability may pose a big problem for the industry, similar to those resulting from probing attacks in the mid 1990s and power analysis attacks in the late 1990s.

A standard SRAM cell consists of six transistors. Two pairs of p- and n-channel transistors create a flip-flop, while two other n-channel transistors are used to read its state and write new values into it. The layout of the cell is shown on Fig. 7.7. The transistors VT1 and VT2 create the CMOS inverter; together with the other similar pair, they create the flip-flop which is controlled by the transistors VT3 and VT6.

If the transistor VT1 could be opened for a very short time by an external stimulus, then it could cause the flip-flop to change state. By exposing the transistor VT4, the state of the cell would be changed to the opposite value. The main difficulties we might anticipate are focusing the ionizing radiation down to several micrometers spot and choosing the proper intensity. In the original experiments the Microchip

PIC16F84 microcontroller with 68 bytes of on-chip SRAM memory was used [18]. The light from a photoflash lamp was focused using the microscope optics. By shielding the light from the flash with an aperture made from aluminum foil the state of only one cell can be changed. The array, under maximum magnification, is shown in Fig. 7.7. Focusing the light spot from the lamp on the area shown by the white circle caused the cell to change its state from "1" to "0," with no change if the state was already "0." By focusing the spot on the area shown by the black circle, the cell changed its state from "0" to "1" or remained in state "1."

EPROM, EEPROM and Flash memory cells are even more sensitive to fault injection attacks. This happens because the currents flowing inside the floating gate cell are an order of magnitude smaller that inside the SRAM cell. There are some new optical fault injection attack techniques introduced recently. One is local heating attacks [19] which use lasers to implement modification attacks on EEPROM and Flash memory devices. This was achieved with inexpensive laser-diode module mounted on a microscope. By locally heating up a memory cell inside a memory array, the contents of the memory can be altered. As a result, the security of a semiconductor chip can be compromised. Even if changing each individual bit is not possible due to the small size of a memory cell, cryptographic keys can still be recovered with brute force attacks. Another is bumping attacks [20] aimed at data extraction from secure embedded memory, which usually stores critical parts of algorithms, sensitive data and cryptographic keys. As a security measure, read-back access to the memory is not implemented leaving only authentication and verification options for integrity check. Verification is usually performed on relatively large blocks of data, making brute force searching infeasible. By attacking the security in three sttif, the search space can be reduced from infeasible 2^{100} to affordable 2^{15} guesses per block of data. This progress was achieved by finding a way to preset certain bits in the data path to a known state using optical fault injection.

Existing high-end chip-defense techniques, such as top-layer metal shielding and bus encryption, may make an attack using these techniques more complicated, but are not enough. A sophisticated attacker can defeat metal shielding by using IR light or X-rays, while bus encryption can be defeated by attacking registers directly.

7.6.4 Optical Side-Channel Analysis

Transistors emit photons when they switch. This has been well known for decades and is actively used in failure analysis. So far, observation of such emissions was associated with sophisticated and expensive equipment, because only a very limited number of photons emitted per every switch – usually 10^{-2} to 10^{-4}. The peak of emission is in the near-infrared spectrum (900 to 1200 nm) and this poses restrictions on sensors selection. The emission comes from an area close to the drain and primarily from the n-channel MOS transistor. Optical emission significantly increases at higher power supply voltages [21].

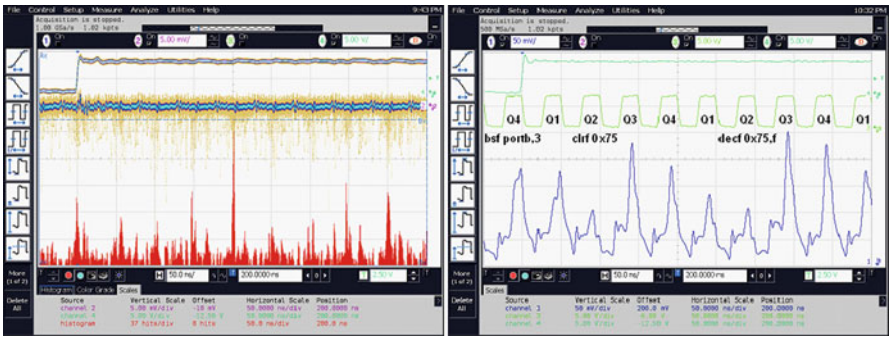


Fig. 7.8 Photon emission analysis and power analysis results for PIC16F628 microcontroller

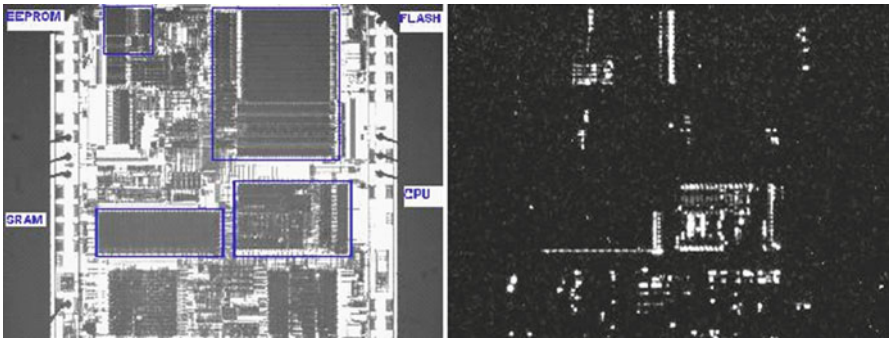


Fig. 7.9 Optical image and photon emission from PIC16F628 microcontroller

Optical emission has good correlation with power analysis and can be used for characterization of leaking areas for later improvement of protection against power analysis attacks. The results, presented in Fig. 7.8, reveal that optical emission has higher bandwidth and thus data appearing at different times can be separated for further analysis.

Modern low-cost charge-coupled device (CCD) cameras are adequate for detecting photons emitted by modern CMOS circuits. Photomultiplier tubes (PMT) are very fast, but they have limited sensitivity in the IR region. Monochrome CCD cameras have good IR sensitivity and low dark current, which is important with long exposure times.

A standard microscope setup with a CCD camera mounted on top and a sample in a test socket can be used for analysis. Hobbyist astronomical cameras with active cooling appeared to be best suited for low-cost optical emission analysis by having good IR sensitivity and extremely low dark current. The emission acquired from a microcontroller using a 2× objective lens is presented in Fig. 7.9. A closer look with a 10× objective revealed that the value of data read from the internal memory is clearly visible (Fig. 7.10).

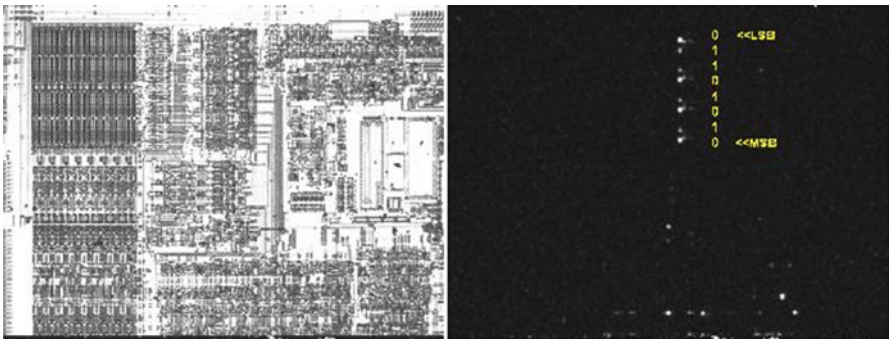


Fig. 7.10 Optical image and photon emission from EEPROM in PIC16F628 microcontroller

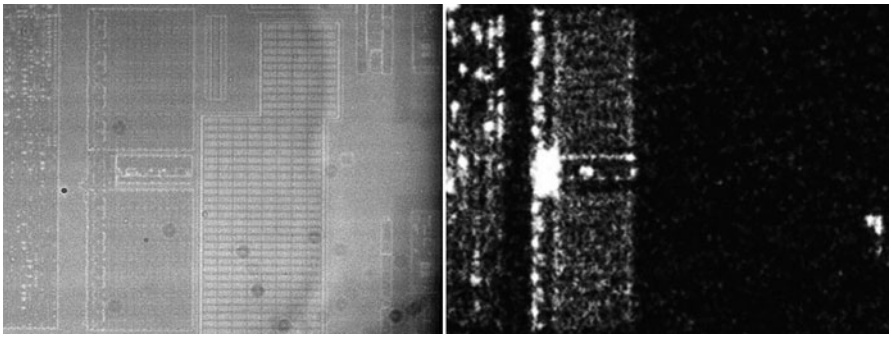


Fig. 7.11 Backside infrared optical image and photon emission from SRAM in A3P250 FPGA

Modern deep-submicron chips emit photons as well. However, the front-side approach no longer works due to multiple metal layers which completely block the emission. For chips built with 0.35 μm and smaller technology, a backside approach is required. In order to achieve reasonable emission, the power supply voltage must be increased by 30–50%. The example of optical emission acquired from the backside of 130 nm FPGA chip above the SRAM area is presented in Fig. 7.11.

Optical emission analysis can lead to possible data extraction from semiconductor chips. That way, the security can be compromised in various chips from microcontrollers and smartcards to FPGAs and ASICs. Possible countermeasures include asynchronous designs and employing data encryption.

7.6.5 Optically Enhanced Position-Locked Power Analysis

This is a great example of combining noninvasive and semi-invasive attack methods for better result [22]. Optical enhancement of power analysis is a new and innovative technique that allows the current through an individual transistor to become visible

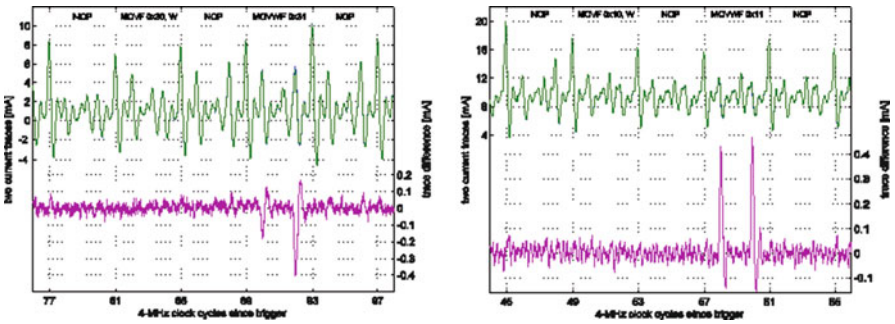


Fig. 7.12 Laser-enhanced influence on Write operation and power analysis of a single-bit change during Write

in the IC power trace. In conventional power analysis, power consumption is measured for a whole chip rather than on a small area of interest. As a result, power transitions in areas that are not relevant to the data processing also affect the power trace. Also, the power fluctuations are affected by the number of bits being set or reset (Hamming weight of data), rather than the actual value of the manipulated data.

By focusing a laser on a specific area on the chip surface, it is possible to monitor the logic state of an individual transistor, as well as the activity of a particular memory cell. This is highly useful for security analysis, allowing faster and less expensive solutions.

When a laser is focused on the VT1 transistor of the PIC16F84 SRAM memory cell (Fig. 7.7) and writing into this cell happens, an about 0.4 mA change in the power trace can be observed (Fig. 7.12). For comparison, the conventional power analysis results showed a similar change in the power consumption for a single-bit difference in the memory contents (Fig. 7.12). However, the same technique applied to the memory cells being read, did not produce any noticeable results, unless many power traces were averaged. This is because writing into an SRAM cell causes a significantly larger current response than a read operation with low laser-injected current.

By focusing the laser on the region between the VT1 and VT4 transistors, significantly higher changes in the power trace can be observed (Fig. 7.13), and what is more important, for both write and read operations (Fig. 7.13), thus allowing access detection. This happens because the timing characteristic of the cell changes when both inverters are influenced with a laser.

Interesting results can be achieved with the laser focused on the area between the cell-select transistors (VT3 and VT6). In this case, access to any of the memory cells inside the memory array column will produce a very noticeable difference in the power consumption. The same response can be obtained when a laser with higher power is focused between VT1 and VT4. These approaches can be used for access-event triggering.

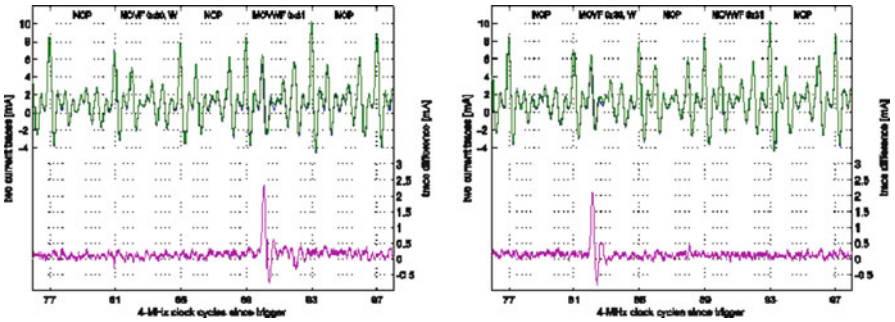


Fig. 7.13 Laser-enhanced VT1+VT4 influence on Write and Read operations

Modern chips normally have multiple metal layers over their active areas, preventing direct access with a laser beam. Accessing the chip from its rear side can circumvent this obstacle and the power-trace difference is very similar to the one from the front side.

7.7 Countermeasures Against Physical Attacks

Since the late 1990s, smartcard vendors have made invasive attacks much more difficult. Smartcards typically have a sensor mesh implemented in the top metal layer, consisting of a serpentine pattern of sensor, ground and power lines. All paths in this mesh are continuously monitored for interruptions and short circuits, and cause reset or zeroing of the EEPROM memory if alarmed. In some recent smartcards further protection against microprobing attacks is used such as EEPROM data memory bus encryption. Even if the attacker manages to pick up the signals from the data bus he will not be able to recover passwords, secret keys or other sensitive information from it. This protection was aimed at preventing invasive and semi-invasive attacks. At the same time noninvasive attacks could still be successful as the CPU normally has full access control to unencrypted information.

Another improvement worth mentioning is moving from the standard building-block structures like CPU instruction decoder, register file, ALU and I/O circuits, to a complete ASIC-like logic design. This design approach is called “glue logic” and it is widely used in smartcards. Glue logic makes it virtually impossible to tap into the card’s information by manually finding signals or nodes to attack physically (Fig. 7.14). The glue logic design could be done automatically with using special design tools.

Technological progress on its own is increasing the costs to the attackers. Ten years ago it was possible to use a laser cutter and a simple probing station to get access to any point on the chip surface, but for modern deep submicron semiconductor chips very sophisticated and expensive technologies must be used.

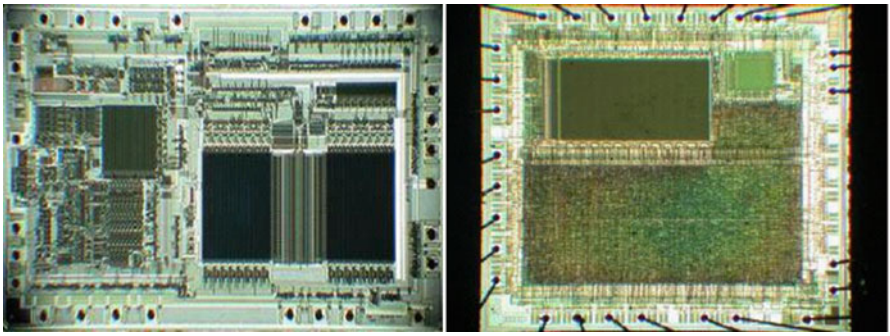


Fig. 7.14 MC68HC705P6A microcontroller with building blocks and SX28 microcontrollers with glue logic

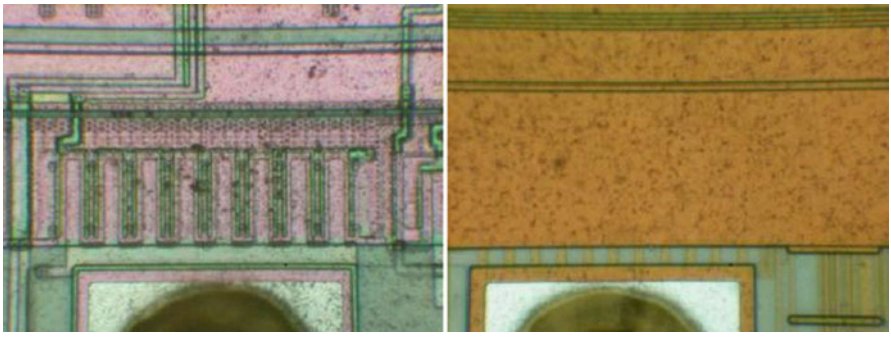


Fig. 7.15 PIC16F877 and PIC16F877A microcontrollers

That excludes most potential attackers. For example, the structure of the Microchip PIC16F877 microcontroller can be easily observed and reverse engineered under a microscope (Fig. 7.15). The second metal layer and polysilicon layer can still be seen even if buried under the top metal layer. This is possible because each subsequent layer in the fabrication process follows the shape of the previous layer. Under a microscope the observer sees not only the highest layer but also edges that reveal the structure of the deeper layers. In 0.5 μm and smaller technologies, for example in the Microchip PIC16F877A microcontroller, each predecessor layer is planarized using chemical-mechanical planarization (CMP) process before applying the next layer. As a result the top metal layer does not show the impact of the deeper layers (Fig. 7.15). The only way to reveal the structure of the deeper layers is by removing the top metal layers either mechanically or chemically.

Modern smartcard protection features now typically include internal voltage sensors to protect against under- and over-voltages used in power glitch attacks; clock frequency sensors to prevent attackers slowing down the clock frequency for

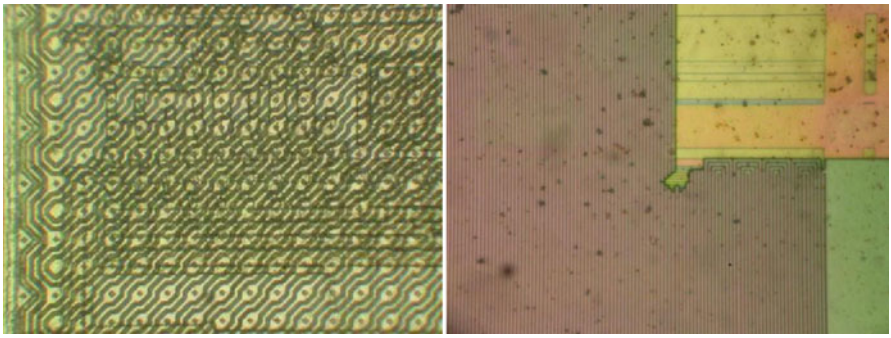


Fig. 7.16 Top metal sensor meshes in smartcard chips

static analysis and also from raising it for clock glitching attacks; top-metal sensor meshes (Fig. 7.16); internal bus hardware encryption to make data analysis more difficult; and light sensors to prevent an opened chip from functioning. Software access to internal memory is often restricted by passwords, so that simple hacks to read out all the memory on the bus are no longer available.

7.8 Conclusion

There is no such thing as absolute security. A determined hacker can break any protection given enough time and resources. The question is how practical it would be. If it takes ten years to break a device which in three years is replaced by a successor with even better security, then the defense has won. On the other hand, the vulnerability could be buried within the design blocks itself. What if your secure system was designed from insecure components? In the end, the overall security of your system is determined by the least secure element. Even if you implement a provably secure protocol, your system could be broken if the key can be easily extracted from the hardware by mechanical or optical probing. Therefore, whenever you design a secure system, proper security evaluation of all the components must be performed. Of course it is impossible to avoid all problems; a reasonable goal is to make the process of breaking your design more expensive and time-consuming. With luck, potential attackers will switch to other products rather than spending money and effort on breaking yours.

The first step in designing adequate protection is to understand motivations of prospective attackers and possible attack scenarios. Then depending on the most likely class of attackers – outsiders, insiders or funded organizations, more realistic estimation of their capabilities can be made. From that a decision on what to protect and the level of protection can be made.

References

1. Abraham DG, Dolan GM, Double GP, Stevens JV (1991) Transaction Security System. IBM Syst J 30(2): 206–229
2. U.S. Department of Commerce. Security requirements for cryptographic modules. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf. Accessed 10 January 2011
3. Common Criteria Evaluation and Validation Scheme. http://www.niap-ccevs.org/. Accessed 10 January 2011
4. Skorobogatov S (2005) Semi-invasive attacks – a new approach to hardware security analysis. In: Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005
5. Kocher PC (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Advances in Cryptology, CRYPTO'96, LNCS, vol 1109. Springer-Verlag, Berlin, Heidelberg, New York, pp 104–113
6. Dhem J-F, Koeune F, Leroux P-A, Mestre P, Quisquater J-J, Willems J-L, A practical implementation of the timing attack. In: Proceedings of CARDIS'98, Smart Card Research and Advanced Applications, 1998
7. Chaum D (1983) Blind signatures for untraceable payments. Advances in Cryptology: Proceedings of Crypto 82. Plenum Press, NY, USA, pp 199–203
8. Goodspeed T (2008) Side-channel Timing Attacks on MSP430 Microcontrollers. Black Hat, USA
9. Quisquater J-J, Samyde D (2002) Eddy current for magnetic analysis with active sensor. In: UCL, Proceedings of Esmart 2002 3rd edn., Nice, France, September 2002
10. Skorobogatov S (2002) Low temperature data remanence in static RAM. In: Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory, June 2002
11. Skorobogatov S (2005) Data remanence in flash memory devices. Cryptographic Hardware and Embedded Systems Workshop (CHES 2005), LNCS 3659. Springer, Berlin, Heidelberg, New York, pp 339–353
12. Wagner LC (1999) Failure Analysis of Integrated Circuits: Tools and Techniques. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
13. Chipworks. http://www.chipworks.com/. Accessed 10 January 2011
14. Blythe S, Fraboni B, Lall S, Ahmed H, de Riu U (1993) Layout reconstruction of complex silicon chips. IEEE J Solid-State Circuits 28(2): 138–145
15. Fournier JJ-A, Loubet-Moundi P (2010) Memory address scrambling revealed using fault attacks. In: 7th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010), IEEE-CS Press, USA, August 2010, pp 30–36
16. Wills KS, Lewis T, Billus G, Hoang H (1990) Optical beam induced current applications for failure analysis of VLSI devices. In: Proceedings International Symposium for Testing and Failure Analysis, 1990, p 21
17. Ajluni C (1995) Two new imaging techniques promise to improve IC defect identification. Electr Design 43(14): 37–38
18. Skorobogatov S, Anderson R (2002) Optical fault induction attacks. In: Cryptographic Hardware and Embedded Systems Workshop (CHES 2002), LNCS 2523, Springer-Verlag, Berlin, Heidelberg, New York, pp 2–12
19. Skorobogatov S (2009) Local heating attacks on flash memory devices. In: 2nd IEEE International Workshop on Hardware-Oriented Security and Trust (HOST-2009), San Francisco, CA, USA, IEEE Xplore, 27 July 2009
20. Skorobogatov S (2010) Flash memory 'bumping' attacks. Cryptographic Hardware and Embedded Systems Workshop (CHES 2010), LNCS 6225, Springer, Berlin, Heidelberg, New York, pp 158–172, August 2010

21. Skorobogatov S (2009) Using optical emission analysis for estimating contribution to power analysis. In: 6th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009), IEEE-CS Press, Switzerland, pp 111–119
22. Skorobogatov S (2006) Optically enhanced position-locked power analysis. Cryptographic Hardware and Embedded Systems Workshop (CHES 2006), LNCS 4249, Springer, Berlin, Heidelberg, New York, pp 61–75

Chapter 8
Side Channel Attacks and Countermeasures

Ken Mai

8.1 Introduction

Side-channel attacks bypass the theoretical strength of cryptographic algorithms by exploiting weaknesses in the cryptographic system hardware implementation via nonprimary, side-channel inputs and outputs. Commonly exploited side-channel outputs include: power consumption, electromagnetic (EM) emissions, light, timing, and sound (Fig. 8.1). Commonly used side-channel inputs include: supply voltage, temperature, light, and other primary signal inputs unrelated to the cryptographic block. The attacks themselves combine observation of side-channel outputs, manipulation of side-channel inputs, observation of primary outputs, and manipulation of primary inputs with increasingly complex analysis techniques to discover secret information from the cryptographic system. Attacks that exploit side-channel output are often termed *passive* side-channel attacks, while attacks that exploit side-channel inputs are called *active* side-channel attacks or fault injection attacks.

In this chapter, we will focus our discussion on passive side-channel attacks against integrated circuits and hardware countermeasures. We take the perspective of a custom ASIC designer building a block that performs a cryptographic operation seeking to safeguard their design against side-channel attacks. Passive side-channel attacks have been highly successful in compromising cryptographic systems due to their largely noninvasive nature, often modest equipment requirements, and the general availability of high-performance computing for data analysis. Additionally, unless a cryptographic system has been designed specifically for low side-channel emissions, its emissions in a particular side-channel can be quite high. Side-channel attacks and algorithmic countermeasures specific to microprocessor systems (e.g., cache attacks, some timing attacks) are outside of the scope of this chapter, but are

K. Mai (✉)
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
e-mail: kenmai@andrew.cmu.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_8, © Springer Science+Business Media, LLC 2012

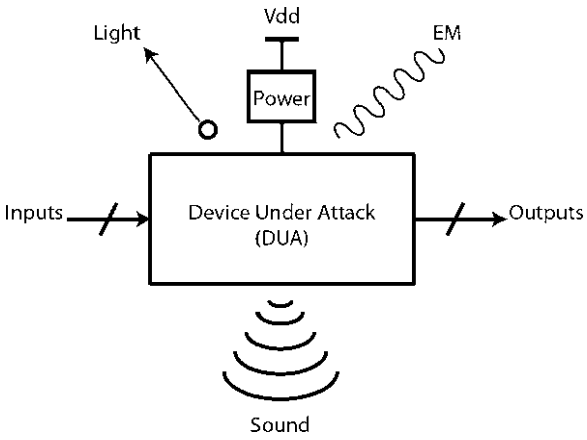


Fig. 8.1 Example side-channel emissions from device under attack (DUA): optical, power, EM, and acoustic

covered in Chap. 11. Active side-channel attacks and fault injection are covered in Chap. 7.

While this chapter presents an introduction to passive side-channel attacks and hardware countermeasures, the extant work on this topic could fill multiple large volumes. We will cover the most commonly exploited side-channel outputs and hope to serve as a jumping off point for more in-depth understanding and research. We first discuss the basic origin of some common side-channel emissions and measurement techniques. Next, we examine how side-channel emissions information can be used to attack a cryptographic block. Then, we explore hardware countermeasures against these attacks.

8.2 Side Channels

During World War II was one of the first times that side-channel phenomena were studied in-depth particularly in relation to cryptographic systems. A recently partially declassified 1972 NSA document describes how in 1943, Bell Labs employees working on encryption systems noted that whenever the system activated, spikes appeared on an oscilloscope in another part of the lab, which could be interpreted to recover the plaintext data [1]. This began a long series of government-sponsored research on side-channel attacks and countermeasures including the NSA TEMPEST program. Much of this early work demonstrated attacks and countermeasures still used in modern systems. In the 1990s, side-channel attacks gained prominence in academic and industrial circles with the publication of a number of papers and demonstrations of successful attacks [2, 3]. In the remainder of this section, we discuss a number of commonly exploited side-channel emissions.

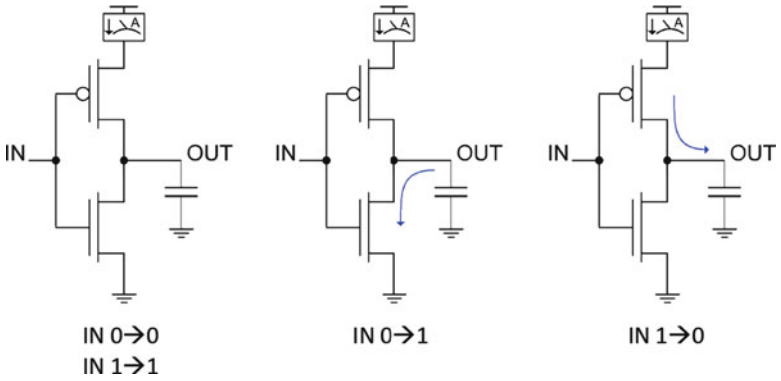


Fig. 8.2 Static CMOS inverter. No dynamic power is dissipated if input does not change. For a 0-to-1 transition on IN, charge is dumped to Gnd. For a 1-to-0 transition on IN, charge is pulled from Vdd

8.2.1 Power Consumption

One of the most successfully exploited side-channels in the modern era has been the power consumption of the cryptographic system. Power consumption in any IC can be classified as either dynamic or static power. Dynamic power is power consumption due to the charging or discharging of on-die capacitances. Any circuit node in a system has capacitance due to the transistor terminal parasitics (gate or diffusion) and wires attached to it. As this node changes voltage in a digital system to indicate a 1 or 0 status, charge is either drawn from Vdd to this node or dumped from this node to Gnd. Figure 8.2 shows an illustrative example with a static CMOS inverter.

For a given node, the dynamic power is calculated as $P(\text{dynamic}) = 1/2^* C^* \text{Vdd}^* \Delta V^* f$, with C being the capacitance of the node, Vdd being the supply voltage, ΔV being the swing in voltage of the node, and f being the frequency of toggling of the node. The $1/2$ factor is included, since we only care about half of the transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$, depending on whether we are measure charge drawn from Vdd or charge dumped to Gnd. For an entire system, this is more generally written as $P_{\text{tot}}(\text{dynamic}) = \Sigma(1/2^* C^* \text{Vdd}^* \Delta V^* f)$. This is often rewritten with f being the clock frequency of the system, and the activity factor merged into the C (changing C to C_{eff}).

Static power arises from either circuits that intentionally draw static power or leakage current in the transistors. Examples of circuits that intentionally draw static power would be analog circuits with current sources and certain types of digital logic gates (e.g., ECL gates, pseudo-NMOS gates). Leakage current in transistors can be caused by subthreshold leakage in the device channel, reversed-biased PN junction leakage, and gate leakage. Process technologies at 45 nm and below have used high-K gate dielectrics (and subsequently required metal gate) to increase the

transistor gate thickness in order to greatly reduce the gate leakage. In the recent past, leakage power has become a significant concern, and is a primary obstacle to ideal supply voltage scaling. This in turn, leads to reduced supply voltage scaling and limited power savings due to process scaling. In some designs, circuits that burn static power are clocked or power gated in order to only activated them when needed.

The power supply distribution network on a chip typically consists of one or more large grids of metal wires on multiple levels of metal. Usually a pair of upper level metals will be used for the global distribution network, as they are the thickest, lowest resistance wires. One or more lower level grids will be used to directly connect to the transistors. In addition to the distribution gird, large capacitors (called decoupling capacitors) sit between Vdd and Gnd to stabilize the Vdd against noise and serve as a fast local supply of charge owing to the inductive parasitics in the power supply distribution network. The chip internal power supply gird is fed by a large number of chip pads through either solder bumps or bond wires. Typically about half of all chip pads are used for the power supplies. The chip pads are connected to the board via the chip package. Finally, systems have voltage regulator modules (VRM) on the board to step down high externally supplied voltages to the lower voltages needed by the chip internals. Many systems also use dynamic voltage and frequency scaling (DVFS) to dynamically adjust their power consumption based on the current workload. DVFS systems require adjustable Vdd and clock frequency, and hence a VRM and adjustable clock source.

The idea behind a side-channel power consumption attack is that logic circuits typically consume differing amounts of power based on their input data. By observing the power consumption during a cryptographic operation an attack may be able to distinguish which functional blocks are active and some information about the data they are operating on. Notably, the most commonly used logic family, static CMOS, will dissipate no dynamic power if their inputs remain stable, but dissipate significant dynamic power if the inputs change to a combination that requires that the output state change.

The most common measurement point for power traces is either immediately before or after the board VRM, since the external power supply connects at a single point and the regulated voltage exits at a single point. Beyond the on-board VRM, power is usually distributed as a PCB plane, then through multiple package pins (or bumps) to the IC, and thus difficult to isolate. The simplest measurement technique is to place a resistor in-line with the power supply and measure the voltage across the resistor. The power supply current and power consumption can then be calculated. A low resistance resistor is optimal to reduce the voltage drop across the resistor, but this reduces the measured voltage magnitude. Alternately, a current probe, sometimes called a current clamp, can be used noninvasively to measure the current through a wire from the power supply to the board. Current probes based on the Hall Effect can be used to measure AC (kHZ range) and DC currents, although use of these sensors may be considered to be measuring the EM side-channel rather than power consumption.

Given the complexity and size of modern multicore CPUs and SoCs, some attackers have sought to better localize the power measurement of the cryptographic block. Most large ICs have multiple (often many hundred) Vdd and Gnd connections to the IC package. If the one closest to the cryptographic block could be isolated then the attacker would likely be able to attain a higher SNR power trace. There are a number of techniques including use of small inductive loops that can help isolate individual power supply connections [4–6]. Additionally, this would bypass the high frequency filtering of the on-board power supply network (i.e., decoupling capacitors and VRMs).

8.2.2 Electro-Magnetic

Electro-Magnetic (EM) emissions originate from the acceleration of charges in a conductor, referred to as the antenna. In the near-field, or the space approximately two wavelengths or less from the antenna, the electric and magnetic field effects are dominant over the radiated EM wave. However, their intensity falls off quickly with distance from the antenna, at a rate greater than the square of the distance. At frequencies of interest, the near-field region can be quite large. For, example, a 1 MHz signal has a wavelength of 300 m. However, high frequency signals have a much smaller near-field region; a 10 GHz signal has a wavelength of only 3 cm. The space from approximately two wavelengths to infinity is termed the far-field, where the radiated EM wave power is dominant. The radiated EM wave's intensity falls off as square of the distance. In general, the signal intensity is significantly lower in the far-field than the near-field, and thus sensing of the EM emissions in the near-field is preferable.

EM emissions, particularly via near-field inductive and capacitive coupling, can also modulate other signals on the die. This gives an indirect method of observing signals of interest, by using other signals as carriers. Researchers have observed both amplitude modulation (AM) and frequency modulation (FM) of carrier signals due to EM coupling [7, 8]. The clock signal and power supply rails are particularly good carriers due to their high EM emissions. Agarwal et al. classified EM emissions into *intentional*, direct EM emissions from the signal of interest, and *unintentional*, EM emissions resultant from a mixture of coupling of the signal of interest on to carrier signals and direct near- and far-field emissions [8].

At modern IC fabrication geometries, signal source localization can be challenging given the high density of radiating antennae. Careful placement and orientation of the sensor and/or use of multiple sensors can aid source localization as well as boost signal intensity and SNR. The placement and orientation of near-field sensors can have a large impact on the achieved SNR. In ICs, the antennas are primarily made up of the metal wire interconnects that are fabricated in planes parallel to the silicon surface. These interconnects can range in length from less than a micron to many millimeters. Thus, the emitted fields are primarily perpendicular to the silicon surface near the chip (Fig. 8.3). When placed close to the chip, near-field electric and magnetic field sensors are optimally oriented in a plane parallel with the plane

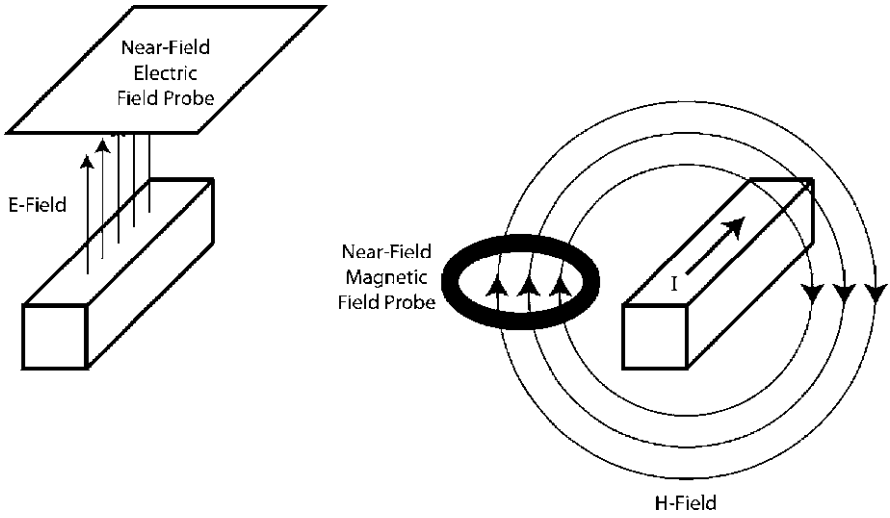


Fig. 8.3 Optimal near-field electric and magnetic field sensor orientation is in a parallel plane with the integrated circuit [9]. Both electric and magnetic fields exit the plane of the chip vertically

of the silicon [9]. For higher signal-to-noise-ratio, decapsulation of the chip may be necessary to both get physically closer to the chip and to remove any packaging materials that may be attenuating the signal of interest. Additionally, high frequency signals are generally preferable to low frequency ones due to the lower noise at higher frequencies [8].

Attacks that exploit EM side-channels can be classified as either simple EM attacks (SEMA) or differential EM attacks (DEMA) [7]. SEMA uses a single EM trace to determine the internal operation, while DEMA is a higher-order attack that uses multiple traces to boost the fidelity of the captured signal. Agarwal et al. additionally noted that EM side-channels are highly correlated with power side-channels, noting that in every instance that a device under attack (DUA) leaked in the power side-channel, EM side-channel leakage was also observed [8]. However, power side-channel countermeasures do not necessarily reduce the EM side-channel emissions, and in some instances may even increase them.

8.2.3 Optical

That mobile hot carriers, both electrons and holes, in a FET channel may cause visible or infrared light emissions has been known for many years [10]. Recently, projects such as the IBM Picosecond Imaging Circuit Analysis and work at Intel have exploited these phenomena to aid in IC test and debug by collecting the photon emissions across the surface of the die [10–12]. Because emissions are rare, the IC is repeated cycled with the same inputs and an emissions map is formed over

many cycles. This technique requires an array of photomultipliers over the chip to provide a 2D map of photon emissions. Alternately, a CCD camera sensitive to the near infrared range can be used [13]. With these techniques individual gate or device activations can be detected, but require a significant number of repeated input vectors and thus long acquisition times. Measurement of the photon emissions from MOSFET device switching presents a significant challenge given the small number of photons emitted from the devices as well as the large number of densely packed devices on a large die fabricated in a modern process.

Further, the optical properties of silicon can be modulated by altering the voltage or current in the silicon [11,14]. Specifically, the reflectivity changes at p-n junctions have been probed from the chip backside using short light pulses from mode-locked lasers to detect node voltage changes in laser voltage probing (LVP) systems [14]. These techniques are limited to a small number of simultaneous measurements. LVP techniques can be used for direct read-out of internal IC storage node values as well as observation of intermediate calculation node values during operation.

Because of the low probability of capture of emitted photons from switching FETs (some researchers place the probability of capture at less than 0.1%), optical observation techniques rely on operation of the IC in a repeating synchronous loop. Design techniques that introduce random delay variations into the logic can make optical observation more difficult and imprecise, requiring longer observation runs to develop reasonably accurate observations.

Optical side-channels are discussed in more detail in Chap. 7.

8.2.4 Timing and Delay

Timing attacks exploit data-dependent differences in calculation time in cryptographic algorithms. Kocher et al. first discussed these attacks in 1996 [2] and kick-started a significant research effort in industry and academia in side-channel attacks and countermeasures. Timing attacks exploit the time required for a computation to complete. If an operation exhibits a data dependent control flow decision wherein the different branches have different delays, delay information can reveal the state of the control variable. These attacks have generally been applied to CPU-based systems with the computation delay measured at the granularity of CPU clock cycles.

However, in the context of dedicated hardware, the delay of a particular logic block can also be measured and exploited. In synchronous systems, individual block delays are not externally visible at the primary outputs, as functional blocks typically are bounded by clocked storage elements (e.g., flip-flop). However, the delay of the block can still be determined by observing another side-channel (e.g., optical emissions). In asynchronous systems, the delay of a functional block or the entire operation may be observable at the primary output, so care must be taken in the design of the system to avoid leaking timing information. This is further discussed in the asynchronous logic portion of the Countermeasures section.

An alternate method of determining the delay of a functional block in a synchronous system would be manipulation of the clock. With access to the test and post-silicon tuning apparatus of a modern clock distribution tree, an attacker could shrink or lengthen a particular clock cycle of a calculation [15, 16]. The minimum operating clock period would be reflective of the delay of the operation.

8.2.5 Acoustic

Attackers have been using acoustic side-channel attacks against macro-scale systems for decades. In the 1950s, British intelligence agents listened to the resetting of the key wheels of Egyptian encryption machines to deduce their starting positions, which enabled them to crack the encryption [17]. More recently, researchers have used acoustic side channel emissions to determine the text printed by dot matrix printers [18, 19] and which key on a keyboard was struck [20–22]. However, exploitation of acoustic side-channel emissions in microelectronic systems has only seen limited research, notably by Shamir and Tromer [23, 24].

In their work, they placed microphones near a standard home-built PC to see if they could detect when RSA encryption was being run. Not only could they distinguish between other activity and RSA encryption but also they could distinguish between runs of RSA with different keys. They speculated that the acoustic emissions were the result of the piezoelectric properties of ceramic capacitors used on the motherboard for power supply filtering and AC to DC conversion. The acoustic emissions may then be a byproduct of power supply current draw, and thus would be similar to power supply analysis, but potentially in a low-pass filtered form. Thus, countermeasures targeting power analysis could also be effective for countering acoustic side-channel analysis.

One unexplored avenue of research is whether acoustic waves could be used as a side-channel input to a system. In theory, just as the piezoelectric effect can be used to snoop on the power supply activity, a mechanical vibration may be able to induce noise on the power supply through the same channel. Further, any portion of the system that uses mechanical components (e.g., the hard disk drive) may be susceptible to acoustic/mechanical vibrations to induce faults or otherwise affect its behavior.

8.3 Attacks Using Side-Channel Information

Attacks using information gleaned from side channel analysis can be classified as either simple or differential [3]. In a simple side-channel attack, the attacker attempts to directly map the results from a small number of traces [1] of the side-channel to the

[1] A side-channel trace is a set of side-channel measurement samples taken during the cryptographic operation of interest.

operation of the device under attack (DUA). For example, if a control flow branch (either in software or in hardware) activates distinctly different functional blocks with different side-channel leakage, a trace could identify which block was activated and hence the state of the control variable. If the side-channel signal-to-noise ratio (SNR) is sufficiently high, even a single trace may allow a successful attack. Attackers can use simple averaging on multiple traces to filter out noise and boost the SNR. Having only a limited number of traces are generally indicative of not having direct access or possession of the DUA. These attacks rely on a precise understanding of the DUA implementation, the side-channel leakage effects, and a high SNR signal. Simple attacks are generally difficult to mount given these requirements.

Differential side-channel attacks do not require a precise understanding of the implementation or high SNR side-channel traces. Knowledge of the cryptographic algorithm is all that is generally required A differential attack exploits the correlation between the data values being processed and the side-channel leakage. An attacker observes the cryptographic device under operation and records side-channel emission traces. The attacker then constructs a model that estimates side-channel emissions depending on the observable input or output (i.e., plaintext, ciphertext) and the key. Because differential attacks generally require a large number of side-channel traces, they typically imply possession of the DUA or observation over a long period of time. There are several enhancements to differential attacks that improve their effectiveness including template attacks [25–27] and higher-order differential analysis [28, 29]. In these attacks, the side-channel traces must be aligned in time as precisely as possible. Differential attack can use statistical techniques and error correction to overcome some misalignment. These attacks require a triggering signal for trace alignment.

Chapter 11 gives a detailed description of a differential power analysis attack.

8.4 Countermeasures

As the initial discovery of EM side-channel leakage from US encryption machine during WWII, researchers have been developing countermeasures to foil side-channel attacks. These countermeasures fall into a number of basic categories:

- Hiding
- Masking/Blinding
- Design partitioning
- Physical security

8.4.1 Hiding

The goal of an attacker exploiting a side-channel output is to gain sufficient information from the side channel to determine some secret data about the chip

operation. To that end, they are seeking to recover a signal from the side-channel, which typically has a large amount of noise. This process can be viewed as attempting to boost the signal-to-noise ratio (SNR) of the side-channel information as much as possible. Thus, many countermeasures seek to reduce that SNR by either increasing the noise or reducing the signal.

8.4.1.1 Noise Generators

One way to decrease the SNR is to increase the noise. A number of researchers have proposed adding noise generators to secure ICs in order to foil side-channel attacks. For power analysis attacks, designers have proposed adding circuits that draw random amounts of power during chip operation or that keep the total power dissipation constant by filling in power dissipation if it is lower than a set amount [30]. For timing attacks, researchers have proposed adding circuits that have a random delay into the logic path [2, 31]. However, adding this type of circuit may pose a problem for synchronous systems, which must complete all logic paths by the end of the clock period. Randomized delays will also make any side-channel observation that relies on subsampling (e.g., optical emission) or trace alignment more difficult. For EM attacks, EM noise can be added to reduce the SNR. The difficulty is that the emission spectrum is quite broad required a large amount of added noise across a wide frequency range [8]. Such an EM noisy chip poses a problem for system integrators whose designs must pass governmental tests for electronic interference. Further, this would require a significant amount of power, if not die area, for the noise generators. Finally, unless the noise generators are carefully placed, they may not add noise to all of the unintentional EM emissions due to signal coupling. While adding noise to the side-channel can be sufficient to make simple side-channel attacks infeasible, differential attacks will still be possible, but require more traces or more advanced signal processing.

8.4.1.2 Balanced Logic Styles

Another way to decrease the SNR is to make the logic gates' side-channel emissions independent of the data being processed. The basis of many side-channel attacks is the data-dependent difference in the side-channel emissions (e.g., power, EM, timing). If the logic emissions were independent of the processed data then the side-channel could no longer be exploited. There have been a large number of proposed logic families that have low side-channel leakage in a specific side-channel, particularly power. However, care must be taken in the design of hardened logic styles, as reducing emissions in one side-channel may increase emissions in another. A number of surveys of these logic families have been published [32, 33].

Many hardened logic styles use a form of dual-rail precharged (DRP) logic, wherein the gate has two outputs (*out* and *out_b*) and operated in two phases (*reset* and *evaluate*). The gate nodes are first reset to known values, then in the

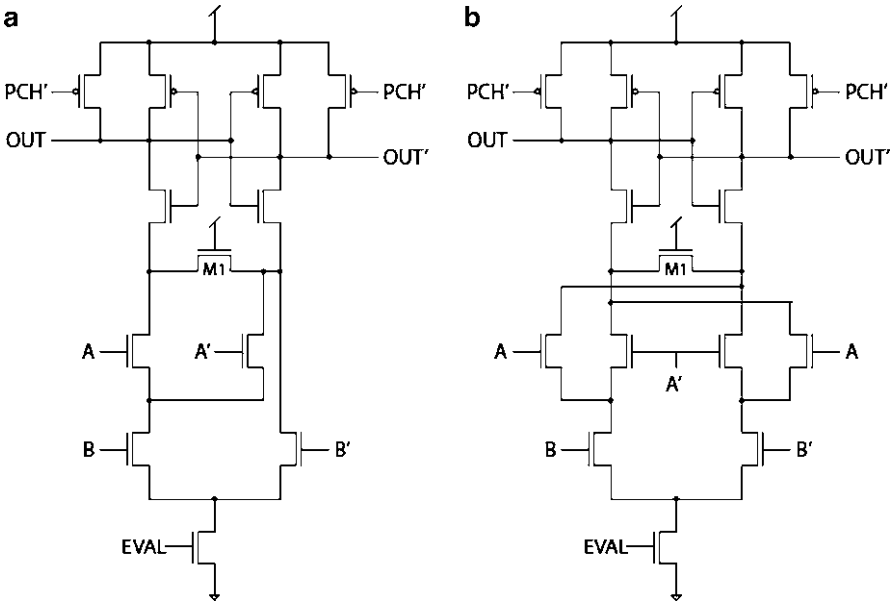


Fig. 8.4 Example dual-rail precharged logic gates: Sense Amplifier Based Logic (SABL) [36]

evaluate phase either *out* or *out_b* transitions, but not both. The hardened gates are constructed such that the transition of *out* or *out_b* ideally have the same side-channel emissions. For example, a gate hardened against power analysis would be constructed such that the *out* and *out_b* nodes had the same capacitance and thus would dissipate the same amount of dynamic power on a switching event. Especially in a standard cell place and route synthesis flow, the matching of *out* and *out_b* is difficult and requires specialized CAD tools [34, 35]. Mismatches in the discharge of internal gate capacitances can also lead to side-channel emissions [32]. Examples of this type of logic family include WDDL [36], SABL [37], and dual-spacer logic [38]. Figure 8.4 shows example NAND and XNOR gates in SABL.

To alleviate the need to balance the outputs, researchers developed three phase logic families such as TDPL [39] and TSPL [40]. They operate in three phases (*precharge*, *evaluate*, *discharge*) and do not require balancing of the output loads. The first two phases operate the same as DRP, but in the third phase all output nodes are discharged to ground. This ensures that all outputs see a charging and discharging event every cycle, independent of the input data, and thus the power dissipation is the same for every cycle. Figure 8.5 shows example NAND and XNOR gates in TDPL. Three-phase operation has also been adapted to operate at the block level (Fig. 8.6) allowing designers to use standard logic styles in the core functional block [41], but the core functional block would still be vulnerable to EM attack.

These logic families also require three phase nonoverlapping clocks, which is not a standard clocking style, but there are techniques to generate such clocks

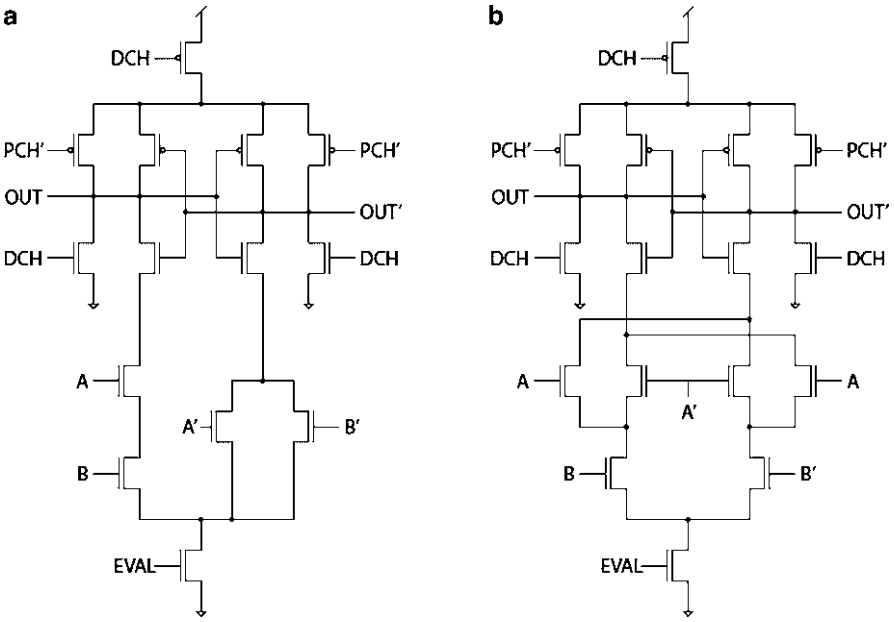


Fig. 8.5 Example three phase logic family: Three phase Dual-rail Precharged Logic (TDPL) [41]

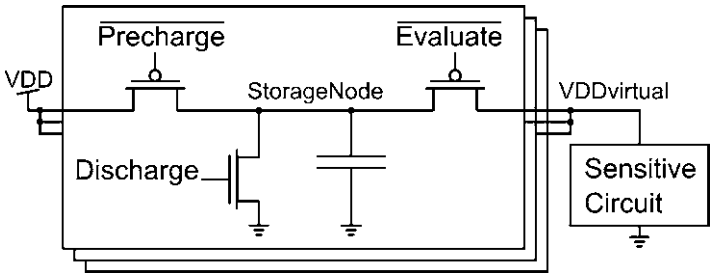


Fig. 8.6 Three phase power supply filter scheme using switched capacitors [44]. Note that this scheme allows the use of any standard logic family in the functional block

[42]. However, three phase logic styles do rely on the attacker being unable to distinguish between phases of operation. For systems that run at high frequencies (i.e., in the GHz range) and that have power supply stabilization apparatus (e.g., decoupling capacitance, voltage regulators), distinguishing between the phases can be challenging. However, this implies that designs that employ these types of logic families must have safeguards against an attacker down-clocking the system.

An additional issue with secure logic families is the early propagation effect (EPE) leading to information leakage via the output switching timing [43, 44]. Depending on the logic function and input values, a logic gate may switch its output to its final value before all of the inputs have arrived. For example, an AND gate

with input starting state $(in_a, in_b) = (1, 1)$ and input ending state $(in_a, in_b) = (0, 0)$, will immediately switch its output from 1 to 0 once either input falls to 0 irrespective of when the other input falls. Thus the gate will exhibit data dependent switching time, which may be observed using any number of side-channels (e.g., power, EM, optical) and used in a timing attack. Use of arbiters to prevent the gate from firing until all inputs have arrived has been proposed as a fix for this issue [45]. Alternately, all gates can be fired on a fixed delay step guaranteed to be larger than the longest gate delay in the previous stage [40].

A final issue of note is that the VLSI overheads in area, power, and delay of secure logic families over conventional logic styles is significant, often 3x or even more [32, 33]. Further, hardened logic families often require additional control signals and have additional timing constraints not found in conventional logic. Thus, these logic families will likely not find main-stream use in all part of a large SoC or CPU design, but rather will only be used in portions of the design that require secure operation.

8.4.1.3 Asynchronous Logic

Asynchronous gates have also been proposed for use in secure systems due to their power analysis resistance (most asynchronous logic families use dual rail or 1-of-n signal encoding), lower EM emissions due to lack of clock synchronization, resistance to fault injection, and ability to encode an alarm state into the data [31, 46–48]. One often cited general advantage of asynchronous logic is that it finishes a calculation as quickly as possible based on the input data. From a side-channel analysis standpoint this is a disadvantage, because it reveals information about the input data based on the calculation time. With careful micro-architecture design, insertion of arbiters (e.g., Muller C-elements), or insertion of random delays, the circuit delay for variable input data can be equalized [31].

8.4.1.4 Low Power Design

A method to reduce the side-channel emissions is to dampen the overall emissions from the chip thus making them harder to sense [7]. A simple way that this can be accomplished is by reducing the overall power dissipation of the chip, thus reducing the power put into the side-channels. Unfortunately, sensing technology can still recover side-channel emissions from chips using conventional low power techniques [49]. More extreme low power techniques such as subthreshold operation or adiabatic computing can reduce the power dissipation of chips by orders-of-magnitude, but come at the price of radically reduced performance, potentially below the acceptable threshold for most applications. Further, operation in sub-threshold may exacerbate side-channel leakage by magnifying transistor mismatch due to process variability. This in turn may increase side-channel leakage in the power, EM, or timing.

8.4.1.5 Shielding

Another method of reducing the overall emissions from the chip is physical shielding or filtering of the side-channel leakage. For power emissions, this could entail adding additional decoupling capacitors the power supply rails or on-die voltage regulators for active filtering [50]. These are currently used to step-down the high external voltages (e.g., 1.8 V) to the lower core voltages needed for 45 nm and smaller process geometries (e.g., 1.0 V).

Various methods of reducing the EM emissions via shielding have been suggested [7], including use of the existing upper-level metal layers as a shield. Unfortunately, large SOCs and CPUs typically use the upper-level metal layers for distribution of clock and power supplies, the very signals that indirect EM side-channels tend to exploit. Despite having upward of ten metal layers in a modern semiconductor fabrication process meant for logic, dedication of one or more of those layers for EM shielding is somewhat expensive. Additionally, this does not shield the substrate side of the chip. Additionally, any metal layer used as a shield would have to be somewhat porous to allow for I/O and power supply signals access to the area bond pads used in the majority of large modern chips. It is notable that the DEC Alpha 21264 microprocessor did dedicate two metal layers for Vdd and Gnd planes for lower resistance power supply distribution [51]. While this was not meant specifically as an EM shield, this type of structure has been manufactured previously. The chip could be surrounded by an external Faraday cage for shielding, but again this would have to be somewhat porous for I/O and power supply connections. Further, this type of shielding would be difficult to implement in some physical form factors (e.g., Smartcards).

Acoustic shielding could be accomplished using sound dampening materials in the system casing [23]. For optical side-channels, the normal chip metal layers shield the top-side of the chip from optical probing, but the back (substrate) side may be exposed. While various packaging options may be feasible to hinder backside optical probing, the effects of those countermeasures on heat dissipation during operation (as the backside is the side of the die that heatsinks are attached to) and testability as a number of optical testing technologies rely on backside optical probing. Backside probing would require thinning of the die for better SNR, but there are a number of existing techniques that can accomplish this [52].

8.4.2 Masking/Blinding

Masking or blinding is a countermeasure that seeks to remove the correlation between the input data and the side-channel emissions from intermediate nodes in the functional block [53–56]. This can be accomplished on a per-gate basis or on a per-block basis. Per-gate schemes use specialized masking logic families (e.g., RSL [57] or MDPL [58]) that use a globally distributed random masking bit to conditionally invert the gate outputs (i.e., mask the output by XORing it with the mask bit).

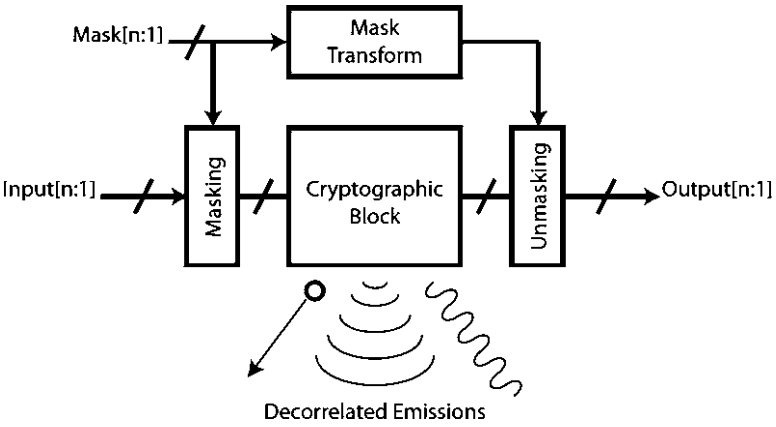


Fig. 8.7 Per-word masking scheme that decorrelates intermediate values in the cryptographic block from input data. Side channel emissions cannot reveal information about original data

Per-word schemes (Fig. 8.7) randomize the input data with a random bit vector before entering the cryptographic block. The cryptographic block operates on the randomized data, and even if side-channel information is leaked, that information is not correlated with the original input data. The data undergoes the opposite transform upon exiting the cryptographic block to recover the output data. An attacker may be able to determine the intermediate values used in the cryptographic block, but as long as the mask is kept secret, no information about the original data can be attained. These operations are relatively simple for linear functions, but are still possible, albeit more challenging, for nonlinear functions such as the AES S-Box [59]. Masking countermeasures have shown to be vulnerable to a number of attacks including glitch exploitation, template attacks, and higher-order differential attacks [60–64].

8.4.3 Design Partitioning

Some side channels (e.g., unintentional EM emissions) leak information due to the coupling of secret signals onto other nodes. To reduce these emissions, designers can separate regions of the chip that operate on plaintext from regions that operate on ciphertext [1, 7]. In NSA TEMPEST parlance, this is the classic RED (plaintext) separation from BLACK (ciphertext). In addition to physical separation of the portions of the chip, this separation could also include separation of any typically shared infrastructure. This would include power supply infrastructure (distribution grids, regulators, pads/pins), separate clocking infrastructure (phase-locked loops, distribution network, pads/pins, board crystals), and separate testing infrastructure (flip-flop scan chain, built-in self test). This type of separation is not atypical in some

mix-signal (analog and digital) ICs that have extremely sensitive analog circuits that do not want noise from the digital portion of the chip on the power supply rails or clocking signals. This noise has strong distinct spectral components at the clock frequency and its harmonics.

A more radical separation of RED and BLACK zones can be achieved using 3D die stacking. The two zones can be fabricated on entirely separate die, then bonded together to form a die stack. There are a number of commercially available technologies and techniques for 3D chip stacking. The separate die can communicate using a number of technologies including through-silicon vias (TSV) that offer high density, low latency connections [65]. Using 3D stacking to implement secure systems has also been suggested as a way to solve manufacturing assurance and supply chain vulnerabilities by manufacturing a portion of the design in a untrusted foundry and a portion in a trusted foundry, then combining the two die using 3D stacking.

8.4.4 Physical Security and Anti-Tamper

For the highest SNR side-channel sensing, the attacker typically needs to have prolonged physical access to the DUA and may need to apply invasive techniques (e.g., decapsulate, alter the board or chip). Thus denial of proximity, access, and possession are key to reducing the attacker's ability to mount side-channel attacks [66].

For example, since the highest power EM emissions must be sensed close to the chip, physical security of the zone around the chip can hinder EM side-channel sensing. In recently released TEMPEST documents, US government regulations required a zone of 200 feet around cryptocenters [1]. The required secured zone size was dictated more by what the NSA estimated could be practically secured rather than on technical feasibility of EM emissions sensing. Relatively low frequency signals (e.g., 1 MHz) have wavelengths and hence near-field zones that extend beyond 200 feet. Further, anti-tamper and techniques to prevent/detect decapsulation can hinder an attacker from exploiting the EM side-channel.

In the previous work on acoustic side channels, researchers noted a need to place the microphone in close proximity to the DUA. This requirement could be seen as a limitation of this type of attack, but there are a number of long range microphone technologies using reflected light (e.g., infrared beam or laser) to detect sound vibrations [67, 68] that would obviate the need for physical proximity of even direct physical access to the DUA. Thus, acoustic shielding of the environment around the DUA would also be necessary.

8.5 Conclusions

Attacks on cryptographic systems are more and more using a combination of side-channels emissions, rather than just a single type, as well as fault injection to discern

secret information. Similarly, countermeasure development must take a holistic approach to building secure systems, rather than focus on a particular attack. If a countermeasure reduces side-channel emission of one type, but increases them in another, the overall security of the system has not necessarily been improved. Additionally, countermeasures need to be evaluated not only on their reduction in side-channel emissions but also on the basic VLSI metrics of delay, area, and power. System designers must make trade-offs between the added security afforded by a countermeasures against their cost in delay, area, and power.

Finally, passive side-channel attacks are and will remain a powerful tool in the attacker's arsenal. While hardware countermeasures can significantly reduce the side-channel emissions and increase the number of traces necessary for an attacker to extract useful data, the emissions can never be reduced to absolute zero. Thus, algorithmic countermeasures that ensure that the overall system remains secure even if side-channel leakage occurs will be critical.

References

1. Friedman J (1972) TEMPEST: a signal problem, NSA Cryptologic Spectrum. http://www.nsa.gov/public/pdf/tempest.pdf. Accessed Summer 1972
2. Kocher P (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Advances in Cryptology, CRYPTO'96, LNCS, vol 1109. Springer-Verlag, Berlin, Heidelberg, New York
3. Kocher P, Jaffe J, Jun B (1999) Differential power analysis. In: 19th Annual International Cryptology Conference (CRYPTO), vol 2139. Springer-Verlag, Berlin, Heidelberg, New York, August 1999
4. Xiaoxiao W, Hassan Salmani S, Tehranipoor M, Plusquellic J (2008) Hardware Trojan detection and isolation using current integration and localized current analysis. In: International Symposium on Defect and Fault Tolerance in VLSI Systems, October 2008, pp 87–95
5. Weaver J, Horowitz M (2007) Measurement of supply pin current distributions in integrated circuit packages. IEEE Electrical Performance of Electronic Packaging, October 2007
6. Weaver J, Horowitz M Measurement of via currents in printed circuit boards using inductive loops. IEEE Electrical Performance of Electronic Packaging, October 2006
7. Quisquater J Samyde D ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Smart Card Programming and Security (E-smart), September 2001
8. Agrawal D, Archambeault B, Rao J, Rohatgi P (2002) The EM side-channel(s). Cryptographic Hardware and Embedded Systems (CHES), 2002
9. Carluccio D, Lemke K, Paar C (2005) Electromagnetic side channel analysis of a contactless smart card: first results. Workshop on RFID and Light-Weight Crypto, July 2005
10. Tsang JC, Kash JA (1997) Picosecond hot electron light emission from submicron complementary metal-oxide-semiconductor circuits. Appl Phys Lett 70: 889–891
11. Tsang J, Kash J, Vallett D (2000) Time-resolved optical characterization of electrical activity in integrated circuits. Proceedings of the IEEE, September 2000
12. Rusu S, Seidel S, Woods G, Grannes D, Muljono H, Rowlette J, Petrosky K (2001) Backside infrared probing for static voltage drop and dynamic timing measurements. In: Proceedings of ISSCC, 2001, pp 276–277
13. Skorobogatov S (2009) Using optical emission analysis for estimating contribution to power analysis. In: 6th Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009), IEEE-CS Press, Switzerland, pp. 111–119

14. Eiles TM, Woods DL, Rao VR (2000) Optical probing of flip-chip-packaged microprocessors. In: 2000 IEEE International Solid-State Circuits Conference on Digestive Technology Papers, San Francisco, CA, 2000, pp 220–221
15. Tam S et al (2000) Clock generation and distribution for the first IA-64 microprocessor. IEEE J Solid-State Circuits 35(11): 1545–1552
16. Geannopoulos G, Dai X (1998) An adaptive digital deskewing circuit for clock distribution networks. ISSCC 1998, pp 25.3–1–25.3–2
17. Wright P (1987)Spy catcher: the candid autobiography of a senior intelligence officer. Viking Adult, 1987, p 82
18. Backes M, Dürmuth M, Gerling S, Pinkal M, Sporleder C (2010) Acoustic side-channel attacks on printers. In: Proceedings of the 19th USENIX Security Symposium, August 2010
19. Briol R (1991) Emanation: how to keep your data confidential. In: Proceedings of the Symposium on Electromagnetic Security for Information Protection, 1991
20. Asonov D, Agrawal R (2004) Keyboard acoustic emanations. In: IEEE Symposium on Security and Privacy, 2004, pp 3–11
21. Zhuang L, Zhou F, Tygar JD (2005) Keyboard acoustic emanations revisited. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, 2005
22. Berger Y, Wool A, Yeredor A (2006) Dictionary attacks using keyboard acoustic emanations. In: Proceedings of the 13th ACM Conference on Computer and Communication Security (CCS 2006). ACM, New York, pp 245–254
23. Shamir A, Tramer E (2004) Acoustic cryptanalysis: on nosy people and noisy machines. Eurocrypt 2004 rump session, 2004
24. http://www.wisdom.weizmann.ac.il/~tromer/acoustic
25. Chari S, Rao J, Rohatgi P (2002) Template attacks. Proceedings of CHES 2002, LNCS, vol 2523, 2002, pp 13–28
26. Archambeau C, Peeters E, Standaert F-X, Quisquater J-J (2006) Template attacks in principal subspaces. In: CHES, 2006, pp 1–14
27. Rechberger C, Oswald E (2004) Practical template attacks. In: WISA, 2004 pp 440–456
28. Messerges T (2000) Using second-order power analysis to attack DPA resistant software. In: CHES 2000, LNCS 1965, 2000, pp 238–251
29. Waddle J, Wagner D (2004) Towards efficient second-order power analysis. In: CHES 2004, LNCS 3156. Springer-Verlag, Berlin, Heidelberg, New York, pp 1–15
30. Daemen J, Rijmen V (1999) Resistance against implementation attacks: a comparative study of the ales proposals. In: The Second AES Candidate Conference. National Institute of Standards and Technology, Gaithersburg, MD, pp 122–132
31. Moore S, Anderson R, Cunningham P, Mullins R, Taylor G (2002) Improving smart card security using self-time circuits. In: Proceeding of Eighth International Symposium on Asynchronous Circuits and System. IEEE Computer Society, Silver Spring, MD, pp 211–218
32. Menendez E, Mai K (2010) A comparison of power-analysis-resistant digital circuits. In: IEEE International Symposium on Hardware-Oriented Security and Trust
33. Mangard S, Oswald E, Popp T (2006) Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Berlin, Heidelberg, New York, iSBN 0-387- 30857-1, http://www.dpabook.org/
34. Hwang D, Tiri K, Hodjat A, Lai B-C, Yang S, Schaumont P, Verbauwhede I (2006) AES-based security coprocessor IC in 0.18-μm CMOS with resistance to differential power analysis side-channel attacks. IEEE J Solid-State Circuits 44(4): 781–792
35. Tiri K, Verbauwhede I (2004) Place and route for secure standard cell design. In: Proceedings of the 6th USENIX Smart Card Conference and Advanced Application Conference (CARDIS 2004), 2004
36. Tiri K, Akmal M, Verbauwhede I (2002) A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In: Proceedings of the 28th European Solid-State Circuits Conference ESSCIRC 2002, September 2002

37. Tiri K, Verbauwhede I (2004) Charge recycling sense amplifier based logic: securing low power security ICs against DPA [differential power analysis]. In: Proceeding of the 30th European Solid-State Circuits Conference, 2004
38. Sokolov D, Murphy J, Bystrov A, Yakovlev A (2004) Improving the security of dual-rail circuits. In: Proceedings of the Workshop Cryptographic Hardware Embeded System (CHES), vol 3156, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, pp 282–297
39. Bucci M, Giancane L, Luzzi R, Trifiletti A (2006) Three-phase dual-rail pre-charge logic. In: Proceedings of the Workshop Cryptographic Hardware Embeded System (CHES), vol 4249, Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, pp 232–241
40. Menendez E, Mai K (2008) A high-performance, low-overhead, power-analysis-resistant, single-rail logic style. In: Proceedings of the IEEE International Workshop Hardware-Oriented Security Trust (HOST), 2008, pp 33–36
41. Tokunaga C, Blaauw D (2010) Securing encryption systems with a switched-capacitor current equalizer. IEEE J Solid-State Circuits 45(1): 23–31
42. Glasser LA, Dobberphul DW (1985) The Design and Analysis of VLSI Circuits. Addison-Wesley, Reading, MA
43. Kulikowski KJ, Karpovsky MG, Taubin A (2006) Power attacks on secure hardware based on early propagation of data. In: 12th IEEE International On- Line Testing Symposium (IOLTS 2006), pp. 131–138. IEEE Computer Society, Silver Spring, MD, 10–12 July 2006
44. Suzuki D, Saeki M (2006) Security evaluation of DPA countermeasures using dual-rail pre-charge logic style. In: Goubin L, Matsui M (ed) Crypto- graphic Hardware and Embedded Systems – CHES 2006, 8th International Work-shop, Yokohama, Japan. Proceedings, volume 4249 of Lecture Notes in Computer Science, pp. 255–269. Springer, Berlin, Heidelberg, New York, 10–13 October 2006
45. Guilley S, Hoogvorst P, Mathieu Y, Pacalet R, Provost J (2004) CMOS structures suitable for secure hardware. In: Proceedings of Design, Automation and Test in Europe Conference and Exposition (DATE), pp 1414–1415, February 2004
46. Fournier J, Li H, Moore SW, Mullins RD, Taylor GS (2003) Security evaluation of asynchronous circuits. Workshop on Cryptographic Hardware and Embedded Systems (CHES). Published by Springer in LNCS 2779, New York, September 2003
47. Kulikowski KJ, Su M, Smirnov A, Karpovsky MG, MacDonald DJ (2005) Delay Insensitive Encoding and Power Analysis: A Balance Act. In: ASYNC, 2005
48. Yu Z, Furber S, Plana L (2003) An investigation into the security of self-timed circuits. In: ASYNC, 2003
49. Rabaey JM (2009) Low Power Design Essentials, Series on Integrated Circuits and Systems. Springer, New York
50. Weste N, Harris D (2010) CMOS VLSI Design: A Circuits and Systems Perspective, 4th edn. Addison Wesley, Reading, MA, USA
51. Gronowski P et al. (1998) High performance microprocessor design. IEEE J Solid-State Circuits 33(5): 676–686
52. Perdu P, Desplats R, Beaudoin F (2000) A review of sample backside preparation techniques for VLSI. Microelectron Reliabil 40: 1431–1436
53. Chari S, Jutla CS, Rao JR, Rohatgi P (1999) Towards sound approaches to counteract power-analysis attacks. In: Proceedings of Advances in Cryptology (CRYPTO 1999). Springer, Berlin, Heidelberg, New York, pp. 398–412
54. Goubin L, Patarin J (1999) DES and differential power analysis – the "duplication" method. Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems – (CHES 1999). Springer, Berlin, Heidelberg, New York, pp 158–172
55. Akkar M, Giraud C (2001) An implementation of DES and AES, secure against some attacks. In: Proceedings 2001 Workshop on Cryptographic Hardware and Embedded Systems, (CHES 2001), LNCS 2162. Springer, Berlin, Heidelberg, New York, pp 309–318

56. Golic JD, Tymen C (2002) Multiplicative masking and power analysis of AES. In: Proceedings of the 2002 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), LNCS 2523. Springer, Berlin, Heidelberg, New York, pp 198–212
57. Suzuki D, Saeki M, Ichikawa T (2007) Random switching logic: a new countermeasure against DPA and second-order DPA at the logic level. IEICE Trans. Fundament E90-A(1): 160–168
58. Popp T, Mangard S (2005) Masked dual-rail pre-charge logic: DPA resistance without the routing constraints. In: Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), LNCS 3659, pp 172–186, August 2005
59. Oswald E, Mangard S, Pramstaller N, Rijmen V (2005) A side-channel analysis resistant description of the AES S-Box. Fast Software Encryption 2005, LNCS 3557, 2005, pp 413–423
60. Schaumont P, Tiri K (2007) Masking and dual-rail logic don't add up. In: Proceedings of the 2007 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS 4727, 2007, pp 95–106
61. Tiri K, Schaumont P (2007) Changing the odds against masked logic. Selected Areas in Cryptography 2007, LNCS 4356, 2007, pp 134–146
62. Waddle J, Wagner D (2004) Towards efficient second-order power analysis. In: Proceedings of 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), LNCS 3156, 2004, pp 1–15
63. Chen Z, Schaumont P (2008) Slicing up a perfect hardware masking scheme. In: IEEE International Workshop on Hardware-Oriented Security and Trust, June 2008
64. Chen Z, Schaumont P (2009) Side-channel leakage in masked circuits caused by higher-order circuit effects. In: 3th International Conference on Information Security and Assurance (ISA 2009), June 2009
65. Beyneandetal E (2008) Through-SiliconVia and die stacking technologies for microsystems-integration. In: Proceedings of IEEE International Electron Devices Meeting, 2008
66. Anderson R, Kuhn M (1996) Tamper resistance a cautionary note, In: 2nd USENIX Workshop on Electronic Commerce, 1996
67. Galeyev B (1996) Translated by Vladimir Chudnovsky (1996). Special Section: Leon Theremin, Pioneer of Electronic Art. Leonardo Music Journal (LMJ) 6
68. Glinsky A (2000) Theremin: Ether Music and Espionage. University of Illinois Press, Urbana, Illinois

Chapter 9
Trusted Design in FPGAs

Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak

9.1 Introduction

The last decade and in particular the last year were important for FPGAs and even more for FPGA security. For example, for the first time after a decade of no increase, the FPGA revenues grew by more than one third to surpass the $4 B level. Maybe even more importantly, the number of new designs based on FPGA was 110,000. The colossal size of this number can be best seen from the fact that only 2,500 ASIC designs were initiated. At the same time, FPGA has been recognized as an exceptionally efficient platform due to its flexibility compared to ASICs, and due to its efficiency compared to implementations based on the general purpose microprocessors.

The FPGA security scope is very broad and ranges from technological and architectural issues to applications, from FPGA vulnerability to new types of security primitives and protocols, from relative limitations of FPGA-based systems in terms of security to their strategic and quantitative advantages, and from digital right management (DRM) issues to trusted remote execution. Our objective is to cover various key aspects of this broad space.

Recently, several relevant FPGA security surveys have been published, including [1]. We believe that our survey is complementary to the available summaries in the field while it is unique both in terms of the scope as well as the depth of coverage of key issues. In addition, we have a strong emphasis on hardware-based security.

M. Majzoobi (✉) · F. Koushanfar
Electrical and Computer Engineering Department, Rice University, 6100 Main, MS380, Houston, TX 77005, USA
e-mail: mehrdad.majzoobi@rice.edu; farinaz@rice.edu

M. Potkonjak
Computer Science Department, University of California Los Angeles, Los Angeles, CA 90095-1596, USA
e-mail: miodrag@cs.ucla.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_9,

The remainder of the chapter is organized as follows. The next section outlines the steps of the reconfigurable platform's synthesis flow and its vulnerabilities. Section 9.3 discusses the implementation of hardware cryptographic modules on FPGAs and addresses the relevant attacks. The security primitives that can be used as universal mechanisms for many different protection protocols are discussed in Sect. 9.4. Important primitives such as physical unclonable functions and true random number generation (for the reconfigurable platform) are presented in this section. In Sect. 9.5 we outline the most challenging directions in the field and early results along those directions. The chapter is concluded in Sect. 9.6.

9.2 FPGA Synthesis Flow and Its Vulnerabilities

Efficient design and field usage of FPGAs is enabled by sophisticated computer-aided design (CAD) tools that have matured over the years. To make their devices accessible and easy to use, the FPGA vendors and third-party providers contribute a full set of programs and tools that allow automatic synthesis and compilation from a high level hardware description language such as Verilog or VHDL to a string of bits, commonly termed a *bitstream*.

The FPGA Synthesis flow is shown in Fig. 9.1. The input to the synthesis flow is the hardware specification, design constraints, and sometimes some FPGA-specific commands. The set of inputs is symbolically denoted by hardware Description Language (HDL) on the flow figure but it contains the aforementioned knowledge

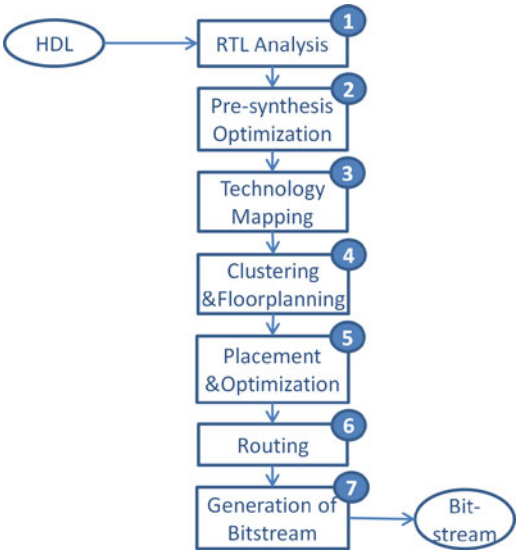


Fig. 9.1 The FPGA synthesis flow

of design constraints and specifications. The design constraints include the timing bounds between the input and the output pads, between the inputs and the registers, and between the registers and the outputs. The designer may also specify additional details such as mutli-cycle paths. Another set of possible constraints are location-dependent where a designer may limit the implementation of a specific part of the design to a target part of the device to achieve a certain design objective or for an optimization reason. The FPGA specific commands specify the selection of the underlying FPGA device that also impacts the design metrics including the timing, power, and cost. Even the same manufacturer often offers a range of devices with differing characteristics that has to be carefully selected for the application at hand.

Although there has been a trend toward using higher level abstraction models such as SystemC and behavioral synthesis tools, they are yet to be widely adopted. The legacy IPs and contemporary designs that are used across a spectrum of applications in industrial, commercial, and defense sectors are predominantly designed at the RTL level. A relatively small number of designs are developed using higher level behavioral languages including but not limited to general purpose languages such as C or SystemC, or domain-specific languages such as Matlab or Simulink. The behavioral-level specifications are not cycle accurate and generally a high level synthesis tool is used for converting the description to HDL level.

Consider the steps of the design flow as shown in Fig. 9.1 after the HDL input, design constraints, and the specifications are provided. First, a set of analysis at the register-transfer level (RTL) takes place where the control, memory, and the data path elements are considered. Second, a set of presynthesis optimization separately treats each of the identified elements. For example, the datapath optimizations, the control path optimizations including the FSM optimization and retiming, and combinational logic optimizations. Third, the design passes through technology mapping and more detailed optimizations. The control logic is mapped to the basic logic elements. The datapath logic is mapped mostly to dedicated on-chip modules including the multipliers, adders with dedicated carry chains, and embedded memory.

Forth, the location of each element in the floorplan of the mapped netlist is determined. The basic logic elements maybe clustered into logic blocks before the floorplanning. Fifth, the placement is originally done according to the floorplan which is subject to a number of optimization steps. The optimizations are incrementally done post interconnect placement where a better timing profile becomes available. The optimizations at this stage include rewiring, restructuring, and duplication after which typically another round of incremental placement takes place. Sixth, the routing is performed where the signal paths are connected using the available on-chip programmable routing structure. Lastly, the results of mapping, placements, and routing are encoded into a bitstream that can be used to configure the logic and wires to implement the target design. A comprehensive review of the FPGA design automation can be found in [2].

9.2.1 Vulnerabilities

There is a number of possible attacks that can be envisioned on the design flow and the design artifact described earlier in the section. We now briefly mention the plausible adversarial acts and the common generic countermeasures taken against the vulnerabilities. Note that the emphasis of this section is on the attacks that are specific to FPGAs; there is a number of vulnerabilities that apply to most implementations of cryptographic functions, such as system-level attacks on the protocols. In the interest of brevity and conciseness, we focus our discussions to the FPGA domain. Before we delve into discussion, we make a distinction among three types of IPs: soft, firm, and hard IPs. According to the standard definitions since an IP in a hardware description language is still in a program format, is considered to be a "soft IP." The phrase "firm IP" is used to refer to an IP that is still presynthesis but has a fixed RTL level description. A "Hard IP" refers to the IP that is in form of a full layout post synthesis, placement, and routing, that is ready to be implemented (a bitstream in case of an FPGA) [3].

9.2.1.1 HDL-Level IP Theft and Tampering

Attacks at the HDL level include stealing the IP, inserting malware in an IP to disrupt its operation, or inserting malware/spyware to extract information and data out of the IP.

The common methods for addressing the attacks against stealing of the soft IP cores include watermarking of the soft IP, license agreement, and encryption of the cores that are transferred between parties. As the soft IP by itself is just a datafile, any other method that is applied to transferring and storage of data files can be used for protecting the transfer and safeguarding of this kind of information. The Trojans/spyware inserted at the HDL level code are either trivial or very hard to detect, based on the availability of designer's information and trust in the designer. It is worth noting that the designer inserted Trojans are very hard to detect in very complex codes, and even the best verification tools may not be able to detect the additional states and functions added by a designer [4,5]. Often times, the designer does not provide the full specification of the IPs, and therefore, there may not be a basis for comparing the soft IP at hand to a Trojan-free (golden) model. If the designer is trusted, standard cryptographic protocols for integrity checking (e.g., digital signatures) can be applied for ensuring that the original designer's code is not modified. In the final section, we discuss the recent efforts for creation of provably trusted IP.

If the user of an HDL code acquires the program from a certified vendor that has the certificates and can show integrity proofs, there is no need to worry about the HDL-level Trojans. Unfortunately, such proofs and certificates are not always available for third-party IP and reuse scenarios. Therefore, the soft IP trust is a standing complex problem that is not necessarily specific to FPGA; it is a universal problem that threatens almost all soft IP cores that are not from trusted sources or certified vendors.

Aside from classic encryption [6, 7], another set of methods for thwarting the soft IP theft and piracy attacks is based on watermarking [3]. Watermarking hides a hard to forge or remove digital signature in the IP, such that the owner of the datafile can be later recognized based on his/her signature [8]. Methods applied to watermarking during pre- or during synthesis can be directly integrated within the FPGA synthesis tools. Generally speaking, a watermark may be applied at the HDL level, at the netlist level, or at the bitstream level. Depending on the insertion point of the watermark, it can provide a proof of ownership for the legitimate author. For example, an HDL level watermark may be inserted by the core designer, while a bitstream level watermark is likely to be embedded by the tool vendor who is able to easily integrate the watermark within the synthesis flow.

The work in [9] provided the first known methods for FPGA bitstream watermarking and fingerprinting. Fingerprint is a mark that not only identifies the design owner, but also is able to identify the instance of the design. In the FPGA case it can identify the specific device where the design is embedded. Note that the watermark and fingerprint have to satisfy a number of properties including difficulty of forging, difficulty of tampering or removal, uniqueness of the signature sequence and ease of evaluation. A detailed discussion of hardware IP and FPGA core watermarking and fingerprinting is outside the scope of this chapter. We refer the interested readers to excellent comprehensive sources on the topic [1, 3, 10] and Chap. 9 of this book.

9.2.1.2 Synthesis-Level IP Theft and Tampering

By synthesis level IP theft we mean all the stages between the RTL level descriptions to routing (Steps 1–7 in Fig. 9.1). Both firm and hard IPs may also be a subject of piracy and malware insertion attacks. A suit of methods based on watermarking can provide ownership proof, but would not be enough to actively deter from piracy. A class of methods that is intended to functionally deter firm IP theft is called active hardware metering [11–13]. Active hardware metering integrated the behavioral description of a design with the unclonable device-specific signatures such that the IP is only tied to one IC. Transferring the IP to another IC would render the device nonfunctional. For a comprehensive discussion on metering, we refer the interested readers to Chap. 8 of this book.

Another set of IP protection methods based on the use of PUFs attempt at using the inherent and unclonable physical disorders of the device for generating a secret key based on the unclonable device variations. Thorough discussion of IP control based on the PUF signatures is provided in Chap. 7 of this book. A number of defense studies and industrial reports have expressed concerns about the possibility of insertion of hardware malware during the design. Following the suggestion by a Defense Science Board Report [14] and the followup proposal solicitations by DARPA [15], the common trust model in the field became trusted designer (system integrator), untrusted optimization and synthesis tools, untrusted third party cores, and untrusted components-off-the-shelf. The common assumption is that the

devices can be trustfully tested for functionality and to ensure they carry on the intended computations, and it can be tested for Trojan detection. A full discussion of Trojan models, detection, and isolation is provided in Chaps. 15–17 of this book.

9.2.1.3 Bitstream-Level Theft and Tampering

The circuit configuration data is encoded into the bitstream. In the widely used SRAM FPGA technology, because of the underlying volatile memory, at each power up incident the device should read and load the bitstream from an external nonvolatile source, typically a Flash device or an EEPROM [6]. The uploaded bitstream typically goes under the functional and parametric tests before being shipped to the users. From this point on, the only active interaction between the provider and the user is via occasional updates by field reconfiguration that can be remotely performed [16]. The common threat model in this area is to assume that the user maybe untrusted [15].

The conventional bitstream uploading methods are independent of the FPGA device, as long as the device is from a certain family and of the same size. Therefore, an adversary could launch an attack targeted at tapping the bitstream during the upload phase and later cloning the stream on other FPGAs. Cloning has been shown to be practically feasible and inexpensive to do for skillful engineers with conventional devices such as probes and logic analyzers. Not only cloning and overbuilding harms the revenue of the original design company but also the counterfeit devices are often of lower quality and could cause system reliability problems.

Device counterfeiting may also be done at the hardware level, by mislabeling the devices. A common attack is to mislabel a lower quality or an earlier generation device to the current generation. The two generations can be distinguished by structural tests, but such tests are difficult to conduct infield and most customers cannot afford the time and expenses of the testing equipment. The chips are likely indistinguishable based on the functional tests since the input/output specifications (and not performance) of the two chips would be similar. The exact statistics for the percentage of counterfeit components is not exactly known; a few years ago, the Alliance for Gray Market and Counterfeit Abatement (AGMA) estimated that about 10% of the electronic products on the market are counterfeit [17]. It was also reported that the percentage of counterfeit components are growing, emerging as a serious threat to the Integrated Circuits and electronics market.

Another potential form of tampering with the bitstream is *reverse-engineering*. The detailed format of the bitstream for a specific FPGA family is typically considered proprietary to the vendor. Even though the bitstream generation or device configuration details are not commonly published and the complexity of the designs often deters a full reversal of the bitstream, the bitstream alone does not provide any provable security. In some sense, vendor specific bitstream generation only provides a level of obscurity, that is not sufficient for providing protection against reverse-engineering. Given enough time and learning algorithms, bitstream reverse

engineering is computationally feasible. Therefore, hiding data and information in the bitstream (i.e., security by obscurity) does not yield a strong protection guarantee.

Full bitstream reversal would expose the IP to unintended parities. Even though the authors are not aware of any tool or method that would offer a full reversal of FPGA bitstream at the time of writing this article, partial reversals of FPGA bitstream were reported earlier. As an example, about 20 years ago, a startup Clear Logic used Altera's bitstreams to produce smaller and cheaper laser programmed devices; however, they had to halt their operations because of a successful lawsuit by Altera [1, 18, 19].

Partial decoding of the bitstream data is also possible by studying the RAM and LUT content [20–22]. An example of how this can be done is reported by Ulogic project that attempted an iterative process that manipulates the available files in the Xilinx Design Language (XDL) format and partial conversion to bitstream. It is also possible to perform a *read-back* function, which is the process of retrieving a snapshot of an operating FPGA's present state. Note that this snapshot just gives the information about configuration and states in one moment and is different from the original bitstream. However, this mechanism, if repeatedly applied, provides an effective characterization tool for testing, verification, and also partial reverse-engineering [1].

9.3 FPGA-Based Applied Cryptography

With the proliferation of personal computing, mobile devices, and Internet, and with the booming of the global information and knowledge, storing and processing of digital functions and data increasingly demands new computing devices. As many of these devices and services are integrated within our daily lives and our personal data, it is not surprising that protection and security are needed in several key applications, including Internet, secure email, secure wireless access, data centers, electronic financial transactions, and grid computing. As a result, several National and International organizations have been working on developing standards for protecting these applications, such as Advances Encryption Standard (AES), Elliptic Curve Cryptography (ECC), and the recent NIST efforts for standardizing the next generation hash functions [23].

Processing of cryptographic algorithms often takes a large amount of system processing time and resources especially for cases where a large amount of data and information is involved, or where the platform is power constrained to satisfy portability and mobility [23]. Furthermore, many applications require real-time secure processing of data which places additional constraints on the system and processor timing. As a result, in many real world scenarios, the hardware implementation is preferred over software. The comparable high throughput and power efficiency of hardcoded modules compared to their programmable counterparts makes the hardware the natural choice in such scenarios.

It is worth noting that while a software implementation is not the most performance efficient option, it is inexpensive, easy to debug, and induces a short time to market. VLSI hardware solutions provide the high throughput and power efficiency, but they are expensive, they have a long development cycle, and they do not provide much flexibility for design alterations. the reconfigurable hardware has become the platform of choice for many cryptographic modules and security processing tasks. This is because of FPGA robustness, comparative low-cost, and shorter time-to-market compared with the ASICs solutions, simultaneously combined with reconfigurable device throughput and power advantages compared with the software and general purpose computing solutions.

There are a number of other reasons for selecting reconfigurable solutions for cryptography and security applications, including: (1) the effectiveness of the FPGA's cell structure for implementing bit-wise logical operations that are needed in many cryptographic algorithms; (2) the large amount of memory blocks built-in the state-of-the-art FPGA devices that ease the implementation of memory intensive substitution operation required by the standard encryption algorithms; (3) the reconfigurable platforms that not only eases interfacing of the security cores to other cores on the same device by allowing reprogrammability but also provides a flexible solution that can be integrated into a larger platform with other components.

9.3.1 Vulnerabilities

The standard cryptographic algorithms are designed to be secure against algorithmic attacks that target the steps and flows of the security procedure. Unfortunately, while conventional cryptography methods have been resilient to attacks on the security algorithm, they have been demonstrated to be vulnerable to attacks that target some aspects of their implementation, including the side-channels, fault injection, and physical attacks. The security cores programmed as softcore, reconfigured on FPGA, or realized in ASIC have all been target of implementation-level attacks. In the remainder of this subsection, we briefly mention the attacks and provide references for further reading on the subject.

9.3.1.1 Side-Channel Attacks

Once a reconfigurable device is programmed to function as a certain circuit, it is possible to extract external measurable manifestations of the incident computations performed in the circuit. The term side-channel is used to refer to quantities that can be measured from the circuit in operation; those measured external quantities are correlated with the circuit computations, and therefore, could provide additional (side-channel) information about the internal circuit values. Examples of common side-channels used for attacking the secure hardware cores include power analysis, timing analysis, and electromagnetic emanation analysis. In all cases, multiple measurements of the side-channel for different inputs and in different conditions

are needed. An important performance measure for the side-channel attacks is the amount of useful information one can get from each round of attack, and the number of required inputs/outputs to successfully accomplish the attack's objectives.

Power Analysis. The CMOS gates consume two types of power: static and dynamic. The static (leakage) is the power leaked away because of the device imperfections. For each gate, the leakage power is a function of the gate-type and its incident input vector. The dynamic (switching) power is incurred when the state of one gate transitions from one value to the next. The dynamic power for each gate is also a function of the gate type and the transition input incident to the gate. Both the static and dynamic power can be externally measured by monitoring the current drawn from the circuit's supply pins.

Generic dynamic power measurement results on the widely used SRAM-based FPGAs had demonstrated that a significant portion of the transitional power on those devices is due to the interconnect routing, while the logic switching and clock transitions composed the remaining parts of dynamic power consumed. The leakage power for the logic was not a significant portion in earlier technologies, but the aggressive miniaturization of transistors is drastically increasing the static power significance in newer technologies [24]. The early work in [25] demonstrated that both simple power analysis (SPA) and differential power analysis (DPA) could reveal information about the secret values and operations performed during the execution of cryptographic operations on FPGA. In SPA, the patterns in the power traces incident to individual inputs are processed. In DPA, the differences among the power trace patterns of two or more input sets are processed. A large body of work on attacking the chips based on SPA and DPA has followed, including [26–31].

Simultaneously, many researchers are working on developing countermeasures against the power analysis attacks [32, 33]. It was shown that if the core is not run in isolation and if there are other sources or cores in the circuit contributing to the power, or even when the core is run in parallel, it is harder to distinguish the contributions of each component. In general, the power analysis attack can be thwarted if the functions that depend on the secret values and information have the same power signature as other operations. Following this principle, two effective countermeasures against the power analysis attacks are: (1) randomization so that the impact of one computation cannot be easily distinguished among the many operations, and (2) equalization such that all computations consume the same amount of power. For each implementation, both methods incur undesirable power and timing overheads which needs to be mitigated while there is also a need to provide proofs for efficiently obfuscating the secret values. Both overhead mitigation and proof of hiding (randomness) are active research topics.

Timing Analysis. The gate timing is also a function of its type and internal values. It was shown that by careful path timing signature measurements, one could be able to reveal the secret values that are processed by the gates [34,35]. The countermeasures for this type of attack are similar in nature to power analysis, and consist of timing equalization and timing randomization. Both methods may incur additional overhead and should be carefully studied and analyzed.

Electromagnetic Emanation Analysis. The movement of electrons during the execution of computations would generate electromagnetic field that can be externally measured by placing antennas outside the chip. Electromagnetic emanation analysis (EMA) was shown to be able to successfully predict the secret values and computations done while executing the security functions [36–41]. Such attacks were also reported for FPGAs. Most countermeasures against this attack are based on disturbing the EM field by changing the device properties or by adding layers. These methods cannot be directly applied to conventional reconfigurable hardware. The proposed methods for thwarting this attack on FPGA rely on distributing the computations across the FPGA area to avoid localizing the events. Last but not least, we note that it was demonstrated that by combining multiple side-channels, one may be able to launch a much stronger attack [42, 43].

9.3.1.2 Fault Injection Attacks

Several forms of operational faults can be induced in circuits performing the secure processing. A fault maybe generated by a number of methods, including controlling of the voltage, inducing an electromagnetic field close to the device, or exposing the device to radiations. If carefully injected, such faults can reveal aspects of the secret. We briefly mention some of the important ongoing work in this area that are also applicable to FPGAs.

Glitch Analysis. The objective of such analysis is to force a device to execute faulty operation(s), or to leave the device in a state that can lead to leaking of secret information. The common techniques for induction of glitch include changing the external clock, and altering the supply voltage. Such attacks were shown to be successful on microcontrollers [44], and if not carefully considered, they can be adopted for FPGA and ASIC implementations. An effective countermeasure against this attack is to ensure that all the states are properly defined in models and in implementation, and to verify that the glitches cannot alter the execution order of the events. Another class of countermeasures is to avoid fault injection by implementing tamper detection mechanisms that would report (or prevent, or even correct) altering of clock pulses or voltage levels.

Ionizing Radiation Analysis. Radiation-induced faults have shown to cause single-event upsets in the CMOS circuits [45–47]. Such single (or multiple) event upsets may cause transient delay faults, or may cause the memory bits to flip (known as soft errors). As the FPGAs are SRAM-based, such memory flips would alter the device's functionality. Ionizing radiation is a way to induce the faults and hence, change the memory content. If targeted accurately, it could be used for changing the secret, or to trace back a secret. The complexity of the integrated circuits and small size of the individual components renders this attack very difficult. Many methods for detection and removal of soft-errors are in development which could additionally deter this type of attacks.

9.3.1.3 Physical Attacks

For an attacker with access to costly and high precision measurement and testing equipment, it is possible to physically probe or alter the device so that the secret information can be extracted [48]. There are at least two major hurdles in performing such an invasive probing. First, very costly higher precision Focused Ion Beam (FIB) measurement equipments are needed to precisely target the specific parts of the chip [49]. Second, the device has to be depackaged and the passivation layers that protect the metal interconnects from oxidation needs to be removed. Depackaging and delayering is challenging for certain class of package technology and interconnect deposition methods. Miniaturization of CMOS to nanometer scales and the added layers of interconnect are rendering this attack extremely difficult for newer technology nodes.

There is also a possibility of performing a *semi-invasive* physical attack. These attacks also need the device packaging to be removed, but then they adopt techniques from thermal analysis, imaging, and other side-channel studies to conclude the properties of the chip [48]. Unlike the invasive attacks that need very costly equipments mainly owned by governments or mega-companies, the semi-invasive attacks are much less costly and more accessible to general public. It is worth noting that both invasive and semi-invasive attacks pose real threats to electronics and new methods for thwarting and circumventing these attacks are under research and development.

9.4 FPGA Hardware Security Primitives

Security on reconfigurable platforms has emerged as a challenging security paradigm in system design. Systems implemented on FPGAs like any other systems could require secure operations and communications. However, as we discussed in the previous section, on reconfigurable systems in addition to concerns regarding the compromise of data confidentiality and integrity, the system itself can be subject to malicious architectural alterations to the hardware and to design theft during the operation or even before the design is loaded. As a result, it is critical to establish security of configuration data and maintain design integrity against malicious changes. Several existing solutions govern different trade-offs between security and the market requirements on cost and performance. In this section, we discuss a number of mechanisms and protocols that can be used as the underlying primitives for many FPGA security protocols and modules.

Every FPGA relies on certain programming technology that enables the control and configuration of programmable switches inside the FPGA which in turn program the functionality of the underlying logic. Historically used programming technologies include EPROM [50], EEPROM [51, 52], flash [53], static memory (SRAM) [54], and antifuse [55, 56]. Among these technologies, mainly the flash memory, the static memory, and the antifuse are used in modern FPGA devices.

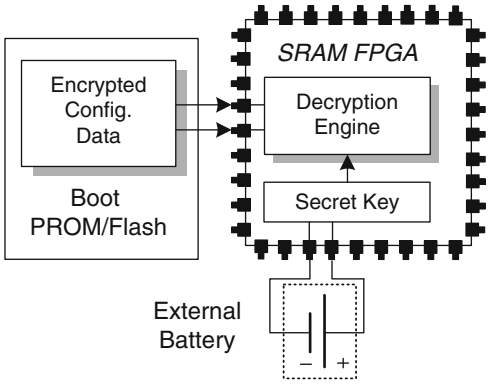


Fig. 9.2 Embedded key storage on SRAM-based FPGAs

The dominant family of FPGAs is realized using volatile SRAM-based memories. Upon power-up, the FPGA is configured by loading and storing the desired functionality inside the SRAM memory cells. The SRAM cell values define the logic functions by means of initializing a set of truth tables or *lookup tables (LUT)* and by enabling/disabling connections through switch matrices. Once the FPGA is powered off, the content of the SRAM cells is lost. In other words, the SRAM-based FPGAs must be constantly powered to retain the configured functionality and they need to be reprogrammed every time the power is lost.

The lack of nonvolatile embedded storage mechanisms on SRAM-based FPGAs thwarts permanent storage of secret keys which is required to establish a secure channel for sending the configuration data. Without the use of encryption, the configuration bitstream has to be communicated to the FPGA at start-up through a nonsecure channel. This is specially important in applications in which systems and IPs must be protected against piracy or unauthorized read-out as well as against malicious changes to tweak the system functionality.

Integration of nonvolatile memory on SRAM-based FPGAs is costly because integration of state-of-the-art nonvolatile technologies on standard CMOS process requires more complicated fabrication steps and wafer processing. As a result, nonvolatile storage is often not available on lower-end devices [6]. In order to store keys on SRAM-based FPGA, an external battery is typically attached to the device to constantly provide energy to the SRAM cells containing the secret key (s). The concept is shown in Fig. 9.2.

Antifuse technology uses a layer of amorphous silicon in the via, which causes an isolation between the metal layers [57]. In the un-programmed state, the amorphous silicon has very high resistance, thus isolating the metal layers. After programming voltage is applied, the amorphous silicon resistance drops significantly, creating a metal to metal interconnect. Compared to other technologies and even ASICs, the antifuse FPGAs enjoys the highest level of security, because of the following reasons: (1) As the FPGA can be configured once and shipped by the system designer to the end-user, there's no need to transfer the configuration over an insecure channel. (2) The fabric of the FPGA (i.e., the interconnection, routing, and

Table 9.1 Comparison of current programmable technologies

	SRAM	Flash	Antifuse
Volatile?	Yes	Yes	No
Reprogrammable?	Yes	Yes	No
Area	High	Moderate	Low
Power	High	Low	Low
Manufacturing process	Standard CMOS	Flash process	Special development
Programming yield?	100%	100%	>90%
Security	Low	Moderate	High

placement of the programmable elements) reveals no information about the design (in contrast with ASICs). This is due to the fact that all the design data is internal to the device and it is stored at programmable links. Invasive reverse engineering methods such as etching that take away the surface will only reveal the top of the vias and not the state of the amorphous antifuse silicon; thus, such techniques do not expose much information on the chip functionality. Non-invasive attacks that use advanced imaging and probing techniques such as SEM theoretically might have a chance to monitor the device. The imaging technique attempt to determined the state of antifuse links by looking for any deformations in the amorphous silicon vias. With millions of links on each device, it is still not an easy task to scan every single link of the FPGA. For example, Actel's AX2000 antifuse FPGA contains approximately 53 million antifuse elements.

As the antifuse FPGAs can only be programmed once, it takes away a great advantage of in-field FPGA reconfigurability feature. Table 9.1 summarizes the properties of different programming technologies.

In the rest of this section, we focus our attention on SRAM FPGAs because they currently have the largest market share in the reconfigurable hardware domain.

9.4.1 Physical Unclonable Function

Physical Unclonable Functions (PUFs) provide an alternative mechanism for key storage on SRAM-based FPGAs. PUFs overcome the inherent vulnerability of key storage on nonvolatile memories against various attacks as well as the extra technology cost overhead of nonvolatile memory integration onto SRAM-based devices. PUFs use the inherent and embedded nano- and micro-scale randomness in silicon device physics to establish and define a secret which is physically tied to the hardware. The randomness is introduced by the existing uncertainty and lack of precise control during the fabrication process that lead to variations in device dimensions, doping, and material quality. The variation in device physics transfers itself into variations in electrical properties, such as transistor drive current, threshold voltages, capacitance, and inductance parasitics. Such variations are unique for each IC and device on each IC. PUFs typically accepts a set of input

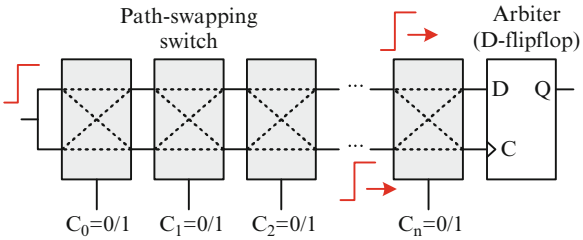


Fig. 9.3 Arbiter-based PUF introduced in [58]

challenges and map them to a set of output responses. The mapping is a function of the unique device-dependent characteristics. Therefore, the responses two PUFs on two different chips produce to the same set of inputs are different. A comprehensive review of PUF concept and literature is provided in Chap. 7 of this book. In the remainder of this chapter, we focus on the work covering the FPGA PUFs. Our discussions are complementary to the material presented in the earlier chapter.

A common way to build a PUF in both ASICs and FPGAs is by measuring, comparing, and quantifying the propagation delays across the logic elements and interconnects. The variations in delays appears in forms of clock skews on clock network, jitter noise on the clock, variations in setup and hold times of flipflops, and the propagation path delays through the combinational logics.

The work in [58] was the first to exploit the unique and unclonable delay variations of silicon devices for PUF formation. The PUF, known as arbiter PUF or delay-based PUF, is shown in Fig. 9.3. The PUF uses the analog differences between the delays of two parallel paths that are identical in design and prior to fabrication, but the physical device imperfections make the delays different. Beginning the operations, a rising transition is exert at the PUF input producing a racing condition on the parallel paths. An arbiter at the end of the paths generates binary responses based on the signal arrival times. To enable multiple path combinations and generate an exponential number of challenge/response pairs, the paths are divided into multiple sub-paths interleaved by a set of path swapping switches. The challenges to the PUF control the switches and, therefore, how the varying paths are formed.

A successful implementation of this type of PUF was demonstrated on ASICs platforms [59]. It is critical to note that the differences in delays should be solely coming from manufacturing variation and not from design-induced biases. To obtain exact symmetry on the signal paths and to equalize the nominal delays, careful and precise custom layout with manual placement and routing is required for implementation on ASICs. The lack of a fine control over arbitrary placement and routing on FPGA has resulted in difficulty in balancing the nominal delays on the racing paths within the arbiter-based PUF. Implementation on FPGA was troubled because of the constraints in routing and placement imposed by the rigid fabric of the FPGA as studied in [60, 61].

However, the recent work in [62] has addressed this problem by demonstrating a working implementation of the arbiter-based PUF on FPGA that utilizes a nonswapping symmetric switch structure as well as a precise programmable delay

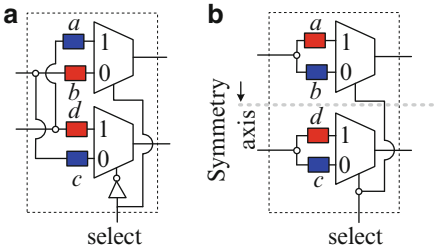


Fig. 9.4 Two implementation of path selecting switches (**a**) Asymmetric path-swapping switch (**b**) Symmetric non path-swapping switch

line (PDL) component to cancel out the systematic delay biases. The path-swapping switch previously used in the arbiter-based PUF of Fig. 9.3 can be implemented by two multiplexers (MUX) and one inverter as depicted in Fig. 9.4b. However, due to cross wiring from the lower half to the upper half (diagonal routing), maintaining symmetry in path lengths for this type of switches is extremely difficult. To avoid diagonal routings, a nonpath swapping switch with a similar structure was introduced in [62] which uses two MUXes as shown in Fig. 9.4a. As it can be seen on the figure, after applying the method the resulting routings and path lengths are symmetric and identical across the symmetry axis (drawn by the dashed line).

Despite using a symmetric switch structure, systematic biases in delay would still exist due to the asymmetries in routing from the last switch to the arbiter flipflop and/or before the first switch. To eliminate such delay skews, a highly accurate programmable delay line (PDL) was introduced in [62]. The PDL was implemented by a single LUT and can achieve a resolution of better than 1 picosecond. The PDL works by slightly incrementing/decrementing the signal propagation path length inside the LUT. Figure 9.5 shows an example PDL implemented by a three-input LUT. The LUT implements an inverter logic where the output of the LUT reflects negation of A_1. However, the inputs A_2 and A_3 functionally serve as don't-cares while they can change the signal prorogation path inside the LUT and cause slight in the propagation delay.

In contrast to the arbiter-based PUF where racing condition is formed by signal propagation through two independent paths, the FPGA PUF introduced in [63, 64] which referred to as time-bounded PUF, compares the signal propagation speed through a combinational logic against the system clock speed. The time-bounded PUF uses the standard at-speed delay test circuit (delay characterization circuit) shown in Fig. 9.6a. The at-speed delay test circuit consists of one *launch*, *sample*, and *capture* flipflop. At the rising edge of the clock, the launch flipflop sends a low-to-high signal through the circuit under test (CUT). At the falling edge of the clock the output of the CUT is sampled by the sample flipflop. The steady state output of the CUT is then compared with the sampled value by an XOR logic. If discrepancies exist, it means that the output was sampled before the signal had arrived at the output of CUT. This condition is referred to as timing error. By sweeping the clock frequency in a linear fashion, one can locate the transition point from error free zone to full error zone. The center of the transition corresponds to the delay of CUT.

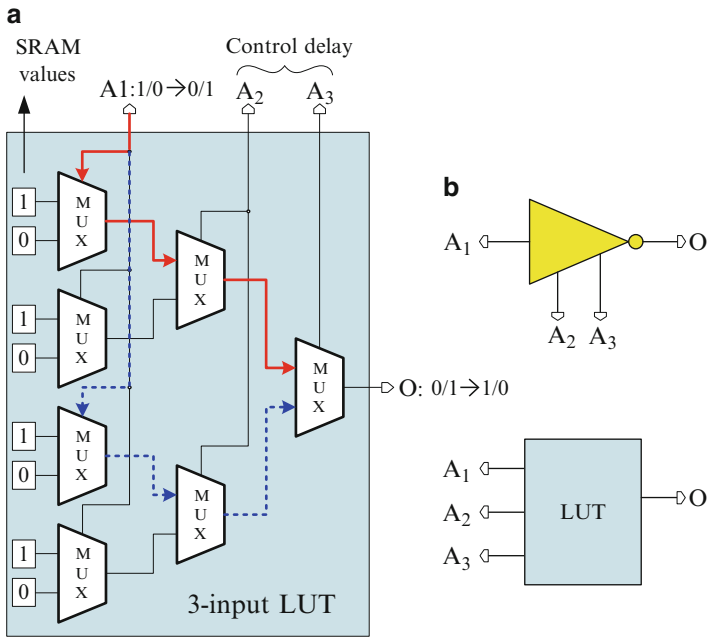


Fig. 9.5 (**a**) LUT-based programmable delay line (**b**) symmetric switch structure

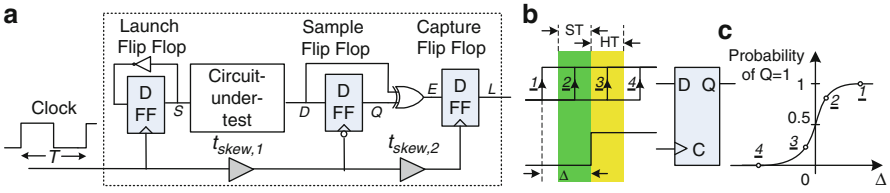


Fig. 9.6 (**a**) Delay characterization circuit based on at-speed delay testing mechanism (**b**) sampling signals with different arrival times (**c**) probability of the flipflop output = 1

If the time difference between the sampling time and the signal arrival time is smaller than the setup and hold time of the sample flipflop, then sample flipflop produces nondeterministic outputs. It was shown in [63, 65] the probability of sampling the correct value in this case is a monotonically increasing function of the time difference between the signal arrival time and the sample time. This is depicted in Fig. 9.6b, c. To estimate the center of this smooth transition curve, statistics on the observed error need to be gathered.

A careful investigation of the characterization circuit reveals that the observability of timing errors go through periodic phases. The measured probability of timing error as a function of half clock period (T/2) on Virtex 5 FPGA is illustrated in Fig. 9.7. The two consecutive transitions from 0% to 50% and 50% to 0%

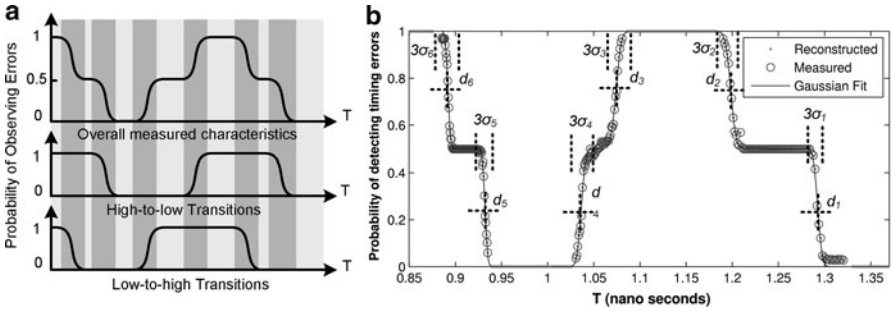


Fig. 9.7 (**a**) Probability of observing timing error for rising/falling edge transition and both transitions as a function of half clock period (**b**) Measured probability of transition as a function of half clock period on Virtex 5 FPGAs

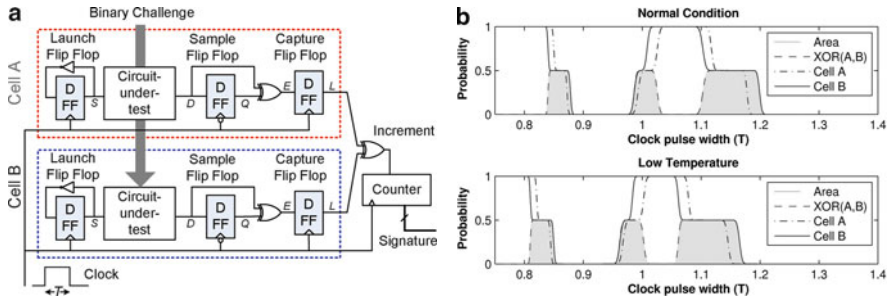


Fig. 9.8 Extracting shift invariant signatures (**a**) Differential timing circuit (**b**) Symmetric nonpath-swapping switch

(and vise versa) are formed by the differences in propagation delays in rising edge and falling edge signals. The measured probability is the net effect of both transitions. The center and slope of each transition point are unique to each circuit on different FPGAs.

The extracted absolute delay parameters are sensitive to changes in environmental variations. In order to obtain more resilient responses and better signatures against such fluctuations, a method to perform linear calibration of the clock frequency according to the current temperature is introduced in [64]. The operating temperature and voltage are obtained by querying the built-in FPGA sensors, and calibration is performed accordingly on the clock frequency. In addition to frequency calibration, a differential structure is further proposed that cancels out the common effect of environmental variations on delays as shown in Fig. 9.8a. The differential circuit consists of two at-speed delay test circuit (Fig. 9.6) whose outputs are tied to an XOR logic. As the absolute delays increase/decrease, extracting shift invariant parameters such as the distance between the centers of transition regions (width), or the area under the curve would result in more robust signatures.

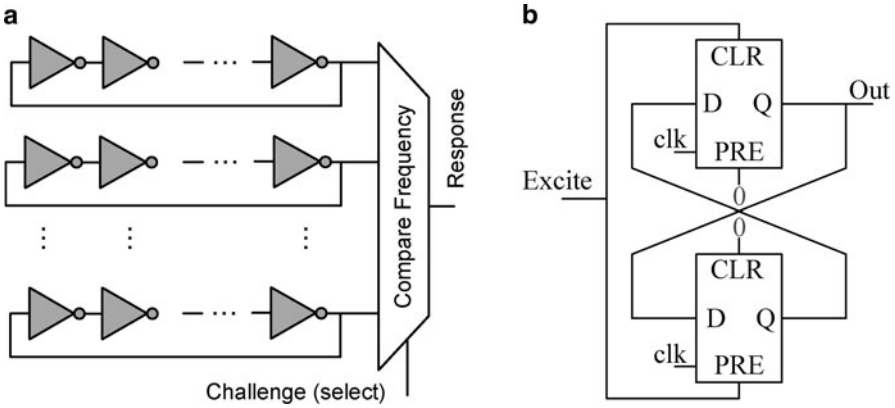


Fig. 9.9 Other delay-based PUFs (**a**) RO-PUF (**b**) Butterfly PUF

The circuit in Fig. 9.8a measures the area under the XOR probability curve using Riemann sum approximation. As it can be observed on the figure, the area under the measured curves stays the same for low and normal operating temperatures.

Another family of PUFs amenable to implementation on digital platforms and in particular FPGAs, is based on ring oscillators (RO-PUF). A ring oscillator is composed of an odd number of inverters forming a chain. Due to variations in delays of comprising logic components and interconnects, each ring oscillates at a slightly different frequency. The RO-PUF measures and compares the unique frequency of oscillation within a set of ring oscillators. A typical structure of RO-PUF is shown in Fig. 9.9a. Most of the work around RO-PUFs is focused on post processing techniques, selection, quantization and comparison mechanisms to extract digital responses while achieving robustness of responses and high response entropy.

One of the early papers to consider and study ring oscillators for digital secret generation is [66]. The work proposes a 1-out-of-k mask selection scheme to enhance the reliability of generated response bits. For each k ring oscillator pairs, the pair that has the maximum frequency distance is chosen. It is argued that if the frequency difference between two ring oscillators is big enough, then it is less likely that their difference changes sign in presence of fluctuations in operating temperature or supply voltage.

In order to achieve higher stability and robustness of responses, extra information can be collected by measuring the oscillation frequency under different operating conditions. Methods presented in references [67, 68] use this information to efficiently pair or group the ring oscillators to obtain maximum response entropy. Specifically, frequency measurement is performed at two extreme (low and high) temperatures and a linear model is built to predict the frequency at middle temperature points.

Systematic process variation can adversely affect the ability of RO-PUF for generation of unique responses. A method to improve uniqueness of ring oscillator

PUF responses is discussed in [69]. A compensation method is used to mitigate the effect of systematic variation by (1) placing the group of ROs as close as possible (2) picking the physically adjacent pair of ROs while evaluating a response bit. Large scale characterization of an array of ROs on 125 FPGAs (Spartan3E) is performed in [70].

The existing inherent race conditions in combinatorial logics with feedback loop are also used in development of other types of PUFs. For instance, a loop made of two inverter gates can have two possible states. At the power-up, the system enters into a metastable state that settles onto one of two possible states. In fact, the faster gate will dominate the slower gate and determine the output. The idea of back-to-back inverter loops is used in SRAM memory cells. SRAM-based PUFs based on the inherent race condition and variations in component delays produce unique outputs at startup. Unfortunately, in SRAM-based FPGAs, an automatic internal reset mechanism prevents using the unique startup value. A more practical implementation that is based on the same concept but uses the logic components on FPGA rather than the configuration SRAM cells, is referred to as a butterfly PUF. The basic structure of a butterfly PUF is shown in Fig. 9.9b. Butterfly PUF is made of two D-flipflops with asynchronous preset and reset inputs. The flipflops are treated as combinational logics. The work in [71] presents a comparative analysis of delay based PUF implantations on FPGA. The work particularly focuses on the requirements of maintaining symmetry in routing inside the building blocks of Arbiter-based PUF, Butterfly PUF, and RO-PUF.

9.4.2 True-Random Number Generator

FPGAs are also suitable platforms for implementing True-Random Number Generators (TRNG). TRNGs are important security primitives that can be used to generate random numbers for tasks such as (1) secret or public keys generation, (2) initialization vectors and seeds for cryptographic primitives and psuedo-random number generators, (3) padding bits, and (4) nonces (number used once). As modern cryptographic algorithms often require large key sizes, generating the key from a smaller sized seed will significantly reduce the effectiveness of the long keys. In other words, by performing a brute-force attack only on the seed that generated the key, one can break the crypto system. Therefore, it is essential to generate the keys from a high entropy source.

Numerous TRNG designs have been proposed and implemented. Each design uses a difference mechanism to extract randomness from some underlying physical phenomena that exhibit uncertainty or unpredictability (or probably a behavior not well-understood). Examples of sources of randomness include thermal shot noise in circuits, secondary effects such as jitter and metastability in circuits, Brownian motion, atmospheric noise, nuclear decay, random photon behavior. In this chapter, we only focus on TRNGs that are implementable on digital platforms and FPGAs.

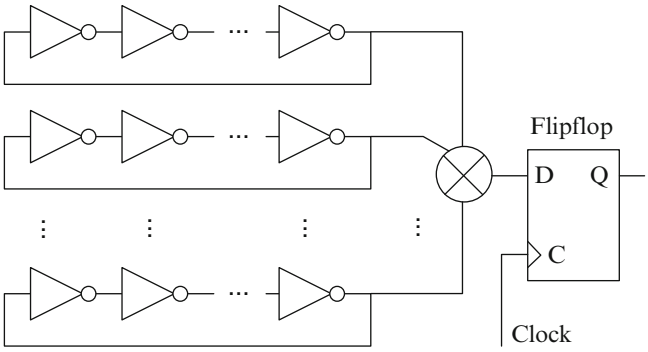


Fig. 9.10 TRNG based on sampling the ring oscillator phase jitter

In general, TRNGs are evaluated using the following typical parameters and measures: (1) entropy source (source of randomness), (2) design footprint (area and energy per bit), (3) predictability of the generated bitstream and its statistical properties, (4) security and robustness of the generated bits against attacks, and (5) ease of implementation.

As discussed in the previous section, one measurable analog quantity on digital platforms is the signal propagation delay. The circuit noise (thermal, shot, and flicker noise) can exhibit their effect on propagation delays. The noise manifest itself as the jitter and phase noise on the systems clock by causing temporal variations in the oscillator frequency.

The approach in [72] uses sampling of phase jitter in oscillator rings to generate a sequence of random bits. The output of a group of identical ring oscillators are fed to a parity generator function (i.e., multi-input XOR). The parity generator output is then constantly sampled by a D-flipflop driven using the system clock. In absence of noise and identical phases, XOR output would be constant (and deterministic). However, in presence of jitter in phase, glitches with varying nondeterministic lengths appear at the XORoutput (Fig. 9.10).

Another type of TRNG is introduced in [73] that is based on the basic arbiter-based PUF structure. Unlike PUFs where reliable response generation is desired, the PUF-based TRNG goal is to generate unstable responses. This is achieved by driving the arbiter into the metastable state essentially through violating the setup/hold time requirement of the arbiter. The PUF-based random number generation method searches for challenges that result in small delay differences at the input of the arbiter which in turn cause highly unreliable response bits.

In order to improve the quality of the output bitsteam and increase the randomness, various post-processing techniques are often performed. [72] introduces resilient functions to filter out deterministic bits. The resilient function is implemented by a linear transformation through a generator matrix commonly used in linear codes. The hardware implementation of resilient function is demonstrated in [74] on Xilinx Virtex II FPGAs. The TRNG after post processing achieves a

throughput of 2Mbps using 110 ring oscillators with three inverters in each. A post-processing may be as simple as von Neumann corrector [75] or may be more complicated such as extractor function [76] or even a one-way hash function such SHA-1 [77]. Von Neumann method is a well-known post-processing technique to removed localized biases in the generated bit sequence. It looks at pairs of bits in the bitstream. If both bits in the pair are identical, the corrector removes both of them from the sequence. If the bits are different then it uses only one of them (e.g., the second bit). The bit rate as a result will be reduced to about 1/4 of the input bit rate on average (this is for the optimistic case where 0s and 1s are equally likely).

Besides improving the statistical properties of the output bit sequence and removing biases in probabilities, post-processing techniques increase the TRNG resilience against adversarial manipulation and variations in environmental conditions. An active adversary attacker may attempt to bias the probability of the output bits in order to reduce the entropy of the generated keys. Post-processing techniques typically govern a trade-off between the quality of the generated bit vs. the throughput. Other online monitoring techniques may be used to ensure higher quality of the generated random bits. For instance, in [73], the probability of the generated bits are constant monitored and as soon as a bias is observed in the bit sequence, the search for a new challenge vector that produces unreliable response bits is initiated.

Although it is almost impossible to analytically and mathematically prove the unpredictability of the generated bit stream, a simple system design, insight on underlying physical randomness, as well as a thorough examination of the bitstream statistical properties and randomness are fundamental to justify the security of the TRNGs. In other words, it is necessary, although not sufficient, to perform a comprehensive set of statistical tests on the generated bit sequence. A well-known and common suites of randomness tests are outlined in DIEHARD [78] and NIST Test Suites [79].

9.5 Top FPGA Security Challenges

In this section we identify and analyze dominant FPGA research and development challenges and opportunities. The challenges are dictated by technological, application, business models, and tools development trends. We start by discussing our top 15 challenges and finish by analyzing uniqueness of FPGA security requirements and degrees of freedom with respect to ASIC and general purpose and application specific programmable processors. As we already stated, it is obvious that each platforms has certain advantages or limitation depending on the security threats and goals as well as a set of overall security desiderata. However, it is important to emphasize that flexibility and configuration capabilities of FPGAs may be instrumental for creation of unique classes of security primitive and protocols.

9.5.1 Algorithmic Cryptographic Security

Algorithmic (mathematical) cryptography is one of the most elegant and effective computer science fields. Numerous ingenious and surprising primitives and protocols have proposed, analyzed, and are in practical use [80–84]. The algorithmic (mathematical) foundations for some protocols such as public-key communication and storage are solid although rarely actual consistent proofs are available. Nevertheless, the chances of breaking modern protocols such as Advanced Encryption Standard (AES) using algorithmic attacks are relatively very small.

However, it is well known that the computer engineering basis of algorithmic security is far less reliable. It has been reported that a variety of physical and side channel attacks easily using inexpensive equipment easily break essentially all algorithmic cryptography protocols. Development of engineering techniques for protection of information leakage is a popular research direction. These techniques are often much less effective for FPGA platforms due to factors such requirements for highly regular routing, relatively sparse and publicly available IC structure, and higher difficulty and cost of TRNG. However, the greatest impediment to any masking technique in particular at the gate level is process variation that prevents matching of gates. In principle, FPGA platforms have here advantage due to their reconfigurability.

Once when the physical security of an FPGA platform is ensured, it would have a high potential for significant energy efficiency, protocols flexibility, and even speed advantages over programmable platforms. In addition, FPGA's ability to facilitate reuse can greatly improve actual security. There is little doubt that at least in short time periods, cost, energy, low latency, and high throughput of algorithmic cryptographical protocols will be of the primary importance.

9.5.2 Hardware-Based Cryptography: Primitives and Protocols

Hardware-based security techniques have been going through several partly overlapping phases. Initially, the emphasis was on creation of unique ID. In the next phase, IDs were used for protection of the platform and applications related to the hardware or software running on the platform including: hardware metering, remote enabling and disabling, and similar tasks. Silicon PUFs initiated a revolution in hardware security [58, 73]. However traditional PUF technique utilize only secret key-based cryptography. More recently several schemes redefined ways how PUFs are constructed and used to enable a variety of public key security protocols. They have been developed under the names of PPUF [13], SIMPL [85, 86], and timed authentication [13, 63, 87]. While the public key PUF approaches have been proposed several years ago, now more and more realistic schemes are analyzed. For example, PPUF-based scheme include not only authentication and public-key private key communication, but also time stamping, place stamping, device

stamping, and more demanding protocols such as coin flipping and oblivious transfer. The crucial observation is that FPGAs are ideal platform for many type of primitives and security protocols due to their reconfiguration capabilities.

In addition to hardware primitives, device-level characterization and conditioning play important enabling roles. For example, it has been demonstrated that leakage energy measurement can be used for fast and very accurate gate-level characterization and for provably comprehensive hardware trojans detection [88–93]. Furthermore, it has been demonstrated that in addition to side channels ways for collecting information, there are device conditioning techniques that can be used to organize accurate and diverse measurements. For example, localized heating can be used for breaking correlation in linear systems of equations in such a way that all gates can be characterized [91]. As another example, very accurate detection bounds based on the submodularity of the objective function can be achieved [90]. These techniques are universal, but the presence on unused hardware on FPGA ICs can additionally facilitate their effectiveness [63, 64]. Finally, hardware primitives can be used for creation of a great variety of security protocols. It is important to note that they have sharply different principles than the identical algorithm-based security protocols.

9.5.3 Digital Right Management of ICs and Tools

There are dominant approaches for protecting hardware IPs. The first is watermarking [8, 94–101]. Numerous hardware watermarking techniques have been proposed at essentially all levels of design abstraction. The current emphasis in on using of side channels for efficient watermark detection [102]. It is important to emphasize that several of early hardware watermarking techniques enable easy watermark detection through minimal modification of outputs or through augmentation of finite state machines [8, 101]. It is easy to see that IP watermarking is often much more difficult task for FPGAs than for ASICs. This is in particular true for techniques that embed watermarks by superimposing additional constraints on design specification. The main reason is that watermarks on higher levels of ASIC synthesis are naturally protected by the nonrecurring engineering (NRE) cost and time-to-market delay. Hardware watermarking is covered in Chap. 9 of this book.

The second task is hardware metering where the goal is to ensure that foundry does not sell unauthorized ICs [103, 104]. There are two broad classes of hardware metering. The first is passive hardware metering where already sold ICs are examined in order to detect illegally produced ICs. Passive metering has techniques are equally difficult for both FPGA and ASIC designs. Active metering techniques do not only detect illegal ICs but also directly enforce DRM rights by requiring specific authentication steps that can be provided only by the designer. While obviously active metering techniques are more effective, they also induce higher operational and design overheads in metrics such as energy or delay. Metering is covered in Chap. 8 of this book.

Finally, the most ambitious DRM step is remote control where the designer or an authorized entity can remotely control which action can or cannot be conducted by the user [105–108]. The stated three types of techniques and many other DRM tasks (e.g., auditing) can be performed on all three implementation platforms (FPGA, ASIC, and programmable processor). One advantage of programmable and configurable platforms is that a more refined control can be conducted on them.

9.5.4 Trusted Tools

In modern and pending synthesis flows, usage of design tools is unavoidable. It is easy to embed a variety of malicious circuitry, malicious functionality, and security vulnerabilities using the CAD tools. A key security task is to derive a trusted set of synthesis and analysis tools. There are two types of trusted tools required for FPGA synthesis. The first type is tools used for realization of FPGA structures themselves. The second is tools that are used for implementation of a specified functionality on FPGA chips. In the case when programmable processors are used on FPGA ICs, we need to also secure the compiler(s) and the operating system(s).

Although at the first look the problem may seem intractable, recently it was addressed in a surprisingly simple way using notions of fully specified design (FSD). FSD is a design where user-specified functionality utilizes all the resources at all times. Therefore, a potential attacker does not have means (hardware and clock cycles) to initiate attacks. FSD can be easily realized using regular synthesis tools. The key idea is that the designer develops simple tools for checking the solutions produced by the complex synthesis tools. For example, the designer can keep updating her specified functionality until all functional units are not used in all clock cycles [109].

Additional efforts are needed to ensure that the produced designs are energy efficient and to provide runtime checking. Of course, this first approach will hopefully provide impetus for other conceptually different techniques for trusted synthesis. The final remark is that one can create numerous abstraction of trusted synthesis and that is an interesting and important problem itself.

9.5.5 Trusted IP

In addition to trusted tools, modern design flows require trusted hardware and software IP. Deriving techniques that provide proofs that a particular IP is trustworthy is essential due to the ever increasing silicon-productivity gap. There are two practical options. One is to require that each IP is fully checkable. For example, one can insist that each IP is created using trusted tools; the test vectors could be included for verification. Another maybe even more realistic, but less secure option is to develop security wrappers in form of additional circuit logic that control all inputs

and outputs from each IP. In that case, the IP user can export to the otherwise untrusted IP or import from it only the data that is functionally specified. Therefore, IP would not get in possession of the privileged information from other parts of the overall design. It still can produce intentionally incorrect results, but these data would not help the attacker to take a control over the overall design. The FPGA flexibility makes this platform better suited for inclusion of trusted IP.

9.5.6 Prevention of Reverse Engineering

One of the often promoted FPGA advantages over other implementation and architectural options is its resiliency against reverse engineering [110, 111]. Today, IC reverse engineering is widely used for tasks such as patents enforcement, technological espionage, and market trend tracing. In particular, antifuse FPGA (such as the ones designed by Actel) are identified as reverse engineering resilient devices and are widely used by several US government agencies. The key argument is that antifuse devices have very small feature sizes and it is very difficult to figure out if a particular devices fused and not. Since the number of fuses is large (several hundred thousands) their accurate classification is at least very demanding. In addition, it is often argued that usually only a small percentage of them is actually fused and that, therefore, this makes them even more difficult for reverse engineering. However, this argument is questionable since the entropy of the scenario is much higher than one where a half of fuses is actually burned.

We anticipate that reverse engineering research will pursue two lines of attacks. The first is indeed technological where the goal is to develop technologies that are difficult (ideally impossible) for reverse engineering. The second line will explore diversity in functionality specification and realization that will make each IC unique. Hence, merging imperfect technological information from multiple ICs will not bring results. A prime candidate for this purpose are N-version synthesis techniques [112–114].

9.5.7 Trojan Detection and Diagnosis

Recently, hardware Trojan detection and diagnosis attracted a great deal of attention. Currently, a major emphasis is on added ghost circuitry that is used in a variety of detrimental ways to alter the functionality of the initial design. In addition, more subtle attacks that employ device aging or resizing and crosstalk have been proposed and analyzed. The detection techniques can classified in two broad classes: (1) side channel-based; and (2) ones that use functional or delay testing as their starting points. Silicon foundries are often cited as a major potential security attacker. It was argued that FPGA automatically provide protections against hardware trojans since the designer consequently configures FPGA in such a way that this information is

not available to potential attackers. In addition, the regular FPGA structures makes embedding of hardware trojans difficult. However, that is only to a certain extent true because the attacker can also alter nonfunctional crucial components of designs such as power supply network.

It is important to note that hardware Trojans detection is much more difficult than functional or manufacturing testing because malicious alterations are intentionally conducted such that their analysis is difficult or maybe even infeasible. There are two main conceptual and technical difficulties. The first is that the ratio of the number of gates vs. input/output pins keeps increasing and as a result, the controllability and observability is consistently reduced. The second is that many discrepancies between the measurements and the simulations can be easily explained as the impact of process variations.

Nevertheless, one can comfortably state that many types of structural hardware Trojans can be already detected and even diagnosed [4]. We expect that the next generation of functional Trojan horses where malicious circuitry is partly or fully merged with circuitry that is actually used for targeted functionality will create much more severe security requirements. The Trojan related topics are comprehensively addressed in Chaps. 15–17 of this book.

9.5.8 Zero Knowledge and Oblivious Transfer

There is an interesting and important class of cryptographical protocols that are unfortunately too complex for widespread use in their software implementation. This class includes zero knowledge and oblivious transfer. Hardware-based security primitives such as PPUFs when realized on FPGA have the potential to create ultra efficient realization of these protocols [13, 64]. We expect that a variety of these and similar protocols will not only be proposed but also be realized and demonstrated. Interestingly, in many of these applications, protocols, and security primitives the role of flexibility is essential. Therefore, FPGAs will be often the preferred implementation platform for those types of protocols.

9.5.9 Self-Trusted Synthesis

A variety of trusted modules and platforms have been developed. In some situations there is not even a specific itemization and quantification of what and who is trusted.

It has been demonstrated recently that one can easily create PUF structures in such a way that all decisions about its parameters and, therefore, the relationship between challenges and responses is completely controlled by the user. The procedure is surprisingly simple. It is sufficient to allow a user to age each PUF delay segment either randomly or to its own specifications [92, 115–117].

While, of course, the path from the devising and implementing a PUF to designing and implementing an arbitrary design may be complex and complicated, we believe that soon such solutions will be created. The FPGA flexibility is essential for such tasks although one could also create flexible ASIC solutions.

9.5.10 New FPGA Architectures and Technologies

There is a large number of different FPGA architectures in terms of combinatorial logic blocks, interconnects, and embedded memories. There are also several technologies that are used for configuration (e.g., SRAM and fuses). FPGA can be used to implement a great variety of applications. However, it appears that no consideration for security primitives and protocols has been used as the design objectives and/or constraints. After several decades of stable silicon CMOS technology, it seems that we are on the brink of revolutionary changes. For example, technologies such as graphene, III–V and graphene nanotubes, memristors, phase change materials, photonics, and plasmonics may fundamentally alter the design objectives and design process. We already see that the process variation greatly complicates detection of hardware Trojans and enables PUF existence and optimization. These technological changes will greatly impact FPGA trade-offs and architectures. In addition, 3D technologies might qualitatively alter the FPGA architectures and could have influential security ramifications.

9.5.11 FPGA Tools for Hardware-Based Security

Development and rigorous analysis of FPGA security tools is a difficult and complex task. For example, process variation (PV) often plays a crucial role. PV impacts all the design metrics and has a complicated nature that keeps changing with each new technological node. For example, the effective channel length depends on several highly correlated factors. On the other hand, the threshold voltage consistently follows an uncorrelated Gaussian distribution. Other models such as device aging are also of high importance. In addition, tools for reverse engineering may have a crucial importance.

In general, one has two options: implementation and/or simulation. As FPGA implementation platform is greatly preferred because of its sharply lower cost and flexibility. There is, at least among one class of researchers, the philosophy that implementation is ultimate proof of any concept and the value of simulation is minimal. There is, obviously, some advantages in the implementation. If nothing else, it implies that the technique works at least on one platform. At the same time, any statistical proof based on one or very few points is at best of questionable value. Also, very little insight and knowledge is obtained from so limited experiments.

Simulation models are widely used in industry and have convincingly demonstrated their practical values. They can be used not only for well established technologies but also for the pending ones. Therefore, simulations are of great theoretical, conceptual, and practical values. Still, in order to obtain maximal benefit, comprehensive and up-to-date modeling and simulation tools are needed. We believe that it is crucial to have sound and fast FPGA security models that are shared among various groups of researchers. There are already activities along these lines, including Trust-Hub[1] that aims to provide both FPGA platforms that are remotely accessible as well as simulation and benchmark capabilities. For example, a collection of most effective attacks may greatly improve the development of security techniques. Finally, it is important to emphasize that these tools must find ways to transparent synthesis and analysis of FPGA-based systems.

9.5.12 Side Channels

Side channels are effective mediums and mechanisms that drastically increase the observability of the inside of the pertinent IC [118]. There is a large and ever increasing number of side channels modalities including delay, power, electromagnetic emanation, substrate noise, and temperature. Side channels greatly facilitate the detection and analysis of malicious circuitry. They also impose an even stronger threat to cryptographical protocols and hardware-based security techniques.

Side channels are in particular effective against FPGA implementations because the structure of the circuitry is known. The density is relatively lower than ASICs, and one can embed additional circuitry to augment the side channels.

9.5.13 Theoretical Foundations

Numerous interesting and sometimes surprisingly elegant hardware security techniques have been proposed. In addition, several classifications for hardware Trojan attacks and defense mechanisms, TRNGs, PUFs, and other hardware security primitives and protocols have been published. Nevertheless, we are still far from establishing sound foundations and identifying the universally beneficial paradigms.

Currently, innovative but ad-hoc techniques dominate the development of hardware-based security techniques for both ASIC and FPGA platforms. More complex structure of FPGA synthesis flow and much higher number of heterogeneous design results in more pressing need for development of sound foundations and standardized ways for analysis of synthesis and analysis flows.

[1] http://trust-hub.org/.

9.5.14 Physical and Social Security Applications

The strong emphasis of both classical algorithmic security and emerging hardware-based security techniques is on data and electronic system protection. These goals are, of course, tremendously important. However, two types of new application classes are of even higher importance. The first type consist of securing physical, chemical, and biological entities [119]. The second is related to personal and social security. Interestingly, hardware based security has the potential to play an essential role in many such application. For example, it has been demonstrated that many objects such as paper, DVD, optical fiber, and wireless radios can be used as PUFs. An important alternative is to integrate silicon (in particular FPGA) PUFs for protection of physical or biological systems. Even parts (e.g., blood vessels) of individual human bodies can be used as PUFs. FPGA-based platforms would be most often used as a staring point due to their low NRE cost.

9.5.15 Recovery and Long-life Enabling Techniques

Faults masking techniques such as built-in-self-repair (BISR) play an important role in enhancing the yield or lifetime of integrated circuits. For example, BISR mechanisms and circuitry are widely used in dynamic random access memory (DRAM) ICs. We anticipate that analogous techniques may be similarly relevant in protection against the Trojan horses. FPGA are ideally suited for BISR Trojan masking due to their reconfiguration capabilities. If there is a Trojan in FPGA hardware, one could place and configure the hardware in such a way that its impact is eliminated. If there is a Trojan in the bitstream, after its detection, characterization, and removal, one can quickly create a new Trojan-free system.

Recently, several powerful aging techniques were proposed for creation of PUFs. Aging techniques provide numerous advantages over process variation-based PUFs including enabling that user herself creates her own PUFs, much higher entropy, prevention of precomputation, and much longer lifetimes in presence of intentional and nonintentional device aging. Currently, only transistor aging is considered, but we can expect that other types of aging including electromigration will be soon pursued.

9.5.16 Executive Summary

It is difficult to consistently and in a uniform way analyze the advantages and limitations of the three principal implementation platforms (ASIC, FPGA, and programmable processors). The various security features have drastically differing requirements. Still, there is a reasonable arguments that FPGAs may emerge as a

security platform of choice due to their desirable features including flexibility and post-silicon realization of functionality. While the development of new and practical hardware-based security techniques is still in very early phases, it may result in new and revolutionary ways for both system and data security. In the meantime, support for realization of classical algorithmic protocols and DRM issues will be of primary importance.

9.6 Conclusions

We have surveyed a selection of the most important issues related to FPGA security. Specifically, we placed emphasize on security primitively (PUFs and TRNGs), analysis of potential vulnerabilities of FPGA synthesis flow, digital rights management, and FPGA-based applied algorithmic cryptography. We also analyzed the most challenging and beneficial research and development techniques related to FPGA and FPGA-based security platforms. While, of course, it is very risky to publicly state firm predictions, we expect that the system and hardware-based security of and by FPGAs is bound to emerge as a premier research and development direction.

References

1. Drimer S (2008) Volatile FPGA design security – a survey (v0.96). http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf. Accessed April 2008
2. Chen D, Cong J, Pan P (2006) FPGA design automation: a survey. Found Trends Electron Design Automation 1: 139–169
3. Qu G, Potkonjak M (2003) Intellectual Property Protection in VLSI Design. Springer, Berlin, Heidelberg, New York
4. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. IEEE Design Test Comput 27(1): 10–25
5. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: identifying and classifying hardware trojans. IEEE Comput 43(10): 39–46
6. Trimberger S (2007) Trusted design in FPGAs. In: Design Automation Conference (DAC), pp 5–8
7. Hori Y, Satoh A, Sakane H, Toda K (2008) Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems. In: Field Programmable Logic and Applications (FPL), September 2008, pp 23–28
8. Oliveira A (2001) Techniques for the creation of digital watermarks in sequential circuit designs. IEEE Trans Comput Aided Design Integr Circuits Syst 20(9): 1101 –1117
9. Lach J, Mangione-Smith WH, Potkonjak M (1998) Fingerprinting digital circuits on programmable hardware. In: Information Hiding (IH), pp 16–31
10. Lach J, Smith WHM, Potkonjak M (2001) Fingerprinting techniques for field-programmable gate array intellectual property protection. IEEE Trans Comput Aided Design Integr Circuits Syst 20(10): 1253–1261
11. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. In: Information Hiding (IH), pp 81–95

12. Dabiri F, Potkonjak M (2009) Hardware aging-based software metering. In: Design, Automation and Test in Europe (DATE), pp 460–465
13. Beckmann N, Potkonjak M (2009) Hardware-based public-key cryptography with public physically unclonable functions. In: Information Hiding. Springer, Berlin, Heidelberg, New York, pp 206–220
14. Defense science board (DSB) study on high performance microchip supply. http://www.acq.osd.mil/dsb/reports/2005-02-hpms_report_final.pdf.
15. D. A. R. P. A. D. M. T. O. (2007) (MTO), TRUST in ICs.
16. Trimberger SM, Conn RO (2007) Remote field upgrading of programmable logic device configuration data via adapter connected to target memory socket, United States Patent Office. http://patft1.uspto.gov/netacgi/nph-Parser?patentnumber=7269724. Accessed September 2007
17. 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)
18. Altera Corporation vs. Clear Logic Incorporated (D.C. No. CV-99-21134) (2005). United States court of appeals for the ninth circuit. http://www.svmedialaw.com/altera. Accessed April 2005
19. Court issues preliminary injunction against CLEAR LOGIC in ALTERA litigation, Altera Corp. (2002) http://www.altera.com/corporate/news_room/releases/releases_archive/2002/corporate/nr-clearlogic.html. Accessed July 2002
20. Gutmann P (1996) Secure deletion of data from magnetic and solid-state memory. In: USENIX Workshop on Smartcard Technology, July 1996, pp 77–89
21. ——, (2001) Data remanence in semiconductor devices. In: USENIX Security Symposium. August 2001, pp 39–54
22. Skorobogatov SP (2002) Low temperature data remanence in static RAM. University of Cambridge, Computer Laboratory, Tech. Rep. 536, June 2002
23. Rodriquez-Henriquez F, Saqib N, Diaz-Perez A, Koc C (2007) Cryptographic Algorithms on Reconfigurable Hardware. Springer, Berlin, Heidelberg, New York
24. Kim NS, Austin T, Blaauw D, Mudge T, Flautner K, Hu JS, Irwin MJ, Kandemir M, Narayanan V (2003) Leakage current: Moore's law meets static power. IEEE Comput 36(12): 68–75
25. Standaert F-X, van Oldeneel tot Oldenzeel L, Samyde D, Quisquater J-J (2003) Differential power analysis of FPGAs: how practical is the attack? Field Programmable Logic and Applications (FPL). Springer-Verlag, Berlin, Heidelberg, New York, pp 701–709
26. Shang L, Kaviani AS, Bathala K (2002) Dynamic power consumption in Virtex-II FPGA family. In: Field Programmable Gate Arrays Symposium (FPGA), pp 157–164
27. Mangard S, Oswald E, Popp T (2007) Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer-Verlag, Secaucus, NJ, USA. http://www.dpabook.org/
28. Standaert F-X, Örs SB, Preneel B (2004) Power analysis of an FPGA implementation of textscRijndael: is pipelining a DPA countermeasure?. In: Cryptographic Hardware and Embedded Systems Workshop, ser. LNCS, vol 3156. Springer, Berlin, Heidelberg, New York, pp 30–44
29. Standaert F-X, Örs SB, Quisquater J-J, Preneel B (2004) Power analysis attacks against FPGA implementations of the DES. In: Field Programmable Logic and Applications (FPL). Springer-Verlag, Berlin, Heidelberg, New York, pp 84–94
30. Standaert F-X, Mace F, Peeters E, Quisquater J-J (2006) Updates on the security of FPGAs against power analysis attacks. In: Reconfigurable Computing: Architectures and Applications, ser. LNCS, vol 3985, pp 335–346
31. Standaert F-X, Peeters E, Rouvroy G, Quisquater J-J (2006) An overview of power analysis attacks against field programmable gate arrays. Proc IEEE 94(2): 383–394
32. Messerges TS (2000) Power analysis attack countermeasures and their weaknesses. In: Communications, Electromagnetics, Propagation and Signal Processing Workshop.

33. Mangard S (2004) Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In: Okamoto T (eds) RSA Conference, ser. LNCS, vol 2964. Springer, Berlin, Heidelberg, New York, pp 222–235
34. Kocher PC (1996) Timing attacks on implementations of DIFFIE-HELLMAN, RSA, DSS, and other systems. In: Cryptology Conference on Advances in Cryptology, ser. LNCS, vol 1109. Springer-Verlag, Berlin, Heidelberg, New York, pp 104–113
35. Dhem J-F, Koeune F, Leroux P-A, Mestré P, Quisquater J-J, Willems J-L (1998) A practical implementation of the timing attack. In: International Conference on Smart Card Research and Applications (CARDIS), pp 167–182
36. Quisquater J-J, Samyde D (2001) ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards. In: International Conference on Research in Smart Cards (E-SMART). Springer-Verlag, Berlin, Heidelberg, New York, pp 200–210
37. Agrawal D, Archambeault B, Rao JR, Rohatgi P (2002) The EM side-channel(s). In: Cryptographic Hardware and Embedded Systems Workshop (CHES), ser. LNCS, vol 2523. Springer-Verlag, Berlin, Heidelberg, New York, pp 29–45
38. Gandolfi K, Mourtel C, Olivier F (2001) Electromagnetic analysis: concrete results. In: Cryptographic Hardware and Embedded Systems Workshop (CHES), ser. LNCS, vol 2162. Springer-Verlag, Berlin, Heidelberg, New York, pp 251–261
39. Carlier V, Chabanne H, Dottax E, Pelletier H (2004) Electromagnetic side channels of an FPGA implementation of AES. Cryptology ePrint Archive, no. 145.
40. De Mulder E, Buysschaert P, Örs SB, Delmotte P, Preneel B, Vandenbosch G, Verbauwhede I (2005) Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In: International Conference on "Computer as a tool" (EUROCON), pp 1879–1882
41. Peeters E, Standaert F-X, Quisquater J-J (2007) Power and electromagnetic analysis: improved model, consequences and comparisons. VLSI J Integr 40: 52–60
42. Agrawal D, Archambeault B, Chari S, Rao JR, Rohatgi P (2003) Advances in Side-channel Cryptanalysis, Electromagnetic Analysis and Template Attacks. vol 6, no. 1, Springer, Berlin, Heidelberg, New York
43. Agrawal D, Rao JR, Rohatgi P (2003) Multi-channel attacks. In: Cryptographic Hardware and Embedded Systems Workshop, ser. LNCS, vol 2779, pp 2–16
44. Anderson RJ, Kuhn MG (1998) Low cost attacks on tamper resistant devices. In: International Workshop on Security Protocols. Springer-Verlag, Berlin, Heidelberg, New York, pp 125–136
45. Karnik T, Hazucha P, Patel J (2004) Characterization of soft errors caused by single event upsets in CMOS processes. IEEE Trans Dependable Secure Comput 1(2): 128–143
46. Lesea A, Drimer S, Fabula J, Carmichael C, Alfke P (2005) The ROSETTA experiment: atmospheric soft error rate testing in differing technology FPGAs. IEEE Trans Device Mater Reliabil 5,(3): 317–328
47. Fabula J, Moore J, Ware A (2007) Understanding neutron single-event phenomena in FPGAs. Military Embedded Systems
48. Skorobogatov SP (2005) Semi-invasive Attacks – A New Approach to Hardware Security Analysis, University of Cambridge, Computer Laboratory, Tech. Rep. 630, April 2005
49. Soden JM, Anderson RE, Henderson CL (1997) IC failure analysis: magic, mystery, and science. IEEE Design Test Comput 14(3): 59–69
50. Frohman-Bentchkowsky D (1971) A fully-decoded 2048-bit electrically-programmable MOS ROM. In: IEEE International Solid-State Circuits Conference (ISSCC), vol XIV, pp 80–81
51. Cuppens R, Hartgring C, Verwey J, Peek H, Vollebragt F, Devens E, Sens I (1985) An EEPROM for microprocessors and custom logic. IEEE J Solid-State Circuits 20(2): 603–608
52. Scheibe A, Krauss W (1980) A two-transistor SIMOS EAROM cell. IEEE J Solid-State Circuits 15(3): 353–357
53. Guterman D, Rimawi I, Chiu T, Halvorson R, McElroy D (1979) An electrically alterable nonvolatile memory cell using a floating-gate structure. IEEE Trans Electron Dev 26(4): 576–586

54. Carter W, Duong K, Freeman RH, Hsieh H, Ja JY, Mahoney JE, Ngo LT, Sze SL (1986) A user programmable reconfiguration gate array. In: IEEE Custom Integrated Circuits Conference (CICC), May 1986, pp 233–235
55. Birkner J, Chan A, Chua H, Chao A, Gordon K, Kleinman B, Kolze P, Wong R (1992) A very-high-speed field-programmable gate array using metal-to-metal antifuse programmable elements. Microelectron J 23(7): 561–568, http://www.sciencedirect.com/science/article/B6V44-4829XPB-7F/2/3e9f92c100b2ab2f2527c5f039547578
56. Hamdy E, McCollum J, Chen S, Chiang S, Eltoukhy S, Chang J, Speers T, Mohsen A (1988) Dielectric based antifuse for logic and memory ICs. In: International Electron Devices Meeting (IEDM), pp 786–789
57. Design security in nonvolatile flash and antifuse FPGAs. Actel FPGAs, Tech. Rep.
58. Gassend B, Clarke D, van Dijk M, Devadas S (2002) Silicon physical random functions. In: ACM Conference on Computer and Communications Security (CCS), pp 148–160
59. Lee J, Lim D, Gassend B, Suh G, van Dijk M, Devadas S (2004) A technique to build a secret key in integrated circuits for identification and authentication applications. In: Symposium on VLSI Circuits, pp 176–179
60. Morozov S, Maiti A, Schaumont P (2010) An Analysis of Delay Based PUF Implementations on FPGA. Springer, Berlin, Heidelberg, New York, p 382387
61. Majzoobi M, Koushanfar F, Potkonjak M (2009) Techniques for design and implementation of secure reconfigurable PUFs. ACM Trans Reconfig Technol Syst 2: 5:1–5:33
62. Majzoobi M, Koushanfar F, Devadas S (2010) FPGA PUF using programmable delay lines. In: IEEE Workshop on Information Forensics and Security (WIFS), in press
63. Majzoobi M, Elnably A, Koushanfar F (2010) FPGA time-bounded unclonable authentication. In: Information Hiding (IH), pp 1–16
64. Majzoobi M, Koushanfar F (2011) FPGA time-bounded authentication. IEEE Transactions on Information Forensics and Security, in press
65. Majzoobi M, Dyer E, Elnably A, Koushanfar F (2010) Rapid FPGA characterization using clock synthesis and signal sparsity. In: International Test Conference (ITC)
66. Suh G, Devadas S (2007) Physical unclonable functions for device authentication and secret key generation. In: Design Automation Conference (DAC), p 914
67. Yin C-E, Qu G (2010) LISA: maximizing RO PUF's secret extraction. In: Hardware-Oriented Security and Trust (HOST), pp 100–105
68. Qu G, Yin C-E (2009) Temperature-aware cooperative ring oscillator PUF. In: Hardware-Oriented Security and Trust (HOST), pp 36–42
69. Maiti A, Schaumont P (2010) Improved ring oscillator PUF: an FPGA-friendly secure primitive. J Cryptol 1–23
70. Maiti A, Casarona J, McHale L, Schaumont P (2010) A large scale characterization of RO-PUF. In: Hardware-Oriented Security and Trust (HOST), June 2010, pp 94–99
71. Morozov S, Maiti A, Schaumont P (2010) An analysis of delay based PUF implementations on FPGA. In: Sirisuk P, Morgan F, El-Ghazawi T, Amano H (eds) Reconfigurable Computing: Architectures, Tools and Applications, ser. Lecture Notes in Computer Science, vol. 5992, pp 382–387. Springer, Berlin, Heidelberg
72. Sunar B, Martin WJ, Stinson DR (2007) A provably secure true random number generator with built-in tolerance to active attacks. IEEE Trans Comput 58: 109–119
73. Odonnell CW, Suh GE, Devadas S (2004) PUF-based random number generation. In: In MIT CSAIL CSG Technical Memo 481 (http://csg.csail.mit.edu/pubs/memos/Memo-481/Memo-481.pdf, p 2004
74. Schellekens D, Preneel B, Verbauwhede I (2006) FPGA vendor agnostic true random number generator. In: Field Programmable Logic and Applications (FPL), pp 1–6
75. von Neumann J (1963) Various techniques used in connection with random digits. In: von Neumann Collected Works, vol 5, pp 768–770
76. Barak B, Shaltiel R, Tromer E (2003) True random number generators secure in a changing environment. In: Cryptographic Hardware and Embedded Systems workshop (CHES). Springer-Verlag, Berlin, Heidelberg, New York, pp 166–180

77. Jun B, Kocher P (1999) The Intel random number generator. In: CRYPTOGRAPHY RESEARCH, INC.
78. Marsaglia G (1996) DIEHARD: A battery of tests for randomness. http://stat.fsu.edu/~geo
79. NIST (2000) A Statistical Test Suite for Random and Pseudorandom Numbers. Special Publication
80. Menezes A, van Oorschot P, Vanstone S (1996) Handbook of Applied Cryptography. CRC Press, Boca Raton
81. Goldreich O (2001) Foundations of Cryptography, Volume 1: Basic Tools. Cambridge University Press, Cambridge
82. Schneier B (1996) Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, NY, USA
83. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inform Theory IT-22: 644–654
84. Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2): 120–126
85. Rührmair U, Chen Q, Stutzmann M, Lugli P, Schlichtmann U, Csaba G (2010) Towards electrical, integrated implementations of SIMPL systems. In: Workshop in Information Security Theory and Practice (WISTP), pp 277–292
86. Rührmair U (2011) SIMPL systems, or: Can we design cryptographic hardware without secret key information? In: SOFSEM, pp 26–45
87. Chen Q, Csaba G, Lugli P, Schlichtmann U, Stutzmann M, Rührmair U (2011) Circuit-based approaches to SIMPL systems. J Circuits Syst Comput 20: 107–123
88. Alkabani Y, Koushanfar F (2009) Consistency-based characterization for IC trojan detection. In: International Conference on Computer-Aided Design (ICCAD), pp 123–127
89. Koushanfar F, Mirhoseini A, Alkabani Y (2010) A unified submodular framework for multimodal IC trojan detection. In: Information Hiding (IH)
90. Koushanfar F, Mirhoseini A (2011) A unified framework for multimodal submodular integrated circuits trojan detection. In: IEEE Transactions on Information Forensic and Security
91. Wei S, Meguerdichian S, Potkonjak M (2010) Gate-level characterization: foundations and hardware security applications. In: ACM/IEEE Design Automation Conference (DAC), pp 222–227
92. Wei S, Potkonjak M (2010) Scalable segmentation-based malicious circuitry detection and diagnosis. In: International Conference on Computer Aided Design (ICCAD), pp 483–486
93. ——, (2011) Integrated circuit security techniques using variable supply voltage. In: ACM/IEEE Design Automation Conference (DAC), to appear
94. Kahng AB, Lach J, Mangione-Smith WH, Mantik S, Markov I, Potkonjak M, Tucker P, Wang H, Wolfe G (1998) Watermarking techniques for intellectual property protection. In: ACM/IEEE Design Automation Conference (DAC), pp 776–781
95. Kahng AB, Mantik S, Markov I, Potkonjak M, Tucker P, Wang H, Wolfe G (1998) Robust IP watermarking methodologies for physical design. In: ACM/IEEE Design Automation Conference (DAC), pp 782–787
96. Hong I, Potkonjak M (1998) Technique for intellectual property protection of DSP designs. In: International Conference on Acoustic, Speech, and Signal Processing (ICASSP), pp 3133–3136
97. Koushanfar F, Hong I, Potkonjak M (2005) Behavioral synthesis techniques for intellectual property protection. ACM Trans Design Automation Electron Syst (TODAES) 10(3): 523–545
98. Lach J, Mangione-Smith W, Potkonjak M (2000) Enhanced FPGA reliability through efficient runtime fault recovery. IEEE Trans Reliabil 49(49): 296–304
99. Kahng AB, Mantik S, Markov IL, Potkonjak M, Tucker P, Wang H, Wolfe G (2001) Constraint-based watermarking techniques for design IP protection. IEEE Trans Comput Aided Design Integr Circuits Syst 20(10): 1236–1252

100. Kirovski D, Potkonjak M (2003) Local watermarks: methodology and application to behavioral synthesis. IEEE Trans Comput Aided Design Integr Circuits Syst 22(9): 1277–1284
101. 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
102. Ziener D, Assmus S, Teich J (2006) Identifying FPGA IP-cores based on lookup table content analysis. In: International Conference on Field Programmable Logic and Applications (FPL), pp 1–6
103. 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
104. 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
105. Koushanfar F, Qu G, Potkonjak M (2001) Intellectual property metering. In: International Workshop on Information Hiding (IHW). Springer, Berlin, Heidelberg, New York, pp 81–95
106. Koushanfar F, Qu G (2001) Hardware metering. In: Design Automation Conference (DAC), pp 490–493
107. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX Security Symposium, pp 291–306
108. Koushanfar F (2011) Active integrated circuits metering techniques for piracy avoidance and digital rights management, ECE Department, Rice University, Tech. Rep. TREE1101
109. Potkonjak M (2010) Synthesis of trustable ICs using untrusted CAD tools. In: ACM/IEEE Design Automation Conference (DAC), pp 633–634
110. Wong J, Kirovski D, Potkonjak M (2004) Computational forensic techniques for intellectual property protection. IEEE Trans Comput Aided Design Integr Circuits Syst 23(6): 987–994
111. Kirovski D, Liu D, Wong J, Potkonjak M (2000) Forensic engineering techniques for VLSI CAD tools. In: IEEE/ACM Design Automation Conference (DAC), pp 580–586
112. Alkabani Y, Koushanfar F (2008) N-variant IC design: methodology and applications. In: Design Automation Conference (DAC), pp 546–551
113. Alkabani Y, Koushanfar F, Potkonjak M (2009) N-version temperature-aware scheduling and binding. In: International Symposium on Low Power Electronics and Design (ISLPED), pp 331–334
114. Majzoobi M, Koushanfar F (2011) Post-silicon resource binding customization for low power. ACM Transactions on Design Automation of Electronic Systems (TODAES), to appear
115. Nelson M, Nahapetian A, Koushanfar F, Potkonjak M (2009) SVD-based ghost circuitry detection. In: Information Hiding (IH), pp 221–234
116. Potkonjak M, Nahapetian A, Nelson M, Massey T (2009) Hardware trojan horse detection using gate-level characterization. In: ACM/IEEE Design Automation Conference (DAC), pp 688–693
117. Potkonjak M, Meguerdichian S, Nahapetian A, Wei S (2011) Differential public physically unclonable functions: architecture and applications. In: ACM/IEEE Design Automation Conference (DAC), to appear
118. Vahdatpour A, Potkonjak M, Meguerdichian S (2010) A gate level sensor network for integrated circuits temperature monitoring. In: IEEE Sensors, pp 1–4
119. Potkonjak M, Meguerdichian S, Wong J (2010) Trusted sensors and remote sensing. In: IEEE Sensors, pp 1–4

Chapter 10
Security in Embedded Systems

Yunsi Fei and Juan Carlos Martinez Santos

10.1 Introduction

As networking connections become pervasive for computer systems and embedded software contents increase dramatically, it becomes more convenient for hostile parties to utilize software vulnerability to attack embedded systems, such as personal digital assistants (PDAs), cell phones, networked sensors, and automotive electronics [1]. The vulnerability of embedded systems carrying sensitive information to security attacks, ranging from common cybercrimes to terrorism, has become a very critical problem with far-reaching financial and social implications [2]. For example, security is still the largest concern that prevents the adoption of mobile commence and secure messaging [3, 4]. In addition to the traditional metrics of performance, area, and power consumption, security has been regarded as one of the most important design goals for networked embedded systems [4]. Compared to the general purpose and commodity desktop system, an embedded system presents advantages in allowing deployment of meaningful countermeasures across system architecture design. Building a secure embedded system, however, is a complex task that requires multidisciplinary research across different system layers and spanning various design stages, including circuits, processors, Operating System (OS), compiler, system platform, etc. It is especially challenging to find efficient solutions granting system immunity to a broad range of evolving attacks,

Y. Fei (✉)
Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA
e-mail: feiyunsi@gmail.com

J.C. Martinez Santos
Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, USA

Currently on leave from Universidad Tecnologica de Bolivar, Cartagena, Colombia
e-mail: jsm07006@engr.uconn.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_10,

considering the stringent constraints of embedded systems on computing capability, memory, and battery power and the tamper-prone insecure environment.

Our research on security in embedded systems is to establish both effective and efficient architectural support in embedded processors for secure processing. Typical software security attacks include buffer overflows, fault injections, Trojan horses, and data and program integrity attacks [6–10]. These attacks exploit system vulnerabilities to allow malicious users to overwrite program code or data structures, leaking critical information or launching malicious code. Compared to the existing inefficient software solutions and inflexible hardware measures, our approaches enhance the processor security inherently. Processor-based mechanisms can be less susceptible to attacks since they can be made transparent to upper-level software attacks. They can also minimize error propagation and enable fast error isolation and robust recovery, as they can be carried out at a finer granularity without much execution time overhead. Our investigations of a security engine inside processors focus on two issues. First, how to integrate hardware augmentation and software support for dynamic information flow tracking to protect critical control information of programs, like return address, indirect jump address, and system call IDs, from being overwritten by malicious users? Second, how to leverage embedded speculative architectures to protect noncontrol data efficiently?

We next introduce our secure computing model and threat model, and then discuss the importance of data properties, followed by an overview of our approaches.

10.1.1 Secure Computing Model and Threat Model

Here, we consider embedded systems built around a single processor with external memory and peripherals. In our secure computing model, the processor itself is assumed to be secure and protected from physical attacks, like side-channel attacks which observe implementation properties such as power consumption and execution time to infer critical information. The trusted computing base (TCB) consists of the processor and the security kernel of a given operating system (OS). External memory and peripherals are assumed to be insecure, e.g., keyboard, disk, and network card, which may be tampered with by an adversary via physical attacks or software attacks by exploiting the weakness in software execution. A survey in [5] has reviewed various software attacks. The typical direct consequence of software security attacks is memory corruption, which can lead to malicious code injection, sensitive information leaking, user privilege escalation, etc.

We assume that an encryption and decryption mechanism has been implemented between the on-chip caches and off-chip memory bus, which prevents system-critical information from being retrieved, addressing the confidentiality issue [11, 12]. In this chapter, we will focus on protecting the data properties from software security attacks that can gain random write access to an arbitrary address in the program address space. We do not address the separate trusted platform module chip designed by the Trusted Computing Group [11, 13].

10.1.2 Protection of Program Data Properties

Malicious users mainly gain access to a system through either control data or noncontrol data attacks. Control data include the target addresses that can be loaded to the processor's program counter (PC) at run-time to change the flow of program execution, e.g., at the sites of procedure calls, returns, local jumps, and special nonlocal jumps (setjump/longjump), and system call IDs which may be altered by attackers to break into the system for critical information or system crash. In current processor architectures, control flow transfers are blindfolded without validation. A quick survey of the CERT/US-CERT security advisories [14, 15] and Microsoft Security Bulletin [16] shows that control data attacks are the most dominant security threats today. Recent analysis has also demonstrated that the emerging attacks on noncontrol data [17–20], which may gain the privilege of the victim process and result in security compromises, are as severe as traditional control-data attacks. Security-critical noncontrol data includes configuration data, user identity data, user input data, and decision-making data [17]. For example, decision-making data usually is just a Boolean variable in a register or memory location to decide the direction of a branch instruction, which can be attacked to change the program execution or escalate the user privilege.

10.1.3 Hardware/Software Approach for Secure Processing in Embedded Systems

We will take a hierarchical hardware/software approach to enforce secure processing in embedded systems. At the low-level of instruction execution, we will employ a novel and efficient dynamic information flow tracking (DIFT) mechanism to protect critical control data [72]. DIFT is to tag (taint) memory data from untrusted sources, such as I/O, networks, and keyboards, and track its propagation through the system. If tainted data is used in a potentially unsafe manner, e.g., dereferencing a tagged pointer, a security exception is raised. Desired characteristics of the low-level security mechanism should include small area/power overhead, little performance overhead, few false positives or false negatives, and practicality (working with real-world code and software models, including program binaries, dynamically generated, and self-modifying code). We expect our approach to achieve all the goals aforementioned. Second, at a higher level of super block (a group of consecutive basic blocks where only the last basic block ends with an indirect branch instruction), we propose a novel approach to monitor control flow transfers and execution paths [73]. To enable fast validation, we leverage the existing on-chip branch target buffer (BTB) as a cache for the legitimate control flow in memory. The modification to the processor architecture is minor on top of the branch prediction structure and is transparent to the upper-level OS and programs.

10.2 Secure Page Allocation for Efficient Dynamic Information Flow Tracking: PIFT

We observe that one essential feature of memory corruption attacks is that they often use untrusted data illegitimately to hijack the normal program control flow. The vulnerabilities exist when the actual size of the user inputs is not checked, and the memory adjacent to the intended region for the input may be corrupted, or when the user inputs are not correctly filtered and string inputs are treated as statements or commands instead of regular data.

Dynamic information flow tracking (DIFT) approaches have been presented to differentiate sources of data (trusted or untrusted), track the data flow at run-time, and check the usage of different kinds of data. Each data is associated with a taint, which is usually a one-bit field that tags the data as trusted (0) or untrusted (1). Data taints are initialized according to their input sources – data from trusted sources, like local disks, starts out as trusted, and data from untrusted sources that can be utilized by malicious users, like networks and keyboards, starts out as untrusted. Taints are then propagated along the program execution and stored in memory and registers temporally. To detect attacks, critical control instruction sites are checked for use of tainted values, e.g., address for function returns, system call IDs, or control flow transfer target address. Using tainted value in these places is considered unsafe because they may allow the attacker to change the control flow of the application, like executing inserted code, etc.

The DIFT technique has been widely adopted and implemented in both software [21–23] and hardware [24–28]. Software-based approaches are flexible with taint propagation and checking policies. However, software approaches incur large code (memory) overhead and performance degradation and have difficulties in handling cases like self-modifying code, JIT compilation, and multithreaded applications [29]. Hardware-based approaches address these drawbacks and reduce the performance degradation, but require drastic redesign of the processor core for taint storage and processing (including propagation and checking), and they are limited by certain preset rules to avoid false alarms and cannot countermeasure new emerging attacks. Most of the hardware-assisted DIFT schemes couple the taint storage tightly with the data, e.g., extend the width of memory, cache, and register file to accommodate the extra taint bits. In some schemes, taints are stored in a dedicated memory region without changing the storage and communication bus. The granularity of data for a taint is normally at the byte level or word level.

We propose a flexible, efficient, and light-weight approach to perform DIFT based on secure page allocation, PIFT. Our approach differs from previous approaches in two ways. First, rather than tracking information (taints) flow at run-time, i.e., taint tag update of both registers and memory, our approach is a compile-time process that identifies taints of memory data and allows the compiler to allocate trusted/untrusted information into different memory pages. The run-time taint propagation is only for setting registers' taints. Second, instead of associating each data value with a taint bit (either tightly coupled with the data or mapped

into another location in a special array), we aggregate data according to their taints, i.e., putting trusted data in trusted memory pages, and untrusted data in untrusted memory pages. The whole page has only one taint bit stored in the page table. In this way, the memory (space) overhead is reduced significantly. As a result, our approach demands less OS support and little hardware augmentation for the DIFT taint processing.

10.2.1 Related Work

A lot of research work has been done for enforcing secure program execution through DIFT. Software approaches may involve compiler and OS augmentation. Run-time monitoring dynamically detects and halts invalid flow during program executions, and it may require architectural enhancement and corresponding OS support.

Software approaches allow much flexibility in policies for taint propagation and checking, taint bits, etc., and cover most of the known vulnerabilities [21–23, 30–32]. However, they suffer significant performance slowdown, e.g., 30% to 4x. In addition, they cannot track information flow inside binary libraries or system call functions [22, 32], and do not support multithreaded code [21, 23]. Another disadvantage is that there is extra program code needed to support dynamic flow tracking in software approaches. In GIFT [31], wrapper functions are added for initialization, propagation, and checking. To reduce the amount of information that must be tracked and also the size of extra code, [30] uses static analysis and a declarative annotation language to associate symbolic tags with data at run-time. In [31], a value range propagation algorithm is combined with taint analysis to help programmers apply the defensive programming style.

Some hardware-assisted approaches reduce the execution time overhead by introducing changes inside the processor core for taint storage and processing. The changes include widening the memory, register file, and buses, and introducing logic to initialize, propagate, and check the taints [24,33–36]. The taint initialization could be done by the Operating System [24, 35], or by new instructions [33]. The policies for propagation and checking could be set up at the beginning of the execution [24], or reconfigured at run-time [33]. Other approaches use a special memory region to store the taints [29, 34] instead of widening the memory and buses. They may introduce more latency for cache misses when the processor needs to retrieve the taint for data from the memory.

There also exists some hardware-software codesign work that utilizes some inherent processor architecture without changes to the processor or memory system. An approach presented in [37] leverages the speculative execution hardware support in modern commodity processors, like the *Itanium* processor, for DIFT. They utilize the *deferred exceptions* mechanism, using the exception token bit extended on each general purpose register for the taint bit. However, they have to set up a bitmap for data memory for their taints. They use software-assigned security

Table 10.1 Comparison between different DIFT techniques

DIFT steps	Software	Hardware-assisted	Our PIFT
Initialization	SW (application extension/compiler)	OS support	Compiler
Propagation	Application extension	Extra HW logic	Extra HW logic
Storage	Dedicated memory region	Widened memory or dedicated region	Page table
Checking	Application extension	Extra HW logic	Extra HW logic

policies and software techniques for taint propagation between the processor and the memory system. Moreover, the applications require recompilation, and the approach does not support multithread applications. In [38], the logic needed for doing DIFT is decoupling onto a simple and separated coprocessor. The application runs without any modification but with a small performance degradation due to the synchronization between cores.

To reduce the performance degradation of DIFT process, one recent approach is to separately run the normal application and the DIFT mechanism on different cores utilizing multi core architecture [34, 39–41].

Table 10.1 summarizes how DIFT is implemented in different techniques and what kind of supports are needed for each one. Our approach is a hardware-software codesign solution. Basically, it needs minimal hardware support for taint propagation and checking, but taint initialization and allocation is done at compile-time. In addition, the overhead for memory taint storage is very small compared to both existing hardware and software approaches because of the coarse granularity at the page level. We believe our approach is an effective one that integrates all the three parts, compiler (without reprogramming), OS support, and hardware changes, in a way that the advantages of both software and hardware-assisted approaches can be obtained.

In summary, our contributions include:

1. We propose a hardware-software codesign solution to perform DIFT based on compile-time static taint analysis and secure page allocation with minimal hardware changes.
2. By aggregating data according to their taints and using only one taint bit for each page, we can reduce the memory overhead significantly.
3. We demonstrate that our approach can reach the same security level as hardware-assisted approaches, i.e., addressing self-modifying code, JIT compilation, third-party libraries, as long as the attributes (trusted or untrusted) of the sources of these code are known a prior.
4. Different from software approaches, our approach only involves system software, including novel compilation passes and OS support, without reprogramming the applications. Therefore, the performance degradation of application execution is very small.

10.2.2 Our PIFT Approach

As analyzed before, the current hardware-assisted DIFT approaches have large area overhead, e.g., an overhead of 3.125% in off-chip memory, on-chip storage including the cache and register file, and bus for tagging at the word (4-byte) level [33, 42], or incur great execution overhead by accessing the dedicated memory for taints in addition to normal data [29, 34]. We propose a novel approach, Page-level Information Flow Tracking, to taint the memory system at the granularity of page, i.e., normally at the size of 4 KB, and allocate data to memories according to their attributes, *trusted* (TR) or *untrusted* (UTR), at compile-time.

10.2.2.1 General Idea

The taint attributes of data are obtained from a static taint analysis of the source code. The compiler statically divides all the data into two categories: *trusted* (TR) and *untrusted*(UTR) according to their sources. At program load time, the *loader* (one OS service) allocates them to trusted and untrusted memory pages and initializes the page tag. At execution time, when a data is accessed, the *memory controller* (another OS service) uses the address to retrieve the taint value. The OS allocates dynamic pages also according to their taints. Only one taint bit is needed for a page, which takes one unused bit in the page table. Hence, overhead of our approach in taint storage is zero. We do not need to widen the buses or access the memory twice, one for data and one for tag if they are stored separately. Because the taint processing is done in parallel with the normal instruction execution by the enhanced processor, we can expect little effect on the performance as well.

By default, the text segment, containing instructions, is trusted (TR). The data segment, the stack, and the heap are divided into two types of regions, according to the data that they will keep. Trusted/untrusted data will be allocated into trusted/untrusted pages accordingly. Figure 10.1 shows the mapping of the virtual address space to physical memory, which is managed by the page table where a tag is set for each page.

To propagate taints between the off-chip memory and register file inside a processor, we have to augment the processor architecture slightly. The register file is widened by one bit to hold the taint. We also include logic for taint propagation. Many propagation rules, e.g., those in [33], can be employed and selected through a configuration register. In this work, we adopt the OR-rule on all the involved operands. Different from other DIFT approaches, our system checks abnormal conditions during program taint propagation, for example, the case of untrusted information flowing into a trusted memory location. This kind of situation indicates possible memory corruption attacks, and the processor will raise an exception which allows OS or other service routine to examine the situation further or terminate the application directly. Our approach does not need to check specific control sites at run-time, e.g., function return and system calls. By allocating the

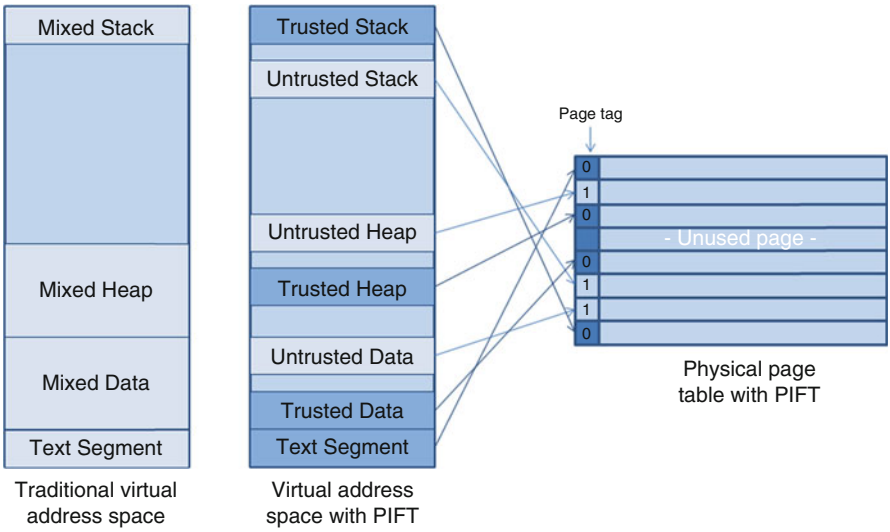


Fig. 10.1 Virtual address space and page table with PIFT

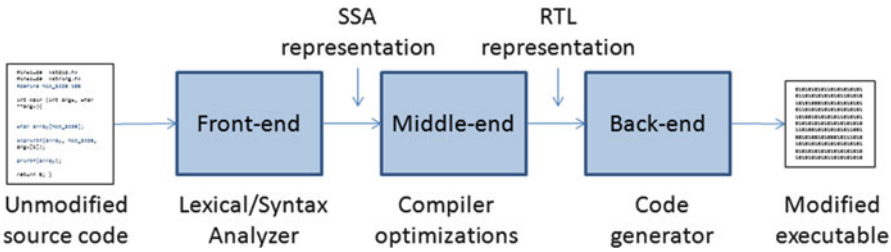


Fig. 10.2 Code transformations at compile-time

critical information, like the return address and system call IDs, into trusted pages and enforcing the taint checking policy, the critical data in memory is protected systematically. However, our approach still needs to check jump target, as they are in registers and the taints are updated at run-time.

10.2.2.2 Compilation Stage

In this section, we discuss several salient design steps in PIFT at compile-time for static taint analysis and memory map generation, which includes data attribute assignment, instruction duplication, and data duplication.

Figure 10.2 shows the basic code transformations done by a modified compiler. Our approach does not change the source code; instead, it changes the compiler front-end so that annotations are added to each variable of the program in the

Static Single Assignment (SSA) representation. These annotations will contain the taint information for each variable and will be used by newly added passes in the middle-end to do static taint analysis. The taint information is held during the complier optimization passes. It is later passed on to the back-end to generate the memory map.

When identifying the data sources and consequently the initial taint values, our approach also does annotations for critical C standard library functions which are found vulnerable to memory corruption attacks [43]. Although the vulnerable function is given as a black box, each time it is invoked, the compiler checks how the taints of the arguments affect the taints of the return results (if available). For example, function *malloc*(*size*) returns a pointer to a newly allocated block *size* bytes long, so the attribute of the returned pointer depends on the taint of variable *size*. In this way, although the source code of the functions is unknown, the compiler assigns a taint to the results.

Memory Map and Data Attributes

Normally, the memory space is classified into four segments: text, data, heap, and stack. We handle each segment specifically to support dynamic information flow tracking, adhering to certain overwriting policy that we set for memory integrity protection. In this work, we adopt a policy that data is not allowed to be exchanged between trusted and untrusted memory pages. We show later in Sect. 10.2.3 that by enforcing this policy, various memory corruption attacks are caught by our PIFT.

By default, the text or code segment is trusted. Therefore, it is allocated on memory pages whose taints are trusted. In general, this segment is read-only for program instruction fetching. In the case of self-modifying applications, the text segment could be overwritten and the new generated code must be from trusted sources. In the case that a piece of code is invoked from another section (i.e., the stack segment), a similar policy is applied, i.e., the code must come from trusted pages.

The data segment, also called Block-Started-by-Symbol (BSS) segment, normally contains constants and global variables used by the program. The constants are tagged as trusted and allocated on trusted memory pages. In contrast, although global variables may be initialized to some values, the values may change at execution time, and their attributes could be either trusted or untrusted at different times. If the static taint analysis shows that a global variable can have both trusted and untrusted attributes, our approach duplicates the global variable, one copy for trusted uses and the other for untrusted uses.

Our approach implements intra- and inter-procedural data-flow analysis at the middle-end to get a list of variables and their attributes, and then identify which global variables need to be declared twice. This duplication results in data overhead and slight code size increase.

Heap segment is a memory section dynamically allocated to store temporary information. If the information comes from a trusted source, a trusted page (or pages) will be allocated. Otherwise, untrusted pages will be allocated. The taint

is associated with the pages until the pages are released. At compile-time, the *linker* (the last step in the *back-end*) assigns the right attribute to each memory section, so the OS can allocate these heaps on different pages.

In addition to dynamic data, the heap also holds critical meta-data used by the Memory Management Unit (MMU) to handle the memory chunks. Attackers can exploit programming errors to overwrite meta-data, like forward and backward pointers which link to available memory chunks, and such overwrites can change the execution flow. If the MMU stores these critical pointers in a trusted page, our approach can avoid the possibility that they are corrupted by a heap overflow or double free. In [44], it shows that by modifying the memory allocation manager and using a lookup table, this chunk of information can be stored in a different memory region. We propose to hold the link list in trusted pages instead of using *guard pages* between memory chunks as employed in [44].

The last segment, the stack, requires special considerations. At the beginning of back-end code generation, some variables are assigned to the stack, and some are mapped onto registers. The stack variables could be function arguments, local variables, and special registers. Hence, the stack segment can hold both trusted and untrusted information. For example, frame pointer and return address are trusted. The attribute of function arguments and local variables depends on the functions. In order to separate trusted data from untrusted data, the compiler modifies the way each variable is allocated in the stack. Trusted variables are allocated without modification; however, untrusted variables are allocated with an extra offset. The size of the offset is large enough to avoid stack overlapping, and should be page-aligned. These modifications help to protect critical data, including return address, system call arguments, and function call pointers. The idea of multiple stacks has been presented in the previous work [45], where data are placed on different stacks according to their types, like array, integer, etc.

Code Duplication

In the cases where a variable must be duplicated, the compiler also has to generate a new statement to use the duplicated variable. Due to statement duplication, the size of the code can increase a little bit.

There is an issue with function calls because the arguments' attributes may be trusted or untrusted at different sites. Figure 10.3 shows an example. If the attributes for the function arguments do not change during execution, the program runs without any overhead. However, if the argument attributes change from *MAIN* to *LOOP*, the compiler changes the way the *function A* is being called. When an argument has different attribute, as shown in Fig. 10.3 for *var* 1 and *var* 3, the function call needs to be duplicated. Although the statement duplication suggests an exponential increase of the code, our static taint analysis shows that function duplication is a rare condition.

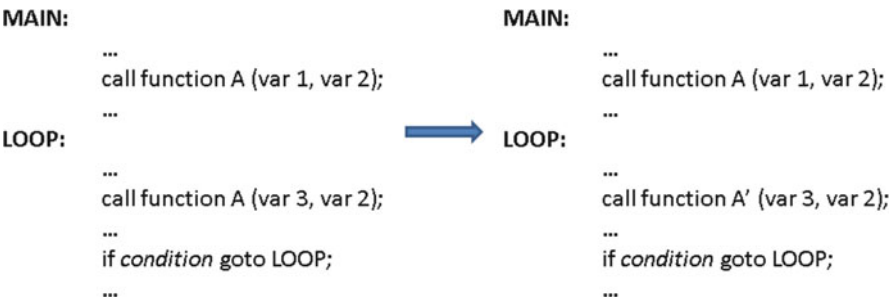


Fig. 10.3 Function argument at execution time

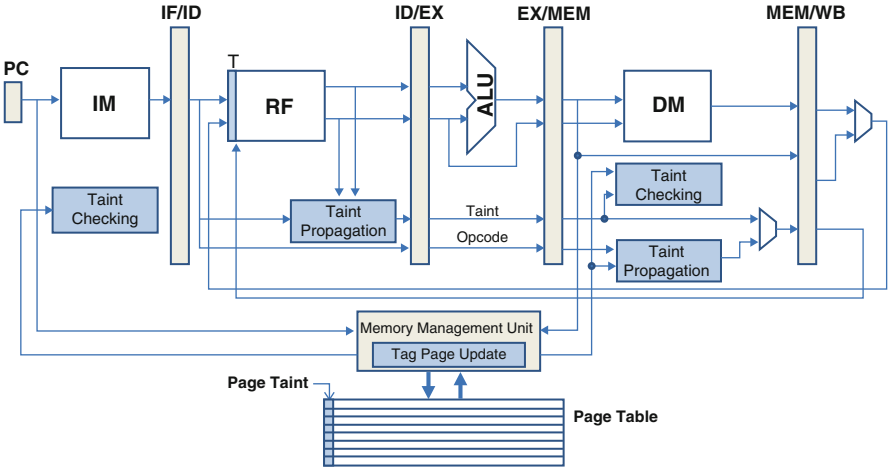


Fig. 10.4 Architecture design of PIFT

10.2.2.3 Run-time Execution with Architectural Augmentation

In the earlier sections, we explain how the compiler sets the memory attributes, and how the variables are aggregated to different kind of pages. In this section, we also show how the OS, especially the memory controller, should handle each segment of the memory map at run-time. We describe what changes are needed inside the processor to support taint propagation and checking for PIFT to detect security attacks on the application.

Figure 10.4 shows the architecture design of PIFT. Our approach allows the compiler to aggregate data based on their taints. At run-time, when the OS allocates memory for the current process, it initializes the page taint accordingly in the *Page Table* (in memory). Each time the processor fetches an instruction from the memory, the *Memory Management Unit* (managed by the OS) retrieves the *Page Taint* from the *Page Table*. The taint for the instruction has to be trusted in order to prevent malicious code injection and execution. This is ensured by a *Taint Checking*

module at the IF stage. During instruction execution, the tag is propagated inside the processor by the *Taint Propagation* module. There are two locations for such modules. One is at the ID stage where the taints of the source operand registers are known and the current instruction is decoded. The other is at the MEM stage where the taint of memory data (e.g., for LOAD instruction) is retrieved from the page table. For taint checking, the *Memory Management Unit* module ensures that when data is being written to the memory (in the MEM stage), the combination of taints for the sources and destination is allowed by the overwriting policy. In addition, the *Taint Checking* module checks that the jump target address is always trusted. Otherwise, an exception is raised.

Details about architectural changes are given later.

1. *Wider Register File:* Each register is widened by one bit to hold the taint tag. For taint propagation, we include glue logic to read and write these taints. "OR" operations are performed on the source taints except several special cases. When the instruction has only one source operand, the taint is propagated directly without any operation. Another special case is when a destination register is cleared, like using the XOR instruction *xor* $r1$, $r1$, $r1$. In this case, the register becomes trusted and its taint is independent on the source operands' taints. In addition, when an instruction loads an immediate value (*movl* $*10, (%esp)*), the destination register is always trusted. All the special cases are all considered in the *Taint Propagation* module.
2. *Memory Taint Bit Retrieval:* During taint propagation between registers and memory, the taint associated with a variable (in fact, with the memory page that holds the variable) should be retrieved on any LOAD or STORE instruction. At sites of LOAD instructions, when the memory address is used to look up the data memory/cache, it is used to look up the page table for the taint as well. At sites of STORE instructions, the taint for the memory location is retrieved in a similar manner, and the overwriting policy of no trusted/untrusted data to untrusted/trusted page is enforced with security alarms raised on violations. In systems that support virtual memory and page tables, there already exists a page table look-up mechanism whenever the instruction directs to access the memory. Therefore, there is no extra overhead for retrieving the taint bit stored in the page table. The system supports dynamically generated code, like self-modifying code, JIT compilation, etc., if and only if the generated code is from trusted sources.

10.2.3 Security Analysis and Attack Detection

There are two considerations in evaluating the effectiveness of our approach. First, the approach should not produce false alarms when no attack is launched upon the system. Second, the approach should be able to detect attacks that attempt to overwrite critical information.

We run the benchmarks from SPEC CINT2000 [46]. For each application, the source code is compiled twice, one with a regular compiler and the other with PIFT's compiler. The results obtained are the same and the applications run without false alarms. Details about how the PIFT's compiler affects the execution are given in Sect. 10.2.4. To evaluate the security effectiveness of our approach, we use three micro-benchmarks for DIFT [26] to test if our system can detect overwrites of critical memory positions by untrusted information. To simulate dynamic information flow tracking at run-time, we used a modified version of *tracegring*, which is a plug-in developed by Avalanche project [47] for Valgrind [48], an instrumentation framework to track the flow of tainted data.

Our PIFT can capture all the security attacks. Under stack buffer overflow attacks on an X86 architecture, the attack is detected when a chunk of data is copied to a memory region, with different taint attributes of the *Source Index Register* (ESI) and the *Destination Index Register* (EDI) used by a system call. In format string attacks, the taint of the format directive register gets to propagate to a register used by a conditional jump in the *printf* function, which our checking policy does not allow. In heap corruption attack, at run-time the OS determines the exact position of memory allocation. When a new chunk of memory is needed, the MMU needs to know what attribute the data will have, and the data can be allocated in the right page. The untrusted array is put in an untrusted page without the possibility of overwriting the critical meta-data in the trusted pages.

For system calls and *return-to-libc* attacks, our approach is also effective because the system ensures that the IDs for system calls are always stored in a trusted page. In addition, for high-level semantic software attacks, our approach is effective as well because the system ensures that only code from a trusted page can be executed.

Our approach can detect the attacks that intend to take control of the application by overwriting critical information like return address, system call IDs, and function pointers. Although the attacker can still overwrite some memory positions, no critical information will be changed due to different page allocation and security policies. Our approach can be complemented with other techniques that watch the execution of a program to avoid incorrect results, information disclosure, or application crashing due to the corruption of untrusted memory.

10.2.4 Experimental Results

In this section, we will perform experiments to evaluate the effect of our approach on performance, storage, code size, and data size.

10.2.4.1 Implementation

Our implementation consists of three parts in compiler: static taint analysis, stack duplication, and global variable separation. PIFT is built on top of Vulncheck [49],

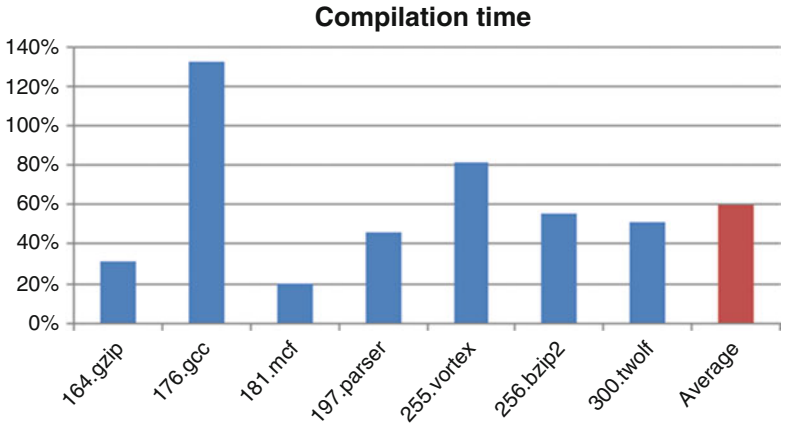


Fig. 10.5 Compilation time overhead with PIFT

an extension of GCC 4.2.1 for detecting code vulnerabilities. The taint analysis and initialization process includes the intra- and the inter-procedural analysis. The intra-procedural analysis builds the taint calculation formulas for the function return values and some output global variables depending on the taints of function call arguments and some input global variables. These formulas are used by the inter-procedural analysis to complete the static taint analysis and duplicate variables where needed. The whole analysis process is done when the code is output in GIMPLE representation, a tree based intermediate language. This representation is then transformed into an RTL representation in the compiler middle-end. At the back-end, the concept of stack first appears and certain stacks get duplicated based on the attributes collected previously in the middle-end.

Finally, to ensure that trusted and untrusted global variables are in different pages, we slightly modified the GNU linker (the last step of the compiler) to rearrange untrusted global data section one page beyond the trusted global data section. We compared our modified compiler with the original GCC 4.2.1. Figure 10.5 shows that the compilation time overhead is on average 60%. We found the time overhead is related with the size of the source code; big programs, like 176.*gcc*, take more time than small ones.

10.2.4.2 Static Memory Overhead

There are two sources of static memory overhead: global variable and statement duplication. Figure 10.6 shows that the overhead for global variables duplication is 6% on average. A small variable duplication results in a small statement duplication, as shown in Fig. 10.7. The average code overhead is less than 1%. Another reason for the small size overhead is that the compiler adds padding to align blocks, and new statements can utilize this padding.

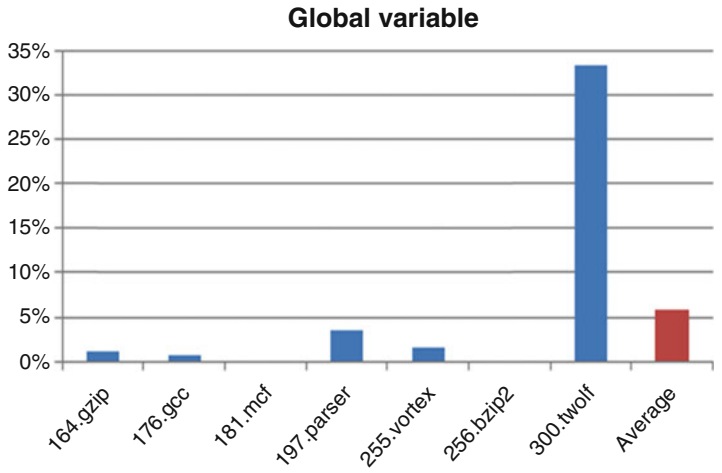


Fig. 10.6 Global variable overhead with PIFT

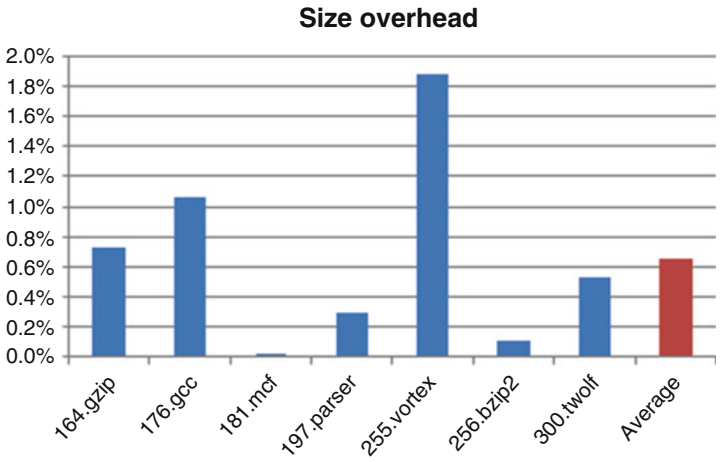


Fig. 10.7 Code size overhead with PIFT

10.2.4.3 Dynamic Memory Overhead: Heap and Stack

The static analysis reveals that no program requires heap duplication. However, that is not the case for stack. Using *drd*, a Valgrind tool for monitoring process behavior [48], we measured how much each stack grows during the execution. Figure 10.8 shows the result. For some applications, the stack is just split into two orthogonal stacks (untrusted and trusted) and the overall effective size does not change, so that the overhead is zero, like 176.*gcc* and 197. *parser*. For some applications, majority of the stack need to be duplicated (e.g., local variables that

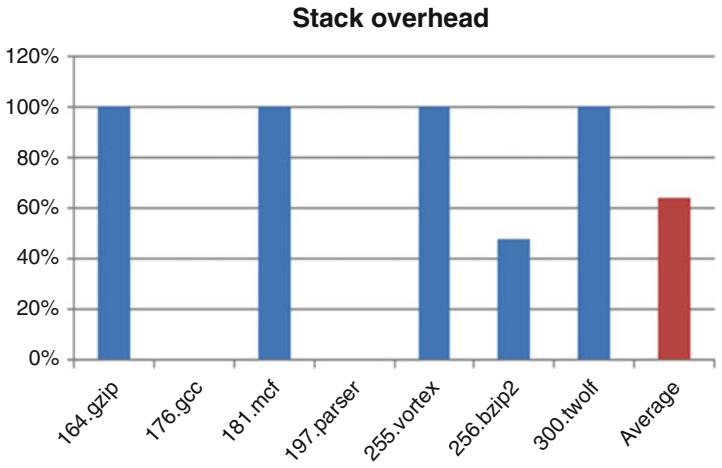


Fig. 10.8 Stack overhead with PIFT

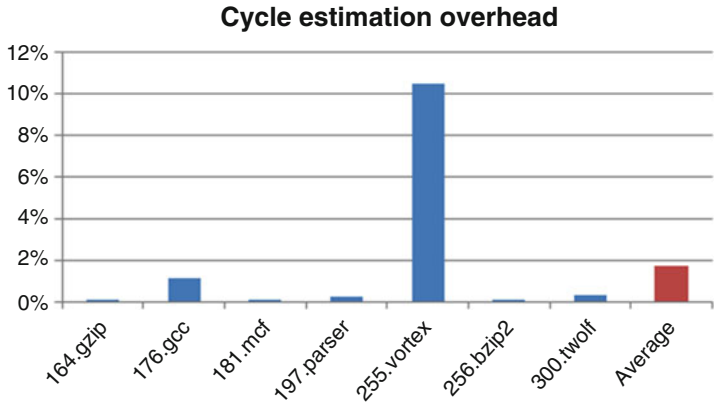


Fig. 10.9 Execution cycle overhead

are trusted and untrusted at different usages), and the overhead is near 100%, e.g., 164.*gzip*, 181.*mcf*, 255.*vortex*, and 300.*twolf*. For other applications, it is a mix of separation and duplication, and the overhead depends on the composites of stacks, e.g., 256.*bzip*2 (Fig. 10.8).

10.2.4.4 Execution Overhead

We test a set of SPEC CINT2000 applications running on Valgrind to measure the performance impact of our approach. Figure 10.9 shows the effect of our approach on the performance degradation, less than 2% on average, which is negligible.

The execution overhead is caused by cache misses. When data with different attributes are allocated into different pages, the original spatial locality is reduced, and hence there will be more capacity-limit cache misses [50]. At the same time, code size may get larger and average memory access time to code pages may increase as well.

10.2.5 Summary

In summary, we propose a flexible, efficient, and light-weight hardware/software codesign solution for Dynamic Information Flow Tracking with compile-time page allocation. We reduce the overhead for taint storage to almost zero without sacrificing security. There is some memory overhead for implementing the proposed page-level taint processing. Although our approach does not address issues related to decision-data attacks, we believe that our approach can be complemented with other signature-based solutions which protect applications based on their legitimate behavior or control flows.

10.3 Leveraging Speculative Architectures for Run-time Program Validation

Memory corruption attacks allow user data to overwrite program code or control-flow structure [51]. Hence, program execution can be directed to the malicious code that is implanted into the program's memory space, opening part or all of the system to the adversary. Control flow transfers have to be validated to defend against control-data attacks. Recently, some exploits have emerged that can change the direction of control instructions by overwriting local variables instead of changing the target address [18, 52, 53]. It is called decision data attack. Although all the control transfers are valid, the global control flow is compromised, and the attackers can extract important system information. This kind of attack is hard to detect by control flow transfer validation only. Therefore, we have to validate the branch decision path at run-time as well.

In this section, we introduce a novel approach of protecting program execution at the micro-architecture level. We propose to monitor control flow transfers and execution paths at the super block granularity, which is a portion of code between two consecutive indirect branch instructions. At each checking point (indirect branch site), dynamic program execution information is compared against a full record set (FRS) stored in a secure memory region, including legitimate target addresses and execution paths. In addition, behavior reference for future program execution is also prefetched for later monitoring. To enable fast validation, we consider introducing a cache architecture for the FRS in the memory. As branch

prediction schemes have been widely adopted in embedded processors, the on-chip branch target buffer (BTB) can be used as a perfect cache for the FRS. We propose a mechanism to ensure security for the BTB, i.e., avoiding any possible pollution from tampered memory. We find that the validation mechanism only needs to be activated for those indirect branches that are mis-predicted by the BTB, in terms of direction, address, or execution history. The rare access of the external FRS results in much less performance degradation than other hardware schemes. The modification to the processor architecture is minor on top of the branch prediction structure and is transparent to the upper operating system and programs. Thus, legacy code can run on the secure processor without recompilations.

Our approach falls into the category of general symptom-based mechanism to detect attacks on data. This kind of black-box approaches monitors program execution at run-time to detect any deviation from its legitimate behavior, thereby allowing us to ignore the complexities of various vulnerabilities and exploits and focus on protecting the critical objects, like control data, decision making data, and system call data. The salient issue is how to properly define the distinction between normal behavior and anomaly to improve the error detection accuracy.

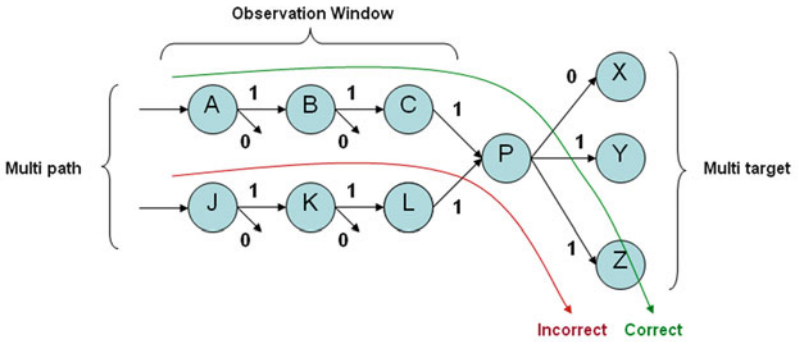
10.3.1 Preliminaries

To prevent control flow attacks, our micro-architecture level mechanism validates both control flow transfers and execution paths. The program execution monitoring is sampled at run-time at the sites of indirect branches, instead of every conditional branch or jump. Thus, the performance and storage overhead incurred by the monitoring may be reduced greatly. However, as the checking is carried out less frequently, the coverage of each checking process has to be adequate for the program to reduce the false negative rate of validation.

10.3.1.1 Complex Program Control Flow

There are several issues in program execution that complicate the validation process of control flow transfers and execution paths, e.g., multiple-path situations, as shown in an example control flow graph in Fig. 10.10, where each node represents a basic block, and each edge is a control flow transfer. Here basic block P ends with an indirect instruction that may go to different target addresses for taken branches. There are multiple paths of basic blocks that lead to block P. Without proper handling, they may result in ambiguities that will increase the false negative rate and degrade the effectiveness of the security countermeasures.

For direct conditional branches, a program control flow can be validated by using only binary decision information (where 1 represents branch taken and 0 branch untaken). However, when indirect branches are present, validation has to be performed against the address path, which requires a lot of memory storage.



	Address Path [32 x n bits]	Branch History [n bits]	Our Approach [(16+32+n) bits]
Correct Path	A-B-C-P	1-1-1	size-C-1-1-1
Incorrect Path	J-K-L-P	1-1-1	size-L-1-1-1

Fig. 10.10 Complex program control flow with ambiguity

An alternative solution is to look backward at the history of branch decisions at indirect branches, and use a binary decision path [53]. However, an attacker can take control of the application and change the normal control flow by exploiting the ambiguity of decision history representation. In Fig. 10.10, there are two paths that lead to basic block P which is the end of a super block; assume the path of A – B – C – P is the correct, and the other path of J – K – L – P is the infeasible. However, their binary history paths are the same: 1–1–1. If we just use the binary history path information, 1–1–1 will be stored in the signature and the incorrect path of J – K – L – P will escape checking. Therefore, the decision history path is insufficient to validate the execution. In addition to the binary path, the size of the observation window, i.e., length of the branch decision history, is important for alleviating ambiguity.

We propose a hybrid solution for the earlier situation: each history path has to be associated with something that provides enough information about the history, and at the same time, allows an observation window as large as possible. We introduce the past branch site address (PBPC) in the history. It allows us to differentiate the correct path from the malicious ones. Moreover, we add a new variable, the number of basic blocks in the super block. With the size of the super block, it is more difficult for the attack to camouflage. As shown in the last column of the table in Fig. 10.10, the correct path representation for basic block P associated with TPC of Z will be size-C-1–1–1, a hybrid of the direction history path, the last branch site address, and the size of the super block. We have modified the SimpleScalar toolset [54] to profile a set of MiBench applications [55] to find how common the ambiguities for binary paths are. The results are shown in Sect. 10.3.4.

In addition, the information on future expected paths (EPs) is obtained at the end of each super block, and will be used to check the decision of basic blocks in

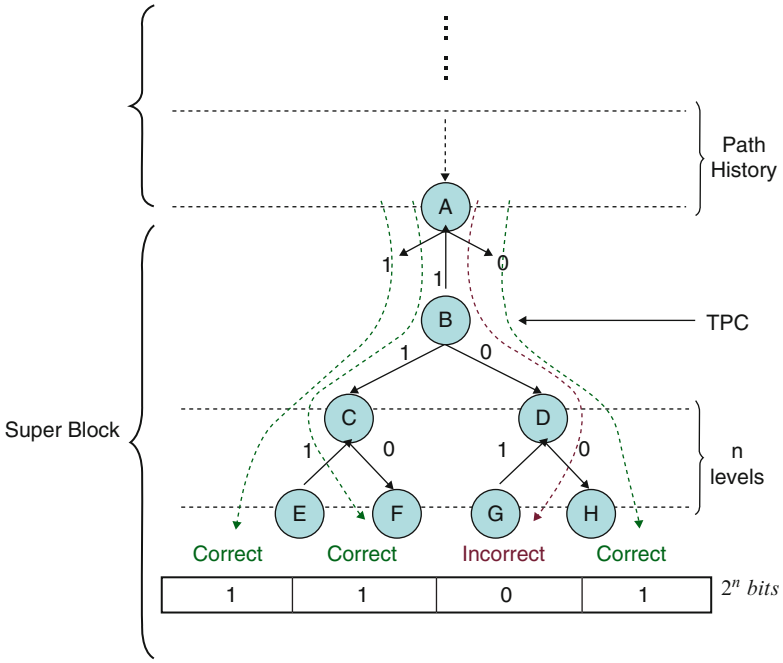


Fig. 10.11 Expected paths vector fetched at an indirect control instruction site

the next super block. Instead of storing all the possible binary paths, we can get an all-path-vector for the current target address (TPC) for a number of levels or until the program execution reaches the end of the super block. Figure 10.11 shows one example of an expected path vector for a depth of 2. Basic block A is the end of last super block. One of the taken branches of A goes to B, and B will be the root of a binary tree. Because all the basic blocks between B and next super block are direct conditional branches, their directions are sufficient for path validation. With a depth of 2, there are 4 paths in total: B – C – E (with a decision path of 11), B – C – F (10), B – D – G (01), and B – D – H (00), and we use a 4-bit vector to represent the validity of each path. For example, if path B – D – G (decision history path is 01) is invalid and all the other three paths are valid, the vector will be 1101. For depth n, the vector size is 2^n bits, where each bit represents the validity of its corresponding path, and the vector size is much smaller than the maximum size of $2^n \times n$ bits for recording all the possible n-level paths.

The vector is used during program execution to validate on the fly, making faster detection of attacks possible. Whenever a conditional branch is encountered and resolved, its direction decision (taken or untaken, i.e., 1 or 0) is used to cut the vector in half, with the higher half kept for decision 1 or the lower half for 0. If the vector reduces to 0, an invalid path has been executed and the anomaly is detected.

10.3.1.2 Related Work

There has been a lot of research done on using specialized software and hardware to detect specific attacks such as buffer overflows [56–59]. These techniques focus on the attack itself and can prevent attackers from exploiting the particular vulnerability. They fall in the category of white-box approaches, where the attack mechanism has been analyzed in detail and particular precautions are taken. However, due to the many sources of vulnerabilities, it is desirable to employ a more general symptom-based mechanism to detect any abnormal operations. This is a kind of black-box approach, where the program execution is monitored at run-time and any deviation from its legitimate behavior is detected, no matter what is the cause.

Researchers have shown that the program behavior can be modeled at different granularity levels, including system call, function call, control and data flow. Simply checking system calls is not sufficient and may cause high false negatives in some cases. In addition, storing them in a Finite State Automata (FSA) and checking against all normal system call traces is a tremendous design effort [60–62]. Some previous works have focused on increasing the reliability of the return address stack (RAS) to defeat the exploits at the function call level [63–65]. However, the control data for other non-RAS indirect branches can also possibly be tampered with by adversaries through contamination of memory. Other finer granularity levels have been proposed to detect abnormal program behavior. There exist some research works that detect control flow errors either by software or hardware support. The software-based approach rewrites the binary code with a checking mechanism in every basic block [66]. Although this technique is flexible, it requires binary translation and increases both the program size and execution cycles dramatically. Hardware support for control flow transfer validation is more efficient in performance, e.g., the hardware-assisted preemptive control flow checking [67]. However, the hardware cost is large because it is necessary to duplicate the entire register file and the PC for validation. Illegal indirect branch transfers may slip the checking mechanism. Another hardware-based approach mainly focuses on direct jumps and uses a sophisticated coprocessor for the complex control flow modeling and checking. For this approach, the storage overhead for reference behavior is also very large.

The most related previous work is the Bloom Filter-based run-time control flow validation [68]. They focus on reducing the storage size for legitimate control flow transfers and thus the hardware access latency. However, their Bloom Filter may introduce false negatives. In their follow-up work [53], validations are only performed at indirect branch sites with binary history path (branch direction). This approach may result in a high false negative rate because it cannot solve path ambiguities due to the binary representation. Our approach reduces the ambiguity by associating the binary history path with the last branch address and super block size. In addition, our mechanism considers branch correlation between adjacent super blocks by associating the history path with the future expected path. Thus, our approach has a higher anomaly detection rate than the previous work in [53]. By utilizing the existing branch target address prediction mechanism, our mechanism achieves negligible execution cycle overhead even with a long latency for accessing the FRS.

10.3.2 Speculative Architecture for Control Flow Transfer and Execution Path Validation

For every indirect control instruction, the system has to validate its legitimacy. As the full record set (FRS) is stored in a secure memory region, frequent off-chip full record access for validation will degrade the program performance greatly. To reduce the performance penalty, we can either reduce the latency for accessing the full record or decrease the number of accesses. Previous work has employed a Bloom Filter scheme to reduce the off-chip hardware access latency to four cycles [68]. In this work, we focus on reducing the access frequency by leveraging the speculative architecture of branch target buffer (BTB).

10.3.2.1 Extraction of Dynamic Program Execution Information for Validation

When validating program execution, the system needs to collect dynamic information on the history of conditional branch directions, the last branch address, and the size of the super block to compare against the normal behavior obtained from static-time analysis or training. We dedicate a small amount of memory to hold the branch sites of the current super block and three registers for dynamic information, which can only be accessed and used by the validation system. The branch history shift register (BHSR) [53] is a shift register that stores the decisions of the past n conditional instructions (assume the length of BHSR is n), and is updated after each direction is resolved. Another register, the past branch program counter (PBPC), records the branch address of the last basic block. The last register, for the size of the current super block, counts the number of basic blocks belonging to the current super block.

At each indirect instruction site, the branch site (BPC) is used to look up the expected behavior either in the BTB or in the FRS. If a BTB entry or an Indirect Control Signature (ICS) in the FRS (which consists of one branch site (BPC) as a tag, one target site (TPC), multiples history paths (HPs), the size of the super block, and the vector for the expected paths (EPV)) is found that matches the tuple of {BPC, TPC, BHSR, PBPC, SIZE}, the target address and history are valid. Otherwise, there is a mismatch, and an alarm is raised for the operating system to take control over the program. In the mean time, the expected path vector is fetched to check the following basic blocks along the execution, and an alarm may be raised at any time within the next superblock if the vector is reduced to 0.

10.3.2.2 BTB Update and Administration

Branch prediction mechanisms have been widely adopted for high-end embedded processors to alleviate pipeline stalls caused by conditional instructions. Predicting the branch direction while caching the branch target address has been the

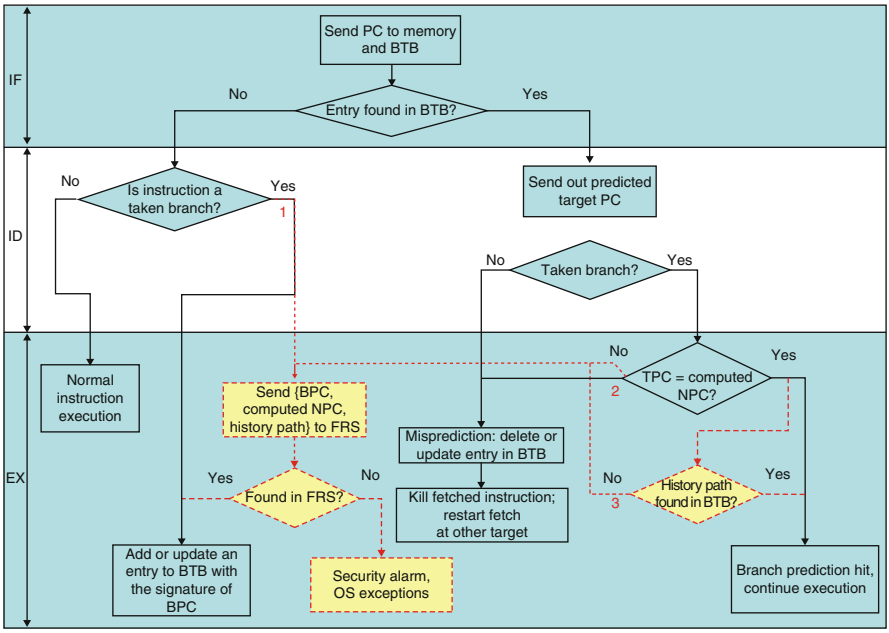


Fig. 10.12 Regular branch prediction flow enhanced with security features

fundamental objective in many designs of efficient pipelined architectures. The branch target address is provided by the BTB, which has a cache structure consisting of a number of sets with the set associativity range from 1 to 8 ways [69]. The upper part of the BPC is used as a tag to access the BTB.

Figure 10.12 illustrates the branch prediction flow using a BTB in a simple five-stage pipeline architecture. The solid objects and lines are for the regular branch prediction scheme. The flow is extended with some enhancements for control flow transfer and execution path validation, as shown in dashed lines and objects, which will replace the original operations. When an instruction is fetched from the instruction memory (in IF stage), the same PC address is used to access the BTB. A hit at this point indicates that the instruction to be executed is a control instruction. The predicted target PC (TPC) is then used as the next PC (NPC) to fetch a new instruction in the next pipeline stage (ID), rather than waiting until the later stage (EX) to use the computed NPC for fetching, to avoid branch stalls. Meanwhile, according to the instruction type, a direction prediction or computation is performed. If the branch is computed to be taken (direction hit), the TPC from the BTB will be compared with the computed NPC in the EX stage. If these two values match (address hit), both the branch direction and the target prediction are correct. However, one more validation has to be done for indirect control instructions in our approach. The system has to compare the BHSR, PBPC, and size against the HPs in the BTB entry. If the history matches as well, it is a history hit, and the program execution will continue as normal. Otherwise, it is a history mis-prediction: the

history paths associated with the TPC in the BTB do not include the history just seen. The tuple of {BPC, computed NPC, BHSR, PBPC, SIZE} has to be sent to the external memory (FRS) for further validation. Only when it misses again will a security alarm be raised.

In traditional architecture, on an address mis-prediction site, the BTB entry is just updated when the next PC is resolved from the instruction. However, since an indirect target address mis-prediction may also be caused by security attacks, in our enhanced architecture, the external memory has to be checked before the corresponding entry in the BTB is updated with the matched ICS. At a direction mis-prediction site where there is a BTB entry for the instruction but the instruction is actually not taken, the entry is deleted and the fetched TPC is squashed. There is normally a mis-prediction penalty for these remedy actions.

On the leftmost side of the flow diagram, if no entry is found in the BTB for the current PC, the instruction may be a regular data path instruction or a control instruction falling through. In the subsequent ID stage, if the instruction is found to be a taken control instruction, it is a direction mis-prediction and the address is not cached in the BTB. Before a new entry is inserted to the BTB, the tuple of {BPC, computed NPC, BHSR, PBPC, SIZE} has to be validated against the record in memory. As the BTB has limited capacity, a replacement policy, e.g., least recently used (LRU), may be employed to find a victim to evict for multiassociativity BTB.

A lot of studies have been presented on improving prediction accuracy to reduce the mis-prediction penalty and thus performance degradation. We observe that for regular speculative architecture, the BTB has served as a target address cache for some of the taken branch instructions, and mis-prediction just affects performance. To turn the BTB into a cache for the full record, we have to first extend it to include the path information as well. More importantly, we have to ensure its integrity, i.e., guarantee the BTB is free of corruption from the external memory. Thus, when we use the BTB as the reference to validate control flow transfers and execution paths at run-time, on any mis-prediction site, including direction, target address, and history path, we have to look up the external full record for the current BPC before the BTB is updated.

10.3.2.3 Architecture Support for Control Flow Validation

Access to the full record should be reduced to the minimum to lower the performance degradation. Thus, a clear distinction has to be made between instructions that will point to a safe target address and the instructions that definitely require validation against the external FRS. Assuming that code integrity checking mechanisms, e.g., [70, 71], have been applied, we can regard direct control instructions as always generating correct target addresses because the addresses are encoded in the instruction and the instruction bits have been ensured correct. Hence, on BTB mis-predictions, these instructions can directly update the BTB with the computed next target address without validating against the full record set. Note that only the target address is needed for the BTB entries. In addition, the full record in memory

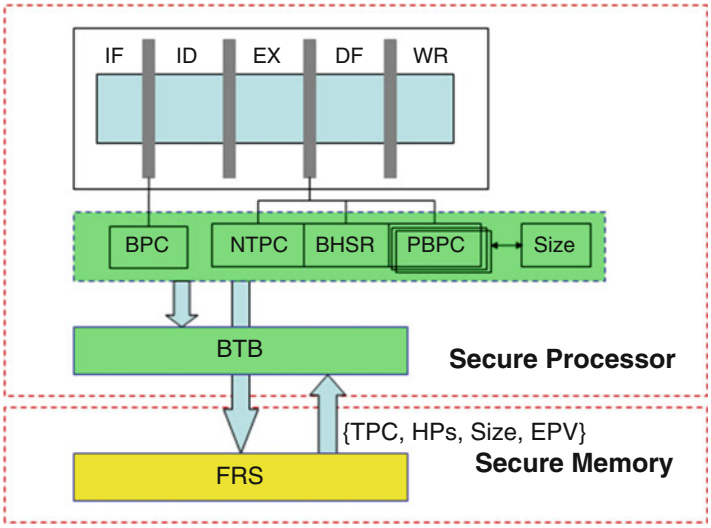


Fig. 10.13 The architecture support in processor pipeline for control flow validation

does not need to keep any information for direct control instructions. Only those indirect control instructions that are mis-predicted by the BTB will possibly incur full record lookups.

Figure 10.12 shows that the tuple of {BPC, computed NPC, BHSR, PBPC, Size} is sent to the full record for validation at three sites (labeled 1, 2, 3), which represent direction mis-prediction, address mis-prediction, and history mis-prediction, respectively. Figure 10.13 illustrates the overview of our architecture support in a processor pipeline for control flow validation.

10.3.3 Experimental Results and Security Analysis

We have modified the SimpleScalar toolset to profile a set of MiBench applications, to determine how common the ambiguities for binary path representation are. In addition, we test a set of SPECINT applications running on a modified cycle-accurate SimpleScalar that models an in-order single-issue processor to measure the performance impact. We consider extra delays for accessing the full record set for validation when the branch prediction unit is enhanced with security features.

10.3.3.1 Ambiguity Alleviation

We define the path ambiguity as two or more different execution paths that have identical branch site (BPC), same target (TPC), and same binary history, but

Table 10.2 Number of ambiguities found

Benchmark	12 bits of history
susan-corner	16
susan-edge	5
susan-smoth	1
bitcount	4
qsort-large	15
qsort-short	22
dijkstra	8
patricia	4

different past branch sites (PBPC). Our profiling shows that several legal paths can share the same tuple of {BPC, TPC, History}. The ambiguity results are shown in Table 10.2 for a set of applications from MiBench suit. To handle the ambiguities and reduce the possibility for a successful attack, we include the past branch site (PBPC) and also the number of basic blocks in the super block (Size). With this information, we dissolve all the ambiguities for the tested applications.

10.3.3.2 Performance Impact of the Run-Time Validation Mechanism

We next examine the performance impact of our run-time validation mechanism. Our BTB configuration contains three parameters, i.e., the number of sets, the set associativity, and the number of history paths in each entry (the path associativity). We evaluate the performance degradation for a set of SPECINT benchmarks with the BTB configuration of 512 sets, direct map, and 1 history path in each BTB entry. Looking up the full record set in memory to validate mis-predicted indirect instruction targets has resulted in negligible performance degradation. Even with the full record memory access time set at 200 cycles, for most of the benchmarks, the execution cycles overhead is 4.82% on average. For application gcc, the overhead is noticeable – up to 24.13%.

We also check the impact of each BTB parameter on the performance. We vary one of the three parameters, the number of sets, the set associativity, and history path associativity, and keep the other two unchanged. Due to page limit, the results are not included in this book chapter.

10.3.3.3 Security Analysis

We assume that the profile for the FRS is complete, and hence the false positive rate is zero (or near zero). In order to evaluate the false positive rate of our approach, we run a set of applications from MiBench. We profile all possible options in each case; then, we feed the application with a different input and check the result. The system does not raise an alert when there is no attack.

Table 10.3 Vulnerable programs

Program	Attack	Site detection
Traceroute	Double free	Different past branch site (PBPC) at *getenv* function
Polymorph	Buffer overflow	Different target address (TPC)
WU-FTPD	Format string	Unknown signature at *vf printf* function

To evaluate the effectiveness of the attack detection, we tested real programs with well known vulnerabilities: Traceroute (double free), Polymorph 0.4 (buffer overflow), and WU-FTPD (format string). Table 10.3 shows details of the attack capture sites. In all cases, the attacks were detected. The results indicate that our approach is effective in detecting different kinds of memory corruption attacks.

10.3.4 Summary

In current processors, control flow transfer is a blindfold without any validity check. With more software vulnerabilities being exploited by adversaries, control data and decision data attacks have become the most dominant and serious threats to computer system security. In this chapter, we have proposed a practical micro-architecture-based approach for run-time control flow transfer and execution path validation. With the aid of the speculative architecture of the branch target buffer (BTB), we narrow down the insecure instructions to indirect control instructions, and sample the validations only at indirect control sites. The anomaly detection rate of our approach is higher than previous related work because our approach not only validates the history path, but also monitors the next branch decisions at run-time. Our approach results in very little performance degradation with minor storage overhead.

References

1. Mobile Threats, Secure Mobile Systems. http://www.smobilesystems.com/homepage/home.jsp
2. Computer Crime and Security Survey, Computer Security Institute (2006). http://www.gocsi.com. Accessed 2 March 2007
3. Epaynews – mobile commerce statistics (2005) http://www.epaynews.com/statistics/mcommstats.html. Accessed 12 July 2005
4. Ravi S, Raghunathan A, Kocher P, Hattangady S (2004) Security in embedded systems: design challenges. ACM Trans. Embedded Comput Syst: Special Issue Embedded Syst Security 3(3): 461–491
5. Younan Y, Joosen W, Piessens F (2005) A methodology for designing countermeasures against current and future code injection attacks. In: Proceedings of the International Workshop on Information Assurance, March 2005, pp 3–20

6. Baratloo A, Singh N, Tsai T (2000) Transparent run-time defense against stack smashing attacks. In: Proceedings of the USENIX Annual Technical Conference, San Jose, CA, June 2000, pp 251–262
7. BBP, BSD heap smashing. http://freeworld.thc.org/root/docs/exploitwriting/BSD-heap-smashing.txt. Accessed Oct 2006
8. Dobrovitski I (2003) Exploit for CVS double free() for linux pserver. http://seclists.org/lists/bugtraq/2003/Feb/0042.html. Accessed Feb 2003
9. Shankar U, Talwar K, Foster JS, Wagner D (2001) Detecting format string vulnerabilities with type qualifiers. In: Proceedings of the USENIX Security Symposium, USENIX Association, Berkeley, pp 201–220
10. Younan Y (2003) An overview of common programming security vulnerabilities and possible solutions. Master's thesis, Vrije Universiteit Brussel
11. Suh GE, Clarke D, Gassend B, van Dijk M, Devadas S (2003) AEGIS: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the International Conference on Supercomputing, pp 160–171
12. Suh GE, Clarke D, Gassend B, van Dijk M, Devadas S (2003) Efficient memory integrity verification and encryption for secure processors. In: International Symposium on Microarchitecture, pp 339–350, Dec 2003
13. Trusted Mobile Platform, NTT DoCoMo, IBM, Intel Corporation. http://xml.coverpages.org/TMP-HWADv10.pdf. Accessed 2004
14. CERT Security Advisories. http://www.cert.org/advisories. Accessed 1998–2004
15. United States Computer Emergency Readiness Team. Technical cyber security alerts. http://www.uscert.gov/cas/techalerts/. Accessed 2004–2011
16. Microsoft Security Bulletin. http://www.microsoft.com/technet/security. Accessed 2000–2011
17. Chen S, Xu J, Sezer EC, Gauriar P, Iyer RK (2005) Non-control-data attacks are realistic threats. In: Proceedings of the Conference on USENIX Security Symposium, Baltimore, MD, July/Aug 2005, pp 12–26
18. Feng HH, Giffin JT, Huang Y, Jha S, Lee W, Miller BP (2004) Formalizing sensitivity in static analysis for intrusion detection. In: Proceedings of the IEEE Symposium on Security & Privacy
19. Zhang T, Zhuang X, Pande S, Lee W (2005) Anomalous path detection with hardware support. In: International Conference on Compilers, Architecture, & Synthesis for Embedded Systems, Sept 2005, pp 43–54
20. Shi Y, Lee G (2007) Augmenting branch predictor to secure program execution. In: IEEE/IFIP International Conference on Dependable Systems & Networks, June 2007
21. Livshits B, Martin M, Lam MS (2006) Securifly: runtime protection and recovery from web application vulnerabilities. Stanford University, Technical Report, Stanford University, Sept 2006
22. Xu W, Bhatkar S, Sekar R (2006) Taint-enhanced policy enforcement: a practical approach to defeat a wide range of attacks. In: Proceedings of the USENIX Security Symposium, July–Aug 2006, pp 121–136
23. Qin F, Wang C, Li Z, Kim H, Zhou Y, Wu Y (2006) Lift: a low-overhead practical information flow tracking system for detecting security attacks. In: IEEE/ACM International Symposium on Microarchitecture, pp 135–148
24. Suh GE, Lee JW, Zhang D, Devadas S (2004) Secure program execution via dynamic information flow tracking. In: Proceedings of the International Conference on Architectural Support for Programming Languages & Operating Systems, pp 85–96
25. Crandall JR, Wu SF, Chong FT (2006) Minos: architectural support for protecting control data. ACM Tran. Arch Code Opt 3(4): 359–389
26. Xu J, Nakka N (2005) Defeating memory corruption attacks via pointer taintedness detection. In: Proceedings of the International Conference on Dependable Systems & Networks, pp 378–387
27. Vachharajani N, Bridges MJ, Chang J, Rangan R, Ottoni G, Blome JA, Reis GA, Vachharajani M, August DI (2004) RIFLE: an architectural framework for user-centric information-flow security. In: Proceedings of the International Symposium on Microarchitecture, pp 243–254

28. Shi W, Fryman J, Gu G, Lee H-H, Zhang Y, Yang J (2006) InfoShield: a security architecture for protecting information usage in memory. International Symposium on High-Performance Computer Architecture, Feb 2006, pp 222–231
29. Venkataramani G, DoudalisI, Solihin Y, Prvulovic M (2008) Flexitaint: a programmable accelerator for dynamic taint propagation. In: Proceedings of the International Symposium on High-Performance Computer Architecture, Feb 2008, pp 173–184
30. Chang W, Streiff B, Lin C (2008) Efficient and extensible security enforcement using dynamic data flow analysis. In: Proceedings of the Conference on Computer & Communications Security, Oct 2008, pp 39–50
31. Lam LC, Chiueh T-C (2006) A general dynamic information flow tracking framework for security applications. In: Proceedings of the Annual Computer Security Applications Conference, pp 463–472
32. Pozza D, Sisto R (2008) A lightweight security analyzer inside gcc. In: Proceedings of the International Conference on Availability, Reliability & Security, pp 851–858
33. Dalton M, Kannan H, Kozyrakis C (2007) Raksha: a flexible flow architecture for software security. In: Proceedings of the International Symposium on Computer Architecture, June 2007, pp 482–293
34. Chen S, Kozuch M, Strigkos T, Falsafi B, Gibbons PB, Mowry TC, Ramachandran V, Ruwase O, Ryan M, Vlachos E (2008) Flexible hardware acceleration for instruction-grain program monitoring. In: Proceedings of the International Symposium on Computer Architecture, June 2008, pp 377–388
35. Katsunuma S, Kurita H, Shioya R, Shimizu K, Irie H, Goshima M, Sakai S (2006) Base address recognition with data flow tracking for injection attack detection. In: Proceedings of the Pacific Rim Interernational Symposium on Dependable Computing, Dec 2006, pp 165–172
36. Ho A, Fetterman M, Clark C, Warfield A, Hand S (2006) Practical taint-based protection using demand emulation. In: EUROSYS'06
37. Chen H, Wu X, Yuan L, Zang B, Yew P-C, Chong FT (2008) From speculation to scurity: practical and efficient information flow tracking using speculative hardware. In: Proceedings of the International Symposium on Computer Architecture, June 2008, pp 401–412
38. Kannan H, Dalton M, Kozyrakis C (2009) Decoupling dynamic information flow tracking with a dedicated coprocessor. In: Proceedings of the International Conference on Dependable Systems & Networks, June 2009, pp 105–114
39. Nightingale EB, Peek D, Chen PM, Flinn J (2008) Parallelizing security checks on commodity hardware. In: Proceedings of the International Conference on Architectural Support for Programming Languages & Operating Systems, March 2008, pp 308–318
40. Ruwase O, Gibbons PB, Mowry TC, Ramachandran V, Chen S, Kozuch M, Ryan M (2008) Parallelizing dynamic information flow tracking. In: Proceedings of the Annual Symposium on Parallelism in Algorithms & Architectures, June 2008, pp 35–45
41. Huang R, Deng DY, Suh GE (2010) Orthrus: efficient software integrity protection on multi-cores. Comput Archit News 38(1): 371–384
42. Newsome J (2005) Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: International Symposium on Software Testing & Analysis
43. Wilander J, Kamkar M (2002) A comparison of publicly available tools for static intrusion prevention. In: Proceedings of the 7th Nordic Workshop Secure IT System, Nov 2002
44. Younan Y, Joosen W, Piessens F (2006) Efficient protection against heap-based buffer overflows without resorting to magic. In: Proceedings of the International Conference on Information & Communication Security, Dec 2006
45. Younan Y, Pozza D, Piessens F, Joosen W (2006) Extended protection against stack smashing attacks without performance loss. In: Proceedings of the Annual Computer Security Applications Conference, Dec 2006, pp 429–438
46. "Spec CINT 2000 benchmarks," http://www.spec.org/cpu2000/CINT2000/. http://www.spec.org/cpu2000/CINT2000/

47. Isaev IK, Sidorov DV (2010) The use of dynamic analysis for generation of input data that demonstrates critical bugs and vulnerabilities in programs. Program Comput Software 36(4): 225–236
48. Nethercote N, Seward J (2007) Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of the Conference on Programming Language Design & Implementation, June 2007, pp 89–100
49. Sotirov A (2005) Automatic vulnerability detection using static source code analysis Ph.D. dissertation, University of Alabama
50. Younan Y, Joosen W, Piessens F (2005) A methodology for designing countermeasures against current and future code injection attacks. In: International Workshop on Information Assurance, March 2005, pp 3–20
51. One A (1996) Smashing the stack for fun and profit. Phrack 7: 49
52. Feng HH, Giffin JT, Huang Y, Jha S, Lee W, Miller BP (2004) Formalizing sensitivity in static analysis for intrusion detection. In: Proceedings of the IEEE Symposium on Security & Privacy
53. Shi Y, Lee G (2007) Augmenting branch predictor to secure program execution. In: IEEE/IFIP International Conference on Dependable Systems & Networks
54. Austin T, Larson E, Ernst D (2002) SimpleScalar: an infrastructure for computer system modeling. IEEE MICRO 35(2): 59–67
55. Guthaus MR, Ringenberg JS, Ernst D, Austin TM, Mudge T, Brown RB (2001) Mibench: a free, commercially representative embedded benchmark suite. In: IEEE International Workshop on Workload Characterization, pp 3–14
56. Cowen C, Pu C, Maier D, Hinton H, Walpole J, Bakke P, Beattle S, Grier A, Wagle P, Zhang Q (1998) StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In: Proceedings of the USENIX Security Symposium, pp 63–78
57. Pyo C, Lee G (2002) Encoding function pointers and memory arrangement checking against buffer overflow attacks. In: Proceedings of the International Conference on Information & Communication Security, pp 25–36
58. Suh GE, Lee JW, Zhang D, Devada S (2006) Secure program execution via dynamic information flow tracking. In: ACM Proceedings of the International Conference on Architectural Support for Programming Languages & Operating Systems, pp 85–96
59. Tuck N, Cadler B, Varghese G (2004) Hardware and binary modification support for code pointer protection from buffer overflow. In: Proceedings of the International Symposium on Microarchitecture
60. Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA (1996) A sense of self for UNIX processes. In: Proceedings of the IEEE Symposium on Security & Privacy
61. Mao S, Wolfe T (2007) Hardware support for secure processing in embedded systems. In: Proceedings of Design Automation Conference, pp 483–488
62. Michael c, Ghosh A (2000) Using finite automata to mine execution data for intrusion detection: a preliminary report. In: Proceedings of the International Workshop on Recent Advances in Intrusion Detection, pp 66–79
63. Lee R, Karig DK, McGregor JP, Shi Z (2003) Enlisting hardware architecture to thwart malicious code injection. In: Proceedings of the International Conference on Security in Pervasive Computing, pp 237–252
64. Park Y, Zhang Z, Lee G (2006) Microarchitectural protection against stack-based buffer overflow attacks. IEEE Micro 26(4): 62–71
65. Ye D, Kaeli D (2003) A reliable return address stack: microarchitectural features to defeat stack smashing. In: Proceedings of the Workshop on Architectural Support for Security & Antivirus, pp 73–88
66. Bori E, Wang C, Wu Y, Araujo G (2005) Dynamic binary control-flow errors detection. ACM SIGARCH Comput Arch News 33(5): 15–20
67. Ragel R, Parameswaran S (2006) Hardware assisted preemptive control flow checking for embedded processors to improve reliability. In: Proceedings of the International Conference on Hardware/Software Codesign & System Synthesis, pp 100–105

68. Shi Y, Dempsey S, Lee G (2006) Architectural support for run-time validation of control flow transfer. In: Proceedings of the International Conference on Computer Design
69. Perleberg C, Smith AJ (1993) Branch target buffer design and optimization. IEEE Trans Comput 42(4): 396–412
70. Fei Y, Shi ZJ (2007) Microarchitectural support for program code integrity monitoring in application-specific instruction set processors. In: Proceedings of the Design Automation & Test Europe Conference, pp 815–820
71. Lin H, Guan X, Fei Y, Shi ZJ (2007) Compiler-assisted architectural support for program code integrity monitoring in application-specific instruction set processors. In: Proceedings of the International Conference on Computer Design
72. Martinez Santos JC, Fei Y (2008) Leveraging speculative architectures for run-time program validation. In: Proceedings of the International Conference on Computer Design, Oct 2008, pp 498–505
73. Martinez Santos JC, Fei Y, Shi ZJ (2009) PIFT: Efficient dynamic information flow tracking using secure page allocation. In: Proceedings of Workshop on Embedded System Security, Oct 2009

Chapter 11
Side-Channel Attacks and Countermeasures for Embedded Microcontrollers

Patrick Schaumont and Zhimin Chen

11.1 Introduction

While trustworthy hardware helps to establish the basis of trustworthy computing, most applications in embedded security rely to a significant extent on software. Smart-cards are an excellent example. A smart-card is an embedded computer in the form factor of a credit card with an integrated microcontroller. In contrast to a credit card, a smart-card thus has an active component. This allows the card to execute one side of a cryptographic protocol, such as digital signature-generation. Crypto-protocols are build using crypto-algorithms including symmetric-key and public-key encryption, random number generation, and hashing. A smart-card may implement these building blocks in software or, in some cases, in dedicated hardware. Software is often preferred because of two different reasons: reducing the design cost and supporting flexibility. This chapter will discuss the implementation of side-channel attacks on such microcontrollers, as well as some common countermeasures. The objective is to introduce the reader to this exciting field of research within the limits of a book chapter. In-depth discussion of side-channel analysis on microcontrollers can be found in the literature, e.g., [1].

The main insight of this chapter is the following. Even though the cryptographic buildingblocks on microcontrollers are developed as software, the side-channel leakage originating from them is generated by hardware, in this case by the microcontroller architecture. Hence, understanding and mitigating side-channel leakage requires understanding the low-level interaction of crypto-software with the microcontroller hardware. By carefully elaborating on this insight, side-channel countermeasures can be created that require a minimum amount of architecture specialization, yet that are fully flexible and algorithm independent. This is a novel

P. Schaumont (✉) · Z. Chen
ECE Department, Virginia Tech, Blacksburg, VA 24061, USA
e-mail: schaum@vt.edu; chenzm@vt.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_11, © Springer Science+Business Media, LLC 2012

insight since (a) typical countermeasures in hardware require significant architecture specialization and (b) typical countermeasures in software are highly dependent on the crypto-algorithm.

11.2 Side-Channel Leakage in an Embedded Microcontroller

Figure 11.1 shows the generic architecture of a load-store microcontroller. This type of architecture implements memory-operations (memory-load and memory-store) as explicit instructions. The central element in this architecture is a register file. On the right of the register file is a datapath to implement microcontroller instructions. On the left of the register file sits a controller to fetch instructions from an instruction memory and dispatch them on the datapath. A cryptographic software program can be thought of as a series of transformations on data stored in the central register file. In this architecture, one can distinguish memory-store instructions, memory-load instructions, arithmetic (register-to-register) instructions, and control-flow instructions.

Now, assume that a secret value is stored in one of the locations of the register file; such a value is called a *sensitive* value. Any operation that involves the sensitive value may potentially result in side-channel leakage. This is because the processing of the sensitive value (storing it in memory, performing an arithmetic operation with it, and so on) will require some energy, which is correlated to the sensitive value. A side-channel leak can be observed in several ways, such as by measuring the power dissipation of the microcontroller, or by monitoring its electromagnetic radiation. The following analysis shows how each instruction type can cause side-channel leakage.

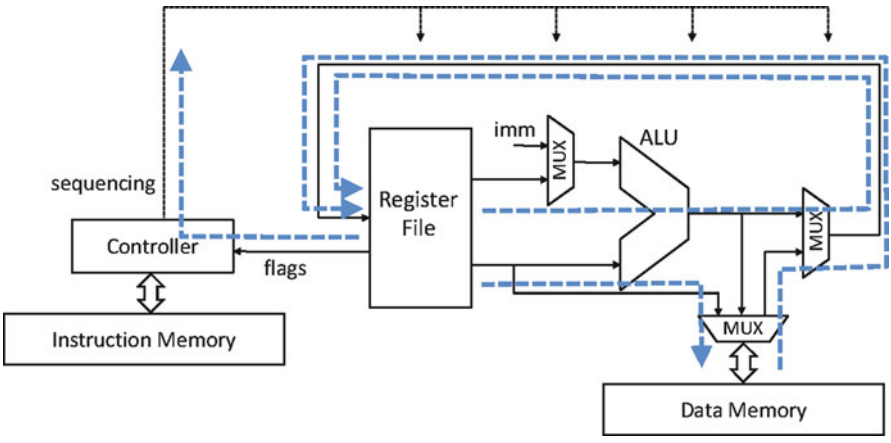


Fig. 11.1 Source of side-channel leakage in a microcontroller

1. *Memory-store operations* transfer data from the register file to data memory. A memory-store operation has at least two arguments: a memory address and a data item to write into that designated memory location. Side-channel leakage will occur when the sensitive location is used as either the memory-store address or the memory-store operand.
2. *Memory-load operations* transfer data from the data memory to the register file. A memory-load operation requires at least one argument, namely a memory address to read the data from. Power-based side-channel leakage may occur when the sensitive value is used to form the address, or even when the sensitive value is being overwritten by data fetched from memory. The latter may sound surprising, but is in fact easy to explain. In CMOS, energy consumption is proportional to signal transitions. Hence, overwriting a sensitive value (for example, with an all-ones pattern), will result in side-channel leakage.
3. *Arithmetic and Bit-wise operations* transfer data from the register file to the register file. Power-based side-channel leakage will occur when the sensitive value is used as either a source operand or when it is being overwritten by the resulting operation.
4. Finally, *control-flow instructions* may conditionally modify the execution path of a program. If the execution path is changed based on the value of a sensitive location, side-channel leakage will occur.

Obviously, once the register file contains a sensitive value, there are multiple sources of side-channel leakage possible. Hence, any countermeasure developed against side-channel leakage will need to jointly address all of these sources.

The remainder of this chapter will first discuss side-channel attacks and next consider side-channel countermeasures. The discussion will be focused on power-based side-channels, and use symmetric-key cryptography as the driving example. In relation to the analysis presented earlier, this implies that the focus is on datapath and memory operations. The assumption is made that conditional control flow will never depend on a sensitive location. For symmetric-key encryption, this is fortunately easy to achieve.

11.3 Side-Channel Attacks on Microcontrollers

This section shows how to mount a side-channel attack on microcontroller software. After a review of the available statistical methods, there's a specific example: a side-channel attack on a software implementation of the Advanced Encryption Standard (AES).

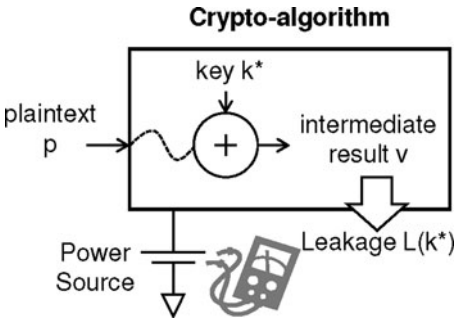


Fig. 11.2 Power-based side-channel analysis

11.3.1 Side-Channel Analysis

Figure 11.2 illustrates how a side-channel attack works. In the example attack of Fig. 11.2, the objective is to retrieve the internal secret key k^* of a crypto-algorithm. The adversary can observe the input p, as well as the overall power consumption. The crypto-algorithm is known beforehand; only the secret key is unknown.

Based on a known input p, the attacker will find an intermediate result v that depends on both p and (part of) the secret key k^*. By observing the side-channel leakage (the data-dependent power-consumption) of v, a hypothesis test on the key value k^* can be created. The leakage caused by v is a function of the key value k^*, and it can be expressed as follows [2]:

$$L(k^*) = f_{k*}(p) + \varepsilon.$$

The function f_{k^*} is dependent on the crypto-algorithm as well as on the nature of the implementation in hardware and software. The error ε is an independent noise variable, defined by measurement errors as well as other unrelated activity in the crypto-implementation. Several types of power-based side-channel analysis have been formulated starting from this relation.

- In Differential Power Analysis (DPA), the adversary will select a small part of the key and build a hypothesis test based on the difference of means for $f_{k^*}(p)$ depending on the value of that part of key [3].
- In Correlation Power Analysis (CPA), the adversary builds a linear approximation $\tilde{f}_k$ for f_k, and builds a hypothesis test based on linear correlation of $\tilde{f}_k$ and $L(k^*)$[4]. A CPA can reveal multiple key bits at once but is sensitive to the approximation error made by $\tilde{f}_k$.
- In a Template Attack, the adversary characterizes the probability density function (pdf) of the Leakage $L(k)$ for every possible key, which requires an approximation of f_k as well as the error. When this pdf is known, the adversary can then select the key k^* using a Maximum Likelihood Approximation [5–7].

What makes a side-channel attack so much more efficient than a brute-force attack is that it can provide a conclusive hypothesis test over part of the key. For example,

assume that the key k contains 128 bits. Assume also that the attacker has identified an intermediate value v which is dependent on a single key byte (8 bits out of 128 bits). In this case, the side-channel attack hypothesis test needs to select 1 out of 256 possible key bytes. Using 16 such tests, the attacker can then proceed in guessing each byte of the 128-bit key. In comparison with a brute-force attack, the overall search space is now reduced from 2^{128} to $16 * 2^8 = 2^{12}$.

Another powerful concept of a side-channel attack is that only weak assumptions are made on the implementation details of the cryptography. For example, CPA and DPA do not assume how exactly the intermediate sensitive value v will occur during the execution of the algorithm; they only assume that it will occur at some point. Thus, small variations in the implementation of an algorithm will typically not affect the attack. The example attack on AES, presented further in this chapter, will demonstrate this. One needs to take note that this benefit is not available for a Template Attack. As the leakage $L(k)$ needs to be precisely characterized for every possible key, a template attack is implementation-specific.

11.3.2 The Advanced Encryption Standard Mapped on a PowerPC

This and the following sections will discuss a concrete implementation of a side-channel attack on a software implementation of the Advanced Encryption Standard. This includes the implementation of AES on a 32-bit microprocessor, the analysis setup, and the analysis of attack results.

The Advanced Encryption Standard is well known, and detailed descriptions of the algorithm have been published [8]. A summary of the major steps in the algorithm is as follows.

AES operates on a block of 128 bits at a time. It takes a block of 128 bits of plaintext and iterates that data through several rounds to produce ciphertext. Each round includes four transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. A description of these operations may be found in literature. AES organizes a block of 128 bit data as a 4x4 matrix of 16 bytes. Figure 11.3 illustrates the dataflow within a single AES round. The AES-128 algorithm, which is investigated in this paper, includes ten such rounds.

Each of the s-operations in Fig. 11.3 is a 256-entry substitution table. This substitution is mathematically defined: an S-box operation on an input byte is found as a finite-field inversion (over GF (2^8)) of that byte followed by an affine transformation. In each AES-128 round, there are 16 S-box substitutions. Such a formulation of AES leads to byte-wide computations over the entire round, which is inefficient on a 32-bit processor.

The S-box is therefore frequently implemented as a T-box, which includes each of the shaded operations in Fig. 11.3. A T-box transforms one byte into four bytes, each of which represent part of a MixColumn result. In each AES round, four

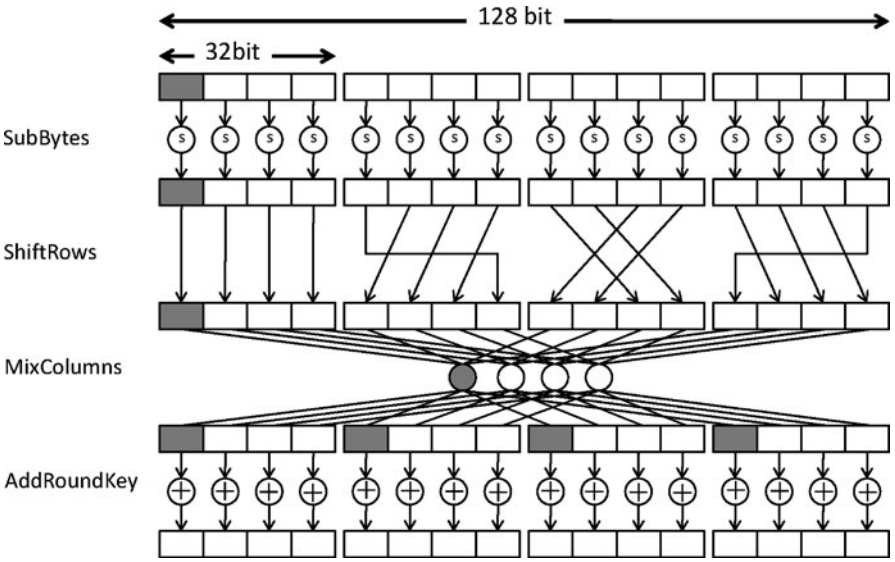


Fig. 11.3 Dataflow in AES. A single T-tbox operation includes the shaded operations

Table 11.1 AES performance summary on a 24 MHz PowerPC core

	S-box	T-box
Latency (μs)	310	60
Throughput (kbit/s)	412	2,133
Footprint (kB)	1.9	5.2

T-box operations can be combined to obtain a single row of the AES state matrix. Due to the detailed implementation of MixColumn, this approach requires four different T-box lookup tables. However, this approach provides a significantly better utilization of the 32-bit datapath of a processor.

Table 11.1 illustrates the implementation results of implementing the two versions of AES (one using S-box, and another using T-box), on a 32-bit PowerPC microprocessor running at 24 MHz. The PPC is implemented inside of an FPGA, with data-memory and instruction-memory mapped into on-chip FPGA memories. Hence, the PowerPC does not need a cache memory. Table 11.1 demonstrates – as expected – that the T-box implementation is significantly faster than the S-box implementation, at the cost of increased footprint.

Figure 11.4 shows a test setup for side-channel analysis. This setup shows a SASEBO-G board, a digital oscilloscope, and a PC to control the measurement. The power consumption of the FPGA board is captured by means of a current probe. A CPA or DPA requires the collection of a large amount of power traces from the microcontroller while it is executing AES. This proceeds through the following steps. The PC sends a sample plaintext to the PowerPC on the FPGA for encryption. During the encryption, the digital oscilloscope captures the power consumption from the board. After the encryption is completed, the PC downloads

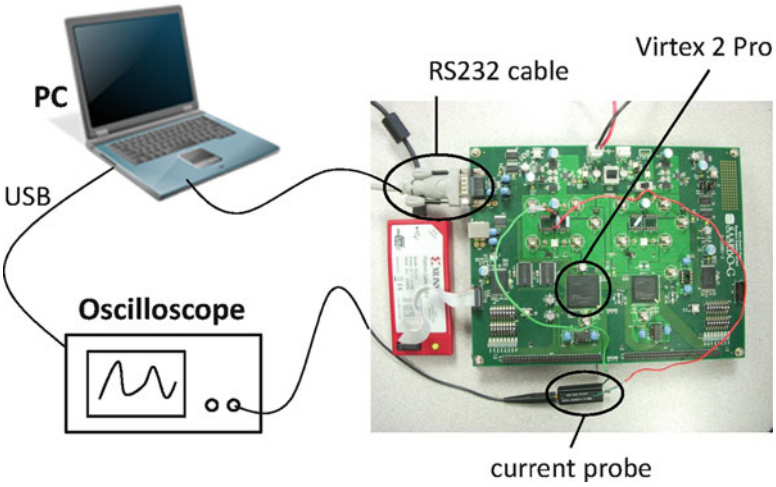


Fig. 11.4 Side-channel analysis setup for AES

the resulting power trace from the oscilloscope, and proceeds with the next sample plaintext. A setup like this can run fully automatic and captures several traces per second. Thus, it is easy to obtain thousands of measurements in a short time.

11.3.3 Side-Channel Analysis: Power Model Selection

This section introduces the actual side-channel analysis of AES executing on the PowerPC processor. The specific attack used is Correlation Power Analysis. Two important aspects of a practical CPA will be covered. The first is the selection of the power model, and the second is the definition of the attack success metric (measurements to disclosure, MTD).

The objective of CPA is to compare measurements, obtained from a prototype, with power estimates, generated using a power model. The power model is chosen so that it has a dependency on a part of the secret key.

A good candidate is the output of the substitution step. Assume that the attacker is observing AES encryption. In the first round of AES encryption, the output of the substitution step is given as follows.

$$out[i] = subbytes\,(in\,[i]\,xor\,key\,[i])$$

where *in[i]* is the *i* th byte of the 128-bit plaintext, *key[i]* is the *i* th byte of the 128-bit key, and *out[i]* is the *i* th byte of the 128-bit AES state. In this formula, *in* is known, while *key* and *out* are unknown. However, *out* is indirectly observable through the power consumption of the algorithm. Hence, the attacker can create a hypothesis

test for a key byte, using a power model for *out*. By making a guess for the *i*th key byte, the attacker can infer the value of *out[i]*. The power model used in the CPA is an estimate for the power consumption of the hypothesized *out[i]*.

In CMOS technology, where power is consumed as a result of state transitions, it is common to choose the Hamming Distance as a power model [1]. For example, when performing power analysis on a processor, the attacker could use the transitions of a processor register. However, this knowledge of both the previous- and next-value of the processor register. If the attacker estimates *out[i]* then only the next-value for that register is known. To know the previous-value of the register, the attacker would need to know exactly how the AES software is mapped into processor registers, which is a very strong requirement. Therefore, for software implementations, the attacker may use the Hamming Weight (bit count) as a power model, rather than the Hamming Distance. If one assumes that the previous value of the register could be any value then the Hamming Value is still a good statistical approximation of the Hamming Distance.

A second difficulty in selecting the power model is that a software implementation of AES typically may take many instructions. As a result, the power trace of an AES encryption may cover several hundred clock cycles. The attacker cannot predict, in general, at which clock cycle *out[i]* will be computed and stored in a processor register. To address this issue, the attacker simply extends the power model over all the clock cycles of a power trace, by repeatedly using the Hamming Weight of *out[i]* for every clock cycle that needs analysis.

Finally, when attacking an unknown AES implementation, the specific variation of the algorithm that is executing is unknown to an attacker. For example, an AES implementation could be using either an S-box design or a T-box design. The S-box implementation produces the intermediate value *out[i]* as specified earlier, and therefore the Hamming Weight of *out[i]* is a valid power model. The T-box based implementation, on the other hand, does not have such an intermediate value, because it collapses the S-box substitution with several other AES operations. For a T-box, a more appropriate power model would be one that is based on a T-box lookup table. However, in practice, this is a minor issue, because an attacker can simply try out multiple power models, and select the one for which the strongest hypothesis conclusions can be made. The experiments further in this chapter, therefore, always selected the best power model available.

11.3.4 Side-Channel Analysis: Practical Hypothesis Tests

Once the attacker has chosen a power model, he can combine the measurements with the power estimates and perform the hypothesis test required for a CPA. A hypothesis test does not give an exact answer, but rather one which is correct with a high probability. This raises the practical problem of how to define a practically useful metric to express the success rate of the side-channel analysis.

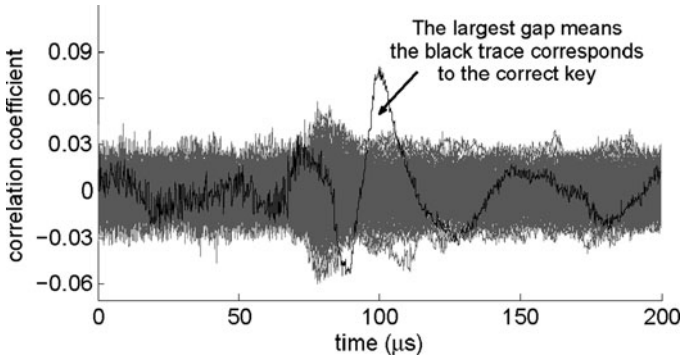


Fig. 11.5 An example of 256 correlation coefficient traces. Around time 100 us, the black trace which corresponds to the correct key byte emerges from all the other 255 traces

A commonly used metric is called Measurements to Disclosure (MTD). The basic idea is that the more measurements that are required to successfully attack a cryptographic design with side-channel analysis, the more secure that design is. Suppose the attacker has a set of random plaintext blocks (*pt[1... n]*). Each of them is used for one measurement. The attacker also obtains *n* power traces, each of which contain *m* sampling points (*tr[1... n][1... m]*).

In the experiment, the CPA takes *pt* and *tr* as inputs, and discovers AES's key byte by byte by focusing on the first round of AES. The details are presented in Algorithm 1. The basic process is to take *pt[1... n]* and a guess value of a key byte (*key_guess*) to calculate an intermediate value of the AES software *iv[1... n]* (*iv[i] = f (key guess, pt[i])*). By sampling a complete power trace, the attacker guarantees that the operations on the intermediate value occur during the sampling process. The power dissipated by *iv* is approximated using the Hamming Weight of *iv* (*iv_hw[1... n]*) to approximate the power dissipated by *iv*. The attacker then calculates the correlation coefficient of *iv_hw* and the actual measured power traces at each sampling point. This way, a correlation coefficient trace *corr[1... m]* is obtained (*corr[i]* is the correlation coefficient of *iv_hw[1... n]* and *tr[1... n][i]*). The correlation traces correspond to one guess value of the key byte. As there are 256 possible values for one byte of key, the attacker can, therefore, obtain 256 coefficient traces. By grouping these coefficient traces together, the attacker is able to identify one coefficient trace from all the other 255 traces. In particular, at some points (when the operations on the intermediate value occur), the abstract value of the coefficient trace corresponding to the correct key guess is much larger than all the other traces.

Figure 11.5 gives an example of the correlation coefficient traces. Around time 100 μs, one trace emerges from all the other traces. This black trace corresponds to the correct key byte.

Table 11.2 Attack results summary: showing the number of bytes discovered

Measurements	S-box	T-box
2,048	2	2
5,120	4	4
10,240	8	6
25,600	11	8
40,960	16	12
51,200	16	13

Algorithm 1 Correlation power attack on one key byte

Require: *pt[1... n]* contains the random plaintext for encryption; *tr[1... n][1... m]* contains the sampled power traces; f maps the inputs to the intermediate value *iv*; g calculates the correlation coefficient.
Ensure: key *gap[j]* is the CPA-attacked key byte.
/* Obtain correlation coefficient traces *corr* */
for key guess = 0 to 255 **do**
  for i = 1 to n **do**
    iv[i] = f (key guess, pt[i])
    iv hw[i] = HammingW eight(iv[i])
  end for
  for i = 1 to m **do**
    corr[key guess][i] = g(iv hw[1 . . . n], tr[i][1 . . . n])
  end for
end for
/* Find the correct key byte */
for i = 1 to m **do**
  find $|corr[key1][i]| = max(|corr[0 . . . 255][i]|)$
  find $|corr[key2][i]| = second\ max(|corr[0 . . . 255][i]|)$
  gap[i] = corr[key1][i] - corr[key2][i]
  key gap[i] = key1
end for
find *gap[j] = max(gap[1 . . . m])*
return key *gap[j]* as the correct key byte

11.3.5 Side-Channel Analysis: Attack Results

Table 11.2 presents a comparison between the S-box AES implementation and the T-box AES implementation, in terms of resistance to side channel attack. It can be clearly observed that both the S-box and the T-box can be broken through CPA. The T-box requires more measurements for a similar level of success as the S-box implementation. This is expected, because the T-Box implementation combines SubBytes, ShiftRows and part of MixColums into one highly compressed lookup table. This implies that the sensitive data exploited by the side-channel analysis is present in the system for a very short time as compared to the S-box implementation.

Figures 11.6 shows typical correlation coefficient plots for the S-box and T-box implementation.

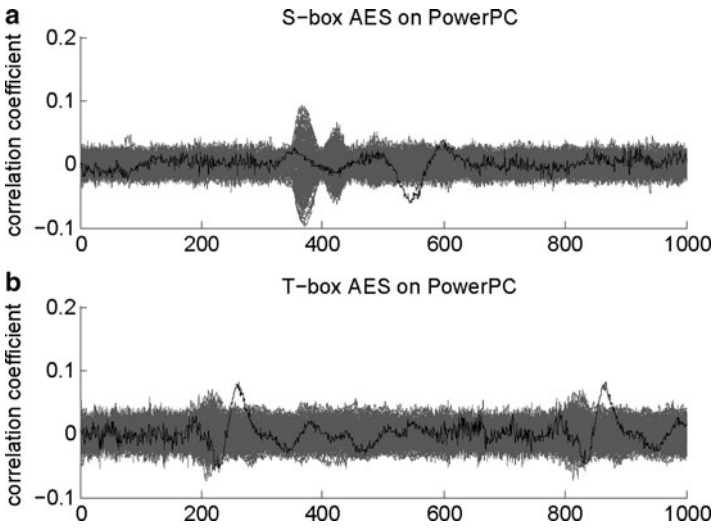


Fig. 11.6 Correlation coefficient plots for side-channel attack (number of measurements = 5,120) on two different AES implementations on PPC

This completes the overview of a standard side-channel attack on unprotected AES. The next section presents possible countermeasures against this side-channel analysis.

11.4 Side Channel Countermeasures for Microcontrollers

As SCA is a passive attack, the attack itself cannot be detected; it can only be mitigated by avoiding the generation of side-channel leakage. This section describes how to design such countermeasures in a microcontroller.

The design space to address this problem is surprisingly large. Consider an AES algorithm written in C. The algorithm is first compiled into executable machine instructions, which run on the microprocessor. There are two different kinds of countermeasures that avoid the generation of side-channel leakage. A first set of countermeasures are algorithm-level countermeasures. Such countermeasures transform the C program so that the generation of dangerous side-channel leakage is avoided. A second set of countermeasures are architecture-level countermeasures. Such countermeasures create a better microcontroller, for example using special circuit techniques, so that no side-channel leakage is generated.

Algorithm-level countermeasures are attractive to microcontroller software programmers, because they do not require special hardware. Instead, they apply a clever transformation of the C program, such that the side-channel leakage of the C program bears no relation to the embedded secret key. A common technique

is the use of randomization, which involves mixing unknown random values in the execution of the C program. For example, the operations of the C program can be masked in a reversible manner [9–11]. Another technique randomizes the time at which a sensitive value appears [12], so that it becomes difficult to get a stable side-channel trace. An attacker who does not know the exact value of the random numbers can no longer harvest useful side-channel leakage. Randomization techniques become ineffective if an attacker can undo the effect of the randomization [13], or even simply guess the value of the random numbers with a better than 50% probability [14]. Moreover, the application of masking is algorithm specific, and the transformations often require a detailed understanding of the crypto-algorithm.

Architecture-level countermeasures are independent of the crypto-algorithm, but instead rely on a correct-by-construction principle: if a single logic gate can be adequately protected then an entire circuit build using such gates will be protected as well. Architecture-level countermeasures are based on specialized hardware or microarchitecture modifications. Most research so far has addressed this using special circuit styles in hardware [15–18]. Such circuit styles aim to make the power consumption data-independent (hiding) or to make it appear random (masking). However, all of them are characterized by a large overhead in area, power, and performance. Compared to unprotected circuits, their area and power is at least two or three times larger.

This section will cover a middle-ground between algorithm-level countermeasures and architecture-level countermeasures. As will be shown, it is possible to use one of the well-known generic circuit techniques as an algorithm-level countermeasure. The advantage of algorithm-level countermeasures over architecture-level countermeasures is that the former are more flexible, and can be deployed as needed. A detailed description of this technique, as well as experimental validation results, may be consulted in [19, 20].

11.4.1 A Circuit-Level Implementation of the Hiding Countermeasure

This section will first describe an architecture-level technique to suppress the generation of side-channel leakage. Later, it is shown how to build the equivalent of this technique in software. The technique is called Dual Rail Precharge, or DRP for short. Figure 11.7 explains the operation of DRP using a NAND gate. In this example, static and dynamic power dissipation of a logic gate is approximated through the Hamming Weight and Hamming Distance of its output, respectively. In the case of a single NAND gate (Fig. 11.7a), the static and dynamic power dissipation depend on the input values of the gate. For example, if the static power is 0, both inputs must be 1. This side-channel leakage is the basis for SCA.

Figure 11.7b shows the same test case on a DRP NAND gate. In this case, the circuit encodes each logic value with a complementary pair (A_p, A_{pbar}).

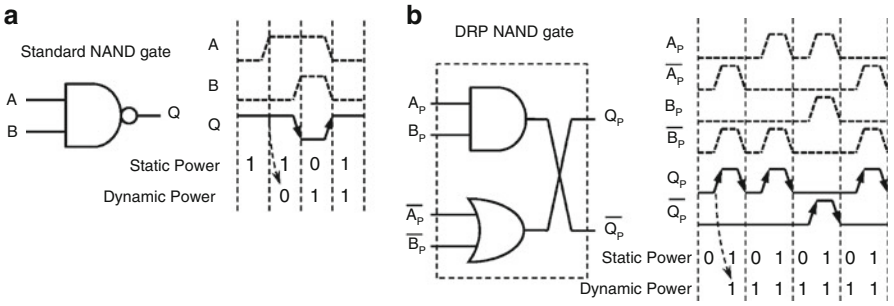


Fig. 11.7 (**a**) A CMOS standard NAND has data-dependent power dissipation; (**b**) A DRP NAND gate has a data-independent power dissipation

Furthermore, each pair is precharged to (0,0) in each clock cycle before evaluation. As a result, each clock cycle, every DRP signal pair shows exactly one transition from 0 to 1 and another one from 1 to 0. The resulting static and dynamic power dissipation are now independent of the input values of the DRP gate.

The DRP technique has been broadly used in hardware as secure circuits, for example in SABL [21], in WDDL [22], and in MDPL [23]. However, it is not possible to directly carry over the DRP technique into software. The major reason for this is that DRP requires the execution of the direct and complementary datapaths in parallel. Regular microcontrollers do not have such complementary datapaths.

11.4.2 VSC: Porting DRP into Software

This section shows how a microcontroller should be modified in order to support DRP. The concept is illustrated in Fig. 11.8. Similar to the DRP circuits, the processor has two parts: one part performs direct operations and the other part performs complementary operations. Such a processor is called a *balanced* processor. Every direct operation in the first part has a complementary counterpart in the second part. A direct operation takes input *in* and generates output *out*. The complementary operation takes input in_{bar} and generates output out_{bar}. $in_{bar} = NOT(in)$ and $out_{bar} = NOT(out)$. *NOT* means the bit-wise inversion operation.

The balanced processor has a set of instructions, referred to as balanced instructions. Each balanced instruction chooses a pair of complementary operations from two parts of the processor and executes them at the same time. The cryptographic algorithm is programmed with this set of balanced instructions. Therefore, the cryptographic algorithm has both a direct execution path and a complementary path. To run the cryptographic algorithm, both direct and complementary plaintext (balanced input) are supplied to these two paths. Finally, the algorithm generates both the direct and the complementary encryption results (balanced outputs), shown in Fig. 11.8a.

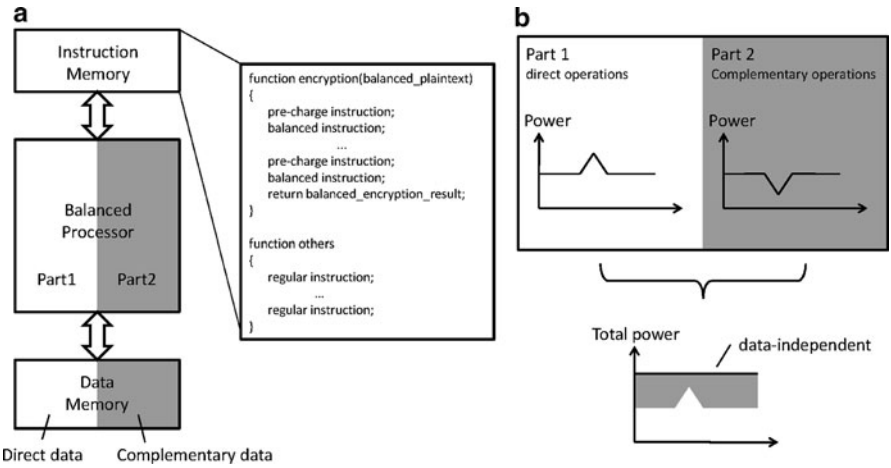


Fig. 11.8 (**a**) Concept of balanced processor and VSC programming; (**b**) The balanced processor does not show side-channel leakage

Besides the balanced instructions, the processor also has another set of instructions that perform the precharge operations in the same way as the DRP circuits. Before executing balanced instructions, a set of precharge instructions first clears the execution datapath. Following that, the balanced instruction finishes the calculation.

The earlier concept leads to a software version of DRP circuit. For a balanced instruction, similar to the DRP gate shown in Fig. 11.7, the power dissipation from the direct operation always has a complementary counterpart from the complementary operation. The sum of these two is a constant, shown in Fig. 11.8b.

11.4.3 Implementing VSC

Given the concept of DRP, and the software counterpart, VSC, the next step is to define how to write VSC software. Such software needs to be written using a set of balanced instructions. Each balanced instruction executes a pair of complementary operations. To make the balanced datapath compatible with the regular datapath, the width of either the direct or the complementary datapaths is designed to be half of the regular datapath's width. The following instructions can be easily included in the VSC balanced instruction set.

- Among the logic operations, NOT is the complementary instruction for itself. This means that if half of the input is direct value and the other half is complementary value, the output of NOT is still a pair of complementary values. Therefore, the regular NOT instruction can also be used as the balanced NOT instruction.

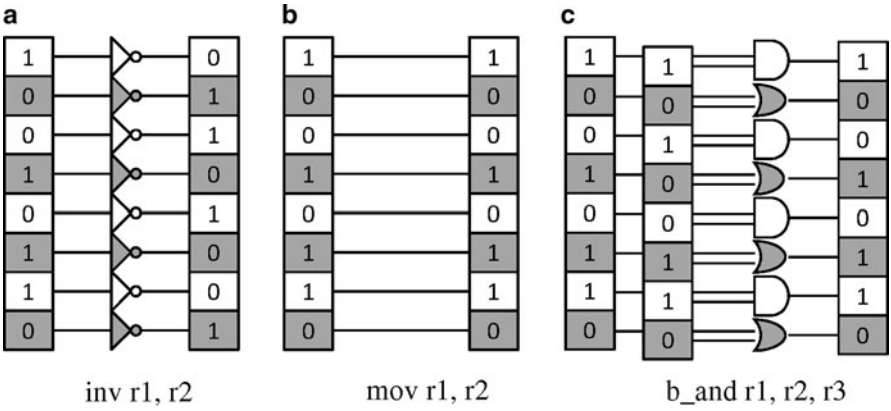


Fig. 11.9 (**a**) Regular INV can be used as a balanced INV; (**b**) Regular MOV can be used as a balanced MOV; (**c**) Balanced AND instruction uses half AND operators and half OR operators

- The AND operation has OR operation as complementary counterpart and vice versa. Therefore, the balanced AND instruction (b and) should be half AND for the direct input and half OR for the complementary input. Similarly, the balanced OR instruction (b or) should be half OR and half AND, shown in Fig. 11.9. With balanced AND, OR, and NOT, any complex logic function can be realized.
- Shift and data-move instructions are similar. They both move the programs' intermediate values from one storage location to another. As "move" operations do not change the values of the operands, their complementary counterparts are themselves. Similar to NOT instruction, shift and data-move instructions are shared by both balanced and regular instructions, shown in Fig. 11.9.
- Precharge operations do not need special instructions. To precharge a balanced instruction in VSC, simply set the input operands of different balanced instructions to all-0 before any active operation.

As will be shown, this set of instructions (AND, OR, NOT, data-move) is sufficient to implement symmetric-key cryptography. Hence, there is no need to create complementary versions of the arithmetic instructions. In addition, for symmetric-key cryptography, it is reasonable to assume that the control flow instructions in a VSC do not show sensitive side-channel leakage.

11.4.4 Mapping AES onto a VSC

To implement AES as a VSC, it needs to be broken into elementary AND, OR, and NOT operations. It turns out that this technique is common for high-performance cryptographic software, and is called bit-slicing. The concept of bit-slicing for symmetric key ciphers was first introduced by Biham et al. for the DES algorithm

[24]. In bit-slicing, an algorithm is broken down into bit-level computations (AND, OR, NOT, XOR) and each of these bit-level operations is implemented as a machine instruction. For example, a bit-level AND operation will be implemented with an AND instruction. As the word-length of a processor is n bits (e.g., 32 bits), this implies that n parallel copies of the same bit-level AND operation can be evaluated in parallel. Hence, using bit-slicing, n parallel copies of the algorithm can be evaluated in parallel. The suitability of an algorithm for bit-slicing depends on the hardware-level complexity of the algorithm. Essentially, bit-slicing treats an algorithm as netlist of combinational logic, and simulates this netlist using processor instructions.

Bit-slicing of AES has been discussed extensively in literature [25–29]. There are two motivations for it: bit-slicing improves performance, and it provides timing based side-channel resistance. The latter property is achieved when lookup tables (memory-load operations) are replaced with computations (logical operations). In that case, data-dependent cache-behavior is avoided, resulting in AES constant-time execution. However, the VSC method applies bit-slicing for the purpose of obtained a constant-power (VSC) version of AES.

To implement a VSC using bitslice programming, one should map the direct and complementary copies of an algorithm into alternate bit slices. The concept of this transformation is illustrated in Fig. 11.10. The shaded register locations in this figure indicate values of interest. A regular software program consists of two AND operations on 3-bit values. The bitslice transformation, in Fig. 11.10b, turns these operations "on their side," so that three AND operations on 2-bit values are obtained. Finally, the VSC version of this program will insert one complementary version for each bit-slice. Hence, the VSC program in Fig. 11.10c works on 4-bit values.

11.4.5 Experimental Results

The section demonstrates the VSC programming method using a prototype in FPGA. In order to implement the balanced AND and OR instructions, a Leon-3 embedded processor is customized. The prototype was realized on a FPGA board with a Xilinx Spartan 3E-1600 FPGA. The balanced Leon3 processor was implemented with FPGA resources, clocked at 50 MHz, while the software applications under test were stored in the on-board DDR2 SRAM memory.

With the earlier measurement results, a SCA was mounted called Correlation Power Attack (CPA) [30]. In the experiments, CPA focused on the outputs of the SubBytes step (16 bytes) in the first round of the AES algorithm. Figure 11.11 shows an example of the attack results on an unprotected AES with 5,120 measurements. A clear correlation peak can be distinguished.

The same attack on the VSC version of the AES implementation is shown in Fig. 11.12. In this case, 5,120 traces are not sufficient to clearly distinguish the correct trace. Hence, VSC-AES remains unbroken using this amount of

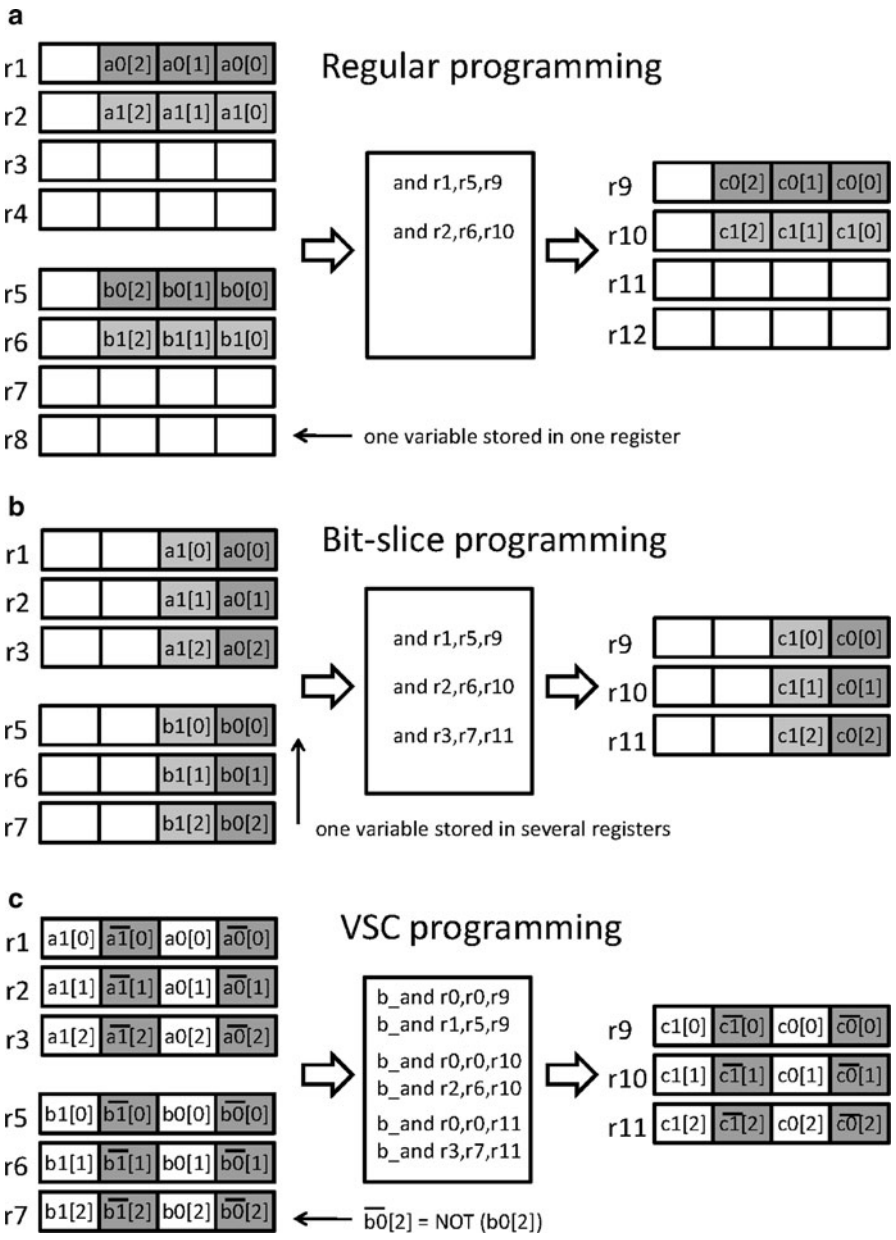


Fig. 11.10 (**a**) Regular programming: example with two AND instructions; (**b**) Bit-sliced version of the program of (**a**); (**c**) VSC version of the bit-sliced program of (**b**)

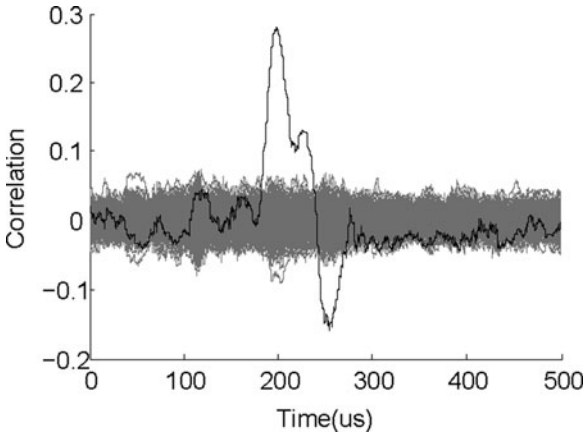


Fig. 11.11 Attack result on unprotected AES using 5,120 measurements

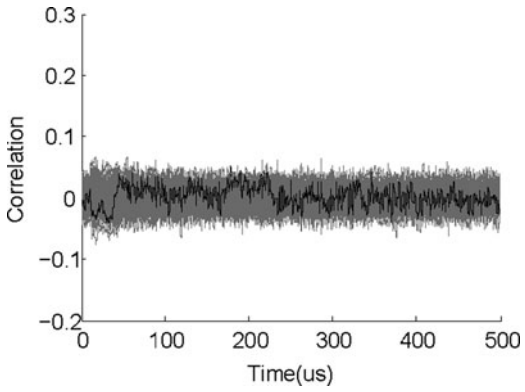


Fig. 11.12 Attack result on VSC-protected AES using 5,120 measurements

measurements. The authors experimentally verified that at least 25,600 traces are needed in order to uncover all key bytes for VSC-AES, while only 1,280 traces are needed to uncover all key bytes for the unprotect AES. Therefore, one can conclude that VSC-AES increases the measurements to disclosure (MTD) with a factor of 20 over the unprotected software. Detailed measurement results of this experiment can be consulted in [19].

11.5 Summary

This chapter discussed the problem of side-channel leakage in the context of embedded microcontroller. Just as side-channel leakage affects cryptographic hardware, it also presents a risk to cryptographic software. A side-channel analysis setup is not hard to build, and it's a practical and effective method of cryptanalysis. The design of countermeasures is therefore an important and vital area of research. The chapter demonstrated an example countermeasure to suppress the generation of side-channel leakage. The method defines only two new instructions on a micro

controller, yet is able to reduce the amount of side-channel leakage with a factor of 20 compared to an unprotected circuit. Future generations of microcontrollers will contain such specialized instructions as a defense against implementation attacks, next to specialized instructions that are intended to improve performance.

References

1. Mangard S, Oswald E, Popp T (2007) Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Berlin
2. Prouff E, Rivain M (2009) Theoretical and practical aspects of mutual information based side channel analysis. In: ACNS'09: Proceedings of the 7th International Conference on Applied Cryptography and Network Security, Springer-Verlag, Berlin, pp 499–518
3. Kocher PC, Jaffe J, Jun B (1999) Differential power analysis. In: CRYPTO, pp 388–397
4. Brier E, Clavier C, Olivier F (2004) Correlation power analysis with a leakage model. In: CHES, pp 16–29
5. Archambeau C, Peeters E, Standaert F-X, Quisquater J-J (2006) Template attacks in principal subspaces. In: CHES, pp 1–14
6. Chari S, Rao JR, Rohatgi P (2003) Template attacks. In: CHES'02: 4th International Workshop on Cryptographic Hardware and Embedded Systems, Springer, London, UK, pp 13–28
7. Rechberger C, Oswald E (2004) Practical template attacks. In: WISA, pp 440–456
8. Daemen J, Rijmen V (2002) The Design of Rijndael. Secaucus, NJ, USA: Springer, New York, Inc.
9. Akkar M-L, Giraud C (2001) An implementation of DES and AES, secure against some attacks. In: CHES 2001, vol LNCS 2162, pp 309–318
10. Oswald E, Schramm K (2005) An efficient masking scheme for AES software implementations. In: WISA 2005, vol LNCS 3786, pp 292–305
11. Gebotys CH (2006) A split-mask countermeasure for low-energy secure embedded systems. ACM Trans Embed Comput Syst 5(3): 577–612
12. Ambrose JA, Ragel RG, Parameswaran S (2007) rijid: Random code injection to mask power analysis based side channel attacks. In: DAC 2007, pp 489–492
13. Clavier C, Coron J-S, Dabbous N (2000) Differential power analysis in the presence of hardware countermeasures. In: CHES 2000, vol LNCS 1965, pp 252–263
14. Schaumont P, Tiri K (2007) Masking and dual-rail logic don't add up. In: Cryptographic hardware and embedded systems (CHES 2007), vol 4727 of Lecture Notes on Computer Science. Springer, Berlin, Heidelberg, New York, pp 95–106
15. Tiri K, Verbauwhede I (2004) A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004, vol 1, pp 246–251, February 2004
16. Popp T, Mangard S (2007) Masked dual-rail pre-charge logic: DPA resistance without the routing constraints. In: Cryptographic Hardware and Embedded Systems – CHES 2007, vol 4727. Springer, Berlin, Heidelberg, New York, pp 81–94
17. Suzuki D, Saeki M, Ichikawa T (2007) Random switching logic: a new countermeasure against DPA and second-order DPA at the logic level. IEICE Trans 90-A(1): 160–168
18. Popp T, Kirschbaum M, Zefferer T, Mangard S (2007) Evaluation of the masked logic style MDPL on a prototype chip. In: CHES'07: Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, New York, pp 81–94
19. Chen Z, Sinha A, Schaumont P (2010) Implementing virtual secure circuit using a custom-instruction approach. In: Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems, CASES'10. ACM, New York, NY, USA, pp 57–66

20. Chen Z, Schaumont P (2010) Virtual secure circuit: porting dual-rail pre-charge technique into software on multicore. Cryptology ePrint Archive, Report 2010/272. http://eprint.iacr.org/
21. Tiri K, Verbauwhede I (2003) Securing encryption algorithms against DPA at the logic level: next generation smart card. In: CHES 2003, vol LNCS 2779, pp 125–136
22. Tiri K, Verbauwhede I (2004) A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: Proceeding of DATE 2004, vol 1, pp 246–251
23. Popp T, Mangard S (2005) Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In: CHES 2005, vol LNCS 3659, pp 172–186
24. Biham E (1997) A fast new DES implementation in software. In: FSE'97: Proceedings of the 4th International Workshop on Fast Software Encryption. Springer, London, UK, pp 260–272
25. Konighofer R (2008) A fast and cache-timing resistant implementation of the AES. In: CT-RSA'08: Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology. Springer, Berlin, pp 187–202
26. Matsui M (2006) How far can we go on the x64 Processors? In: FSE, pp 341–358
27. Kasper E, Schwabe P (2009) Faster and timing-attack resistant AES-GCM. In: CHES, pp 1–17
28. Matsui M, Nakajima J (2007) On the power of bitslice implementation on Intel Core2 processor. In: CHES'07: Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, New York, pp 121–134
29. Rebeiro C, Selvakumar D, Devi A (2006) Bitslice implementation of AES. In: CANS, vol LNCS 4301, pp 203–212
30. Brier E, Clavier C, Olivier F (2004) Correlation power analysis with a leakage model. In: CHES 2004, vol LNCS 3156, pp 16–29

Chapter 12
Security for RFID Tags

Jia Di and Dale R. Thompson

12.1 Introduction

Radio frequency identification (RFID) is an automatic identification method for retrieving and accessing data using devices called RFID tags, sometimes called transponders. The basic RFID system includes tags, readers, and associated interfaces, as shown in Fig. 12.1. Tags are attached to objects that we want to identify. Readers query the tags using a radio frequency (RF) signal to obtain an identifier. RFID applications include item management, physical access control, travel documents, finance and banking, sensors, animal tracking, human identification, and product counterfeiting countermeasure.

The dominant requirement in most passive RFID systems is the low-cost tag. Therefore, there are challenging power and design constraints for a passive RFID tag that provides not much more than a unique identifier when queried by a reader. The use of passive RFID tags in a system that requires security is even more challenging because the relative hardware overhead for implementing modern cryptographic algorithms on a tag to enable encrypted communication and authentication is too high [1]. This has led researchers to search for low-cost cryptographic algorithms, authentication protocols, or other novel techniques.

12.1.1 History of RFID

The first application of RFID can be traced back to the 1930s when US and Britain attached Identify Friend or Foe systems to airplanes, which were active beacons to identify friendly aircraft during war. Such automatic identification was restricted

J. Di (✉) · D.R. Thompson
Computer Science and Computer Engineering Department, University of Arkansas, Fayetteville, Arkansas, USA
e-mail: jdi@uark.edu; d.r.thompson@ieee.org

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_12, © Springer Science+Business Media, LLC 2012

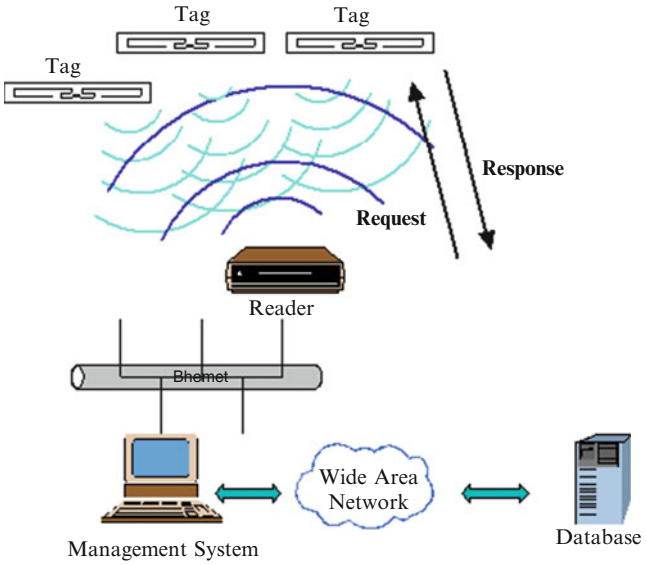


Fig. 12.1 RFID system

to large and expensive systems for many years until 1998 when MIT found it was easier for robots to navigate if the objects in the area could identify themselves. They designed low-cost RFID tags that had no battery, no transmitter, and had simple circuits. Instead of a battery, the tags harvest the direct energy from the RF signal of a reader to operate the simple circuits. Instead of an on-board transmitter, the tags reflect a modified version of the RF signal sent by the reader back to the reader to convey information, referred to as backscattering communication.

Other researchers realized that identifying all objects with RFID tags had commercial use in identifying retail products in the supply chain, which is the network that brings a product to market. They named the identifier the Electronic Product Code (EPC). MIT created the Auto-ID Center in 1999 and the 900 MHz range was chosen as the best overall frequency for use in retail based on cost, read range, and capability. The Auto-ID Center was supported by Proctor & Gamble and Wal-Mart joined in 2001. The Auto-ID Center outgrew the academic environment and EPCglobal Inc. formed in 2003 as a joint venture between UCC and EAN (UCC and EAN is now merged into GS1). EPCglobal Inc. is a nonprofit corporation that leads supply chain standards in RFID [2].

12.1.2 Internet of Things

Merging the physical world with the cyber world has been a dream of researchers that is enabled by RFID. In 2005, the ITU issued a report proposing the Internet of Things in which devices had embedded sensors and transceivers that can be

uniquely identified and can communicate with individuals and other devices [3]. The required technologies include a system to uniquely identify things (like RFID), sensors to detect environment conditions, embedded intelligence, a network, and miniaturization. The devices will collect information and will be queried remotely with a wireless signal. The sensors and transceivers will become less and less visible like the infrastructure for cell phones. Some of the major challenges for the Internet of Things are standardization and protection of data and privacy. Identity matters in the Internet of Things and RFID will be one of the requirements.

12.1.3 RFID Applications

The most successful RFID application is item management, which includes the supply chain, logistics, inventory control, and equipment management. The supply chain is the network that brings a product to market and includes suppliers, manufacturers, shippers, storage facilities, distributors, and retails. Logistics is the management of a supply chain. Inventory control is the process of ordering, receiving, controlling, and maintaining the correct amount of each item in stock, which requires tracking the amount and type of stock. Equipment management is tracking the repair and maintenance of equipment as well as charging its usage to a particular job. All of these areas within item management use RFID to track the flow of objects through a system in a contactless and automatic way.

RFID provides visibility to the supply chain. Visibility refers to the added detail of providing what, when, and where for each tagged item as it travels from the manufacturer to the retailer. With more visibility, businesses can identify bottlenecks in the system, correct them, and save money. The benefit of visibility in the supply chain led to two different mandates that accelerated the introduction of RFID. First, the U.S. Department of Defense (DoD) mandated tagging expensive cases by 2005. DoD was already using RFID for tracking shipping containers and recognized the benefit of using RFID for expensive cases inside the containers. Then, in June 2003 Wal-Mart requested that their top 100 suppliers use RFID at the pallet and case level by January 2005. Wal-Mart continues to request that suppliers expand their use of RFID and in July 2010 announced that it would be placing removable RFID tags on jeans and underwear to optimize inventory.

Another RFID application is physical access control, which includes unlocking and starting an automobile, authorizing and charging for access to toll roads, unlocking the door to a building, or entering a prepaid parking garage. RFID tags embedded in automotive keys provide an additional form of authentication, the unique tag identifier, to help reduce car thefts. Semipassive RFID tags are used to identify and charge automobiles on toll roads. Badges with RFID tags are commonly used to control access to a building or entrance to a paid parking garage.

In 2006, the U.S. began issuing electronic passports (e-passport) that have embedded RFID tags. The industry prefers calling them smart cards instead of the term RFID because they are high-end RFID tags that have a secure embedded

Fig. 12.2 PayPass Key Fob

microcontroller capable of performing cryptographic algorithms and protocols [4]. The smart cards are also designed to hold digital representations of biometrics such as a digital photograph of a face, a fingerprint, or retina.

A low-cost alternative to passports to meet the requirement of the Western Hemisphere Travel Initiative called the DHS Peoples Access Security Services card (PASS card or passport card) contains a passive ultra-high frequency (UHF) RFID tag [5]. The PASS card is issued to US citizens that can use it to enter the US from Mexico, Canada, the Caribbean, and Bermuda. The UHF tag has a nominal read range of 20 feet unlike the e-passport that has a nominal read range of four inches. One can imagine that an individual with a UHF tag could cross the border in a car easier than using a tag with a smaller read range. In addition, RFID tags will be in future state-issued driver licenses. The Real ID Act of 2005 established guidelines for state-issued driver licenses and ID cards to be accepted as valid forms of identification [6].

The finance and banking industries are using RFID. Many major credit cards are now offering embedded RFID tags based on the smart card standards. Instead of handing over the credit card, the user can wave the card or a key fob (shown in Fig. 12.2) near a reader in front of the cash register to pay for items. The contactless payment method appears to be much faster than the using the magnetic strip.

Sensor tags that combine sensors and RFID tags are useful in certain applications. For example, KSW Microtec has a sensor tag that combines a temperature sensor with a tag. The tag has a battery and is programmable. The tag is programmed to wake up at certain time intervals and log the temperature into the nonvolatile memory. The tag will continue to take measurements until the battery is drained. However, the data remains in the memory and can be retrieved even after the battery is drained because the sensor tag has a passive embedded tag that can harvest the energy from the reader for communication.

RFID is used for tracking livestock such as cows and pets. Like in the supply chain, RFID provides visibility of where the animals have been. Today, it is difficult to trace an individual animal as they are bought and sold. Therefore, it is difficult to determine the origin of disease. If all livestock were tagged then any disease outbreaks could be traced to an individual farm instead of shutting down a large number of farms. It is common to tag pets such as dogs with an RFID tag that is

Fig. 12.3 Pet chip

inserted with a special syringe under the skin near the neck, as shown in Fig. 12.3. The tag is encased in a glass capsule and the capsule has a special chemical that bonds with the skin so that the tag does not move. The unique identifier is logged in a database so that the pet can be identified and the owner contacted.

VeriChip sells a human-implantable RFID tag operating at about 134 KHz because at these frequencies the RF signal can penetrate mud, blood, and water. It is about the size of an uncooked grain of rice and appears to be very similar, if not identical, to the chip implanted in pets. In 2002, the US Food and Drug Administration ruled the VeriChip was not a regulated device and in 2004 ruled the serial number in the tag could be linked to healthcare information. The healthcare applications marketed by VeriChip include implanted medical device identification, emergency access to health information, portable medical records access including insurance information, in-hospital patient identification, medical facility connectivity via patient, and disease/treatment management of at-risk populations such as vaccination history. The VeriChip was used to tag bodies after a hurricane to prevent human error and to log the location of the body [7].

Counterfeiting is a $600 billion per year problem that occurs in industries such as automotive, apparel, luxury goods, pharmaceuticals, food, software, and entertainment [8]. Tagging items with RFID has been proposed to mitigate counterfeiting. For example, the Food and Drug Administration estimates that 10% of all pharmaceutical drugs in the US are counterfeit. Therefore, it has been proposed to use RFID to create an electronic pedigree (e-pedigree) for pharmaceutical drugs. A tag would be embedded in the bottle at the manufacturing site. Then, the drug could be automatically tracked from the manufacturer, to the wholesaler, and finally to the pharmacy to create a record or e-pedigree. This would help reduce the introduction of counterfeit drugs into the system.

12.1.4 RFID Reference Model

It is useful to group different functions of an RFID system together in a six-layer RFID Reference Model shown in Fig. 12.4. The RFID Reference Model consists of an application layer, middleware layer, network layer, reader layer, media interface layer, and tag layer. The application layer refers to the different uses of RFID such as item management or anticounterfeiting. The middleware layer refers to the software that translates and/or filters data from the reader to the application. For example, this includes software that translates the output of different model readers to a standard format for processing by the application. The network layer refers to the

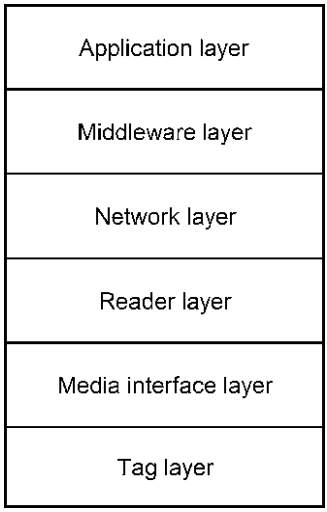


Fig. 12.4 RFID reference model

communication pathway between the reader and the application residing on a server. It most often uses standard communication protocols such as Ethernet, WiFi, and Internet protocols. The reader layer refers to the architecture of a reader. A reader is a computer and transceiver combined into one package connected to one or more antennas. The media interface layer refers to the way the reader controls access to the media, where the media is most often wireless. The media interface is complex because a reader must use the wireless channel to coordinate communication to each tag in a tag group. For example, the EPCglobal Class 1 Generation 2 (Gen 2) protocol includes a random access protocol that is similar to Ethernet [2]. The tag layer refers to the architecture of the tag including the power harvesting circuit, modulator, demodulator, logic, and memory layout.

12.1.5 Types of RFID Tags

There are three categories of RFID tags: passive, semipassive, and active. Passive RFID tags used for item identification are low-cost because there is no battery or transmitter, and they have simple circuits. Instead of a battery, the tags harvest the direct power from the RF signal of a reader to operate the on-tag circuits. Instead of a transmitter, the tags module and reflect the energy from the reader. This is called backscattering communication. The most common identifier for item identification is the EPC that was standardized by EPCglobal Inc. [2]. Semipassive tags have a battery for powering the circuit but use passive-style backscattering to communicate with the reader. The advantage of having the battery is the increased sensitivity of the circuit that increases the read range and the faster turn-on time. Therefore, they are used in applications such as toll road payments so that the readers can read

the tag in the window of a fast moving car. Not having a transmitter, but reflecting the energy using backscattering extends the life of the battery. Finally, there are active tags, which have a battery and transmitter. The battery powers the circuits and transmitter. These tags are more expensive than the passive or semipassive tags and have the disadvantage of having a battery that reduces the lifetime. However, having a battery and transmitter makes them more sensitive and able to transmit their identifier further.

12.1.6 Social and Privacy Implications of RFID

There are important social and privacy implications of RFID. Imagine a world where every item around you and that you carry or wear has sensors, is uniquely identifiable, and can be queried over the Internet. The good aspects of such a transparent society are that missing children would be found, many crimes would be solved, and we would be able to locate friends and family easily. The bad aspects would be that the government could watch us and we would have little privacy. The main privacy threat by RFID is that it enables tracking, profiling, and surveillance of individuals on a large scale. This is what the public fears, whether it is justified or not. RFID privacy threats can be divided into tracking, tracing, hotlisting, and profiling. Tracking is determining where individuals are located. Tracing is determining where individuals have been. Hotlisting is singling out certain individuals because of the items they possess. Finally, profiling is identifying the items an individual has in their possession.

12.2 Security Attacks to Passive RFID Tags

Ever since the introduction of RFID, security has been a major consideration in applications employing various RFID systems. For passive tags, their stringent cost limitation and power budget make it unrealistic for complex digital and analog electronics to be integrated on board. Although many attempts have been made to implement strong cryptography in tag's hardware, such tags are rarely adopted by most prevailing RFID applications. Moreover, it has been demonstrated that even having a hardware-based cryptographic module on tag is not sufficient for securing a RFID system [9]. Due to the relatively simple on-tag circuits and the wireless (or contactless) communication nature, RFID systems have a large amount of vulnerabilities and are susceptible to a wide range of malicious attacks, threatening their confidentiality, integrity, and availability. The targets of these attacks can be the reader, the tag, the communication protocol, the middleware, or the database. In this chapter, various security attacks to RFID tags are discussed, with special focus on hardware-based attacks. In addition, excellent surveys can be found in [10, 11].

12.2.1 Attacks for Impersonation

In an impersonation attack, the adversary tries to imitate the identity of the target tag in order to communicate with the reader as if the reader is accessing the authentic tag. Four major attacks are in this category: tag cloning, tag spoofing, relay attack, and replay attack.

12.2.1.1 Tag Cloning

An adversary can easily copy the memory content (including the unique identifier) of an authentic tag to create an identical yet cloned tag because the EPC Class I tags have no mechanism for preventing cloning. In simple passive RFID systems, cloned tags are indistinguishable from authentic ones. Even for tags with extra security features, adversaries are still able to fool the reader to accept the clone as a legitimate one through more sophisticated attacks [10]. Many tag cloning practices have been published in the literature including the demonstration of cloning e-passports. It is apparent that having two tags with same identifiers will confuse the system and jeopardy its integration.

12.2.1.2 Tag Spoofing

Tag spoofing, or tag emulation, can be considered as a variation of tag cloning, as both attacks try to gain access to the service using an illegitimate device. Different from tag cloning where a physical reproduction of the authentic tag is made, tag spoofing may use a custom designed electronic device to imitate, or emulate, the authentic tag. Same as tag cloning, spoofing attacks also threaten the integrity of RFID systems, e.g., an emulated tag can be used to fool the automated checkout system into thinking a product is still on a shelf. Tag spoofing requires the adversary to have full access to legitimate communication channels as well as knowledge of the protocols and secrets used in the authentication process if any [11].

12.2.1.3 Relay Attack

Passive RFID tags can only communicate with the reader in close proximity ($<\sim$25 feet). Therefore, when a reader successfully accesses a tag, it assumes the tag is within its reading range. This assumption, however, can be utilized by an adversary to create a "virtual clone" through relay attacks, thereby gaining access to all transactions in the communication channel.

As illustrated in Fig. 12.5, to execute a relay attack an adversary needs two devices acting as a tag and a reader, respectively. The *leech* device presents itself to the victim tag as a legitimate reader, while the *ghost* device presents itself to

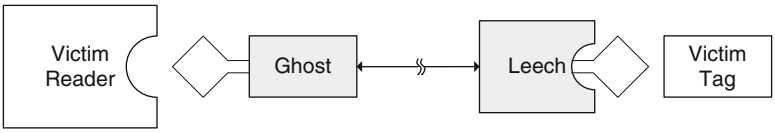


Fig. 12.5 A RFID channel under a relay attack [12]

the victim reader as a legitimate tag [12]. These two devices work as a relay, such that the messages in the authentic communication channel are captured by the leech/ghost, and are sent to the ghost/leech via alternative high-speed channel (e.g., wired communication). While the victim reader and tag are fooled into thinking they are communicating directly with each other, any data exchange between them, even if strongly encrypted and authenticated, can be carried out over the relay channel. Moreover, the illegitimate devices may modify the data during relay.

Relay attacks can be carried out across considerable distances and the relay devices are not bounded by any regulatory standard. Many successful practices have been reported, the victims of which include electronic voting machine, passive keyless entry, and e-passport.

12.2.1.4 Replay Attack

Replay attack is somewhat similar to relay attack, except in a replay attack an adversary may use the captured valid reader–tag communication data at a later time to other readers or tags for impersonation. The valid communication data can be captured through other attacks such as relay attack or eavesdropping attack. Like relay attacks, the tag data protected by strong cryptography or authentication are still vulnerable to replay attacks. As pointed out in [11], a typical scenario of replay attack is to break RFID-based access control systems.

12.2.2 Attacks for Information Leakage

Attacks in this category focus on illegitimately acquiring the data stored on the tag, causing information leakage. Such attacks include unauthorized tag reading, covert channel, eavesdropping, side-channel attack, and tag modification.

12.2.2.1 Unauthorized Tag Reading

Unauthorized tag reading is probably the simplest attack, where an adversary places an illegitimate reader within the proximity of the target tag to access the tag data.

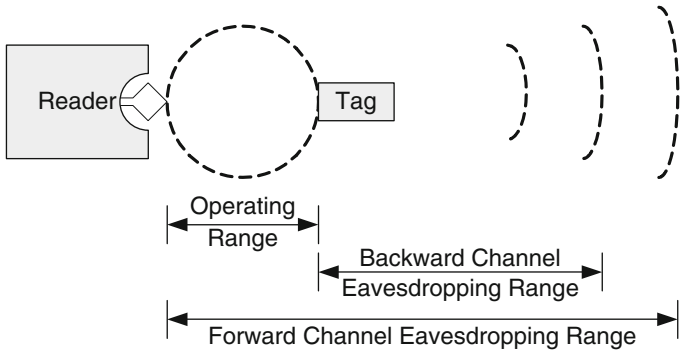


Fig. 12.6 Tag reading range classification [17]

Due to the cost and power constraints, data stored on most passive RFID tags are not protected by encryption or authentication mechanism; in addition, these tags do not have an on/off switch. Therefore, unauthorized tag reading is a simple yet effective attack.

12.2.2.2 Covert Channel

Covert channels are unintended or unauthorized communications paths that can be used to transfer information in a manner that violates system security policies. For passive RFID tags, it is possible to create covert communications channels through the use of their user-defined memory banks. An example of such attacks is to secretly report private medical or social information of a patient carrying an implanted RFID tag [10].

12.2.2.3 Eavesdropping

In an eavesdropping attack, an adversary uses an electronic device with antenna to listen to the legitimate reader–tag communication and record the messages. It can be performed in either reader-to-tag direction (forward channel) or tag-to-reader direction (backward channel). As shown in Fig. 12.6, because the transmission power of a reader is much higher than that of a tag, an adversary can capture the forward channel communication from a much greater distance (hundreds of meters [10]).

Data captured through an eavesdropping attack can be utilized in more sophiscated attacks, e.g., relay attack and replay attack. Practices on eavesdropping attacks have been reported in a number of literatures such as [13].

12.2.2.4 Side-Channel Attacks

In order to enhance the security of RFID tags, many attempts have been made trying to implement advanced cryptography on passive tags. However, having a crypto module of a secure algorithm in hardware on the tag is not sufficient for a secure RFID system [9]. Side-channel attacks refer to those target the hardware implementation of a cryptographic algorithm rather than the algorithm itself. CMOS circuits exhibit different power, delay, electromagnetic interference, and other electrical/logical characteristics while processing different data, which can be recorded by an adversary and processed by chosen statistical algorithms to amplify such differences in order to "guess" the cryptographic key. It has been reported that side-channel attacks were able to defeat most, if not all, standardized cryptographic algorithms, e.g., AES, DES, RSA, and ECC. Popular side-channel attacks include Differential Power Analysis, Timing Analysis, Differential EM Attack, and fault-injection attack. In addition, Differential Frequency Analysis was introduced in [9] for mitigating the randomization countermeasures in passive RFID tags. The wireless nature of RFID systems substantially facilitates the implementation of side-channel attacks as an adversary is able to record the information-leaking electrical characteristics from a distance. Several successful attack practices have been reported [14].

12.2.2.5 Tag Modification

Tag modification is somewhat different from all earlier-mentioned information leakage attacks in that it aims at modifying the data stored on the tag. Various consequences may result from such attacks. For instance, an adversary may wipe out the price stored on the tag attached to an expensive product in a store and write a much cheaper price to it. Another more severe scenario is the required medicine information of a patient stored on a tag is maliciously modified. Passive RFID tags contain user-writeable memory, which can be exploited by an adversary with a malicious reader to modify or delete its content [10]. The effectiveness of this simple attack is apparently dependent on the organization of tag data, and the WRITE protection mechanism implemented on the tag.

12.2.3 Attacks for Denial-of-Service

In a Denial-of-Service (DoS) attack targeting a RFID tag, an adversary tries to break the communication link between the tag and the reader and disable the reader from accessing the tag. Because of the wireless nature of RFID systems, attacks in this category, e.g., *KILL* command abuse, passive interference, active jamming, are relatively simple to implement.

12.2.3.1 KILL Command Abuse

Passive RFID tags usually implement a KILL command, upon executing which the tag stops responding to any further reader queries. This process is protected by a password stored on tag. In reality, a set of tags often share the same KILL password and sometimes a master-password exists for a large amount of tags. These 8- or 32-bit passwords can be obtained by an adversary through more sophisticated attacks, and then be used to issue unauthorized KILL commands, rendering the tag nonresponsive.

12.2.3.2 Passive Interference

Just like other RF signals, the RF communication link between RFID reader and tag is susceptible to interference, such as the presence of water and metal. In addition, radio waves can bounce off surfaces and collide in certain places, canceling each other out if their phases happen to be opposite at that particular point in space [11]. While these types of interference are mostly unintentional, an adversary may use foil-lined bags to shield tags from electromagnetic waves sent from the legitimate reader to block the access.

12.2.3.3 Active Jamming

Different from powerless passive interference, active jamming sends out radio signals to disrupt the reader–tag communication. This type of attack can be carried out through the use of radio noise generators, e.g., power switching supplies or electronic generators [11], radio signals in the same range as the reader, or abuse blocker tags [15] or RFID guardian [16].

12.2.4 Attacks Through Physical Manipulation

While all attacks discussed earlier can be carried out from a distance, attacks in this category require the adversary to have physical contact with the target tag. Such attacks include physical tampering, tag swapping, tag removal, tag destruction, and tag reprogramming.

12.2.4.1 Physical Tampering

The goal of physical tampering attack is to obtain/modify the data stored on tag. These memory-oriented attacks can be done through microprobing, focused ion beam editing, fault injection, and laser cutter microscopes, after the tag is taken

apart [11, 17]. Although such attacks are costly and time-consuming, they are able to bypass all security checks at logical-level, e.g., password and authentication.

12.2.4.2 Tag Swapping, Removal, and Destruction

These three attacks are very simple: tag swapping is to remove the RFID tag from one object and attach it to another one; tag removal is to remove a tag from the associated object; tag destruction is to physically disable the tag through the use of chemicals, the application of excess pressure or tension, or just taking off the antenna [11]. All these attacks take advantage of the poor tag-object adherence, or the physical security of the tag.

12.2.4.3 Tag Reprogramming

Tag reprogramming attack targets those tags that are reprogrammable either through RF or wired interface [11]. An adversary is able to program a tag to either create a clone or cause inconsistency between the data on tag and the data in the backend database. Although such programmable feature is usually protected by passwords, an adversary can acquire these passwords through more sophisticated attacks such as side-channel attacks. An example of tag reprogramming attack can be found in [18].

12.3 Existing Protections for RFID Tags

Ever since the threats and attacks to RFID tags were identified, researchers from industry and academia have been working on various protection mechanisms. In this section a variety of existing protections for RFID tags are introduced, organized by the corresponding attack categories as in the previous section.

12.3.1 Attacks for Impersonation

12.3.1.1 Tag Cloning and Spoofing

Cloning attacks can be mitigated via challenge-response authentication protocols [10]. As the limited on-tag hardware resources lead to weak authentication protocols that are insufficient against these attacks, several other authentication methods have been proposed. Note that the spoofing attacks can be combated in the similar manner.

Physical Unclonable Function (PUF), proposed in [19], exploits the physical characteristics of the silicon and the IC manufacturing process variations to uniquely

characterize each and every silicon chip. A PUF is a function that maps a set of challenges to a set of responses based on an intractably complex physical system [19]. For example, two identical datapaths on a tag laid out in exactly the same length, which should have exactly the same delay, inevitably exhibit different delays from one tag to another due to manufacturing variations. Such timing difference can be captured to create a configuration bit. If multiple configuration bits can be created on a tag, which are unique from tag to tag, these bits can either be used directly to authenticate a tag or can serve as a secret key for cryptographic operations [19]. Such unique configuration or identity is impossible to predict, control, or duplicate as it is based on the physical uncertainty of the silicon IC. The PUF-based authentication consists of an enrollment phase carried out by a trusted party, during which a set of chosen challenges are applied to the tag and the corresponding responses are obtained from the tag. These challenge-response pairs are stored in the database. During authentication, the trusted party selects a previously stored challenge and applies it to the tag, and then compares the response with the stored response to see if they match. Relay attacks can be prevented if each challenge is only used once.

Fragile watermarking [20] generates a watermark using pseudo random number generator based on a few data stored on tag. This watermark is embedded in the reader–tag communication protocol such that the reader is able to use it for tag authentication. If the unique tag IC serial number is used in generating the watermark, tag cloning can be detected. In [21] a synchronized secret method for detecting cloned tag is proposed. Every time a tag is accessed by a reader, a random number generated by the reader is written to the on-tag memory. Upon next access, both static tag identifiers and this random number are checked by the reader. As a cloned tag does not have this synchronized secret, it will be detected by the reader. Another set of techniques for clone detection is tag fingerprinting, which will be discussed in detail in Sect. 4. The major advantages of fingerprinting are: (1) the hardware and software of passive tags do not need to be modified; and (2) there is no additional cost to the tag.

12.3.1.2 Relay Attack and Replay Attack

To combat relay attacks, besides encrypting the communication channel or adding a second form of authentication which degrade the convenience and increase the cost, an effective method is to detect the distance between the reader and the tag because the shorter the distance, the harder for relay attacks to be implemented [10]. A variety of techniques can be used to measure the tag–reader distance, e.g., the round trip delay of the radio signal or the signal strength, ultra-wide band pulse communication-based distance bounding protocol, and another distance bounding protocol based on a side-channel leakage rich challenge-response mechanism involving XOR functions. Similar approaches can be used to mitigate replay attacks as the distance between tag and reader can be applied to distinguish authorized and unauthorized tags/readers. Other methods for mitigating replay attacks include

timestamps, one-time passwords, incremental sequence numbers, clock synchronization, and limiting the direction of radio signals through RF shielding.

12.3.2 Attacks for Information Leakage

12.3.2.1 Unauthorized Tag Reading, Covert Channel, and Eavesdropping

Methods for mitigating unauthorized tag reading can be classified into two categories: one is to break the reader–tag communication link when the tag is not being accessed; the other is to apply access control mechanisms to the tag. The former can be implemented through tag shielding (e.g., use aluminum-lined wallets to protect the tag), blocker tag (i.e., a device capable of simulating many RFID tags to a reader, thereby masks the existence of the actual tag), and RFID Guardian (i.e., a tool capable of acting as a personal RFID firewall that establishes a privacy zone around the user such that only authenticated readers have access to [16]). The latter can be realized through adding cryptographic transactions to the reader–tag communication protocol and permanently deactivating the tags using KILL command [10]. Limiting the access of a tag to only authorized readers also helps prevent information leakage through covert channels. Another method is to reduce the availability of the memory resource on tag, e.g., clearing the unused memory every few seconds or randomizing code and data locations [10]. Eavesdropping attack can be mitigated in similar manner. In addition, reducing the information stored on tag also degrades the effectiveness of eavesdropping attacks [11].

12.3.2.2 Side-Channel Attacks

For mitigating power-based side-channel attacks, the transient power consumption needs to be decoupled from the data being processed. This can be achieved by either power balancing or power randomization. Techniques based on balancing power fluctuation include: (1) new CMOS logic gates, which go through a full charge/discharge cycle for each data processed; (2) asynchronous circuits, especially dual-rail encoded logic, due to its fixed switching activities during each DATA-Spacer cycle; (3) modifying the algorithm execution, (4) compensating current at the power supply node, and (5) using subthreshold operation. Additionally, various techniques for randomizing power data have been proposed. Existing countermeasures for timing-based attacks include inserting dummy operations, using redundant representation, and unifying the multiplication operands.

Although some power-balancing methods also reduce EM fluctuations, masking EM variance is more difficult because of the increased difficulty in matching parasitic reactance. EM attack countermeasures include signal strength reduction

and signal information reduction. Other proposed solutions include limiting the EM emission, increasing the tag IC complexity, and the use of tamper-resistance tags such as the plusID tag [10].

For fault-injection side-channel attacks, in general, most fault-tolerant design techniques, such as temporal and spatial redundancy, can be applied to mitigate certain types of faults. These techniques include Concurrent Error Detection, error detection/correction code, modular redundancy, Built-In Self-Test, and algorithm modification.

12.3.2.3 Tag Modification and Reprogramming

To prevent an adversary from modifying the contents of on-tag memory, a simple solution is to use read-only tags. However, this solution apparently degrades the flexibility of RFID systems and limits its applications. Another solution is to adopt efficient coding schemes or cryptographic algorithms to secure the on-tag data. In addition, reader authentication methods also prevent the unauthorized readers from accessing the tag data.

12.3.3 Attacks for DoS

12.3.3.1 KILL Command Abuse, Passive Interference, and Active Jamming

One way to protect the tag from being disabled by unauthorized reader is to improve the physical security of the authorized reader–tag communication channel, e.g., by using walls opaque to relevant radio frequencies [22]. Such methods are capable of mitigating KILL command abuse, passive interference, and active jamming attacks. In addition, secure password management mechanisms, e.g., avoid the use of master password, help mitigate KILL command abuse attacks [11].

12.3.4 Attacks Through Physical Manipulation

12.3.4.1 Physical Tampering

Protecting the tag IC from physical tampering involves antitampering technologies, e.g., memory protection, chip coating, and self-destruction. These technologies are outside the scope of this chapter. Interested readers may refer to other related publications.

12.3.4.2 Tag Swapping, Removal, and Destruction

These attacks can be prevented by improving the physical resilience of the tags. Such methods include the use of strong mechanical bond or glue and embedding the tags inside the item package. One limitation of the latter approach is the fact that RF signals can be absorbed by materials like water and metal. Embedding tags inside such materials may render the tags inaccessible.

12.4 Fingerprinting RFID Tags for Anticounterfeiting

The electronic characteristics of RFID tags caused by manufacturing differences can be measured to create a fingerprint that is unique to the individual tag to prevent counterfeiting [23]. The fingerprint can be used by itself or in addition to other methods such as traditional authentication protocols. The general steps for enrolling and verifying a fingerprint to prevent counterfeiting is shown in Fig. 12.7. During an enrollment phase, features of the tag are measured to create the fingerprint. During the verification phase, the features of the tag that claims to be the authentic tag are measured and the resulting fingerprint is compared with the enrolled fingerprint using a matching algorithm. If the enrolled and measured fingerprints match, there is a high probability that it is the authentic tag.

12.4.1 Background on Fingerprinting Electronic Devices

An electronic device can be uniquely identified or authenticated by its physical fingerprint based on characteristics such as clock skews [24], transient timing responses [25, 26], frequency responses [25], and other RF characteristics [27]. In [24], it was demonstrated that an individual Ethernet computer network card could be identified from a remote location by its clock skew due to manufacturing variations in its hardware. RF fingerprinting uniquely identifies transmitters by

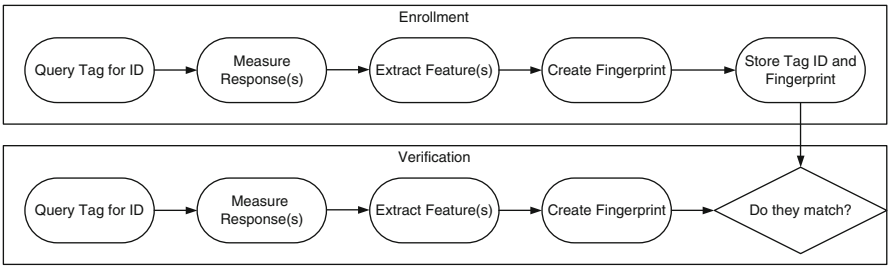


Fig. 12.7 Tag fingerprinting system

variances in RF characteristics such as amplitude, phase, and frequency caused by hardware variations, which can be used by military to uniquely identify and track a transmitter [27], by amateur radio operators to track unintentional and intentional jamming [26], and by cellular telephone companies to prevent fraud caused by cell phone cloning [27]. An intrusion detection system for Bluetooth was proposed in [25] based on RF fingerprinting.

One challenge of fingerprinting a passive RFID tag is that the tag modifies the signal sent from the reader. The work in [28] describes fingerprinting passive HF RFID tags used in E-passports to authenticate tags and detect clones. They were able to distinguish different models of tags but not different tags of the same model. They focused on measuring the transient response that is induced by the tag on the reader antenna. Passive HF RFID tags operate in the near field and use inductive coupling for communication between the reader and the tag much like a transformer. Therefore, the tag has a strong influence on the signal from the reader [28]. Because of this coupling, the induced transient was a good feature for measuring variance between different model tags. However, these HF tags operate in the near field, unlike passive UHF tags that operate in the far field. In UHF, the return signal has much less influence signal on the reader antenna. In addition, an expensive high-bandwidth oscilloscope is required to measure such signals.

12.4.2 Tag Minimum Power Response

In [23], passive UHF RFID tags were fingerprinted using tag Minimum Power Response (MPR). MPR is the set of minimum powers required for a tag to power up and respond at multiple frequencies. The MPR was measured using a bottom-up algorithm, which repeatedly sent signals from the reader to the tag starting at a low power level and increasing the power until a response from the tag was detected. Fifty same-model tags taken off one roll of tags (it is likely that the tags were manufactured at approximately the same time) were each measured six times at 101 different frequencies between 860 MHz and 960 MHz.

After measuring the MPR of 50 tags, the data was analyzed. First, a statistical technique called two-way analysis of variance (ANOVA) was used to determine if different tags had significantly different MPRs. Two-way ANOVA is a statistical procedure for testing the equality (or inequality) of means of several groups. It was applied to test the hypothesis that the MPR of tags are affected by frequency and different tags at the 1% level of significance. The results were that different tags have a significant effect on the MPR and frequency has a significant effect on the MPR.

After determining that the MPR of different tags are significantly different, the authors then use the tag MPR as a fingerprint in an identification system. The K Nearest Neighbor (KNN) was used as the matching algorithm to identify a given tag based on its previously enrolled MPR. This was performed ten times using a technique called stratified tenfold validation to validate the results. The identification system effectiveness was measured using the True Positive (TP) Rate,

Table 12.1 Classification of tags based on tag minimum power response

TP rate	FP rate	ROC AUC
94.4%	0.1%	0.999

False Positive (FP) Rate, and the Area under the Receiver Operating Characteristic Curve (ROC AUC). A True Positive is an instance where the measurement of a tag was positively associated to the correct tag. A False Positive is an instance where a measurement was positively associated to an incorrect tag. An ROC curve is a plot of the fraction of the True Positives against the fractions of False Positives. The area under the curve represents the probability of a randomly chosen positive matching instance being ranked higher than a randomly chosen incorrect instance. As shown in Table 12.1, the identification system had an average True Positive Rate of 94.4%, False Positive Rate of 0.1%, and an ROC AUC of 0.999.

12.4.3 Tag Frequency and Transient Response

Tag Frequency Response was investigated for passive UHF tags in [29]. The results showed differences between different tag models in the third and fifth harmonics but not significant differences between same-model tags. In addition, the author investigated Tag Transient Response to create a fingerprint because this was successful in previous work on fingerprinting transmitters like cell phones [27]. The Tag Transient Response is the part of the signal from the tag when it first begins to backscatter its information to the reader. The results showed that the transient was not significantly different among passive UHF tags. Although fingerprinting methods for other pervasive devices including HF tags have used the transient response, it did not work as a feature for passive UHF tags because these tags do not have an active power source and operate in the far field. Another reason is that these RFID systems use a reader talk-first protocol; unlike an active tag, the passive tag powers on before starting the response.

12.4.4 Tag Timing Response

The tag Timing Response (TR) was used to fingerprint tags in [29] and [30]. In [29], three different model tags were tested by measuring ten tags of each model. The time for the tag to send its ID, checksum, and control bits when queried by a reader was measured and found to be unique for two of the three models of tags that were tested. Two sets of measurements were taken for each tag, one week apart. The TR was measured six times for all tags.

After determining that the TR was different, the authors then use the tag TR as a fingerprint in an identification system for each model. An identification system

Table 12.2 Classification of tags based on length of transmission of EPC, PC and CRC

	TP rate	FP rate	ROC AUC
Manufacturer – 1	99.2%	1%	0.999
Manufacturer – 2	95%	6%	0.998
Manufacturer – 3	47.5%	5.8%	0.91

for each separate model was chosen as a worst-case analysis. KNN was used as the matching algorithm to identify a given tag based on its previously enrolled TR. This was performed ten times using stratified tenfold validation. As shown in Table 12.2, TR can be used for two of the three models to fingerprint tags. Note that the discussions with researchers since [29] was published suggest that the IC chip of the tag model that did not have consistent timing measurements was known to have some problems.

References

1. Juels A (2006) RFID security and privacy: a research survey. IEEE J Selected Areas Commun (JSAC) 24(2): 381–394
2. EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860–960 MHz, ver. 1.0.9, EPCglobal Inc. http://www.epcglobalinc.org/. Accessed 31 Jan 2005
3. ITU Internet Reports 2005: The Internet of Things, Executive Summary. http://www.itu.int/osg/spu/publications/internetofthings/. Accessed 18 July 2011
4. Smart Card Alliance Challenges DHS over RFID Deployment for WHTI PASS Card (2006) Smart Card Alliance. http://www.smartcardalliance.org. Accessed June 2006
5. United States Passport Card (2010) http://www.uspasscard.com/. Accessed October 2010
6. Real ID Act of 2005 Driver's License Title Summary (2010) National Conference of State Legislatures. http://www.ncsl.org/IssuesResearch/Transportation/RealIDAct of2005Summary/tabid/13579/Default.aspx. Accessed October 2010
7. Daigneau E (2006) Tag lines: RFID track Katrina's body count. Governing. http://www. gov erning.com/topics/technology/Tag-Lines-RFID-Track-Katrinas.html. Accessed March 2006
8. The International Anticounterfeiting Coalition (2010) http://www.iacc.org/. Accessed October 2010
9. Plos T, Hutter M, Feldhofer M (2008) Evaluation of side-channel preprocessing techniques on cryptographic-enabled HF and UHF RFID-tag prototypes. In: Dominikus S (ed) Workshop on RFID Security 2008, pp 114–127
10. Mitrokotsa A, Rieback MR, Tanenbaum AS (2009) Classifying RFID Attacks and Defenses, Information System Frontiers, DOI 10.1007/s10796–009–9210-z
11. Mitrokotsa A, Beye M, Peris-Lopez P (2010) Threats to Networked RFID Systems, Unique Radio Innovation for the 21st Century, Part 1, 39–63, Springer Publisher, DOI 10.1007/978-3-642-03462-6_3
12. Oren Y, Wool A (2009) Attacks on RFID-based electronic voting system. IACR ePrint, August 2009
13. Verdule R (2008) Security analysis of RFID tags. Master Thesis, Radboud University Nijmegen, June 2008
14. Oren Y, Shamir A (2007) Remote password extraction from RFID tags. IEEE Trans Comput 56(9): 1292–1296, 10.1109/TC.2007.1050

15. Juels A, Rivest R, Szydlo M (2003) The blocker tag: selective blocking of RFID tags for consumer privacy. In: 10th ACM Conference on Computer and Communication Security, pp 103–111
16. Rieback M, Crispo B, Tanenbaum A (2008) RFID guardian: a battery-powered mobile device for RFID privacy management. In: 13th Australian Conference on Information Security Privacy, July 2008, LNCS 5107. Springer-Verlag, Berlin, Heidelberg, New York, pp 184–194
17. Ranasinghe DC, Cole PH (2006) Confronting security and privacy threats in modern RFID systems. In: 40th Asilomar Conference on Signals, Systems, and Computers, Nov 2006, pp 2058–2064
18. Juels A () Strengthening EPC Tags against Cloning, In: Jacobson M, Poovendran R (eds) ACM Workshop on Wireless Security, LNCS 3982. Springer-Verlag, Berlin, Heidelberg, New York, pp 67–76
19. Devadas S, Suh E, Paral S, Sowell R, Ziola T, Khandelwal V (2008) Design and implementation of PUF-based "Unclonable." In: RFID ICs for Anti-Counterfeiting and Security Applications, 2008 IEEE International Conference on RFID, April 2008, pp 58–64
20. Han S, Chu C (2008) Tamper detection in RFID-enabled supply chains using fragile watermarking. In: IEEE International Conference on RFID
21. Lehtonen M, Ostojic D, Ilic A, Michahelles F (2009) Securing RFID systems by detecting tag cloning. In: The 7th International Conference on Pervasive Computing, May 2009
22. Karygiannis T, Eydt B, Barber G, Bunn L, Phillips T (2007) Guidelines for Securing Radio Frequency Identification (RFID) Systems: Recommendations of the National Institute of Standards and Technology, NIST Special Publication, vol 800(98)
23. Periaswamy SCG, Thompson DR, Di J (2011) Fingerprinting RFID tags. IEEE Transactions on Dependable and Secure Computing, to appear
24. Kohno T, Broido A, Claffy KC (2005) Remote physical device fingerprinting. IEEE Trans Dependable Secure Comput 2(2): 93–108
25. Barbeau JHM, Kranakis E (2006) Detecting rogue devices in Bluetooth networks using radio frequency fingerprinting. In: Proceedings of the IASTED International Conference on Communications and Computer Networks, Lima, Peru, Oct 2006
26. Mallette MC (2006) Sherlock in the XP age. In: CQ VHF, Winter 2006, pp 61–64
27. Riezenman MJ (2000) Cellular security: better, but foes still lurk. IEEE Spectr 39(6): 39–42
28. Romero HP, Remley KA, Williams DF, Wang C (2009) Electromagnetic measurements for counterfeit detection of radio frequency identification cards. IEEE Trans Microwave Theory Tech 57(5): 1383–1387
29. Periaswamy SCG (2010) Authentication of radio frequency identification devices using electronic characteristics. Ph.D. dissertation, Depertment of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR
30. Periaswamy SCG, Thompson DR, Romero HP, Di J (2010) Fingerprinting radio frequency identification tags using timing characteristics. In: Proceedings of Workshop on RFID Security (RFIDsec'10 Asia), Singapore, 22–23 Feb 2010, pp 73–82

Chapter 13
Memory Integrity Protection

Yin Hu and Berk Sunar

13.1 Introduction

With the growth of the personal computers and the internet, computers are now integrated into our lives. With all the conveniences this phenomena also creates more vulnerabilities. An intruder who is physically located on the other side of the earth may gain access to our most secret information by breaking through our firewall and gaining access to the internals of our computing system. Compromised computers may leak our most private information: personal documents, pictures and movies, browsing history, chat history, bank account passwords, etc.

Numerous countermeasures have been proposed to detect and prevent intrusions. However, computing itself is changing. First instance, the owner of the machine may not be trusted either. Consider a multiplayer online game where the player owns the machine, which maintains the state for that particular player for the online game world. The player (owner) of the machine has strong incentive to cheat. He may update the process memory to give his character more power. In this context, isolation techniques such as sandboxing do not mean much because the owner of the platform has full access to the system software and hardware. Game developers are incorporating *monitor* processes to detect cheating. However, it is relatively straightforward to get around such protections. Normally, memory space isolation techniques such as virtual memory are heavily enforced at the hardware level. However, the software platform, e.g., internet browsers and even the OS components, is not perfect either. There are numerous attacks, such as buffer overflow attacks, that exploit bugs to gain unauthorized access to the memory space of other processes.

Even further, *cloud computing* is now taking the threat to a whole new level. In the cloud server may be shared by a number of users running multiple applications

Y. Hu (✉) · B. Sunar
Worcester Polytechnic Institute, Worcester, MA, USA
e-mail: hhyy_best@wpi.edu; sunar@wpi.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_13,

on various operating systems all running on the same physical machine through virtualization. This level of sharing puts even more emphasis on protecting memory at the process level. Just relying on the isolation that comes at the software level with virtual machines and at the hardware level with virtual memory is not sufficient.

In this chapter, we present an overview of techniques for protecting the integrity and security of computer memory systems. Our treatment of memory protection techniques is not exhaustive.

13.1.1 Problem Definition

The most prominent security service is *confidentiality* which simply means that the privacy of the information is preserved in the presence of adversaries. Many schemes have been proposed and numerous commercial software packages are available for this purpose. For instance, Microsoft's BitLocker™ is a harddisk partition encryption tool. When the confidentiality problem is considered in the context of memory systems, the high latency of the encryption algorithm combined with the high frequency of memory accesses create a major bottleneck. Due to the performance bottleneck, sealed memory is rarely used only for high risk applications. While this is a significant research area, it is outside of the focus of this chapter.

The focus of this chapter is another security service, i.e., data integrity, in the context of random access memory (RAM). Very informally, data integrity simply means that unauthorized modifications of the data can be detected with overwhelming probability. The integrity of the data stored in the harddisks is relatively easy to achieve because harddisk access frequencies are relatively low due to the *locality principle*. As we will discuss in the following sections, many effective schemes were proposed for this purpose. In contrast, in the RAM context where access frequencies are typically one to two orders of magnitude higher, it becomes a major challenge to build a integrity protection scheme with reasonable performance.

13.2 Naïve Solution: Using Message Authentication Codes

The standard technique to protect the integrity of data is to use a message authentication code (MAC). In cryptography, a MAC is a short piece of information, or in other words, tag used to authenticate a message. If it is computationally and probabilistically impossible for an adversary to forge a valid MAC for a modified message, the receiver can verify the message by checking whether it meets the MAC or not. When tampering is suspected, the data and its tag can be verified with the aid of a secret key. MACs havebeen employed successfully in ensuring packet integrity

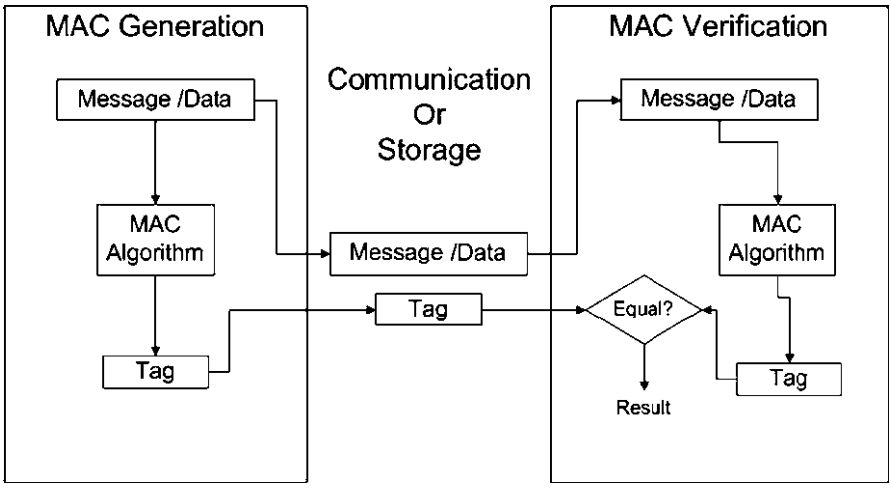


Fig. 13.1 MAC tag generation and verification

in communications for a long time and it is straightforward to adapt them to the stored data setting. A high level depiction of MAC tag generation and verification is shown in Fig. 13.1.

A common technique to generate a MAC is to use a keyed hash function. It accepts as the input a secret key and an arbitrary-length message to authenticate, and outputs a tag that serves as the MAC.

Unfortunately, a MAC alone is an ill fit for the memory integrity protection problem. Consider a memory integrity protection scheme where we keep a MAC for every small block of memory. Each time we access a single block of memory, we authenticate that block of data by verifying the corresponding MAC and calculate new MACs when necessary. This naïve scheme is neither efficient nor secure. The scheme may only protect the integrity of static data. In the next section, we will discuss a practical code authentication scheme built from this idea.

13.2.1 Integrity of Program Codes

In the stored-program paradigm introduced by von Neumann, the executable parts of a program (i.e., instructions) along with data are stored in a single structure. This abstraction is realized physically in the *main memory* by modern day computers. The processor requests the instruction block to be executed by supplying its starting address to the memory. The memory provides the instructions as requested. The memory also stores and supplies data, which can either be generated statically by the software publisher or dynamically by the program itself while in execution. Both instruction and data need to be authenticated for secure execution of programs on the processor.

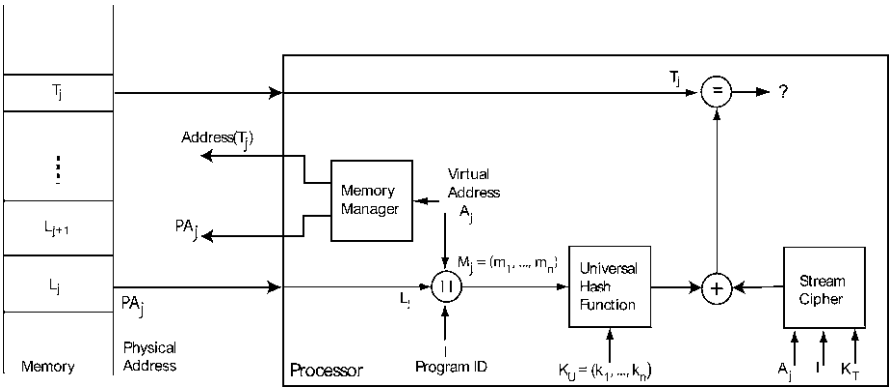


Fig. 13.2 Verification of software blocks

In [1], the authors proposed a practical code authentication scheme based on an efficient family of MACs. In this scheme, the computation latency can be hidden behind the memory access latency allowing the authentication to be performed "on-the-fly." In this scheme, every line of code is associated to an authentication tag. All the codes will be verified using these tags before they are executed. The tags can be generated by the software vendors and transferred to the end user using costly public key schemes. Details about the generation of the tags can be found in [1]. Here we will focus on the "on-the-fly" authentication using the tags.

The outline of software validation is illustrated in Fig. 13.2. When a memory request for an instruction block goes out, the memory access latency is used to generate the pseudo-random pad R_T with the stream cipher $\mathsf{SC}(K_T, A, I)$. The resulting pad R_T is accumulated in a register within the processor.

As instructions are retrieved from the memory into the CPU, they are incrementally hashed using the fast universal hash function. Each incremental hash operation consists of only two additions and one multiplication in $GF(2^w)$ which can be performed extremely fast. After the last block is hashed, the accumulated tag value is computed as $T_j = \mathsf{PR}_{K_U}(M_j) \bigoplus R_T$. The tag T_j computed in the authentication unit must be identical to the tag that are fetched from the memory, in which case the verification is accomplished. As long as the latency of an incremental hash computation operation can be hidden behind the memory access latency, the verification operation will run concurrently. The result of the verification as well as the instructions will be ready right after the last block is brought in from memory and incrementally hashed. Hence, the overhead introduced by this step, is in the retrieval of the authentication tag from memory and in the latency of the last incremental hash and XOR computations.

When an instruction block along with its authentication tag, $(M_j, T_j)$is received, the authentication circuit that is built into the processor first computes the digest $\mathsf{PR}_{K_U}(M_j)$ as defined earlier, where $M_j := (L_j||I||A_j)$ and L_j, I, and A_j are program block, program ID, and block address, respectively. The circuit checks whether the tag verifies and decides to execute (or not) accordingly, as shown in Fig. 13.2.

13.3 Bottlenecks and Limitation

13.3.1 Replay Attacks

A main distinction between codes and dynamic data is that data may be modified from time to time while codes do not normally change. This difference makes real-time data authentication a much harder task to achieve. A major weakness caused by the ever changing data is that the system will suffer so-called replay attacks. The replay attack is achieved by overwriting a new piece of data and its tag with an older piece of data and its tag, in other words by replaying old data. Without a time or an ordering mechanism, there is no way for a simple MAC scheme to tell whether the record has expired or not.

There are two common solutions to this problem: employing a Merkle Tree structure or introducing timestamps. These ideas as well as others can help overcome replay attacks. However, these methods will either require additional storage space or increase the access latency. In many cases, together with the already high overhead caused by the cryptographic hash computation needed to generate the MACs, the latency introduced by the replay countermeasures, memory integrity checks become extremely slow.

The high latency generates a major bottleneck preventing the deployment of real-time memory integrity protection schemes. The tag associated to a memory block usually needs verification and renewal each time the piece of record is accessed. This means that each memory update, will result in a large overhead. Keeping the high memory access frequencies of modern computers in mind, this latency will cause further degradation in the performance.

The code authentication scheme mentioned in Sect. 13.2.1 will not suffer such attacks since it is assumed that the protected data (program code) is static. There will only be one valid record per location. Therefore, there are no old records to replay. For this reason, the code authentication scheme does not need to employ costly schemes to detect replay attacks.

13.3.2 Root of Trust

As we will see in later sections, authentication schemes always need some secure place to save sensitive data, such as the keys used in the MAC, the timestamps, or the root of a Merkle Tree. In practice, this secure place is rather hard to find unless secure hardware is present. Therefore, many schemes rely on a trusted platform module (TPM).

To bring trust into computing systems is a sophisticated paradigm and an associated standard [2] was developed by the Trusted Computing Group. Despite the rapid advance in the standard achieved by strong backing of major device manufacturers and software providers, the techniques proposed in the standard

provide only very general high level descriptions of the security and at this point do not consider performance related issues such as operational efficiency and architectural integration. For instance, a popular means for taking measurements (*attestation*) is to compute the hash of a program or data (state of a program). The computed hash value which is stored in TPM's Platform Configuration Registers (PCRs), are signed by the private key in the TPM to generate a *quote*. Similarly, a program is authenticated using TPM's functionality.

However, as shown in [3, 4], the performance of any operation involving TPM suffers significantly. Although an initial integrity check by the TPM or its occasional use can be afforded at times, the heavy computational and communication (between processor and the TPM) burden eliminates the possibility of "on-the-fly" verification/decryption of code and introduce significant latencies.

Therefore, this secure space is very expensive comparing to regular memory space in terms of both computational and costs. Therefore, it draws another limitation on the authentication schemes that the usage of this secure space should be as few as possible. Clearly, another solution is to integrate the authentication/verification mechanism into the processor architecture essentially embedding some of the TPM functionality (and more that cannot be possible by TPM) into the processor architecture.

13.4 Building Blocks

13.4.1 Merkle Trees

A Merkle Tree is an arrangement for building a hash function that accepts long inputs, from hash functions that accept only fixed size inputs [5]. As shown in Fig. 13.3, the leaf nodes of the tree carry the data blocks while the branch nodes of the tree carry the hash result of the data blocks or hash results of the lower levels.

It is easy to see that if the employed hash function is collision resistant the larger structure will also be collision resistant. The only way to perform unauthorized modification to the data is to change the branch nodes of the tree as well. Therefore, any modification to the tree leaves will result in a different tree root with overwhelming probability. If the root of the tree is kept in a protected memory, i.e., in a place protected by hardware, the integrity of the overall data will also be protected.

A Merkle tree can provide defense against replay attacks as long as the attacker cannot access the root and replay it. Any unauthorized replay of the tags (which will be a child node of the root) can be detected. Another feature of the Merkle tree structure is that it only needs a small space of secure storage, which makes it more practical than adding time stamps to the tags. However, the extra hash operations needed for the multilevel tree structure will slow down the speed of the scheme significantly. Even using the popular SHA-1, which is usually considered fast, in the Merkle tree scheme will cause large overhead.

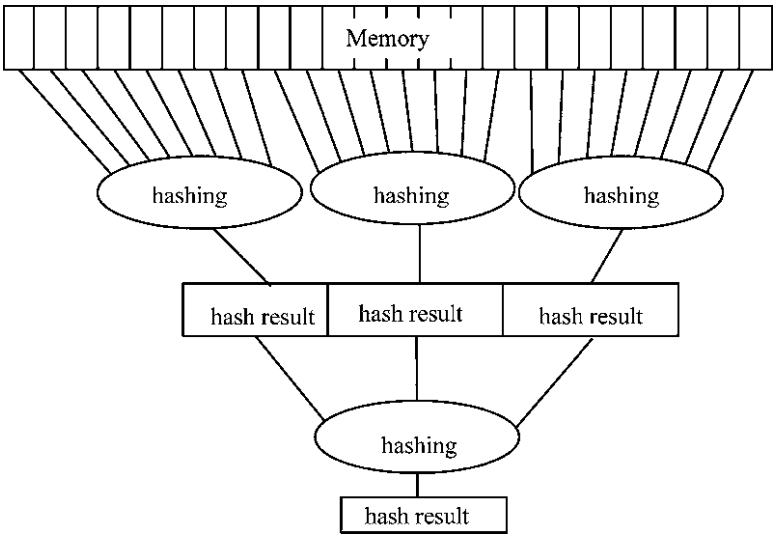


Fig. 13.3 The Merkle Tree

13.4.2 Hash Functions

A secure and efficient hash function is very important to many existing schemes. Roughly speaking, hash functions are procedures or mathematical functions that map arbitrary-length messages into short output strings. It is required that the collision probability of any given pair of messages is small. For cryptographic hash functions such as the SHA-1, it should also be computationally hard to find a collision.

SHA-1 is widely used today and their efficiency is high enough in most cases. However, it is still too slow for memory authentication schemes. To solve this problem, a family of hash functions called universal hash function family is sometimes employed. The universal hash functions are hash functions that have provably small collision probability. Alone universal hash function do not provide any security.

However, universal hash functions can still be used to build unconditionally secure MACs. To achieve this, the communicating parties share a secret encryption key, and a random hash function secretly chosen from the universal hash function family. A message is authenticated by hashing it with the shared hash function and then encrypting the resulting hash using the shared key. Carter and Wegman [6] showed that when the hash function family is strongly universal, i.e., messages are mapped into their images in a pair wise independent fashion, and the encryption is realized by a one-time pad, the adversary (even with unbounded computational power) cannot forge the message with probability better than that obtained by choosing a random string for the MAC.

A universal hash function, as proposed by Carter and Wegman [7], is a mapping from the finite set A with size a to the finite set B with size b. For a given hash

function $h \in H$ and for a message pair (M, M') where $M \neq M'$ the following function is defined: $\delta_h(M, M') = 1$ if $h(M) = h(M')$, and 0 otherwise, that is, the function δ yields 1 when the input message pair collide. For a given finite set of hash functions $\delta_H(M, M')$ is defined as $\Sigma_{h \in H} \delta_h(M, M')$, which tells us the number of functions in H for which M and M' collide. When h is randomly chosen from H and two distinct messages M and M' are given as input, the collision probability is equal to $\delta_H(M, M')/|H|$.

We next quote a number of definitions concerning the universal hash function family. The following two definitions were introduced in [8].

Definition 13.1. The set of hash functions $H = h : A \rightarrow B$ is said to be **universal** if for every $M, M' \in A$ where $M \neq M'$,

$$|h \in H : h(M) = h(M')| = \delta_H(M, M') = \frac{|H|}{b}.$$

Definition 13.2. The set of hash functions $H = h : A \rightarrow B$ is said to be **ε-almost universal** if for every $M, M' \in A$ where $M \neq M'$,

$$|h \in H : h(M) = h(M')| = \delta_H(M, M') \leq \varepsilon |H|.$$

In the past many universal and almost universal hash families were proposed [9–14]. A good example of them is an almost universal hash function family called NH introduced in [13]. The definition of NH is given later.

Definition 13.3. ([13]) Given $M = (m_1, \cdots, m_n)$ and $K = (k_1, \cdots, k_n)$, where m_i and $k_i \in U_w$, and for any even $n \geq 2$, NH is computed as follows:

$$\mathrm{NH}_K(M) = \left[\sum_{i=1}^{n/2} ((m_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m_{2i} + k_{2i}) \bmod 2^w) \right] \bmod 2^{2w}.$$

As a typical universal hash function family, the NH is easy to compute and its collision probability is proven to be low. However, since it is linear, an attacker can extract the keys easily by solving a linear equation if he has access to sufficiently many message-result pairs.

Incremental Hashing: A Merkle Tree structure built using a traditional hash function can achieve a time overhead of $Blog_B(M)$, where M is the size of data to protect and B is the block size. A smaller block size will result in a lower hashing overhead, however it will also will require more levels in the Merkle Tree. This will consume more storage space and may bring extra overhead for the computation of the intermediary level hashes. In contrast, a larger block size means a smaller tree structure, at the cost of a higher latency.

Incremental hash functions may be used to solve this problem. With incremental hash functions, the overhead will no longer have to be proportional to the block size. Very few hash functions posses the ability to incrementally compute the hash result. In [15] the authors have given a definition for the incrementality of hash functions. They also defined that an incremental scheme is called ideal if the running time is polynomial to the size of the word size (the minimum access unit for the data), the hash size, and the logarithm of the number of words hashed. Here we will only discuss the ideal ones since the nonideal algorithms will be too slow to be used in practice. Also, we only focus here on the property of the hash functions and do not consider how the hash function was generated. Thus, we can rewrite the definition as follows.

Suppose a message block M is divided into n words and w is the size of each block. $M\langle j, m\rangle$ means that the jth block of M is updated to m. The division is just a way to index the message and do not necessarily have a relation with the hash algorithm.

Definition 13.4. An incremental hash function is specified by a pair $H = (\mathsf{HashAll}, \mathsf{Update})$ of algorithms where:

- The polynomial time hash algorithm HashAll takes a message $M \in \{0,1\}^{nw}$ and output a k bit string h called the hash of M
- The incremental update algorithm that

$$\forall M \in \{0,1\}^{nw}, j \in \{1,\ldots,n\}, m \in \{0,1\}^{w} \text{ and } h = \mathsf{HashAll}(M)$$

$\mathsf{Update}(M, h, (j, m)) = h'$ can get the same output $h' = \mathsf{HashAll}(M\langle j, m\rangle)$ from HashAll and the running time of the algorithm is polynomial in $w, k, \log(n)$.

The example universal hash function discussed in last section possesses the incrementality property. For instance, if we want to update the m_{2i-1}, m_{2i} to m'_{2i-1}, m'_{2i}, the new hash result can be computed as:

$$\begin{aligned}\mathsf{NH}' = [\mathsf{NH}_{others} &+ ((m_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m_{2i} + k_{2i}) \bmod 2^w) \\ &-((m_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m_{2i} + k_{2i}) \bmod 2^w) \\ &+((m'_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m'_{2i} + k_{2i}) \bmod 2^w)] \bmod 2^{2w}.\end{aligned}$$

If the word size is fixed, the complexity of this update operation will remain constant no matter what block size we choose.

13.4.3 Schemes Other than Merkle Trees

13.4.3.1 Multiset Hash Functions

Multiset hash functions [16] are a new cryptographic tool that can be used to help build the trace-hash integrity checker. Unlike standard hash functions which take strings as input, multiset hash functions operate on multisets (or sets). They map multisets of arbitrary finite size to strings (hashes) of fixed length.

Roughly speaking, the multiset hash functions work as if you are putting balls into a bag. You can put different balls into two bags in any order, and the function can tell you whether the two bags contain the same combination of balls. To protect the integrity of the RAM, we keep one hash for each of what we have written to the RAM and what we have read from the RAM. If the two hashes are the same, we know that there is only tiny probability that we have read something different with what we have written. Therefore, we can claim that the RAM is intact.

A good property of the multiset hash is that they are incremental. When new members are added to the multiset, the hash can be updated in time proportional to the change. Therefore, for the same reason we listed in previous sections for the incremental hash functions, the multiset hash functions are ideal for memory authentication schemes.

This is just the brief outline on how the multiset hash functions may be used for memory integrity protection. A rigorously defined scheme will be needed to claim any security. A detailed treatment of how the multiset hash can be used to provide integrity protection will be given later.

13.5 Survey of Proposed Schemes

13.5.1 The GCM-Based Authentication Scheme

Yan et al. [17] proposed a memory encryption scheme that also provides memory integrity protection achieved by employing a Galois/Counter Mode (GCM)-based [18] authentication scheme. The authors use split counters to improve the encryption efficiency, and use GCM for memory authentication. The authentication scheme is tightly integrated with the memory encryption scheme to hide the authentication latency behind the memory encryption latency.

The definition of the GCM-based hash function GHASH is given later and the GCM is outlined in Fig. 13.4. As we can find out from the figure, the encryption operations can be computed in parallel. The computation that cannot be carried out in parallel is the XORs and the Galois field multiplications. However, an important feature of the Galois field multiplication is that it can be computed in parallel. This parallelism can then be translated into higher speed if additional hardware is available.

Definition 13.5. ([18]) The GHASH function is defined $\mathsf{GHASH}(H, A, C) = X_{m+n+1}$, where H is a string of 128 zeros encrypted using a block cipher, A the data, C the ciphertext, m the number of 128bit blocks in A, n the number of 128-bit blocks in C, and the variable X_i for $i = 0, \ldots, m + n + 1$ is defined as

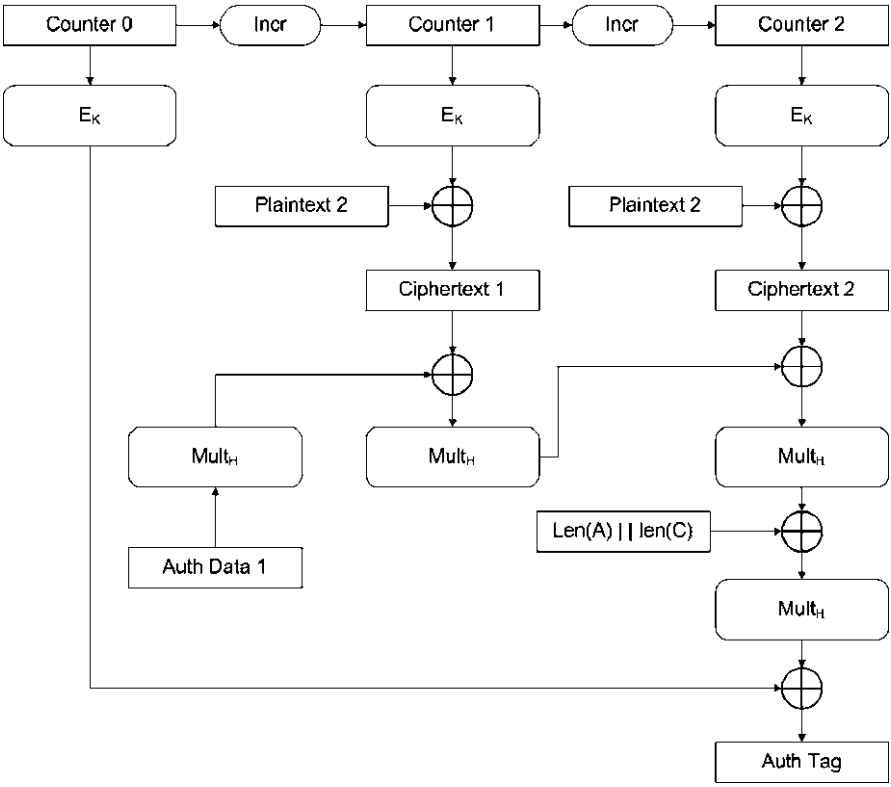


Fig. 13.4 Outline of the GCM

$$X_i = \begin{cases} 0 & \text{for } i = 0 \\ (X_i \oplus A_i) \cdot H & \text{for } i = 1, \ldots, m-1 \\ (X_{m-1} \oplus (A_m^* || 0^{128-v})) \cdot H & \text{for } i = m \\ (X_i \oplus C_{i-m}) \cdot H & \text{for } i = m+1, \ldots, m+n-1 \\ (X_{m+n-1} \oplus (C_n^* || 0^{128-u})) \cdot H & \text{for } i = m+n \\ (X_{m+n} \oplus (\text{len}(A) || \text{len}(C))) \cdot H & \text{for } i = m+n+1 \end{cases}.$$

In the Yan et al. scheme, several words of data in the memory are encrypted in parallel and then hashed using GCM. As we can see in Fig. 13.5, the majority of the computation can be performed in parallel and the rest parts are only XOR operations and the Galois field multiplications, which can be computed efficiently. Therefore, although the amount of calculation required for the hash function is large, the latency caused by the scheme is relatively small.

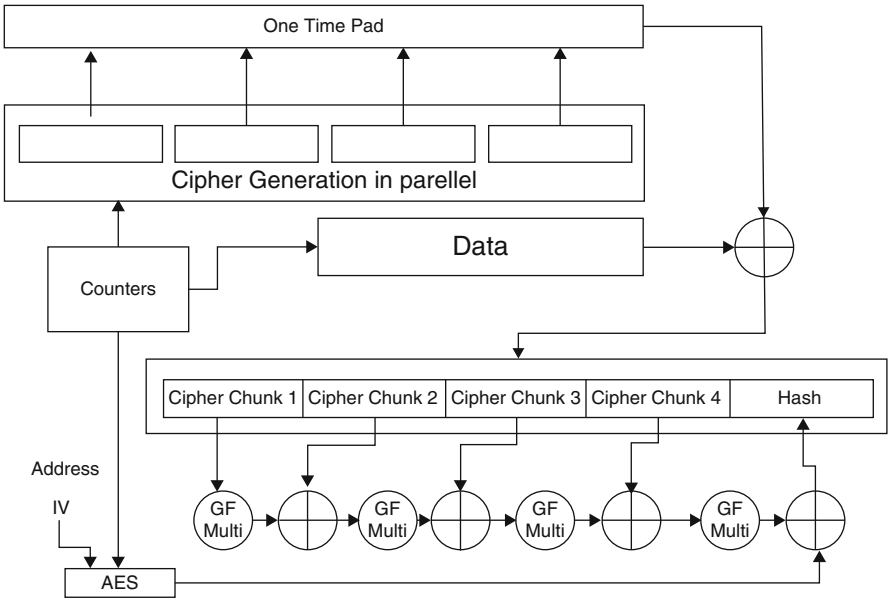


Fig. 13.5 Outline of the Yan et al. memory encryption scheme

13.5.2 The Adaptive Tree-log Scheme

The adaptive tree-log scheme was developed by Clarke et al. in [19]. The scheme adaptively combines a multiset hash-based integrity checker and a Merkle Tree to achieve higher efficiency. As briefly outlined in Sect. 13.4.3.1, the multiset-based trace-hash checker maintains a write trace and a read trace for write and read operations to the external memory. To explain how the multiset-based trace-hash checker is combined with a Merkle Tree, let us start with the details of the multiset hash scheme.

Definition 13.6. ([16]) Multiset Let $(H, +_H, \equiv_H$ be a triple of probabilistic polynomial time (PPT) algorithms. That triple is a multiset hash function if it satisfies:

Compression: H maps multisets of B into elements of a set with cardinality $\approx 2^m$, where m is some integer. Compression guarantees that we can store hashes in a small bounded amount of memory.

Comparability: As H can be a probabilistic algorithm, a multiset need not always hash to the same value. Therefore, we need $\equiv_H$ to compare hashes. The following relation must hold for comparison to be possible:

$$H(M) \equiv_H H(M)$$

for all multisets M of B.

Incrementality: We would like to efficiently compute $H(M \cup M')$ knowing $H(M)$ and $H(M')$. The $+_H$ operator makes that possible:

$$H(M \cup M') \equiv_H H(M) +_H H(M')$$

for all multisets M and M' of B. In particular, knowing only $H(M)$ and an element $b \in B$, we can easily compute $H(M \cup \{b\}) = H(M) +_H H(\{b\})$

As an example, consider the construction of the multiset hash function definition of the MSet-XOR-Hash.

Definition 13.7. ([19]) Let H_k be a pseudorandom function keyed with a seed (key) k. $H_k : \{0,1\}^l \to \{0,1\}^m$. Let $r \xleftarrow{R} \{0,1\}^m$ denote uniform random selection of r from $\{0,1\}^m$. Then define MSet-XOR-Hash as follows

$$H_k(M) = \left(\left(H_k(0,r) \oplus \bigoplus_{v \in V} M_v H_k(1,v) \right); \ |M| \bmod 2^m; \right)$$

$$(h,c,r) \equiv_{H_k} (h', c'r') = (h \oplus H_k(0,r) = h' \ \oplus H_k(0,r') \wedge c = c')$$

$$(h,c,r) +_{H_k} (h',c',r')$$

$$= (H_k(0,r'') \oplus (h \oplus H_k(0,r)) \oplus (h' \oplus H_k(0,r')); c + c' \bmod 2^m; r'')$$

$$\text{where} r'' \xleftarrow{R} \{0,1\}^m.$$

Clarke et al. proposed several types of multiset hash functions in [19]. The hard problem these schemes rely on and their computational efficiency varies. Now that we have the multiset hash function, we can start to build a memory integrity checker from it. Clarke et al. label such schemes "trace-hash" since the schemes protect the integrity of RAM by ensuring a valid access trace.

As one may observe from Fig. 13.6, if the multiset hash functions used are incremental, the operations required for *store* and *load* will be incremental. That means the overhead introduced by these operations are growing only with the word size. As discussed earlier, this is an ideal feature for memory integrity protection applications. However, when a check is required, the checker needs to put the whole block into the "bag" word by word. This will cause a large latency. Therefore, if the application requires frequent integrity checks, the overall performance for this multiset-based scheme will drop significantly to even lower than that of a regular Merkle Tree-based scheme.

Clarke et al. solve this dilemma by introducing an adaptive combination of this multiset hash-based scheme and a Merkle Tree. A more efficient scheme with caches is also introduced in [19]. Here we will only discuss how the two schemes are combined. Briefly speaking, the combined approach, i.e., "tree-trace-checker," initially protects all data with a Merkle Tree. When memory access happens, instead of updating the Merkle Tree, the system will record the accesses via a trace-hash-checker. The Merkle Tree will only be updated when a check is called

The checker's fixed-sized state is:

- 2 multiset hashes: WRITEHASH and READHASH.
  Initially both hashes are 0.
- 1 counter: TIMER. Initially TIMER is 0.

Sub-functions defined to make the process more clear:

put(a, v) writes a value v to address a in memory:
1. Let t be the current value of Timer. Write (v, t) to a in memory.
2. Update WRITEHASH: WRITEHASH$+_H$ = hash(a, v, t).

get(a) reads the value at address a in memory:
1. Read (v, t) from a in memory.
2. Update ReadHash: READHASH $+_H$ = hash(a, v, t).
3. TIMER = max(TIMER, $t + 1$).

Interface of the Trace-Checker:

initialize() initializes RAM.
1. put$(a, 0)$ for each address a.

store(a, v) stores v at address a.
1. get(a).
2. put(a, v).

load(a) loads the data value at address a.
1. v = get(a). Return v to the caller.
2. put(a, v).

check() checks if the RAM has behaved correctly (at the end of operation).
1. get(a) for each address a.
2. If WRITEHASH is equal to READHASH, return true

Fig. 13.6 The trace-hash-checker

for the trace-hash-checker. In that case, the trace-hash will be flushed after being checked and the Merkle Tree will be updated according to the latest data. The entire RAM will be protected by the Merkle Tree again.

The advantage of this scheme is obvious. Instead of protection a whole block of data, now the trace-hash-checker only protects the data accessed between two check calls. If the check operation is called frequently, the amount of data needs

to authenticate will be small, which results in a faster check speed. If the check frequency is low, more data will be protected by the trace-hash-checker and will enjoy the low access latency of the trace-hash-checker.

13.5.3 The UMAC-Based Merkle Tree Scheme

The scheme proposed by Hu et al. [20, 21] proposes to use a Merkle Tree with fast/incremental universal hash functions for more efficient memory integrity protection. For easy reference we will refer to this scheme as the HHS scheme.

The basic idea of this scheme is using incremental universal hash functions. The NH universal hashing scheme introduced in previous sections, exhibits excellent performance in software [8] as it was designed to be naturally compatible with the instructions supported on common processor architectures. Suppose the w in the equation is 32 or 64, the only operations needed for computing the NH are additions and multiplications. However, despite the precise characterization of the statistical properties of universal hash functions, when employed alone they do not provide any cryptographic protection. Thus, the HHS scheme encrypts the results of the universal hash functions using random bit strings. The combination works like a one-time-pad and can provide provable security. Also, Toeplitz technique [22, 23] is used to strengthen the security without significantly increasing the key size.

Roughly speaking, this scheme is an efficiency combination of the Merkle Tree and the UMAC. The idea behind the design is simple. It is the sophisticated design together with the incremental nature of the NH that build up the high performance.

The details of a two level Merkle Tree with the 32-bit NH universal hash function applied twice using the Toeplitz approach as described earlier is outlined in Fig. 13.7. Tags are generated by adding masks to the hash results. The masks are generated by encrypting a random seed using a block cipher. For this task the AES block cipher is employed. The seeds are randomly generated and saved in the unprotected memory in plaintext alongside the produced tags. The tags are then hashed to provide the root hash result. It is easy to extend this scheme to an l-level version by using the tags as the data for the higher level. This scheme is called the *basic scheme* in [20]. We will refer this scheme as the *HHS's Basic Scheme* later.

One problem of the HHS's Basic scheme, as well as other similar authentication schemes is that the data need a check before every access or the system will fail to detect the unauthorized modification. For example, if the data is not checked before every access, the attacker can reverse the modification after the read operation however before the next check. In this way, the system will never be able to detect that it has read falsified data. In a Merkle Tree structure, a check means at least a hash over an entire block. In the HHS scheme, though it is still quite fast, the check operation will become the bottleneck due to the incremental update. However, if the *error inheritance* property holds, we can use it to eliminate this check and can hence boost the performance. This property was firstly mentioned in [20] and discussed in detail in [21].

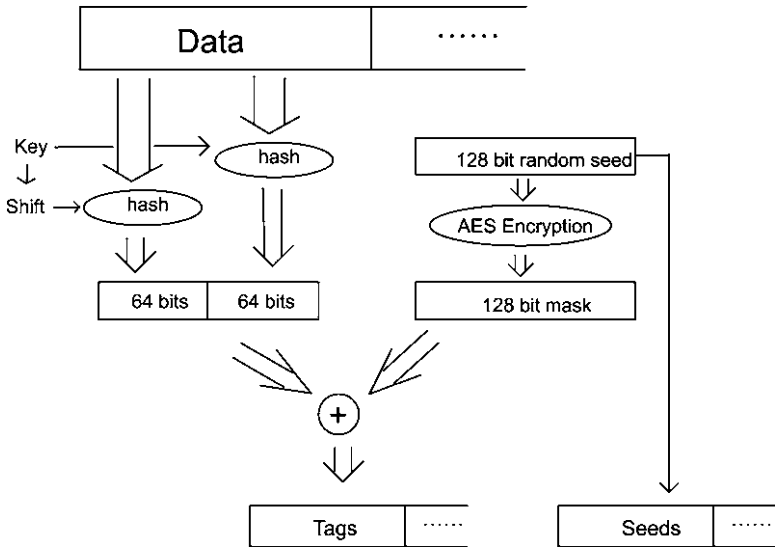


Fig. 13.7 Outline of the HHS's basic scheme in [20]

With this property, any injected error that occurs within the execution will carry forward to the future values of the tag. Even if the tags are checked after a large number of cycles it is still possible to detect the injection of an error at one point in time with high probability. For example, consider the NH scheme. Suppose the attacker has changed data blocks M_1, M_2 to M_1', M_2', and after that the authorized user wants to change the data to M_1'', M_2''. The calculation for the new hash result will be

$$\begin{aligned}\mathsf{NH}' = [\mathsf{NH}_{others} &+ ((m_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m_{2i} + k_{2i}) \bmod 2^w) \\ &-((m'_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m'_{2i} + k_{2i}) \bmod 2^w) \\ &+((m''_{2i-1} + k_{2i-1}) \bmod 2^w) \times ((m''_{2i} + k_{2i}) \bmod 2^w)] \bmod 2^{2w}.\end{aligned}$$

As long as the attacker cannot cancel the term calculated from M_1' and M_2', the attack can be detected. The SHA-1 based scheme on the other hand does not offer such a property. The tags will have to be checked in every update cycle to detect the malicious modification. The HHS scheme and the modification proposed in this paper will allow one to perform integrity checks at a much lower rate without the danger of missing an unauthorized modification, which translates to better performance.

The security of the error inheritance property stems from the fact that the attacker cannot hide the trace of a past attack. In NH, such a trace is the subtraction of the

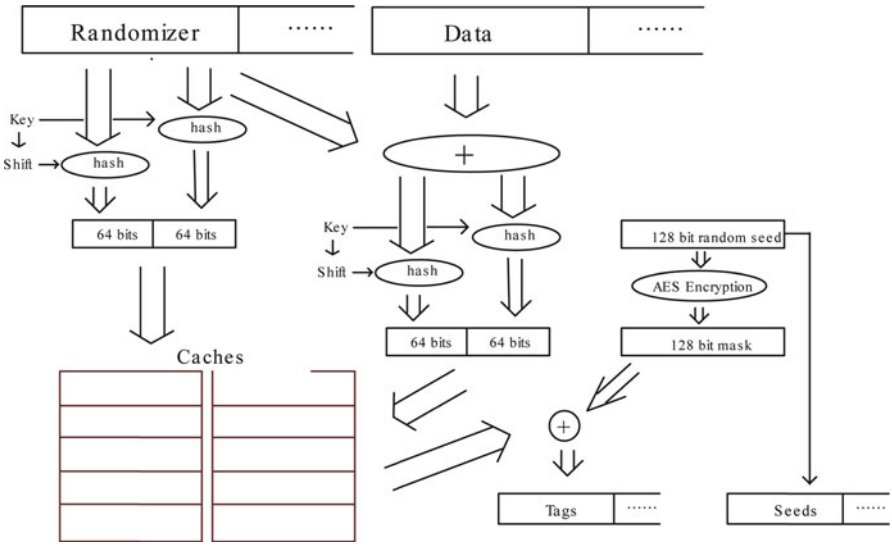


Fig. 13.8 Outline of the new scheme

partial result which is calculated from old attacked data. However, if the attacker can control an authorized update, which may be possible, he can remove the trace by letting the system update to the attacked data.

This attack will compromise the "error inheritance" property in the HHS schemes. However, if the data is selected uniformly at random, which means that the attacker can neither control nor predict the data, the inheritance property can still serve as a way to boost the performance. Normally, the data in the memory is not random so that this property does not work. However, we can randomize the data by modifying the scheme. With randomization, even the check before every read operation can be eliminated.

This can be achieved by keeping random masks for every piece of data that needs to be randomized [21]. These random masks are called randomizers. The randomizers can be generated either by a hardware random source or a PRNG. For each update of the Merkle Tree, instead of the data, the sum of the data and the corresponding randomizer is treated as the leaf of the tree. In addition, the randomizers will also change for each update. If the randomizers are changing randomly, the sum of the randomizer and the data should be also random. In this way, the error inheritance property works. These randomizers do not work as the masks for the hash results, which serve as one-time-pads. They are only used to randomize the data, not encrypt the data. Thus, as long as the randomizers are unpredictable, the error inheritance property will hold. An outline of this improved HHS scheme with caches applied can be found in Fig. 13.8.

As seen from Fig. 13.8, the update of the hash results and the preparation of the masks are independent. Therefore, the mask related calculation can be carried

out in advance or in parallel. If there are enough cores to deal with all these computations, the memory access speed can be extremely fast. In this case, the bottleneck becomes the updating of the NH, which is quite light-weighted. However, in a Merkle Tree, the computations at higher levels have to wait for the lower levels to finish before it can start updating. In other words, higher levels of the tree depend on the results of lower levels. This dependence forms the critical delay and becomes worse as the tree grows larger.

The HHS scheme provides a solution by a small modification to the Merkle Tree. In the HHS schemes, the tag for a block of data is actually composed of two parts, the tag itself and the seed. Normally, the tags should serve as the data for the higher level of the tree. The tags are computed from the data so that it will cause the problem mentioned earlier. However, if the data dependence is moved to the seeds instead of the tags at the higher levels of the tree, this restriction can be removed. The new seeds are selected completely at random so that they will not cause any dependence problems. Each level of the Merkle Tree can start updating without waiting for the result of the lower level. Thus, parallel computation of the Merkle Tree can be employed.

13.6 Beyond Memory Integrity Protection

The schemes we have surveyed in this chapter were introduced for the integrity protection of dynamic memory. However, there is no reason why they cannot be used in other applications. For example, it is possible to readily apply these schemes to protect information stored on other forms of media, whether it is information stored in an RFID or on hard disks. Although there are already a large number of file protection schemes, some new ideas learned from memory integrity protection schemes might be very useful.

A promising application might be protecting data stored in the so-called "cloud." Cloud computing provides greater flexibility and convenience for storage and computing needs of both consumers and the enterprise. For instance, storage services are offered by a number of cloud service providers including the Amazon EC2, the Google Apps, and the Microsoft Azure. Cloud storage users can access terabytes of space whenever needed and release extra storage space when it is no longer required saving costs. However, the cloud computation inherits most information security problems from personal computers.

Existing memory integrity protection schemes can be used to bring integrity services to cloud storage. If we treat the cloud as RAM and the client as the CPU, the two problems will exhibit significant similarities. In addition, the isolated nature of a client from the cloud makes the client an ideal root of trust. Such an application scenario is depicted in Fig. 13.9. On the right the client acts as the CPU while on the left the CSP provides offers a trusted server outside the cloud to act as the CPU.

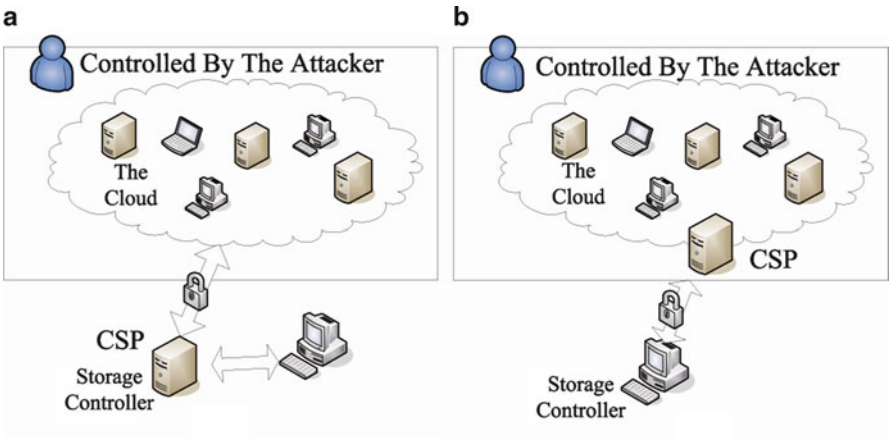


Fig. 13.9 Outline of the model: (**a**) CSP provides the data authentication service. (**b**) Client authenticates the data itself

References

1. Durahim AO, Savas E, Sunar B, Pedersen TB, Kocabas O (2009) Transparent code authentication at the processor level. Comput Digital Tech, IET, 3(4): 354–372
2. Trusted Computing Group, Incorporated. TCG Software Stack (TSS), Specification Version 1.2, Level 1. Part1: Commands and Structures. https://www.trustedcomputinggroup.org/. Accessed 6 January 2006
3. McCune JM, Parno B, Perrig A, Reiter MK, Isozaki H (2008) Flicker: an execution infrastructure for tcb minimization. In: Sventek JS, Hand S (eds.) EuroSys. ACM, New York, pp 315–328
4. McCune JM, Parno B, Perrig A, Reiter MK, Seshadri A (2008) How low can you go?: recommendations for hardwaresupported minimal tcb code execution. In: Eggers SJ, Larus JR (eds.) ASPLOS. ACM, New York, pp 14–25
5. Merkle RC (1980) Protocols for public key cryptosystems. In: Proceedings of the 1980 IEEE Symposium on Security and Privacy
6. Wegman MN, Carter JL (1981) New hash functions and their use in authentication and set equality. J Comput Syst Sci 22(3): 265–279
7. Carter L, Wegman MN (1979) Universal classes of hash functions. J Comput Syst Sci 18(2): 143–154, 1979
8. Nevelsteen W, Preneel B (1999) Software performance of universal hash functions. In: Advances in CryptologyEUROCRYPT99, pp 24–41. Springer, Berlin, Heidelberg, New York
9. Shoup V (1996) On fast and provably secure message authentication based on universal hashing. In: Advances in CryptologyCRYPTO96, pp 313–328. Springer, Berlin, Heidelberg, New York
10. Halevi S, Krawczyk H (1997) MMH: software message authentication in the Gbit/second rates. In: Fast Software Encryption, pp 172–189. Springer, Berlin, Heidelberg, New York
11. Rogaway P (1999) Bucket hashing and its application to fast message authentication. J Cryptol 12(2): 91–115
12. Krawczyk H (1995) New hash functions for message authentication. In: Advances in CryptologyEUROCRYPT95, pp 301–310. Springer, Berlin, Heidelberg, New York

13. Black J, Halevi S, Krawczyk H, Krovetz T, Rogaway P (1999) UMAC: fast and secure message authentication. In: Wiener MJ (ed.) CRYPTO'99, volume 1666 of Lecture Notes in Computer Science, pp 216–233. Springer, Berlin, Heidelberg, New York
14. Etzel M, Patel S, Ramzan Z (1999) Square hash: fast message authentication via optimized universal hash functions. In: Advances in Cryptology-CRYPTO99, pp 786–786. Springer, Berlin, Heidelberg, New York
15. Bellare M, Goldreich O, Goldwasser S (1994) Incremental cryptography: the case of hashing and signing. In: Advances in CryptologyCRYPTO94, pp 216–233. Springer, Berlin, Heidelberg, New York
16. Clarke DE, Devadas S, van Dijk M, Gassend B, Edward Suh G (2003) Incremental multiset hash functions and their application to memory integrity checking. In: Laih C-S (ed.) ASIACRYPT 2003, vol 2894 of Lecture Notes in Computer Science, pp 188–207. Springer-Verlag, Berlin, Heidelberg, New York
17. Yan C, Englender D, Prvulovic M, Rogers B, Solihin Y (2006) Improving cost, performance, and security of memory encryption and authentication. In: Proceedings of the 33rd annual international symposium on Computer Architecture, pp 179–190. IEEE Computer Society
18. McGrew D, Viega J (2004) The Galois/Counter mode of operation (GCM). Submission to NIST http://siswg.net/docs/gcm_spec.pdf. Accessed July 15th 2011
19. Clarke DE, Edward Suh G, Gassend B, Sudan A, van Dijk M, Devadas S (2005) Towards constant bandwidth overhead integrity checking of untrusted data. In: IEEE Symposium on Security and Privacy, pp 139–153. IEEE Computer Society, Silver Spring, MD
20. Hu Y, Hammouri G, Sunar B (2008) A fast real-time memory authentication protocol. In: Proceedings of the 3rd ACM workshop on Scalable trusted computing, pp 31–40. ACM, New York
21. Hu Y, Sunar B (2010) An improved memory integrity protection scheme. Trust and Trustworthy Computing, pp 273–281
22. Krawczyk H (1994) LFSR-based hashing and authentication. In: Advances in Cryptology-CRYPTO94, pp 129–139. Springer, Berlin, Heidelberg, New York
23. Kaps J-P, Yuksel K, Sunar B (2005) Energy scalable universal hashing. IEEE Trans Comput 54(12):1484–1495

Chapter 14
Trojan Taxonomy

Ramesh Karri, Jeyavijayan Rajendran, and Kurt Rosenfeld

14.1 Introduction

With the steady increase in outsourcing of semiconductor integrated circuits (ICs) manufacturing, the concerns for malicious inclusions are increasing within the military and the commercial sectors. Recently, Trojans in ICs used by military equipment have been reported [1]. Trojans can change the functionality of an IC and affect mission critical equipment. Trojans can also disable a system at will. These concerns caused the Defense Advanced Research Projects Agency (DARPA) to initiate the Trust in ICs program. This program focuses on developing Trojan detection methods [2, 3]. To facilitate the development of Trojan mitigation, detection, and protection techniques, it is necessary to systematically categorize hardware Trojans and the benefits are threefold:

- Classification of Trojans will enable a systematic study of their characteristics.
- Trojan detection and mitigation techniques can be developed for each of the Trojan classes.
- Benchmarks targeting each of the classes can be developed. These benchmark techniques can serve as a basis to compare competing Trojan detection methods.

Several research works have proposed taxonomies for hardware Trojans based on their characteristics [4–7]. At the end of this chapter several case studies of Hardware Trojans are discussed.

R. Karri (✉) · J. Rajendran
Polytechnic Institute of New York University, Brooklyn, NY, USA
e-mail: rkarri@poly.edu; jeyavijayan@gmail.com

K. Rosenfeld
Google Inc., New York, USA
e-mail: kurt@isis.poly.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_14,

14.2 Hardware Trojan

It is a malicious addition or modification to the existing circuit elements that can change the functionality, reduce the reliability, or leak valuable information which can be inserted at any phase of the IC design.

Trojans that are triggered usually requires two parts – a trigger and payload. The "trigger" acts like a sensing circuitry, which activates a Trojan to perform a specific task. The "payload" is responsible for the malicious activity of the Trojan. Trojans that are always-on consists of only the payload part.

14.3 Trojan Taxonomy

Hardware Trojans can be classified based on five attributes: (1) Phase of insertion, (2) Abstraction level, (3) Activation mechanism, (4) Functionality, and (5) Location.

The various classes along with their subclasses are shown in Fig. 14.1.

14.3.1 Phase of Insertion

Hardware Trojans can be inserted throughout the development cycle of the chip. Trojans can be classified based on the phases in which they are inserted.

14.3.1.1 Specification Phase

In the specification phase, the characteristics of the system are defined (e.g., target environment, expected function, size, power, delay). While the development of the IC is in this phase, functional specification or other design constraints can be altered. For example, a Trojan at the specification phase might change the timing requirements of the hardware.

14.3.1.2 Design Phase

In the design phase, functional, logical, timing, and physical constraints are considered as the design is mapped onto the target technology. At this phase, designers may use third-party intellectual property (IP) blocks and standard cells.

14.3.1.3 Fabrication Phase

At the fabrication phase, a mask set is made and wafers are produced using the masks. Subtle mask changes can have serious effects. In the extreme case,

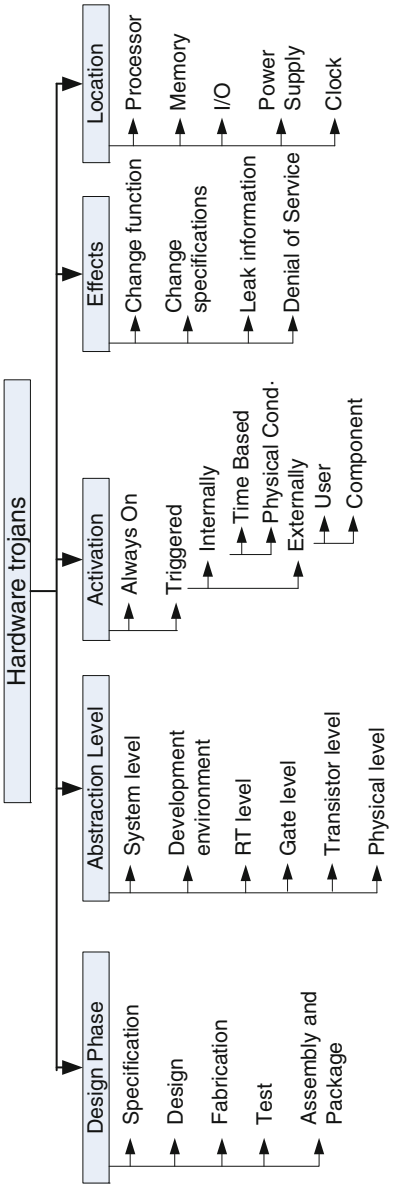


Fig. 14.1 Taxonomy of hardware Trojans

the adversary can substitute his own alternative mask set. Alternatively, chemical compositions may be altered during fabrication to increase the electromigration in critical circuitry like power supply and clock grids which would accelerate failures.

14.3.1.4 Assembly Phase

During the assembly phase, the tested chip and other hardware components are assembled on a printed circuit board (PCB). Every interface in a system where two or more components interact is a potential site for Trojan insertion. Even if all of the ICs in a system are trustworthy, malicious assembly can introduce security flaws in the system. For instance, an unshielded wire connected to a node on the PCB can introduce exploitable electromagnetic coupling between the signal on the board and its electromagnetic surroundings. This can be exploited for information leakage and for fault injection.

14.3.1.5 Testing Phase

This phase is also important for hardware trust, but not because it is a likely phase for Trojan insertion, but because it is opportunity for Trojan detection. Testing is only useful for Trojan detection if the testing is trustworthy. For instance, an adversary who inserted a Trojan in the fabrication phase would want to have control over the test vectors to ensure that the Trojan is not detected during testing. Trust in during testing means that the test vectors will be kept secret, the test vectors will be faithfully applied, and the specified actions (accept/reject, binning) will be faithfully followed.

14.3.2 Abstraction Level of Description

Trojan circuits can be inserted at various hardware abstraction levels. Their functionality and the structure are heavily dependent on the abstraction level at which they are described.

14.3.2.1 System Level

At the system level, the different hardware modules, interconnections, and communication protocols used are defined. At this level, the Trojans may be triggered by the modules in the target hardware. For example, the ASCII values of the inputs from the keyboard can be interchanged.

14.3.2.2 Development Environment

A typical development environment includes synthesis, simulation, verification, and validation tools. The CAD tools and scripts can be used to insert Trojans. Software Trojans inserted in these CAD tools can mask the effects of the hardware Trojans. For example, Trojan components of a circuit may not be revealed to the user by a synthesis tool.

14.3.2.3 Register Transfer Level

At the register transfer level (RTL), each functional module is described in terms of registers, signals, and Boolean functions. A Trojan can be easily designed and inserted at the RTL as the attacker has full control over the functionality of the hardware at this level. For example, a Trojan implemented at this level might halve the rounds of a cryptographic algorithm by making a round counter advance in steps of two instead of steps of one.

14.3.2.4 Gate Level

At the gate level, the design is represented as an interconnection of logic gates. This level allows an attacker to carefully control all aspects of the inserted Trojan including its size and location. For example, a Trojan might be a simple comparator consisting of XOR gates that monitor the internal signals of the chip. Also, the Trojan can be a combinational or sequential type. These Trojans are generally used to alter the functionality of the design and hence they are also called "functional" Trojans.

14.3.2.5 Transistor Level

Transistors are used to build logic gates. This level gives the Trojan designer a control over circuit characteristics like power and timing. Individual transistors can be inserted or removed, altering the circuit functionality. Transistor sizes can be modified to alter circuit parameters. For example, a transistor-level Trojan might be a transistor with low gate width which can cause more delay in the critical path. Trojans at the transistor level are generally used to alter the functionality of the design and hence they are also called as "functional" Trojans.

14.3.2.6 Layout Level

At the layout level, the dimensions and locations of all circuit components are described. This is the physical level of the design where a Trojan can be inserted.

Trojans may be inserted by modifying the size of the wires, distances between circuit elements, and reassigning metal layers. For example, changing the width of the metal wires of the clock grids in the chip can cause clock skew. Trojans at the layout level are generally described as modifications to the parameters of the IC physical design and hence they are also called "parametric Trojans."

Based on the number of gates inserted, a Trojan can also be classified as small or big. Also, based on the distribution of the Trojan in the design, a Trojan can be classified into tightly coupled or loosely distributed. All the components of a tightly coupled Trojan reside within close proximity of each other on the chip (need to change this line). A loosely distributed Trojan is topologically distributed throughout the layout of the chip. It leverages the empty spaces in the layout however they require complex routing mechanisms to interconnect the distributed parts.

14.3.3 Activation Mechanism

Some Trojans are designed to be always on; others may remain dormant until triggered. An "always-on" Trojan, as the name indicates, can affect the chip at any time. The parametric Trojans in which changes are made at the physical layout are considered to be the always on Trojans.

A triggered Trojan needs an event – internal or external to be activated. Once the trigger activates a Trojan, it may remain active forever or return to a dormant state after some time.

1. *Internally triggered*: An internally triggered Trojan is activated by an event that occurs within the target device. The event may be either time-based or physical-condition based. A counter in the design can trigger a Trojan at a predetermined time, resulting in what is known as a time-bomb. The physical-condition based Trojan can be excited by a wide variety of physical conditions such as electromagnetic interference, humidity, altitude, atmospheric pressure, etc. For example, when the chip temperature crosses 55°C, a Trojan might be triggered. A Trojan can also be triggered when a specific state of a state machine is reached.
2. *Externally triggered*: An externally triggered Trojan requires external input to the target module to activate the Trojan. The external trigger can be an user input or a component output. User input triggers can include push-buttons, switches, keyboards, or keywords/phrases in the input data stream. External component triggers may be from any of the components that interact with the target device. For example, a Trojan can be triggered by data coming through external interfaces like RS-232. Usually, the externally triggered Trojans need to have some sensing circuitry to receive the external trigger.

One can also classify the triggered Trojans into two categories – (1) Sensor-triggered and (2) Logically-triggered. Sensor triggered Trojans are triggered by

physical conditions such as temperature, voltage, etc. Logic-triggered Trojans are triggered by logic-conditions such as state of the flip-flops, counter, clock signal, data, instruction, and/or interrupts.

14.3.4 Effects

Trojans can also be characterized based on their undesirable effects. The severity of their effects on the target hardware and/or system can range from subtle disturbances to catastrophic system failures.

1. *Change functionality*: A Trojan can change the functionality of the target device and can cause subtle errors that may be difficult to detect. For example, a Trojan might cause an error detection module to accept inputs that should be rejected. This class also includes Trojans that modify the specification of the design as such modifications result in a different functionality instead of the intended functionality.
2. *Reduce reliability*: A Trojan can downgrade performance by intentionally changing device parameters. They may change the functional, interface or parametric characteristics such as power and delay. For example, a Trojan might insert more buffers in the interconnections of the chip and hence consume more power, which might in turn drain the battery quickly. Trojans can also insert faults such as stuck-at faults, bridging faults, which reduces the reliability of the system.
3. *Leak information*: A Trojan can leak sensitive information. This can occur through both covert and overt channels. Information can be leaked by radio frequency transmission, optical, thermal, power and timing side-channels and also via interfaces like RS-232 and JTAG. For example, a Trojan might leak the secret key of a cryptographic algorithm through the unused RS-232 ports.
4. *Denial-of-service*: Denial-of-service (DoS) Trojans can prevent operation of a function or resource. A Trojan may cause the target module to exhaust scarce resources like bandwidth, computation, and battery power. A Trojan may physically destroy, disable, or alter the configuration of the device. For example, a Trojan might cause the processor to ignore the interrupt from a specific peripheral. DoS may be either temporary or permanent.

14.3.5 Location

A Trojan can be inserted in a single component or spread across multiple components. The Trojans can be located in the processing units, memory, I/O, power grid or clock grid. Trojans distributed across multiple components may act independently

of each other or act as a group. Sometimes, they may be distributed among multiple components and together they can accomplish their attack objectives.

1. *Processing units*: Trojans might be inserted into the processing units. Any Trojan that is embedded into the logic units, that are part of the processing unit, may be grouped under this category. For example, a Trojan in the processor might change the execution order of instructions.
2. *Memory units*: Trojans in the memory blocks and their interface units may be placed under this category. They might alter the value stored in the memory. They might also block read or write access to certain memory locations. For example, a Trojan might change the contents of a PROM in an IC.
3. *I/O units*: Trojans can reside in the peripherals of the chip or within the PCB. These units interact with the external units. This position gives the Trojan a control over the data communication between the processor and the external components of the system. For example, a Trojan might alter the data coming through an RS-232 port.
4. *Power supply units*: Trojans may alter the voltage and current supplied to the chip and cause failure.
5. *Clock grids*: Trojans in the clock grids change the frequency of the clock and/or insert glitches in the clock supplied to the chip, causing fault attacks. Trojans in the clock grids can also freeze the clock signal that is supplied to the rest of the functional modules in the chip. For example, a Trojan might increase the skew of the clock signal supplied to specific parts of a chip.

By combining different categories from different attributes, interesting classes of Trojans can be obtained. Table 14.1 lists some of the classes from the Trojan taxonomy. Using these classes one can easily design some of the interesting Trojans. Specific Trojan detection methods targeting can be designed targeting specific classes of Trojans.

14.4 Case Study of Hardware Trojans

14.4.1 Malicious Off Chip Leakage Enabled by Side Channels (MOLES) [8]

Functionality: Covertly leak the secret keys of a crypto core to a remote attacker using the power side-channel of the IC. The secret key is XORed with a random number, which is generated by the Pseudo Random Number Generator (PRNG), to produce an encoded signal and this encoded signal is fed into a capacitor. The power consumption of the capacitor is directly related with the encoded signal. An attacker can measure the power consumption of the capacitor and can retrieve the encoded signal. As the attacker knows the seed of the PRNG, the attacker can decode the secret key from the encoded signal. For a tester, these signals appear as noise as the seed for the PRNG is not available to the tester. This technique is similar to the spread spectrum technique used in communication.

Table 14.1 Some classes of Trojans based on the different attributes of the taxonomy

Class no.	Trojan class
1	Specification phase – RTL – external-component triggered – leak information – in the I/O
2	Design phase – RTL – user-input triggered – leak information – in the I/O
3	Design phase – RTL – user-input triggered – change function – in the memory
4	Design phase – RTL – user-input triggered – leak information – in the processor
5	Design phase – RTL – user-input triggered – permanently deny service – in the processor
6	Design phase – RTL – user-input triggered – permanently deny service – in the I/O
7	Design phase – RTL – user-input triggered – permanently deny service – in the clock grid
8	Design phase – RTL – user-input triggered – permanently deny service – in the power supply
9	Design phase – RTL – user-input triggered – temporarily deny service – in the processor
10	Design phase – RTL – always on – leak information – in the processor
11	Design phase – RTL – always on – leak information – in the I/O
12	Design phase – RTL – physical-parameter triggered – permanently deny service – in the processor
13	Design phase – RTL – time triggered – temporarily deny service – in the I/O
14	Fabrication phase – transistor level – user-input triggered – change function – in the processor
15	Fabrication phase – transistor level – always on – change function – in the processor
16	Fabrication phase – transistor level – time triggered – change function – in the processor
17	Fabrication phase – physical level – always on – change function – in the processor
18	Specification phase – system level – user-input triggered – change function – in the processor
19	Specification phase – system level – time triggered – temporarily deny service – in the clock grid
20	Design phase – RTL – physical-parameter triggered – change function – in the processor
21	Design phase – RTL – physical-parameter triggered – permanently deny service – in the memory
22	Design phase – RTL – time triggered – change function – in the I/O
23	Design phase – RTL – time triggered – temporarily deny service – in the memory
24	Assembly and package phase – system level – external-component triggered – leak information – in the I/O
25	Assembly and package phase – system level – external-component triggered – permanently deny service – in the power supply
26	Fabrication phase – transistor level – time triggered – permanently deny service – in the clock grid
27	Fabrication phase – transistor level – always on – temporarily deny service – in the clock grid
28	Fabrication phase – physical level – always on – temporarily deny service – in the clock grid
29	Fabrication phase – physical level – physical-parameter triggered – permanently deny service – in the power supply

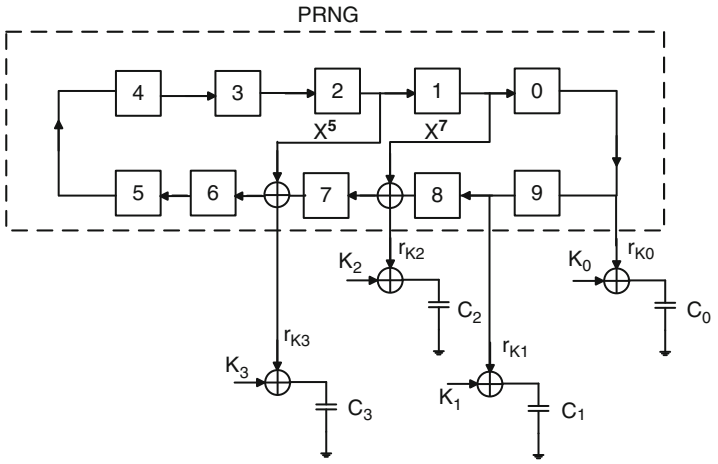


Fig. 14.2 Design of MOLES Trojan

Design: The Trojan, shown in Fig. 14.2, consists of a pseudo random number generator (PRNG) circuit, XOR gates, and capacitors. Each combination of an XOR gate and a capacitor is used to encode and leak a single bit of the secret key. The PRNG is implemented by using a Linear Feedback Shift Register (LFSR) and only the attacker knows the seed for the LFSR. A random number is generated for each clock cycle by the LFSR. The output of the XOR is connected to the I/O pins of the IC as the I/O pins usually have the largest capacitance in an IC.

Working: In every clock cycle, the random number generated by the LFSR is XORed with the key. Based on the generated encoded values, the capacitor can be charged or discharged which leads to additional power consumption. The attacker measures this additional power consumption and decodes the leaked secret key.

This Trojan can be inserted at the design phase of the design. Its complex structure makes it description at the RT level. As it does not contain any triggering part, this Trojan is always-on and its functionality is to leak the secret key.

14.4.2 Secret-Key Leakage Through RS232 Protocol [9]

This Trojan exploits the RS 232 protocol as shown in Fig. 14.3. This Trojan can be inserted both at the specification phase as well as the design phase. It is described at the RT level and located in the RS 232, an I/O unit. A secret key is transmitted at a higher baud rate, which is 12 times faster than that of the original baud rate. One packet of 8-bit data at higher baud rate can be transmitted at one bit interval of original baud rate. Each packet in a RS 232 protocol starts with a mark bit (logic "1"). Hence, during the transmission of the mark bit of the lower baud rate,

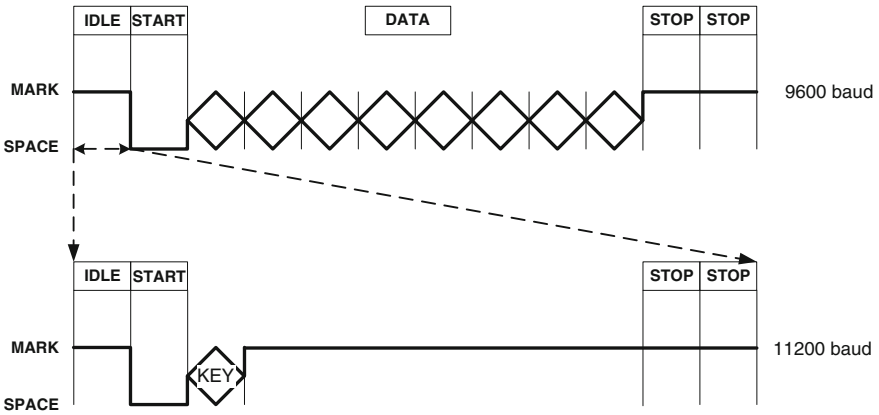


Fig. 14.3 Secret-key leakage through RS232 protocol

a packet at higher baud rate can be transmitted. The flexibility of this Trojan is reduced, due to the fact that the packet at higher baud rate should not affect the shape of the mark bit of the packet of the lower baud rate. But, if only one bit is transmitted in the data packet at the higher baud rate and all the other bits in the data packet are fixed at logic "1", then the packet at higher baud rate matches 11 out-of-12 times at best and 10 out-of 12 times at worst with that of the mark bit of lower baud rate. Thus a normal user, who is monitoring the RS 232 channel at the lower baud rate, can still correctly receive the data packet that is sent at the lower baud rate. But the malicious user, who is monitoring the RS 232 channel at the higher baud rate, can also effectively get the secret key that the adversary intends. Here, the adversary transmits one bit of the secret key in one data packet at higher baud rate.

14.4.3 Synthesis Tool Trojan

This is a software Trojan, that attacks the library of the CAD tool used for development of an IC [10]. For example, the file which is responsible for displaying the logic utilization in an ASIC design can be changed in such a way that the software always displays the logic utilization of the benign device. Even when some other hardware Trojans are introduced into it, the change in logic utilization due to the presence of Trojans, would not been known to the user.

14.4.4 Secret-Key Leakage Through Temperature Side-Channel

This Trojan covertly leaks the secret key which is embedded in a crypto chip through the temperature side-channel [9]. The temperature of the chip is raised to indicate

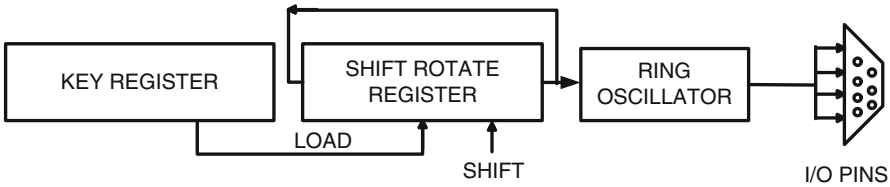


Fig. 14.4 Secret-key leakage through temperature side-channel

logic "1" and then cooled down to indicate logic "0." An attacker uses a temperature probe and monitors the temperature of the device and thereby retrieves the secret key (Fig. 14.4).

Design: The key register of the crypto-core is connected to a shift/rotate register. The LSB of the shift/rotate register acts as an enable signal of a ring oscillator which oscillates at very high frequencies. The output of the ring oscillator is connected to the I/O pins which have a very high parasitic capacitance. When a high frequency signal drives these I/O pins, it causes the parasitic capacitances to charge and discharge quickly and thereby increasing and decreasing the temperature of the IC rapidly.

Working: On power on, the keys from the key register are stored into the shift/rotate register. If the LSB value is logic "'1", the ring oscillator is activated and the I/O pins oscillate at high speed thereby heating up the IC. If the LSB value is logic "0", the ring oscillator is not activated and hence a normal temperature of the IC is maintained. In order for the attacker to measure a logic "1" and logic "0", sufficient time must be allowed for the IC to heat up or cool down. Normally, a time interval of 2–3 min is given to transmit a single bit. Hence, the shift/rotate register rotates at the rate of 2–3 min.

This Trojan can be inserted at the design phase of the design. Its complex structure makes it description at the RT level. As it does not contain any triggering part, this Trojan is always-on and its functionality is to leak the secret key.

14.4.5 Denial-of-Service (DoS) Trojan

This Trojans performs a denial of service (DoS) attack on the chip. This Trojan freezes the clock of the chip once it receives a particular input sequence [11] (Fig. 14.5).

Design: This Trojan consists of a series of XOR gates which compares the input sequence with a pre-defined binary value and an OR gate. One input of the OR gate is connected to the Reset input and other input is held at logic "1" by the comparator (if the input sequence matches with the pre-defined value). The output of the OR gate freezes the clock signal at logic "1".

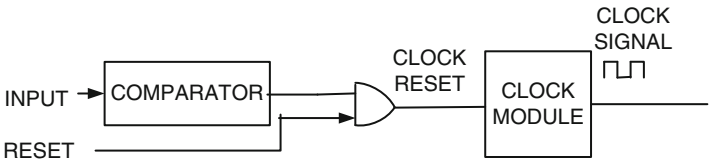


Fig. 14.5 Denial-of-service by freezing the clock

As the clock signal is frozen, the chip does not function unless it is reset. This Trojan can be inserted at design as well as fabrication phase and can be described at gate level. This is an input-triggered Trojan and it is located in the clock module of the chip.

14.4.6 Information Leakage Through VGA Display

The goal of the attacker is to leak the secret key information of the IC through the Video Graphics Array (VGA) display [12]. The refresh rate of the VGA is adjusted slightly above or below the normal refresh rate to indicate a logic "0" or logic "1". For a normal user, in the worst case the effects of variation in refresh rate are reflected as noise or flicker on the attached monitor and in the best case the variations do not cause any visually detectable effects. An attacker observes this variation in the refresh rate using an oscilloscope and decodes the secret key. This Trojan can be inserted at the design phase of the design. Its complex structure makes it description at the RT level. As it does not contain any triggering part, this Trojan is always-on and its functionality is to leak the secret key.

14.5 Conclusion

Hardware Trojans are becoming a serious threat. To understand the characteristics and develop Trojan detection techniques, characteristics of Trojans have to be fully analyzed. In this article, the taxonomy of hardware Trojan is presented which classifies Trojans based on their characteristics. Using the taxonomy the behavior of Hardware Trojans can be studied which leads to develop Trojan detection techniques against them. Several examples of Hardware Trojans with different characteristics are also analyzed which will help in understanding the behavior of Trojans at a large scale.

References

1. Adee S The hunt for the kill switch, IEEE Spectrum, http://www.darpa.mil/mto/solicitations/baa07-24/index.html
2. Defense Science Board, "Defense Science Board Task Force on High Performance Microchip Supply," Feb. 2005. http://www.cra.org/govaffairs/images/2005-02-HPMS_Report_Final.pdf
3. DARPA. http://www.darpa.mil/mto/solicitations/baa07-24/index.html
4. Karri R, Rajendran J, Rosenfeld K, Tehranipoor M (2010) Trustworthy hardware: identifying and classifying hardware trojans. Computer 43(10): 39–46
5. Rad RM, Wang X, Tehranipoor M, Plusquellic J (2008) Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In: IEEE/ACM International Conference on Computer-Aided Design, pp632–639, 10–13 Nov 2008
6. Wolff F, Papachristou C, Bhunia S, Chakraborty RS (2008) Towards trojan-free trusted ICs: problem analysis and detection scheme. In: Design, Automation and Test in Europe, 2008. DATE '08, pp 1362–1365, 10–14 March 2008
7. Wang X, Tehranipoor M, Plusquellic J (2008) Detecting malicious inclusions in secure hardware: challenges and solutions. IEEE International Hardware-Oriented Security and Trust (HOST), 2008
8. Lin L, Burleson W, Paar C (2009) MOLES: malicious off-chip leakage enabled by side-channels. In: IEEE/ACM International Conference on Computer-Aided Design – Digest of Technical Papers, pp 117–122, 2–5 Nov 2009
9. Baumgarten A, Clausman M, Lindemann B, Steffen M, Trotter B, Zambreno J Embedded Systems Challenge. http://isis.poly.edu/~vikram/iowa_state.pdf
10. Tamoney A, Kouttron D, Radocea A RPI team: number crunchers. Embedded Systems Challenge. http://isis.poly.edu/~vikram/rpi.pdf
11. Stefan D, Mitchell C, Almenar C Trojan attacks for compromising cryptographic security in FPGA encryption systems. Embedded Systems Challenge. http://isis.poly.edu/~vikram/cooper.pdf.
12. Gulurshivaram K, Hailemichael R, Jimenez J, Raju A Implementation of hardware trojans. Embedded Systems Challenge. http://isis.poly.edu/~vikram/poly_team1.pdf

Chapter 15
Hardware Trojan Detection

Seetharam Narasimhan and Swarup Bhunia

15.1 Introduction

Emerging trend of outsourcing the design and fabrication services to external facilities as well as increasing reliance on third-party Intellectual Property (IP) cores and electronic design automation tools makes integrated circuits (ICs) increasingly vulnerable to hardware Trojan attacks at different stages of its life-cycle. Figure 15.1 shows the modern IC design, fabrication, test, and deployment stages highlighting the level of trust at each stage. This scenario raises a new set of challenges for trust validation with respect to malicious design modification at various stages of an IC life-cycle, where untrusted components/personnel are involved [1]. In particular, it brings in the requirement for reliable detection of malicious design modification made in an untrusted fabrication facility, during post-manufacturing test. It also imposes a requirement for trust validation in IP cores obtained from untrusted third-party vendors.

The previous chapter describes various types of hardware Trojans and their degree of stealth in terms of testability. In this chapter, the focus is on hardware Trojan detection techniques. The different techniques are described and compared with respect to their capabilities and limitations. Unfortunately, conventional test and validation approaches cannot be used to reliably detect hardware Trojans. Conventional test strategies focus on identifying undesired functional behavior due to manufacturing defects in ICs. They do not target detection of additional functionality in a design due to malicious modification. It is likely that an intelligent adversary would insert hardware Trojans which are stealthy in nature. A cleverly designed Trojan is a relatively unobtrusive circuit which triggers a malfunction only under rare conditions in order to evade detection. It is also possible to design Trojans with no impact on functional outputs, such as a Trojan which

S. Narasimhan (✉) · S. Bhunia
Case Western Reserve University, Cleveland, Ohio, USA
e-mail: sxn124@case.edu; skb21@case.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_15,

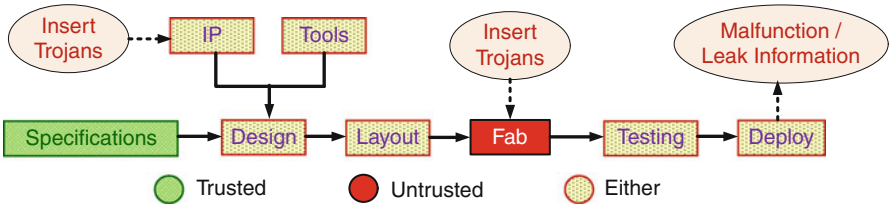


Fig. 15.1 Different stages of the IC design flow, showing level of trust [1]

leaks secret information through current signature [2]. It is extremely challenging to exhaustively generate test vectors for triggering a Trojan and observing its effect for inordinately vast spectrum of possible Trojan instances an adversary can employ. This is especially true for the sequential Trojans or "time-bombs," which are activated only on occurrence of a sequence of rare events. On the other hand, it is possible to detect Trojan circuits by observing their effect on a "side-channel parameter" such as power trace or path delay. An important advantage of such side-channel analysis is that it avoids the requirement of activating a Trojan and observing its effect in output logic values. Hence, test generation for side-channel analysis is expected to be less challenging. However, effectiveness of side-channel analysis is limited by the large process-variation effect in nanoscale technologies and measurement noise which reduce the detection sensitivity for ultra-small Trojans in large multi-million gate designs.

As existing manufacturing test-time solutions, either logic testing or side-channel analysis based, might not provide comprehensive Trojan coverage, it is possible to employ run-time monitoring to improve the level of assurance. Run-time approaches for Trojan detection are based on monitoring execution of critical computations to identify specific malicious behaviors that trigger during long hours of in-field operation. For example, a Trojan, which leaks information from a crypto-chip via wireless channels, might consume large spikes in power during a relatively idle period when no communication is taking place. Hence, run-time monitoring of power traces can be used to detect such Trojan instances.

Trust verification in untrusted third-party IP cores involves additional challenges. Unlike ICs, it is difficult to obtain golden or reference models in case of IPs. For third-party IPs, one can only trust the functional specifications. Hence, one possible approach for trust verification in IP is to apply directed functional tests. However, such tests can only be generated if Trojan trigger conditions and Trojan effects are pre-characterized, which limits the effectiveness of functional validation for arbitrary Trojan circuits. Another alternative approach is formal verification, such as sequential equivalence check, with a high-level reference model, obtained from the functional specification of an IP or its high-level structural information.

Note that existing Trojan detection approaches have their unique capabilities as well as limitations. There is yet no single "silver-bullet" technique available that can be applied to detect all classes of Trojans with high confidence. A possible solution to increase the level of confidence would be to combine various Trojan

detection approaches with complementary capabilities. Logic testing approach can be combined with side-channel analysis. Similarly, manufacturing test solutions can be combined with online monitoring. Finally, test approaches can be enhanced with design-for-security (DfS) solutions to achieve comprehensive protection against various hardware security attacks.

15.2 Hardware Trojan Detection in IC

In this section, an overview of existing Trojan detection approaches for post-Silicon trust validation of ICs is provided. In most of these approaches, it is assumed that the design, layout, and testing steps of the design flow shown in Fig. 15.1 are trusted and the only untrusted component is the fabrication facility. Hence, the golden design and test vectors can be obtained. For side-channel analysis, it is sometimes required to have a set of golden ICs in order to validate the measurement results and calibrate process noise. For such approaches, it is generally assumed that the golden measurement results can be closely modeled from extensive simulations or by destructive reverse-engineering of a few samples of the ICs which are extensively verified to be free from Trojans, and thus, allow for validation of the rest of the ICs non-destructively.

15.2.1 Classification of Trojan Detection Approaches

The overall taxonomy of Trojan detection approaches is shown in Fig. 15.2. They can be broadly classified into *destructive* and *non-destructive* approaches. The destructive techniques subject the manufactured ICs to demetallization using Chemical Mechanical Polishing to extract layer-by-layer images using Scanning Electron Microscope. This is followed by image reconstruction and analysis to identify transistors or gates and routing elements. A bottom-up reverse-engineering approach is utilized, whereby the structural blocks of the IC are first discovered using a template-matching approach, which are grouped together to identify all the implemented functions. These techniques [3] are traditionally used for reverse-engineering the secrets from a competitor's IC in order to gain competitive advantage by designing better products with advanced features. But they can be adapted for the purpose of validating the functionality of an in-house IC, whose specifications are golden, to identify whether any unintended circuit or function is present in the manufactured IC. However, such approaches are extremely expensive and time consuming (destructive analysis of a single chip takes several months [3]) and do not scale well with increase in transistor integration density. Moreover, the results of analyzing a sample cannot be extrapolated to the entire manufactured lot, because an adversary might affect only a small population of the ICs. Hence, each IC needs to be tested individually to ensure its trust. The only use of the destructive

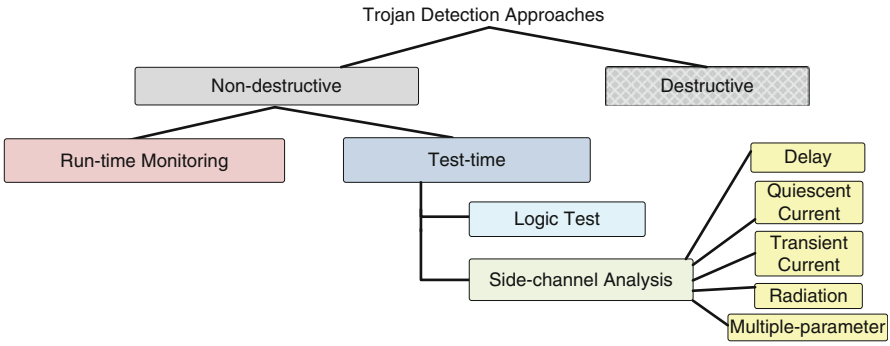


Fig. 15.2 Taxonomy of hardware Trojan detection approaches

techniques for Trojan detection is to test a small sample of manufactured ICs, in order to obtain a set of golden chips which can be used for process calibration and comparison of side-channel parameter with the rest of the ICs.

The non-destructive techniques can be classified into (a) *Run-time Monitoring* approaches and (b) *Test-time* approaches. It should be noted that the run-time approaches [4–7] are typically invasive approaches where some DfS is involved. These run-time monitoring circuits can exploit pre-existing redundancy in the circuit by using a reconfigurable core [4] in multicore systems [7] to avoid an infected part of the circuit. Thus, the reliability of circuit operation can be assured even in presence of Trojans, even though the Trojan-infected chips are not discarded. On the other hand, for mission-critical systems, the chips may be equipped with a self-destructive packaging which can be externally triggered by the user or internally triggered by the run-time Trojan monitor, on the detection of a malfunction. One should, however, be careful to ensure that the run-time monitor or circuitry for triggering destruction of the chips is not compromised and can be controlled by a trusted user alone.

The test-time approaches also can be aided by DfS circuits, similar to Design for Testability circuits, like scan-chains or Built-in Self-Test circuitry. These circuits can enhance the Trojan detection sensitivity or coverage substantially. However, it needs to be ensured that the extra test circuitry is not compromised. Similarly, if there is an easily identifiable signal for enabling test, the attacker could possibly misuse this signal by disabling the Trojan circuit during test. Throughout the test duration, the Trojan will not trigger its malfunction and it might even be gated at the supply during test-time to prevent side-channel information leakage. Typically, the search has been for a non-invasive Trojan detection technique which does not have to alter the design flow or add any overhead to the design, but still can identify any IC with malicious insertion, during post-manufacturing test. Test-time approaches are further classified into (1) *Logic-testing*-based approaches and (2) *Side-channel analysis*-based approaches.

Logic-testing approaches [8, 9] focus on test-vector generation and application for activating a Trojan circuit and observing its malicious effect on the payload at the primary outputs. This method is similar in philosophy to traditional stuck-at-fault (s-a-f) testing; however, the models of Trojans are very different from the fault models. Manufacturing defects are typically modeled as stuck-at-faults where an internal node is stuck at a particular logic value. The difficulty in testing these faults is in terms of exciting all internal nodes to all possible logic values and observing the effect at some primary output. Nodes which are hard to excite or hard to observe are called low controllability and low observability nodes. As the number of gates increases, the number of nodes which are hard to test also increases and it makes testing all nodes for comprehensive fault coverage an exponentially difficult task. On the other hand, Trojans are modeled as a cleverly inserted gate (or set of gates) which trigger under rare conditions and exhibit some malfunction. The number of possible Trojan circuits of a particular type and size is an exponential function of the number of nodes of the circuit. Also, for sequential Trojans which take multiple rare events to activate, it might not be possible to observe the malfunction due to the Trojan during test-time. Finally, as the number of possible Trojans is enormous, the conventional techniques for estimating the coverage of fault detection do not readily apply to Trojan detection coverage.

On the other hand, side-channel analysis approaches are based on the fact that any malicious insertion in the IC should reflect its presence in some side-channel parameter such as leakage current or quiescent supply current (IDDQ) [10–12] or dynamic power trace (IDDT) [13–17] or path-delay characteristic [18, 19] or Electromagnetic radiation (EM) due to switching activity [13], or a combination of these [20]. For example, if the original circuit has N_{orig} gates and consumes I_{orig} quiescent current, the insertion of N_{troj} extra gates in the circuit for implementing the Trojan will increase the current by I_{troj} which can be observed by measuring the supply current under nominal conditions. Various side-channel approaches have been proposed in literature and they concur that the major drawback of these approaches is their susceptibility to error due to process and environment noise. Even the noise due to the measurement setup can interfere with the analysis causing wrong inferences to be drawn regarding presence or absence of Trojan in the circuit. Hence, the Trojan detection problem is posed as a statistical event with the goal being to maximize the *Probability of Detection* (P_{D}) while minimizing the *Probability of False Alarm* (P_{FA}). Test-vector generation can also play an important role in the side-channel analysis approaches in order to increase the Trojan *detection sensitivity* [14–17] especially for ultra-small Trojans in large multi-million gate System-on-a-Chip (SoC) circuits. Typically, Trojan circuits are assumed to be small in size compared to the original design, since they are inserted at the foundry by modifying the layout of the original design. The attackers are assumed to be exploiting the blank spaces in the layout to insert few extra gates and rewiring the circuit to implement their malicious intent. However, the side-channel approaches have major advantage over logic-testing approaches in terms of not having to activate the Trojan to be able to detect it. Hence, they are useful for detecting passive payload Trojans which do not cause malfunction but intend to

leak secret information through unobtrusive side-channels. If the process noise can be calibrated and environmental and measurement noise nullified then the presence of Trojan circuit will surely be reflected in the measured parameter.

15.2.2 Challenges of Trojan Detection

The major challenges with regard to Trojan detection can be classified as follows:

(a) The choice of proper Trojan model.
(b) Test generation to activate Trojan or to increase sensitivity of Trojan in the side-channel parameter.
(c) Elimination/calibration of process, environment, and measurement noise (in case of side-channel analysis-based approaches).

15.2.2.1 Trojan Modeling

Different researchers have chosen different Trojan models to demonstrate the features of their proposed approach. In order to bring uniformity to the process and to enable comparison of different approaches, people have developed structured taxonomy [21] of Trojans based on parameters which describe their size, stealth, activation probability, payload effect, etc. As described in the previous chapter, one can have multiple features for the same Trojan, but most of the detection approaches are based on particular features. For example, if the Trojan's payload is to cause leaking of secret information then its size can be small or large, its activation condition can be combinational or sequential, it may or may not have significant impact on critical path delay, and it may be activated by digital values at some internal nodes or analog conditions like rise in temperature. But from the point of logic testing, the activation condition is important as well as the fact whether the payload causes any change in logic values which is observable at some primary output or whether the Trojan assists in side-channel information leakage [2].

In terms of test generation for logic testing, the digitally triggered, digital-payload Trojan model [8] is most useful. A cleverly designed Trojan will be activated by rare circuit node conditions and will have its payload as a critical node in terms of functionality, but low observable node in terms of testing, to evade detection during normal functional testing. If the Trojan includes sequential elements, such as rare-event triggered counters then the test vectors need to replicate the exact sequence for triggering it, making it harder-to-detect. On the other hand, when choosing rare values at internal nodes as inputs for the Trojan, it should be noted that the combined rare event should be excitable; otherwise the Trojan will never be triggered in real life. This constrains the number of possible Trojans for fixed parameter values of this model.

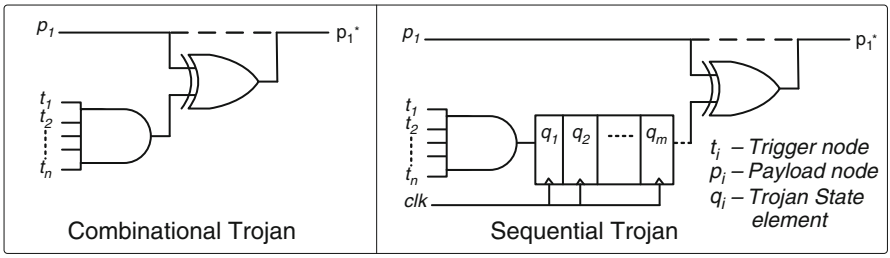


Fig. 15.3 Combinational and sequential Trojan models for logic testing approach

Figure 15.3 shows generic models for combinational and sequential Trojans [8]. The trigger condition is an "n"-bit value at internal nodes which is assumed to be rare enough to evade normal functional testing. The payload is defined as a node which is inverted when the Trojan is activated. To make it harder-to-detect, one might consider a sequential Trojan which requires the rare event to repeat "2^m" times before the Trojan gets activated and inverts the payload node. The sequential state machine is considered in its simplest form to be a counter and the effect of the output on the payload is considered to be an XOR function to have maximal impact. In more generic models, the counter can be replaced by any Finite State Machine (FSM) and the circuit can be modified as a function of Trojan output and the payload node. Similarly, the Trojan trigger condition is modeled as an AND function of several rare nodes in order to make the combined event rarer; however, the attacker may choose to reuse existing logic within the circuit to implement the trigger condition, without adding too many extra gates.

For Trojan detection using side-channel analysis approaches, the Trojan model can be as simple as a current sink or a single inactive gate to model its effect on the quiescent supply current, or an extra node capacitance to model the effect on path delay. However, to detect a Trojan by monitoring transient current or radiation, one needs to induce switching activity in the gates constituting the Trojan. In these cases, the Trojan model can be similar to that used for logic-testing approaches. Usually, the larger the size of the Trojan, the easier it is to detect its effect on a side-channel parameter, like supply current. However, the larger the original design, the impact will be masked more easily by noise. Hence, for determining the scalability of any side-channel approach, one should consider the relative size of the Trojan circuit with respect to the original circuit and its relative impact on the measured side-channel parameter when choosing an appropriate model. Other things which need to be modeled for accurate evaluation of a side-channel analysis approach include the measurement circuitry, the power supply grid, the parasitic capacitances in the original circuit, and the variations in their parameters, which can cause golden chips to have variations in the measured side-channel parameter as well.

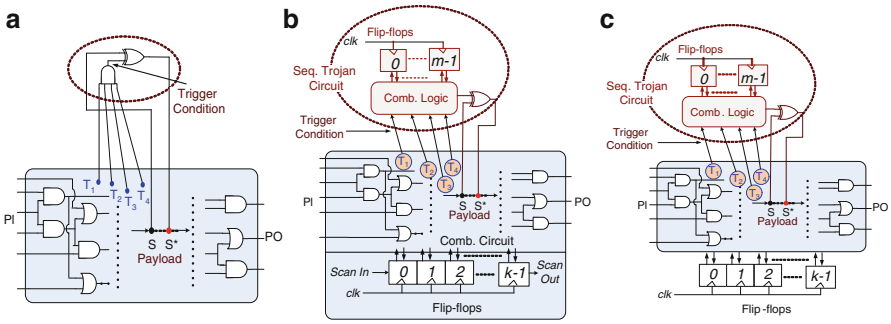


Fig. 15.4 The difficulty of triggering hardware Trojans increases from combinational to sequential Trojans and if the original sequential circuit does not have full-scan; (**a**) combinational Trojan in a combinational circuit; (**b**) sequential Trojan in a full-scan sequential circuit; (**c**) sequential Trojan in a non-scan sequential circuit. *PI*: Primary Inputs, *PO*: Primary Outputs. T_i: Trojan trigger nodes, *S*: Original circuit node which is modified to S^* by the Trojan payload

15.2.2.2 Test Generation

Test generation is an important problem for any Trojan detection approach. It is to be noted that during test-application for Trojan detection, there should not be any clearly identifiable Test Enable (TE) signal. In order to evade detection during chip testing, the Trojan triggering logic can exploit the TE control line to disable the Trojan. Hence, scan-based designs cannot improve the security and functional testing must be involved. As noted previously, the Trojan models are different from stuck-at-fault models used for conventional test generation. Though application of random functional vectors or Automatic Test Pattern Generation (ATPG) tool-generated vectors might be able to detect randomly inserted Trojans following the afore-mentioned models, the challenge is to generate vectors to detect the cleverly designed Trojans which will only be triggered by extremely rare conditions derived from nodes with low controllability and which affect payload nodes with low observability. Different types of Trojans (combinational and sequential) inserted in combinational and sequential circuits (with and without scan-chain) are illustrated in Fig. 15.4. In presence of scan-chain, the original sequential circuit is reduced to a combinational circuit with k extra primary inputs (PI) and primary outputs (PO). The Trojan models are same as those described previously.

The second challenge in terms of test generation is the inordinately large number of possible Trojans which can be derived using a finite set of internal nodes. This enormously large Trojan space makes the generation of an exhaustive set of test vectors to detect all possible Trojans computationally infeasible. As an example, even with the constraint of $n = 4$ trigger nodes and a single payload node, a small ISCAS-85 benchmark circuit c880 with 451 gates can have ${}^{451}C_4 \approx 4.1 \times 10^{10}$ triggers and ${}^{451}C_4 \times (451-4) \approx 1.8 \times 10^{13}$ possible combinational Trojan instances, respectively. As mentioned earlier, not all of the Trojans will be functionally feasible; hence, test generation needs to consider whether the combined rare event

is justifiable from the primary inputs or not. Similarly, the initializing state for the scan-chain flip-flops needs to be a functionally valid state and not an "unreachable state" of the circuit's state machine.

Finally, considering "p" primary inputs and "q" state elements, one can perform exhaustive testing with 2^{p+q} test vectors. This might be sufficient to trigger all possible combinational Trojans, given that $(p+q)$ is reasonably small, which is not the case for typical multi-million gate SoCs. However, the presence of sequential "time-bomb" Trojans which require the rare event to be triggered "m" times cannot be detected even using exhaustive testing. Actually, state-elements in sequential Trojans may not use the global clock; instead they might use rare switching activity of internal circuit nodes as their clock signal. Such design again lowers Trojan trigger possibility rendering them almost transparent with respect to logic-testing-based approaches. Moreover, the presence of memory elements within these SoCs causes the valid state space to explode exponentially. This motivates the use of statistical test generation approaches, as proposed in [8].

15.2.2.3 Process, Environment, and Measurement Noise

In terms of side-channel analysis, the test generation is comparatively simpler, since one does not need to completely activate the Trojan in order to observe its impact on the measured parameter like supply current. However, modern nano-scale technologies suffer from large variations in process parameters due to uncertainties in the manufacturing process. This causes different ICs at different process corners to exhibit wide variations in circuit parameters like delay and current. In fact it has been shown in [22] that process variations in 180 nm technology node caused up to 30% variation in delay and 20X variation in leakage current. The effect of process variations is more on leakage current because of the exponential dependence of leakage current on the transistor threshold voltage. Figure 15.5 shows the effect of process variations on the transistor threshold voltage which can be modeled as Gaussian distributions of random intra-die variations on top of inter-die variations. The simulated current for 1,000 dies shows significant overlap in current showing that the effect of Trojan circuit is masked by the process noise. It also shows the effect of measurement noise on recorded current waveforms for the same set of vectors over multiple time intervals.

As seen in Fig. 15.5c, the variation due to process noise is also a function of the value of the measured parameter. Hence, for a large circuit with high activity, the transient supply current (IDDT) can have large variations, which overlap with the effect due to a small Trojan circuit. The problem reduces to determination of a proper decision boundary or threshold to classify the ICs under test as golden or Trojan. Any golden IC which has higher current than the threshold is misclassified as a Trojan (*false positive*) and any Trojan IC which consumes less current is misclassified as a golden IC (*false negative*). The choice of threshold to account for the process noise can lead to a trade-off between *probability of detection* and

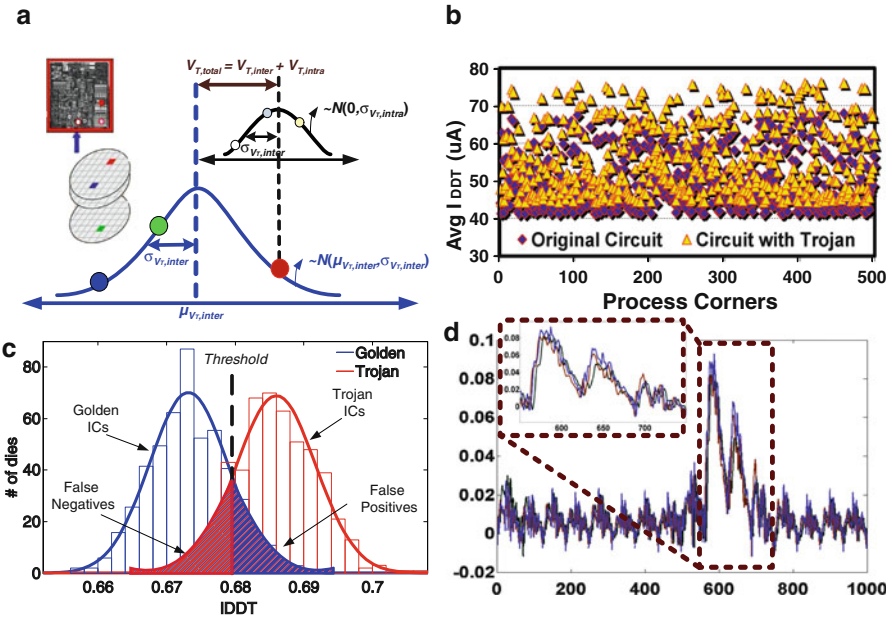


Fig. 15.5 (**a**) Inter- and intra-die process variations can be modeled as variations in the transistor threshold voltage (Vth). (**b**) Considering Gaussian distributions for both components of process variation, Monte Carlo simulations in HSPICE are used to obtain the supply current for 500 dies containing an 8-bit ALU circuit without (Golden) and with (Trojan) an 8-bit comparator as the malicious circuit. (**c**) There is sufficient overlap in the transient current distribution of the 1,000 dies even for a relatively large Trojan, illustrating the masking effect due to process noise. (**d**) The effect of measurement noise on transient current recordings for 16 clock cycles is shown

probability of false alarm. To be on the safe side, it is preferable to discard a golden circuit as a Trojan containing one (more false positives) rather than to allow an infected IC to pass the test (false negative).

One way to reduce the probability of error is to increase the gap between original and Trojan-affected parameter. This is defined in terms of increasing the sensitivity. It should be noted that the values of the side-channel parameters vary widely with circuit conditions; hence the choice of the input vector set for which the parameters are measured plays a significant role. The vectors should ideally be chosen such that the contribution of the original circuit is minimized and the effect of the Trojan is pronounced. Region-based partitioning and directed test vector generation to induce switching activity in possible Trojan instances can cause them to be easily detected. On the other hand, the Trojan detection sensitivity can be increased by decreasing the effect of process noise. Most of the proposed Trojan detection approaches are concerned with calibrating the process noise and reducing the measured side-channel parameter to a common denominator where the effect of the Trojan stands out clearly. Typical techniques involve process calibration by normalization and using statistical averaging techniques to reduce noise. In an ideal scenario, if the

noise can be completely eliminated, the Trojan effect will distinctly stand out from the golden parameter, causing 100% detection, irrespective of the size of the Trojan circuit.

15.2.3 Test and Validation Approaches

First, some of the unique test-time Trojan detection approaches are described and their relative advantages and disadvantages are compared. Most of the proposed Trojan detection approaches belong to this category. Some of them deal with modifications of traditional testing approaches to incorporate detection of Trojans, whereas others deal with identification of Trojan effect in some side-channel parameter.

15.2.3.1 Logic Testing-Based Approaches

Since the problem of Trojan detection by functional testing is different from conventional fault-testing, statistical approaches for vector generation are more suitable. A randomization-based technique to probabilistically compare the functionality of the circuit post-implementation with the original golden design is described in [9]. The technique is derived from probabilistic equivalence checking, wherein each circuit output can be correlated to a unique probability distribution at its input cone in order to obtain a logic "1" value at the output node with a predefined probability. This distribution is used to generate random input vectors to test the untrusted circuit and compare its outputs with those of the golden design. Any mismatch in the circuit's functionality post-implementation is used to identify presence of a Trojan circuit and the input vector causing the mismatch is defined as the *fingerprint* of that particular Trojan. This technique does not consider any model of the Trojan for generating the vectors and is based on finding equivalence between two circuits. The authors claim that the larger the functional disruption caused by the Trojan, the easier it is to detect it. Hence, this technique can be complemented with an approach that generates the random vectors with an intention of triggering possible Trojan inputs to increase detection coverage. This technique is also useful for validating the trust of a third-party IP, given its golden trusted model.

In [8], a statistical vector generation approach for Trojan detection called *MERO* is introduced. It generates an optimal set of test vectors that can trigger each rare node in a circuit to its rare value multiple times (N times, where "N" is a user-specified parameter), similar to the concept of N-Detect test [23]. The rareness and the number of trigger nodes for the Trojan model as well as the nature of the Trojan, whether combinational or sequential, are variable inputs to the algorithm. By individually activating the rare nodes, it improves the probability of triggering a Trojan activated by a rare combination of a selection of the nodes, compared to purely random patterns. Using such logic testing approaches, it is

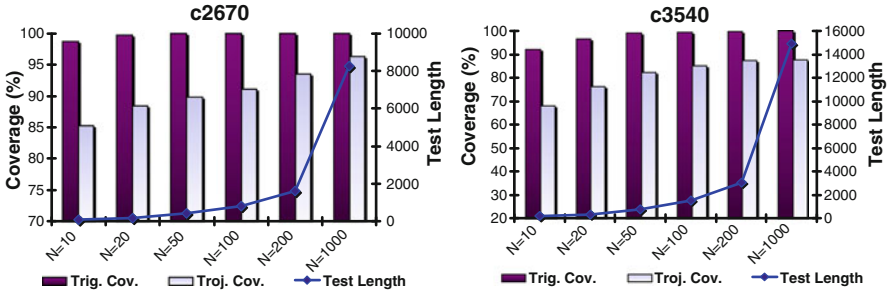


Fig. 15.6 Trigger coverage and Trojan coverage and test length for two ISCAS-85 benchmark circuits for different values of "N," using the MERO approach [8]

easier to trigger a Trojan than to detect its malfunction at the primary outputs (Trojan detection), because the Trojan payload nodes are typically low-observability nodes. This distinction is clarified by considering two parameters to qualify the effectiveness of the test generation approach, namely, *Trigger coverage* and *Trojan coverage* [8]. While computing these coverage metrics, it should be noted that Trojans that can never be triggered should be discarded. The Trojan detection coverage increases for higher values of "N," at the cost of increased test length, as shown in Fig. 15.6 for two ISCAS-85 benchmark circuits. However, it achieves an 85% reduction in test length compared to a weighted random pattern set, for similar Trojan detection coverage. The *MERO* approach also takes advantage of scan flip-flops in a sequential circuit to further decrease the test length and increase Trojan coverage. The relatively higher trigger coverage attained by using the *MERO* approach indicates its potential utility in side-channel analysis-based techniques, for generating vectors which cause switching activity in a Trojan circuit, even if its malfunction is not activated.

An obfuscation-based design approach [24] can be used to prevent the attacker from successfully reverse-engineering the functionality of the IP cores in an IC. As a result, the attackers make incorrect assumptions regarding the rarity of the internal nodes of the circuit and insert Trojans which are either easily triggered or never triggered at all (*benign Trojans*), thus defeating the attackers' purpose. This design technique [24] is shown to increase Trojan detection coverage further, based on the *MERO* approach.

Design for Security techniques can also be used for increasing the testability of a large circuit by enhancing the controllability and observability of probable Trojan trigger and payload nodes. By inserting a control FSM, which is difficult to identify, different modules in a complex design can be selectively tested in uniquely defined *transparent* states. By using specific sequence of inputs, each module is taken to a special *transparent* mode [25], where all other modules are by-passed. The internal nodes at the input of the module are testable from the primary inputs and a compacted signature of the output node values is presented at the primary outputs, which indicates the presence or absence of a Trojan.

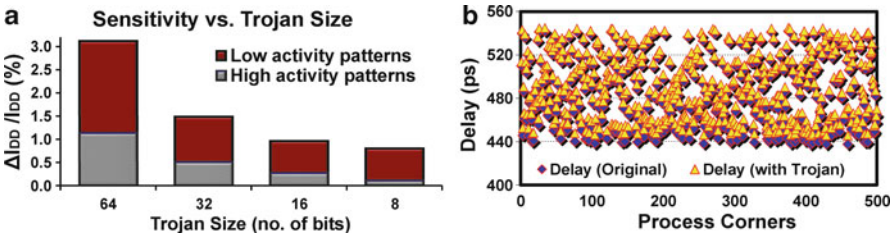


Fig. 15.7 (**a**) Sensitivity of IDDT-based Trojan detection approach decreases with decrease in Trojan size but increases by choosing proper test vectors (**b**) The critical path delay of a circuit varies widely due to process variations, which can mask the effect due to a Trojan circuit

15.2.3.2 Side-Channel Analysis-Based Approaches

All the side-channel analysis approaches are based on observing the effect of an inserted Trojan on a physical parameter such as power supply current (quiescent or transient), power consumption, or path delay. The advantage of these approaches lies in the fact that even if the Trojan circuit does not cause observable malfunction in the circuit during test, the presence of the extra circuitry can be reflected in some side-channel parameter. However, the main challenges associated with side-channel analysis are large process variations in modern nanometer technologies and measurement noise, which can mask the effect of the Trojan circuit, especially for small Trojans. Examples of side-channel parameters, which can be used to detect the presence of extra circuitry, are given later; followed by a description of specific techniques for eliminating the effect of process noise and increasing Trojan detection sensitivity.

IDDQ: The presence of extra circuitry inside a circuit will change the measurable parameters of the circuit, even if the circuit does not activate any malfunction during the actual measurement. This is intuitive in the case of current drawn from the power supply. Any extra gate will consume extra leakage power, which is additive and can be used to distinguish golden circuits from Trojan-infected ones. However, the leakage current effect due to a few extra gates gives very poor sensitivity when measuring the total leakage current for a multi-million gate SoC. To increase sensitivity, one can measure current from multiple power pins, thus effectively reducing the problem to that of detecting few gates in a fraction of the total gates in the IC. The region-based approach also helps localize a small region of the IC, which is infected by the Trojan.

IDDT: Transient power supply current, reflecting the dynamic power due to switching activity, can be measured from the same power pins to give information about the number of gates switching for the specific pair of input vectors. Hence, input vectors can be used to give an additional level of control on increasing the Trojan detection sensitivity by activating smaller sub-blocks of the whole circuit. As shown in Fig. 15.7a, the sensitivity of an IDDT-based Trojan detection approach

decreases with Trojan size, but can be increased by proper choice of vectors. This enhances the scalability of the approach to larger designs and sub-100 nm technologies.

Delay: The third parameter which can be used to detect Trojans is path delay. There are several ways in which the Trojan circuit can affect the path delay. If the Trojan circuit lies on the path whose delay is being measured, it will add several gate delays to the path delay. Even if the applied test vector does not activate the Trojan but contains an input of any Trojan gate, it adds load capacitance to the node and thus, increases the path delay. However, for large path delays, such a minor change in delay might be masked by process variations, as shown in Fig. 15.7b. Hence, it is better if the Trojan is activated by proper input vector control aimed at triggering all possible Trojans according to some fixed model, and then comparing the measured path delays with those of the golden circuit. Also, it is only possible to measure path delays which originate from primary inputs and terminate at primary outputs. Hence, for sequential circuits without full-scan, one needs to adopt invasive design-time techniques for measuring all path delays.

EM: Electromagnetic radiation due to switching activity of different gates can also be observed to detect switching activity in extra Trojan gates in a circuit in a non-invasive non-destructive manner. However, apart from having different measurement circuitry, any technique based on measuring this parameter falls in the same category as that measuring transient supply current.

The sensitivity of a simple side-channel approach based on comparison of measured physical parameter "I" can be used to consider various noise effects and different calibration techniques. For example, in a simple side-channel approach, considering ideal situation with no noise, any golden circuit is expected to have the measured parameter value as I_{orig}. The deviation introduced by an extra Trojan circuit causes the measured value for an infected chip to be $I_{\text{T}} = I_{\text{orig}} + \Delta I_{\text{T}}$. The sensitivity, in the absence of noise, is proportional to ΔI_{T} and inversely proportional to I_{orig}. Now, with the presence of measurement noise (In_{meas}) and process noise (In_{proc}), the measured values of the non-infected circuits can vary from I_{orig} by $In_{\text{meas}} + In_{\text{proc}}$. The process noise In_{proc} is a time-invariant constant which affects different ICs differently. It can further be decomposed to contain inter- and intra-die components, with the intra-die component having systematic and random sub-components, as shown in Fig. 15.5a. The measurement noise In_{meas} has a temporal variation (due to temperature and other factors) for the same IC and a dc offset due to measurement circuitry. Considering a simple side-channel analysis approach, the sensitivity can be defined as:

$$\text{Sens} = \frac{I_{\text{T}} - I_{\text{orig}}}{(I_{\text{orig}} + In_1) - (I_{\text{orig}} + In_2)} = \frac{\Delta I_{\text{T}}}{\Delta In_{\text{meas}} + \Delta In_{\text{proc}}}. \tag{15.1}$$

Existing side-channel approaches tend to perform process calibration by using normalization (or process-corner estimation) and measurement noise calibration by averaging over multiple measurements to get rid of random noise. In order to get

rid of inter-die variations and calibrate systematic intra-die variations, region-based approaches are used where measurements from multiple power pins corresponding to activation of distinct regions, help to compare the measured parameter from the same IC under different circumstances. By using a region-based approach, one can also increase the sensitivity since the I_{orig} value gets reduced and any noise which is proportional to the measured value (e.g., process noise) gets reduced as well. Statistical signal processing techniques can be used to calibrate the process noise and reduce the measured side-channel parameter to be close to the nominal value, for all ICs.

The concept of side-channel analysis for Trojan detection is borrowed from the field of side-channel attacks where secret keys used by cryptographic chips can be estimated by observing a side-channel parameter such as supply current, and performing correlations with current traces for different predicted key values. Techniques like Simple Power Analysis (SPA) or timing analysis can be used for identifying presence of Trojan circuits since any unexpected activity will be reflected in these parameters. Even non-activated Trojans which are large enough to cause an appreciable change in the leakage current can be detected easily. Also, the measurement noise can be reduced by averaging over multiple measurements till the Trojan effect stands out. However, due to process variations, the golden current value is not a single value. In order to identify the effect of the Trojan amidst process noise, one can use advanced signal processing techniques such as Karhunen-Loeve expansion to derive a current-based signature for golden ICs which is violated due to presence of malicious circuitry. The *IC fingerprinting* technique [13] is used to calibrate the process noise and identify subspaces from the transformed power trace signal, which highlight the presence of the Trojan. These subspaces are likely to correspond to test vectors where the Trojan circuit is actively switching while the rest of the circuit has less pronounced current, as the process noise is observed to be proportional to the amplitude of the power trace. From the analysis of power traces [13], it is possible to identify Trojan instances with an equivalent area as small as 0.01% of the total size of the circuit, in the presence of $\pm 7.5\%$ random parameter variations.

To increase probability of activation of Trojan using test vectors applied only at the primary inputs, proper test vector generation needs to be considered. For large sequential circuits, a region-based circuit activation approach is useful for increasing the sensitivity of Trojan detection by side-channel analysis. The circuit can be partitioned into functionally distinct regions or structurally partitioned with minimum overlap between the gates in each partition. These partitions could be in terms of the sequential state elements as well as proximity of gates to each other. Next, directed test vectors are generated to maximize switching activity in the region under consideration while minimizing activity in the rest of the circuit. This increases the sensitivity of the current measurement to detect Trojans present in the activated region. The effectiveness of such a region-based test generation approach compared to random test patterns is demonstrated in [14] in terms of relative switching activity induced in Trojan-containing circuits. A *sustained vector* technique can be used to further increase the Trojan detection sensitivity. Here, the

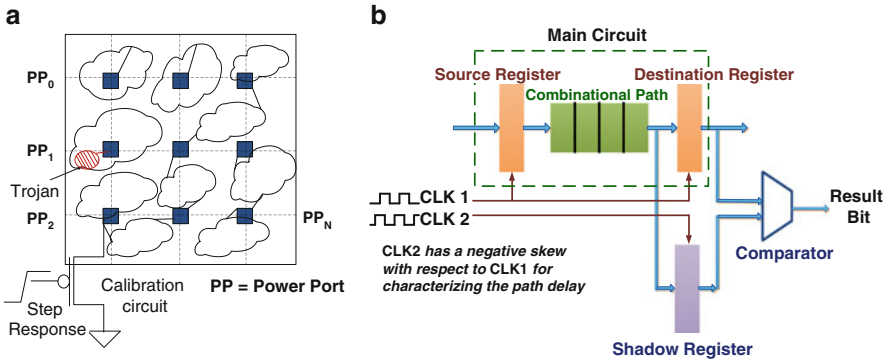


Fig. 15.8 (**a**) A region-based approach using current measurements from multiple power ports [10] can be used for isolating Trojan effect. The shorting transistor is used for calibrating [26] effect of process variations. (**b**) At-speed delay characterization approach [19] for detecting any delay variation due to presence of Trojan circuit

primary inputs are kept constant for multiple cycles when the only switching is due to change in state element values which helps in further reducing the activity within the original circuit, thus reducing I_{orig}. This technique is shown [15] to magnify the power profile difference between the original and the infected circuit by up to thirty times compared to previous approaches.

Another region-based approach for calibrating process noise is defined in terms of the power supply network on a chip [10, 26]. Typically, ICs have a distributed power supply grid at the higher metal layers which contain metal bumps connected to different power supply pins, as shown in Fig. 15.8a. Externally, at the board level, these pins can be connected to a unified power supply. Measurement of current traces from multiple power-ports for different input vectors, followed by integration of the current traces can be used to quantify the charge-transfer during switching activity. Any Trojan-containing circuit accumulates more charge with time, because of extra switching activity compared to the golden circuit and is detected by comparing the integrated current traces for multiple ICs. As the current measurements are done from multiple power ports, the location of the Trojan can also be estimated. The current measurement accuracy can be increased by incorporating current sensors inside the IC at each of the power ports. This allows for calibration of variation in power grid impedance. By measuring power supply transient signals from multiple power ports, one can increase the resolution of the side-channel analysis approach since the current measured from each port corresponds to a localized region containing a manageable fraction of the total number of gates in a large IC. These individual power supply signals are compared with each other to identify the presence of Trojan circuit, which affects only a small region.

Different calibration schemes [26] can be used to eliminate resistive variations in the power ports as well as to calibrate inter- and intra-die process variations. A process calibration circuit (see Fig. 15.8a) consists of a single transistor connected between the power port and the ground which can be selectively turned on using

a scan flip-flop. The measured current for a set of test vectors from multiple power ports are normalized using the calibration matrix and those for orthogonal power ports are compared using scatter-plots to identify a region (*prediction ellipse*) which defines the boundary between golden and Trojan ICs. The technique was capable of detecting around 50% of activated Trojans and 30% of inactive Trojans in an ISCAS-85 benchmark circuit. In order to increase the probability of activation of rare internal nodes of the circuit to their rare values, dummy scan flip-flops feeding AND/OR gates connected to the node can be introduced. This design scheme increases the controllability of the internal nodes and makes it difficult for an attacker to choose rare trigger nodes to hide the Trojan. However, the node which was rare can still be exploited by the attacker to evade detection during testing. A layout-aware scan-cell re-ordering scheme [16] can increase the controllability of switching in particular regions. This increases the probability of switching in any Trojan circuit, thus enhancing the probability of its detection. Similar statistical analysis can be performed on quiescent current (IDDQ) [10] measured from the multiple power ports of ICs containing calibration circuitry in order to detect and localize Trojans, which can be modeled as defects. However, the presence of invasive DfS measures such as calibration circuitry or on-chip current sensors can be easily detected by an attacker who can potentially bypass them to implement a hard-to-detect Trojan.

Path delays measured at output pins for a set of test vectors can be used to identify the presence of a Trojan which lies on the measured path and changes its delay. As shown in [18], the size of such delay information obtained for multiple paths in a circuit can be extremely large, especially for large circuits. Data compression techniques like Principal Component Analysis (PCA) can be used to reduce the data dimensions and construct a convex hull to estimate the boundary of golden chips with process variation. These reduced data points are referred to as *Path-Delay Fingerprints*. Any Trojan which affects the path delays considerably will be detected if the test set excites the particular path and the delay effect propagates to the primary outputs. The procedure is shown to detect 100% of *implicit payload* Trojans, whose gates add to path delay, occupying only 0.13% of the total area under $\pm 7.5\%$ process variations. However, it has relatively poor detection coverage (36%) for *explicit payload* Trojans which only add load capacitance to the measured path delays.

The *at-speed delay characterization* technique [19] introduces shadow registers (see Fig. 15.8b) inside the chip along with comparators to identify the path delay slack for internal register-to-register delays. This DfS technique uses increasing negative skew for the shadow register clock compared to the functional clock and compares the latched output for a series of inputs in order to determine the delay distribution. It can also help calibrate process variation-induced delay variations as statistical path-delay distributions and any hardware Trojan which adds to the measured path delays can be detected.A ring-oscillator can be used as a thermal monitor for calibrating temperature-induced delay variations. This technique needs to be combined with test-vector generation approaches along with appropriate Trojan model in order to achieve high coverage with minimal test-time. Also, the

area overhead for the scheme can be amortized by using the same circuit for IC authentication or environmental variation calibration purposes. Finally, this scheme can also be used at run-time for detecting dormant hardware Trojans which only get activated in-field. This method was shown to be capable of detecting Trojans in an 8×8 array multiplier circuit under ±20% process variations [19]. Another scheme to make the delay effect of Trojan circuits on internal paths visible is to configure the paths into ring-oscillators whose operating frequency changes widely because of internal loading effects or even because of cross-talk with the wires used for implementing the Trojan circuit. The frequency of ring oscillators is measured using on-chip counters. The primary challenge in such schemes is to mitigate the process and environmental variation effects which can cause wide variations in the ring-oscillator frequencies. Also, the attacker can circumvent the technique by inserting the Trojan in such a way that it does not affect the ring-oscillator frequency, or by compromising the frequency counter circuit.

Both path delay and leakage current can be considered as side-channel parameters to implement a *gate-level characterization* technique [12]. The hardware Trojan detection problem can be formulated as a Linear Programming Problem (LPP) with process variation of each gate being represented as a constant scaling factor for its leakage or delay under various input conditions. This technique is capable of detecting single extra gates in ISCAS-85 benchmark circuits with a high level of confidence. The gate-level characterization technique can be further refined with the effect of thermal variations on the leakage current being used for breaking all correlations. Statistical measures are used to improve the detection accuracy and obtain high degree of separation between golden and Trojan-containing instances of various ISCAS-85 and ISCAS-89 benchmark circuits. The gate-level leakage estimation and characterization problem is shown to be NP-complete in [11]. Using a *consistency* metric based on the expected distribution of the estimation error, an iterative procedure can be used for statistical convergence of the gate-level estimation algorithm to find any extra (Trojan) gates in the circuit.

In order to calibrate inter-die process corner of each IC, a *multiple-parameter* approach [20] can be used. The intrinsic correlation between transient supply current (IDDT) and maximum operating frequency (F_{max}) of a circuit under process variations is used to reduce the effect of process noise and increase the Trojan detection sensitivity. As the attacker will not compromise the critical path delay of the circuit, the measured F_{max} values can be used to calibrate the inter-die process corner of the IC. Any variation in IDDT which does not follow the trend due to process variations can be used to identify presence of a Trojan. It is combined with a judicious test-vector generation approach based on a vector-based functional partitioning of the circuit into smaller regions to decrease the background current, and directed test vectors based on MERO approach to increase switching activity within a probable low-activity Trojan circuit. Figure 15.9a shows the correlation among three parameters IDDQ, IDDT, and F_{max} for ±20% inter-die process variations in an ALU circuit. The random intra-die variations cause a slight spread in the scatter plot to cause deviation from the trend line. This spread, is much reduced compared to the spread in IDDT (Fig. 15.5b) or F_{max} (Fig. 15.7b) when

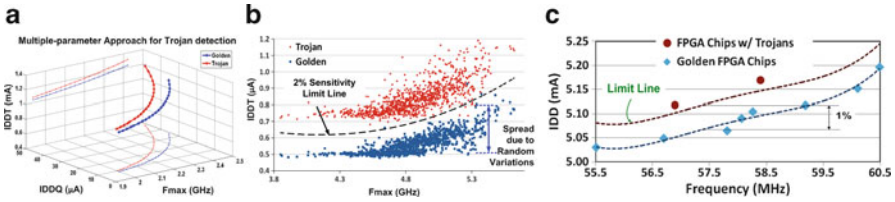


Fig. 15.9 (**a**) Multiple parameters like IDDT, F_{max} and IDDQ can be used to calibrate the process noise and separate the Trojan-infected ICs from the golden ICs. (**b**) In the presence of random intra-die variations, the spread in IDDT values is accounted for by the limit line which sets the lower limit on the Trojan detection sensitivity. (**c**) Similar trend line is obtained with measured IDDT and F_{max} values from ten FPGA chips, two of them containing Trojans [20]

considered in isolation. Hence, this method can be used to increase Trojan detection sensitivity by reducing the value of In_{meas}. Moreover, by proper choice of vectors, one can induce constrained activity within the functional regions, thus reducing I_{orig}, and hence, increasing the sensitivity of the approach. This approach can be used to detect Trojan circuits having an effect on IDDT, larger than the constraint imposed by the limit line, as shown in Fig. 15.9b. Measurement results using a set of Field Programmable Gate Array (FPGA) chips (see Fig. 15.9c) demonstrate the validity of the approach for large benchmark circuits like 32-bit Integer Execution Unit and DLX processor.

In order to further decrease the process noise, a *self-referencing* approach [17] can be used, where the transient current measured for activation of various regions of the circuit can be used with reference to each other. As current measurements from the same IC are compared to calibrate the inter- and intra-die process noise, it leads to further increase in Trojan detection sensitivity. Here, instead of using measured side-channel parameters from golden ICs for comparison, the non-infected regions of the same IC are used to calibrate the process noise and isolate the infected region. By comparing the side-channel parameter for multiple regions with each other, one can isolate the Trojan-infected region. A majority-voting approach is used in case of small Trojans or large regions, where the anomaly does not get reflected in all comparisons, to the same extent. It has been shown that using regions of comparable size gives the best detection probability. This technique can be used hierarchically to delve deeper into a large circuit to increase confidence and to identify the individual gates/transistors which constitute the Trojan in an infected IC. This method can be combined with the region-based vector generation approach proposed in [14] to decrease the background switching activity, the test generation approach in [8] to increase activation probability of rarely activated Trojans and use current measurements from multiple power ports on the same IC as proposed in [26] to increase the sensitivity, enabling ultra-small Trojan detection and scalability of the approach to large sequential circuits.

15.2.4 Run-time Monitoring Approaches

As comprehensive detection coverage of all types and sizes of Trojans during post-Silicon test and validation might be practically infeasible, online monitoring of critical computations can significantly increase the level of trust with respect to hardware Trojan attacks. These run-time monitoring approaches can be used to disable the chip upon detection of malicious logic or to bypass it and allow reliable operation, albeit with some performance overhead, in the presence of unreliable components. One approach for such online monitoring is based on addition of reconfigurable logic, referred to as *DEsign For ENabling SEcurity* (DEFENSE) logic, in a given SoC to enable real-time functionality monitoring [4]. The checks can be performed concurrently with the normal circuit operation and trigger appropriate counter-measures when a deviation from normal functionality is detected. Typically, the reconfigurable core will implement the functionality of the infected logic, which in turn, will be disabled or bypassed.

A combined hardware–software approach to perform runtime execution monitoring has been proposed in [5]. Here, a simple verifiable "hardware guard" module external to the CPU is considered. The work targets primarily Denial of Service (DoS) attacks and privilege escalation attacks, using periodic checks by the operating system (OS) which is enhanced with live check functionality. The scheme can be implemented with only 2.2% average performance overhead based on SPECint 2006 benchmark programs.

In [6], a hybrid hardware/software approach called *BlueChip* is proposed which includes a design-time component as well as run-time monitoring. It tries to identify any unused circuitry with design verification tests, and tags it as suspicious. During run-time, the suspicious circuitry are removed and replaced with exception logic which can trigger a software exception, allowing the system to perform normally by providing a detour around malicious hardware Trojans. This technique is designed to circumvent hardware Trojans, which are similar to software Trojans in purpose. These Trojans are aimed at escalating privilege of any program to superuser mode, granting access to restricted memory space, or initiating DoS attacks, which are similar to allowing execution of malicious code. These Trojan circuits can be inserted in the hardware IP getting mapped to a reconfigurable FPGA or in the instruction code running on an embedded processor.

In terms of multicore processors, a self-scheduling run-time scheme can be implemented [7] whereby functionally equivalent software instances are executed on multiple CPU cores, assisted by dynamic distributed software scheduling. The subtask outputs from different cores are compared to dynamically evaluate their individual trust-levels, with the distributed scheduler undergoing a trust learning procedure over multiple runs. This scheme is capable of successfully completing jobs in a Trojan-infested environment, with improvement in throughput over successive runs. It is similar to the concept of fault-tolerant computing, where faulty or low-reliability cores are bypassed during run-time, without reducing the manufacturing yield.

Table 15.1 Advantages and disadvantages of logic testing and side-channel approaches

	Logic testing approach	Side-channel approach
Pros	(a) Effective for small Trojans	(a) Effective for large Trojans
	(b) Robust under process noise	(b) Test generation is easy
Cons	(a) Test generation is complex	(a) Vulnerable to process noise
	(b) Large Trojan detection challenging	(b) Small Trojan detection challenging

15.2.5 Comparison of Trojan Detection Approaches

Table 15.1 summarizes the relative advantages and disadvantages of logic testing and side-channel approaches for Trojan detection. It is obvious that the two approaches have complementary scope in terms of Trojan detection capability. Hence, approaches that combine the best of both worlds can be the most promising with respect to generic Trojan detection capability. The main advantage of test-time techniques over run-time techniques is that the test-time techniques incur no hardware overhead, while the main disadvantage is the requirement of a "golden" (i.e., Trojan-free) manufactured IC or functional model. Run-time methods typically involve considerable performance and power overhead. However, they provide the last line of defense and are capable of providing 100% confidence in computed results.

15.3 Hardware IP Trust Validation

Hardware IPs from potentially untrusted third-party IP vendors can have malicious alterations and hence need to be verified for trust before use in a system design. The problem can be formulated as follows: Given a third-party IP in register transfer level (RTL) or gate-level or GDS-II format and the functional specification, how can one establish that the IP does exactly as the specification, nothing less nothing more? Note that if a third-party IP is considered as a black-box, the problem of ensuring trust in IP becomes similar to that in an IC, except, in case of IC, one can have a golden reference design, while for third-party it is difficult to have a golden design. While destructive reverse-engineering can be used as a resort to obtain golden IC characteristics; in case of an IP from untrusted vendor, only the specification can be trusted, which is often incomplete. *Sequential equivalence checking* (SEC) turns out to be a natural choice to address this problem. If a complete trusted specification can be obtained, it is possible to derive a high-level golden functional model, which can be used to verify the equivalence with the untrusted IP using formal verification methods like SEC. However, lack of a reliable golden design makes application of such verification approaches infeasible.

Very few investigations have addressed Trojan detection at higher level design descriptions, e.g., RTL IP, also called soft IP. In [27], a structural checking

approach is suggested to verify integrity of third-party IP, but the technique is not easily scalable to large designs [4]. Hardware Trojans could be inserted at higher levels of the design flow including the third-party CAD tools which could also be compromised by a powerful attacker [28]. Such malicious alterations can be detected using formal approaches for obtaining secure Trojan-free designs using untrusted CAD tools. By using simple rules based on good design practices for scheduling, register-binding, fully specified FSMs, etc., one can design the circuit while leaving the attacker less options for effective Trojan insertion.

Instead of modeling and solving the IP trust verification as a formal verification problem, one can model and solve it as test/validation problem. If one has a test solution, they can employ either simulation-based validation or map it to hardware emulators for faster validation. Note that the logic testing techniques which can be applied to activate the Trojan in IC and detect it can be applicable for IPs as well. Hence, for small Trojans, one can use statistical testing approaches to validate the functionality by triggering any undesired malfunction. However, such testing approaches may not be effective for large Trojans, especially for complex sequential Trojans. In addition to statistical testing approaches, one can focus on characterizing specific Trojans' behavior and generate directed tests for them. It would be easier to detect the eventual malicious effects, such as leakage of secret information, corruption of sensitive state, etc. instead of trying to identify the structure or "added function" of the Trojan circuits.

Combination of functional validation with sequential equivalence check can be effective to achieve comprehensive trust verification in IP cores. In [29], a comprehensive approach to verify trust in RTL IP is presented. The approach combines the following steps:

1. Functional vector simulation, to identify suspect signals which correspond to undetected faults, as Trojan inputs are unlikely to be exercised during normal functional testing.
2. N-detect vectors using full-scan ATPG to filter out those nodes which are excitable by multiple vector cubes.
3. Equivalence checking using the remaining nodes which are termed suspect signals. Stuck-at-faults on the suspect signals are activated and observed at the primary outputs. Then, the suspect circuit and specification-derived circuit are compared using FSM unrolling to see if the same behavior is produced when the suspect signal is activated.
4. Finally, the infected region is isolated by using a region-based analysis to identify all nodes affected by the Trojan.

Such a combined methodology is shown to be effective in detecting malicious copies of the untrusted IP and in localizing the region containing the hidden Trojan function, for a set of ISCAS-89 sequential benchmark circuits.

15.4 Summary

The issue of hardware Trojans and effective countermeasures against them have drawn considerable interest in recent times. In this chapter, the major challenges with respect to Trojan detection are analyzed and a comparative study of different Trojan detection techniques reported to date, is presented. The Trojan detection problem can be applicable to both ICs fabricated in untrusted foundry, as well as to hardware IPs obtained from untrusted third-party. The Trojan detection problem is fundamentally different in nature from conventional functional and structural testing, which aim at validating the intended functional behavior of a design. Conventional test generation and application approaches do not address detection of extraneous functionality in a design (imposed by a malicious modification), which may be manifested at the output logic values only under extremely rare conditions or may not affect the output logic at all (e.g., the Trojan instance described in [2]). Hence, detection of Trojan circuits in a design demands novel approaches to sensitize arbitrary Trojan instances in a complex design and observe their effects. Note that destructive approaches through systematic delayering and imaging of ICs can be effective to identify presence of extra circuitry in an IC. However, these approaches are highly expensive, time-consuming, not easily scalable to large designs, and can only be applied to small population of ICs, which are rendered unusable. Considering the fact that an adversary can tamper only select IC instances, destructive approaches cannot provide a general, low-cost solution for pre-deployment trust validation in ICs.

Different non-destructive Trojan detection techniques provide varying capabilities and there is no silver-bullet solution. It is worth noting that side-channel analysis of one or more physical parameters is emerging as a powerful solution for Trojan detection during post-Silicon validation. However, increasing process variations in nanoscale technologies impose a major challenge in isolating Trojan effect in side-channel signature. Hence, significant research efforts have been devoted to developing effective calibration techniques for subsequent elimination of the process noise. Self-referencing approaches [17] can achieve higher detection sensitivity by reducing the effect of process variations (both die-to-die and within-die) due to spatial comparison of current signatures between two regions of the same IC. Furthermore, side-channel analysis approaches require efficient test generation using appropriate Trojan models to increase the signal-to-noise ratio (SNR). This is important particularly for the tiny hard-to-detect Trojans, whose effects in side-channel signature (e.g., supply current) are likely to be barely observable in a multi-million gate design, and hence, can well be below the noise floor of a measurement instrument. In general, the test vector generation routines need to amplify the Trojan effect, while minimizing the background activity. In case of supply current, in addition to judicious vector generation, measurements through multiple supply pins can greatly improve the SNR and hence, detection sensitivity for small Trojans due to the spatially localized effect of Trojan circuits in current signature.

Comprehensive detection of Trojan circuits of arbitrary forms/sizes in a complex multi-million gate design remains an intractable problem. Hence, it is practically infeasible to achieve comprehensive Trojan coverage during post-manufacturing testing. Therefore, a run-time approach, which monitors the critical circuit components for trusted in-field operation at minimal performance/power overhead, can be effective in providing increased level of confidence. These online monitors can detect malicious events in a tampered IC, which evades the testing approaches. Such monitors can also account for unmodeled Trojan circuits or emerging Trojan attacks, e.g., the multi-level Trojan attacks [30], which are mounted at multiple stages of the IC life cycle through nexus among multiple parties.

Considering the varied nature and size of hardware Trojans, it is likely that a combination of detection techniques – both logic testing and side-channel analysis-based techniques – would be required to provide high level of security against Trojan attacks. In case of fabricated ICs, Trojan detection approaches during manufacturing test can be combined with online monitoring of critical computations to increase the assurance level. Finally, the test and online monitoring approaches can be integrated with design for security solutions, further described in the next chapter, to achieve comprehensive protection against Trojan attacks of varying forms and sizes. Design-time approaches can span various levels of design descriptions and can provide both preventive and assistive measures.

Due to increasing usage of third-party hardware IP cores in system-on-chip design, trust verification of these IPs is rapidly emerging as a major concern. Trojan detection in hardware IP cores poses additional problem due to lack of golden or reference model. Therefore, the majority of Trojan detection approaches proposed for fabricated ICs may not work for IP trust verification. As the functional specification of an IP is the only thing that can be trusted, an IP can be verified through directed functional test vectors that can excite malicious changes in critical circuit parts. Another possibility is to apply sequential verification with a high-level model of an IP generated from its microarchitecture or high-level specifications.

Clearly, the area of Trojan detection involves several open challenges and significant research effort is needed to address these challenges to enable trustworthy computing in presence of Trojan attacks. Maximizing the level of confidence during post-Silicon validation would remain a major challenge. Effective combination of DfS measures with test and validation would be an important subject of future investigation. Other future investigations would include: reliable detection of analog Trojans, which can exploit numerous activation and observation conditions; an effective metric to quantify the level of trust that combines both design and test time approaches; approaches for validating trust of an IC or IP core in the absence of a golden model; and finally, an evaluation platform that analyzes a design to identify vulnerable regions for Trojan attacks and the impact of a design change on the level of achievable trust.

References

1. DARPA (2007) TRUST in Integrated Circuits (TIC). http://www.darpa.mil/MTO/solicitations/baa07--24. Accessed 15 Sept. 2008
2. Lin L et al. (2009) Trojan side-channels: lightweight hardware Trojans through side-channel engineering. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systems
3. Chipworks, Inc., Semiconductor Manufacturing – Reverse Engineering of Semiconductor components, parts and process. http://www.chipworks.com. Accessed 20 July 2011
4. Abramovici M, Bradley P (2009) Integrated circuit security – New threats and solutions. In: Proceedings of Workshop on Cyber Security and Information Intelligence Research, pp 1–3
5. Bloom G, Narahari B, Simha R (2009) OS support for detecting Trojan circuit attacks. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust
6. Hicks M, Finnicum M, King ST, Martin MMK, Smith JM (2010) Overcoming an untrusted computing base: detecting and removing malicious hardware automatically. In: Proceedings of the IEEE Symposium on Security and Privacy
7. McIntyre D, Wolff F, Papachristou C, Bhunia S (2009) Dynamic evaluation of hardware trust. In: Proceedings of the IEEE InternationalWorkshop on Hardware-Oriented Security and Trust
8. Chakraborty RS et al. (2009) MERO: a statistical approach for hardware Trojan detection. In: Proc Workshop on Cryptographic Hardware and Embedded Systems
9. Jha S, Jha SK (2008) Randomization based probabilistic approach to detect Trojan circuits. In: Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium, pp 117–124
10. Aarestad J, Acharyya D, Rad R, Plusquellic J (2010) Detecting Trojans though leakage current analysis using multiple supply pad IDDQs. In: Proceedings of the IEEE Transactional Information Forensics and Security
11. Alkabani Y, Koushanfar F (2009) Consistency-based characterization for IC Trojan detection. In: Proceedings of the International Conference on Computer-Aided Design
12. Potkonjak M, Nahapetian A, Nelson M, Massey T (2009) Hardware Trojan horse detection using gate-level characterization. In: Proceedings of the Design Automation Conference
13. Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B (2007) Trojan detection using IC fingerprinting. In: Proceedings of the Symposium on Security and Privacy pp 296–310
14. Banga M, Hsiao M (2008) A region based approach for the identification of hardware Trojans. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust
15. Banga M, Hsiao M (2009) A novel sustained vector technique for the detection of hardware Trojans. In: Proceedings of the International Conference on VLSI Design
16. Salmani H, Tehranipoor M, Plusquellic J (2010) A layout-aware approach for improving localized switching to detect hardware Trojans in Integrated Circuits. In: Proceedings of the IEEE International Test Conference
17. Du D, Narasimhan S, Chakraborty RS, Bhunia S (2010) Self-referencing: a scalable side-channel approach for hardware Trojan detection. In: Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems
18. Jin Y, Makris Y (2008) Hardware Trojan detection using path delay fingerprint. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust
19. Li J, Lach J (2008) At-speed delay characterization for IC authentication and Trojan horse detection. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 8–14
20. Narasimhan S et al. (2010) Multiple-parameter side-channel analysis: a non-invasive hardware Trojan detection approach. In: Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust
21. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Design Test Comput 27(1): 10–25

22. Borkar S et al. (2003) Parameter variations and impact on circuits and microarchitecture. In: Proceedings of the Design Automation Conference, pp 338–342
23. Pomeranz I, Reddy SM (2004) A measure of quality for n-detection test sets. IEEE Trans Comput 53(11): 1497–1503
24. Chakraborty RS, Bhunia S (2009) Security against hardware Trojan through a novel application of design obfuscation. In: Proceedings of the International Conference on Computer-Aided Design
25. Chakraborty RS, Paul S, Bhunia S (2008) On-demand transparency for improving hardware Trojan detectability. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 48–50
26. Rad R, Plusquellic J, Tehranipoor M (2010) A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions. IEEE Transaction on Very Large Scale Integration Systems
27. Smith S, Di J (2007) Detecting malicious logic through structural checking. In: Proceedings of IEEE Region 5 Technical Conference
28. Potkonjak M (2010) Synthesis of trustable ICs using untrusted CAD tools. In: Proceedings of Design Automation Conference
29. Banga M, Hsiao M (2010) Trusted RTL: Trojan detection methodology in pre-silicon designs. In: Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust
30. Ali S, Mukhopadhyay D, Chakraborty RS, Bhunia S (2011) Multi-level attack: an emerging threat model for cryptographic hardware. In: Proc Design Automation and Test in Europe

Chapter 16
Design for Hardware Trust

Yier Jin, Eric Love, and Yiorgos Makris

16.1 Overview

Toward further enhancing the effectiveness of postfabrication hardware Trojan detection solutions and alleviating their limitations, as discussed in previous chapters, several methods which rely on modifying the current IC design flow have been developed by the hardware security and trust community. Collectively termed *design for hardware trust* [1], these Trojan prevention methods aim to prevent insertion and facilitate simple detection of hardware Trojans. In contrast to Trojan detection methods which passively test chips anticipating that the inserted Trojans will be identified based on their abnormal behavior, Trojan prevention methods take a proactive step by changing the circuit structure itself in order to prevent the insertion of Trojans. In order to achieve this goal, the entire IC supply chain needs to be revisited. The resulting modified IC supply chain emphasizes design security to counter Trojan threats and provide a solution for trusted IC design.

Among the proposed Trojan prevention methods, the majority aim to complement side-channel-based Trojan detection methods, by inserting non-functional circuitry during the chip design stage, to facilitate the construction of side-channel signatures. Hereafter, we refer to these methods as side-channel fingerprint-based hardware prevention methods. Some of them can only play an auxiliary role to Trojan detection methods and help in measuring internal side-channel information which is otherwise untestable using off-chip testing equipment. Others go further to not only measure internal side-channel information but also to compare the measured information with predefined threshold values using a special-purpose module which is embedded on-chip and which is responsible for deciding whether abnormality exists within the side-channel signals. Design overhead is the major concern of side-channel fingerprint-based Trojan prevention methods because

Y. Jin · E. Love · Y. Makris (✉)
Department of Electrical Engineering, Yale University, New Haven, CT 06520, USA
e-mail: yier.jin@yale.edu; eric.love@yale.edu; yiorgos.makris@yale.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_16,

side-channel measurement and analysis circuits can be rather complex and consume large on-chip area. Interestingly, for this reason, Trojan prevention methods of this kind typically select path delay as the basis for constructing a side-channel fingerprint of a chip, because delay measurement modules are light-weight in terms of overhead.

Other methods rely on the assumption that attackers will only use rare events to trigger the inserted Trojans. Such methods attempt to enhance traditional functional/structural testing methods by increasing the probability of fully activating inserted Trojans during the test stage. Among them, design obfuscation [2] conceals the circuit so that attackers cannot calculate the probabilities of real events. Alternatively, dummy scan flip-flops [3] and the inverted voltage scheme [4] aim to balance internal signal transition frequencies in order to remove rare events. Along a different direction and in order to avoid relying on such assumptions, a Design for Trojan Test (DFTT) methodology [5] is proposed as a general design hardening scheme.

Lastly, methods that target protection of Intellectual Property (IP) have recently started to appear in the literature. Realizing the importance of protecting third party IP, which constitute a significant portion of the design in most contemporary systems on a chip, the authors in [6, 7] proposed the concept of proof-carrying hardware (PCH), which is based on a well-developed formal software protection methodology, namely proof-carrying code (PCC).

16.2 Delay-Based Methods

16.2.1 Shadow Registers

Shadow registers constitute one of the path delay-based Trojan prevention methods which was first proposed in [8] and then carefully evaluated in [9]. Trojan detection based on path-delay fingerprints was first proposed in [10] where the authors showed that with the help of statistical data analysis, this method could effectively detect hardware Trojans as small as 0.2% of the total on-chip area, even when considering $\pm 7.5\%$ process variation. An obstacle preventing wide usage of this path delay-based Trojan detection method is the difficulty of measuring/observing delays of internal paths (i.e. from register to register but not connected to primary input/output). Without internal path delay information, it is almost impossible to construct the full path-delay fingerprint to which chips-under-test can then be compared. Shadow-register provides a possible solution to this problem by enabling a mechanism for measuring internal path delay.

Figure 16.1 shows the basic architecture of the shadow register Trojan prevention scheme [8]. From this architecture, it can be seen that the basic unit contains one *shadow register* one *comparator* and one *result register*. The *shadow register* is located in parallel with the *destination register* at which a path ends. Different from the *system clock*, shadow registers are controlled by a negative-skewed clock signal

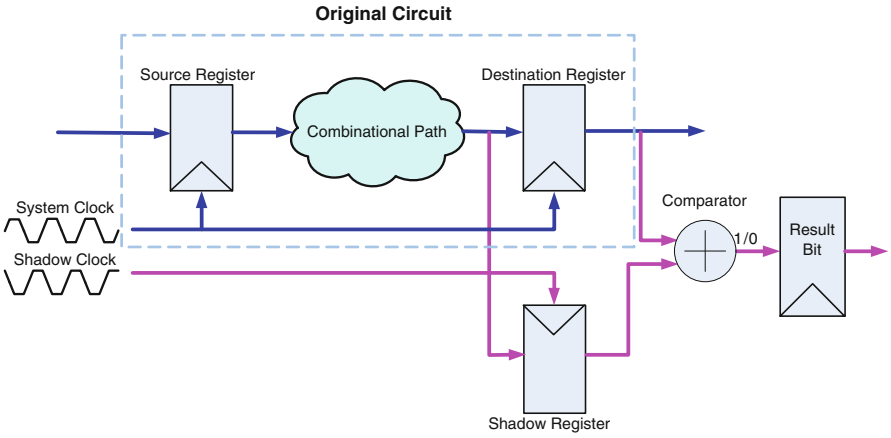


Fig. 16.1 Trojan prevention architecture with shadow registers [8]

the *shadow clock*. The frequency and phase of the *system clock* are kept constant in order to maintain the functionality of the original circuit even when in delay testing mode. The *shadow clock* has the same frequency as the *system clock* but its phase is adjustable. In order to perform internal path delay measurement, an off-chip clock signal generator or an on-chip Digital Clock Manager (DCM) is required. The precision of the signal generator or DCM decides the accuracy of the measured path delays. At the beginning of each internal path delay test, the phase of the *shadow clock* is reset to the same as that of the *system clock* so that the value in the *shadow register* is the same as the *destination register* and the output of the comparator is "0." Then the *shadow clock* is negatively skewed step by step according to the precision of the signal generator or DCM. The adjustment continues until the output of the comparator is "1" indicating that the value in the *shadow register* is now different from the value in the *destination register*. The output "1" is then stored in the *result register* and finally read out through a scan chain (not shown in the figure).

The shadow register architecture significantly improves the observability inside a chip to make it easier to construct path delay fingerprints both for genuine chips and for chips-under-test. Although the authors in [9] demonstrated effectiveness of this Trojan prevention method on a sample target circuit, i.e., an 8x8 Braun multiplier, this prevention scheme suffers from several inherent limitations. These limitations and possible solutions are discussed next.

1. The phase adjustment step size of the signal generator or DCM is critical to the accuracy of the measured delay and also related to the effectiveness of this method. A high-resolution on-chip DCM is preferable but it would consume large area and power.
2. The existence of process variation and measurement noise can easily deteriorate testing results. A popular way of solving this problem is to leverage statistical data analysis methods to deal with the measured delay information.

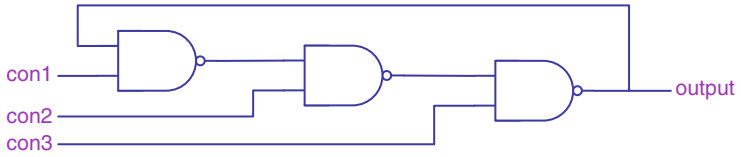


Fig. 16.2 Ring oscillator constructed by three NAND gates

3. As the scale of the original circuit increases, the total number of paths to be measured also increases. More test vectors should be applied to the design in order to cover a large number of paths and improve the testing coverage rate, but at higher testing cost. The continued increase of circuit size and complexity also means that more shadow registers are required to be inserted in the original design. These shadow registers are supported by the *shadow clock*. From a layout point of view, a design with two clock trees would cost large on-chip area and the chip performance may deteriorate due to inefficiencies in clock signal distribution.
4. The question of reading out the information in result registers triggers another concern regarding this Trojan prevention method. Embedding result registers into the original scan chain to reuse the scan chain controller is an efficient way to reduce the area of the prevention scheme, but it forces the chip into scan mode frequently by stopping the normal operation.

16.2.2 Ring Oscillators

In order to lower the testing cost raised by shadow registers but still leverage path delays to prevent the insertion of hardware Trojans, some researchers tried not to measure the existing path delays but insert new paths and measure the delay of these paths instead. For the following reasons, ring oscillators are among the most popular choices to construct the extra internal paths:

1. *Small area*: the area of ring oscillators is much smaller than that of other Trojan prevention architectures, so the total overhead is lower.
2. *Easy insertion*: As the architecture of the ring oscillator is very simple, it is easy for designers to insert ring oscillators inside circuit designs with low impact to the original design. For the same reason, it is easy to predict the behavior of the inserted ring oscillators even when considering large process variation.

Figure 16.2 shows a sample ring oscillator constructed with three NAND gates. Using NAND gates instead of inverters can improve the controllability of the ring oscillator. For this example, there are three control signals, *con1*, *con2*, and *con3*, to control oscillation. Only if all three signals are of high voltage could the ring oscillator start oscillating. Under normal operation, all the inserted ring

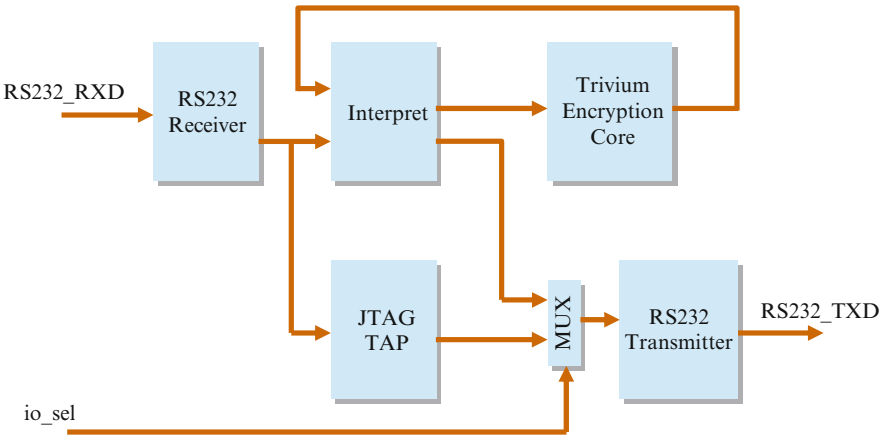


Fig. 16.3 Trivium-based encryption platform

oscillators will be muted to avoid power consumption; while in testing mode, switching the value of controlling signals allows the testers to turn on the selected ring oscillators and then read out the oscillation frequency. In addition to ensuring low power consumption, the added control signals can also prevent attackers from understanding the details of the inserted ring oscillators.

The main idea behind ring oscillator-based Trojan prevention methods is that any malicious modifications to the original design would also change parameters of pre-inserted ring oscillators. These changes include lower power supply voltage, higher input current, longer delay between NAND gates due to rewiring, etc. One question raised by this method is how many ring oscillators are needed and where they should be located inside the chip. The author in [11] uses an example to answer both questions where the target system contains a Trivium encryption core, an interpret module, an RS232 transceiver and a JTAG TAP, as shown in Fig. 16.3.

Three ring oscillators are inserted into the encryption platform: one in the RS232 Transceiver, one in the interpret module and a third in the JTAG TAP. The location selection covers both the datapath and the control logic and emphasizes the security weakness of input/output modules in the target platform. The expectation is that by carefully selecting insertion locations, one can lower the number of ring oscillators needed and increase the efficiency of the Trojan prevention scheme. However, this kind of location selection is solely based on designers' experience and their understanding of the target circuit, and could have limitations that attackers may use to circumvent the entire protection scheme. For example, the insertion of ring oscillators in the RS232 transceiver and JTAG TAP may be able to detect any modifications trying to leak information through the RS232 channel and prevent any malicious change in the control logic of scan chain. Yet malicious addition of a simple shortcut between the plaintext input and the RS232 output channel can easily evade this prevention method and leak the plaintext, as the latter is not protected by any of the three ring oscillators [12].

Another Trojan prevention scheme using ring oscillators, different from the method mentioned earlier, is to construct ring oscillators from gates of the original design by inserting multiplexors (MUXs), NAND gates, and inverters. Rather than inserting ring oscillators into the design in order to detect additional malicious circuitry in an indirect way, this method highly improves the sensitivity of the constructed ring oscillators because any modifications will directly change the internal architecture and result in a significant frequency change for the constructed ring oscillators. At worst, the insertion of malicious circuitry may mute the ring oscillators. The designers can also adjust the coverage rate (the percentage of the on-chip area which is part of the constructed ring oscillators) to control area overhead of the proposed method. To fully present the tradeoff between hardware security level and area overhead, a 4-bit carry look-ahead adder is chosen as the sample circuit in the 2010 CSAW Embedded System Challenge hosted by the Polytechnic University of NYU [13]. Contest organizers proposed three circuit hardening levels by constructing two, four, and six ring oscillators inside the adder. Among them, the low protection level (two ring oscillators) with two MUXs and two inverters covers 62% of the original design's gates (16 of 26 gates). The medium protection level (four ring oscillators) with four MUXs and four inverters covers 85% of the original design's gates (22 of 26 gates). The hard protection level (six ring oscillators) with six MUXs and six inverters covers 92% of the original design's gates (24 over 26 gates). Figure 16.4 shows the gate level architecture of the 4-bit carry look-ahead circuit and one sample ring oscillator constructed by an additional Inverter, an additional MUX and three gates from the original design (an XOR, an AND, and an OR gate). When the ring oscillator control signal *RO* is "0," the circuit performs its normal functionality. When it is set to "1," however, the ring oscillator starts oscillating. Testers can then measure the frequency of the constructed ring oscillators to decide whether the chip is genuine or not.

It is easy to construct internal ring oscillators in small-scale circuits such as the one shown in Fig. 16.4, but for more complex circuits, designers will have to rely on algorithms to automate the insertion process, hence such automation tools should also be developed. Research in this domain is still in progress and more efficient algorithms may be proposed in the future.

Although both ring oscillator insertion and ring oscillator construction are lightweight in terms of overhead, one concern with both methods is the ability to read out the frequency of these internal ring oscillators. Adding pins for each ring oscillator could be a straightforward solution, which can leverage the high precision of external test equipment to measure frequency. Considering the large number of ring oscillators, however, it is not an economic way to use or reuse valuable pin resources. In contrast, most current solutions use on-chip frequency measuring modules to accomplish this task. Counters are among the most popular frequency measuring modules. Figure 16.5 shows a sample measuring module which includes two separate counters, *Clock_Counter* and *RO_Counter* [14]. The global *RST* signal is used to reset both counters. The *Clock_Counter* has its clock pin connected to the system clock and counts the cycles of the system clock while the *RO_Counter* is connected to the output of the ring oscillator. The output of

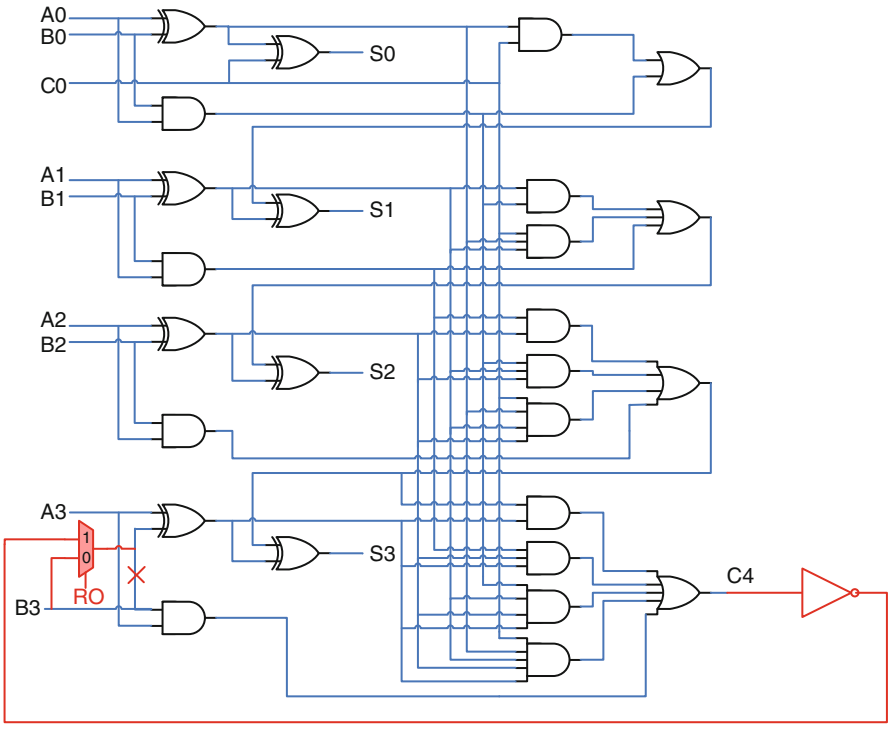


Fig. 16.4 Gate level circuit of 4-bit carry look-ahead adder and an inserted ring oscillator

Clock_Counter is compared to a predefined threshold value N to decide the ON/OFF state of *RO_Counter*. At the testing stage, the ring oscillator is enabled. Both counters start to count, but at different frequencies; one based on the system clock frequency and the other based on the frequency of the connected ring oscillator. When the output of *Clock_Counter* is equal to N, the *RO_Counter* stops counting and its output over N is a measure of a relative frequency to the system clock. Though quite compact, the added frequency measuring modules still increase the overhead of this Trojan prevention method.

All previously introduced hardware Trojan prevention methods try to prevent Trojan insertion by measuring internal path delays. None of them, however, pays attention to the security of the inserted testing circuit itself. As a result, the overall effectiveness of these methods may be lower than expected. The limitations of unsecure protection circuits have already been pointed out by various researchers, who have demonstrated the weakness of these methods by presenting successful hardware Trojan attacks that these prevention methods are unable to detect [11, 12, 14–17]. In fact, the focus of the embedded systems challenge at CSAW 2010 [13] was to find possible security limitations of the ring-oscillator Trojan prevention method. Reports from this competition conclude that currently proposed

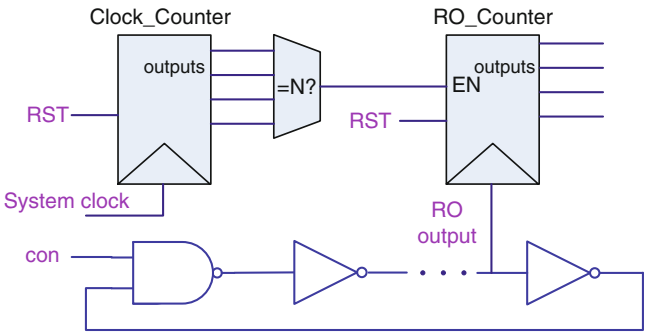


Fig. 16.5 Frequency measuring module for ring oscillator [14]

hardware prevention methods are far less secure than initially believed. More than 200 Trojan designs were submitted to this competition, and many of them succeeded by tampering with the Trojan prevention method.

16.3 Rare Event Removal

Besides the side-channel fingerprint-based Trojan prevention method, other researchers are trying to develop functional/structural Trojan prevention methods. In [18], the author conjectures that attackers will only choose rarely occurring events as triggers and proposes the idea of *Trojan vectors* which can trigger low-frequency events to enhance the detectability of traditional structural testing. Here the basis for launching Trojan attacks is the event probability because attackers will choose low-frequency events to trigger the inserted Trojans. Consequently, if a design hardening scheme increases the difficulty of calculating the event frequency and/or makes rare events scarce, attackers will have to randomly pick trigger events and the probability of the inserted Trojans being activated during the testing phase will increase, thereby deterring attackers from inserting Trojans inside the design. Design obfuscation [2], dummy scan flip-flops [3], and the inverted voltage scheme [4] are three representatives of functional/structural Trojan prevention methods.

Design obfuscation, by definition, means that a design will be transformed to another one which is functionally equivalent to the original, but in which it is much harder for attackers to obtain complete understanding of the internal logic, making reverse engineering much more difficult to perform. In [2, 15], the authors propose a Trojan prevention method that obfuscates the state transition function to add an *obfuscated mode* on top of the original functionality (called *normal mode*). Figure 16.6 shows the obfuscated functionality and normal functionality after the state transition function of the original design is obfuscated. As shown in the figure, the transition arc $K3$ is the only way the design can enter normal operation mode from the obfuscated mode. Thus, only one input pattern is able to guide the circuit

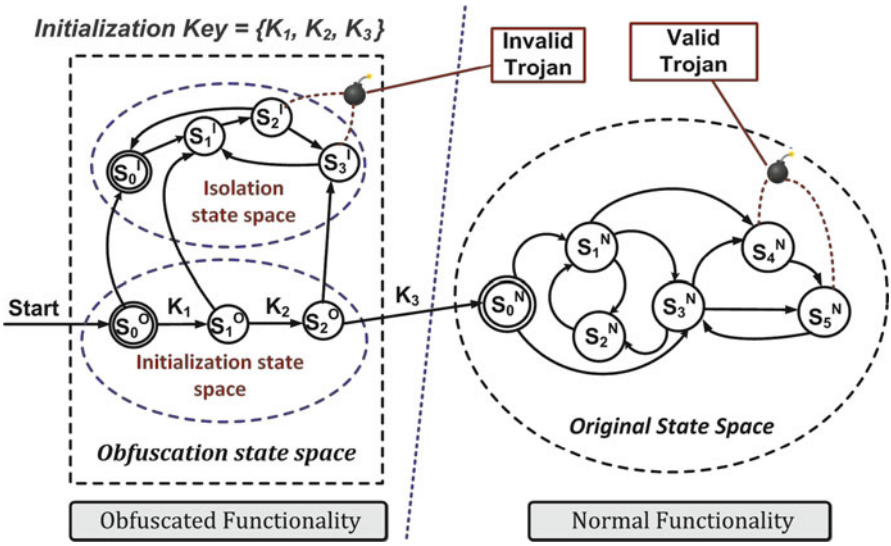


Fig. 16.6 Trojan prevention by design obfuscation [15]

into its normal mode. This special input pattern is called the *initialization key sequence*. Without knowing this key sequence, attackers are unlikely to get into normal mode by randomly picking input patterns, and the event probabilities derived from simulations cannot reflect their real probabilities if only running in normal operation (in order to prevent attackers from finding the initialization key sequence, the size of the obfuscation state space is designed to be very large). After design obfuscation, any inserted hardware Trojans can be divided into two groups, one with all (or part) of the trigger states from obfuscation mode and the other with all trigger states from normal mode. For the former group of Trojans, because the state space in obfuscated mode is unreachable in normal operation, these Trojans will never be triggered at all. For the later group of Trojans, while they are valid, the real event probabilities are likely different from what the attackers expect based on simulation, so it is not necessarily true that these events indeed occur rarely. These two groups of Trojans are labeled as *invalid Trojans* and *valid Trojans* in Fig. 16.6.

The design obfuscation method is easy to implement because it can leverage commercial EDA tools. An algorithm is proposed to automate the whole obfuscation process [2]. However, in many cases, the underlying assumptions of this hardware Trojan prevention method based on design obfuscation may not hold entirely true. One assumption here is that attackers will only use rare events to trigger the Trojan. This assumption omits the always-on hardware Trojans, and the definition of "rare" is rather arbitrary. Note that the other two Trojan prevention methods which we introduce later, namely dummy flip-flop insertion and inverted voltage, also suffer from this problem. Another assumption is that attackers have no knowledge of the obfuscation mechanism but only try to analyze the obfuscated version of the code to

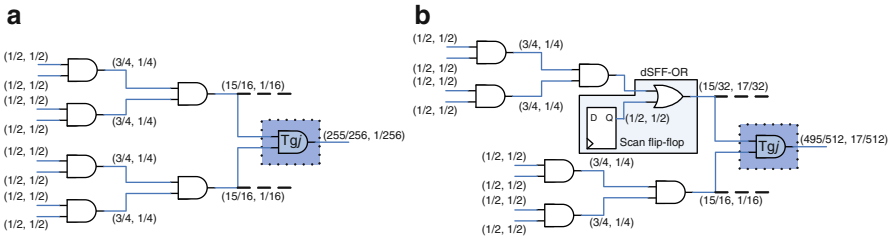


Fig. 16.7 Transition probability analysis on original circuit (**a**) and with dummy flip-flop (**b**) [3]

find rare events. This method assumes that attackers do not choose triggers from the space between exhaustive input patterns and ATPG patterns (plus rare events), even though this space is very large. If any of these assumptions are not valid in reality, the effectiveness of this entire Trojan prevention method will be diminished.

In [3], the authors first model the internal net transition probability using geometric distribution and calculate the value based on the number of clock cycles needed to generate a transition on a net. A Trojan prevention (or more accurately, a Trojan activation) methodology is then proposed which can increase the probability of generating a transition in functional Trojan circuits, if any exist, and analyze the transition generation time. In order to increase the activity of nets with low transition probability, extra flip-flops (called *dummy flip-flops*) are inserted into the original design. These dummy flip-flops are inserted in a way that does not change the design's original functionality. Figure 16.7 shows the procedure to calculate the net transition probability with and without dummy flip-flops. In Fig. 16.7a, the sample circuit contains seven AND gates located in three levels with eight inputs and one output. If it is assumed that, for each input, the probabilities of that input being "1" and "0" are equal, then due to the characteristics of AND gates, the probability of a "0" output is 255/256 while the probability of a "1" output is only 1/256, which is rather low. In Fig. 16.7b, a dummy flip-flop and an OR gate are inserted in one branch of the final AND gate and the probability of a "1" output is increased to 17/512, 8.5 times that of the original circuit.

This dummy flip-flop-based Trojan prevention method can help Trojan detection and Trojan prevention in two ways:

1. *Power-based side-channel analysis*: Due to the insertion of dummy flip-flops and related AND/OR gates, the activity of Trojans under ATPG-generated testing patterns will increase to consume more power during the testing stage. The increased ratio of Trojan power consumption to total power consumption will lead to an easy detection of inserted Trojans and will finally prevent attackers from inserting Trojans even if they know the design is protected by the dummy flip-flop protection scheme.
2. *Functional testing*: The inserted dummy flip-flops can balance the transition probabilities of internal nets so that the probability that the inserted Trojan is fully activated increases. The rare occurrence assumption, which attackers rely on to evade traditional functional testing, can be invalidated and erroneous responses may be observed at a primary output.

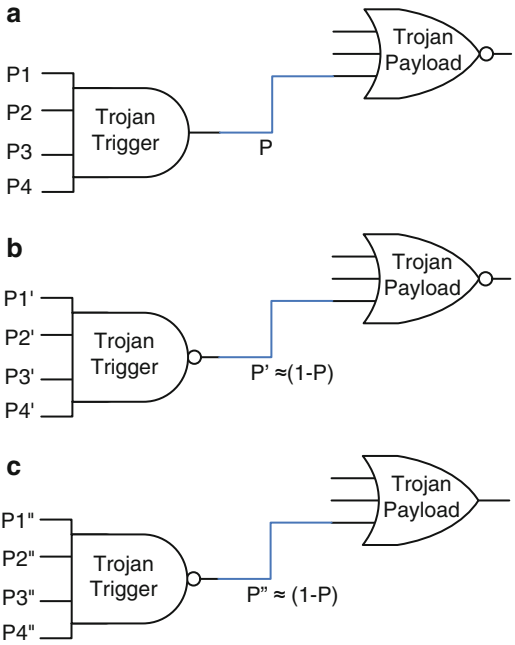


Fig. 16.8 Trojan circuit under normal voltage supply (**a**), only Trojan trigger is affected by inverted voltage supply (**b**), both Trojan trigger and payload are affected by inverted voltage supply (**c**) [4]

In [4], the authors proposed a supply voltage inversion method to magnify Trojan activity without inserting extra gates. The key idea here is that reversing a gate's power supply and ground changes the function of the gate to make low probability outputs occur more frequently. Figure 16.8 shows how this method can help to detect the inserted Trojan by switching the *majority value* and *minority value* of any internal gate [4]. The trigger of the sample Trojan is a four-input AND gate and $P1, P2, P3, P4$ are the probabilities of the four inputs being set to the non-controlling value 1, respectively. So the probability that output equals to its *minority value* 1 is approximately $P = P1 \times P2 \times P3 \times P4$. If the supply voltage to the Trojan trigger is inverted while the supply of the Trojan payload remains the same, the Trojan structure will be changed to that of Fig. 16.8b where the AND gate is converted into a NAND gate with the *majority value* being 1. Now the probability of triggering the Trojan is $P' \approx (1 - P)$. Figure 16.8c shows the case when the supply voltage of both the Trojan trigger and payload are inverted and the payload gate is converted to OR gate but the Trojan activation probability does not change.

One problem of supply voltage inversion in CMOS logic is that the degraded gate potential of one stage of CMOS gates degrades the gate potential on the next stage and the signal might stop propagating after a few stages. To avoid this degradation problem, the authors in [4] modified this method to apply inverted voltage to alternate stages in the original circuit.

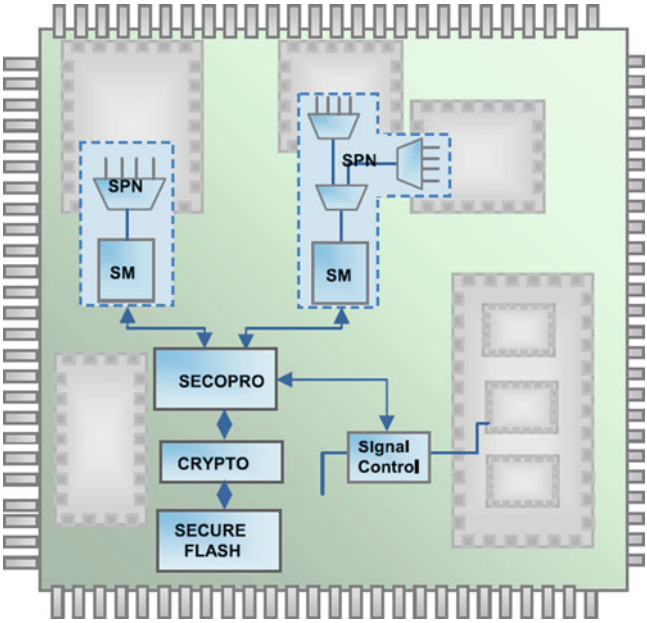


Fig. 16.9 SoC design with DEFENSE logic [19]

In [19], the authors proposed infrastructure logic to perform online security checks during normal operation, focusing on the System-on-Chip (SoC) domain. A reconfigurable logic called design-for-enabling-security (DEFENSE) is added to the SoC platform to implement real-time abnormality monitoring. Figure 16.9 shows the architecture of a hardened SoC chip with DEFENSE. The basic module of DEFENSE logic is the Signal Probe Networks (SPN) – Security Monitor (SM) pair. Here a signal probe network is a distributed pipelined MUX network configured to select a subset of user defined important signals and transport them to Security Monitors, programmable transaction engines configured to implement an FSM to check user-specified behavioral properties of the signals from SPNs. The Security and Control Processor (SECOPRO) reconfigures SPNs to dynamically reselect monitoring signals. All the configurations are encrypted and stored in secure flash memory so that their function is not visible to reverse engineering.

When signal abnormality is detected, the signal control module enables SECOPRO to override the value of any suspicious signals and restore the whole system back to normal operation. The core exhibiting illegal behavior may also be isolated. While effective, the overhead of this method remains is a concern because high coverage of on-chip signals requires a large number of nets to be monitored. Fairly sizeable on-chip area will also be occupied by the DEFENSE logic.

16.4 Design for Trojan Test

The authors in [5] developed a general hardening scheme which follows the paradigm of the widely accepted structural fault testing methodology, Design for Test (DFT). As the target of this method is to prevent attackers from inserting hardware Trojans into circuit designs, it has been termed DFTT. Despite the naming similarity, the DFTT methodology has some key differences from DFT. For DFT methods, test vectors are generated based on the assumption that the CUT (circuit-under-test) is genuine, with no inserted malicious circuits, because the purpose of DFT is to detect manufacturing faults. However, for DFTT, the goal is to generate test vectors with the objective of detecting maliciously inserted circuits.

The central idea of DFTT methods is that any effective hardware Trojans must impose a specific structure on the infected circuit, which the attacker leverages to leak internal information. While this structure is not known to circuit designers, scrutiny of all active logic on-chip with the help of local probing cells may be sufficient to reveal its existence and, thereby, expose the hardware Trojan. This method is robust because even if attackers have a complete picture of how this scrutiny works, it is still difficult to evade. Performing DFTT on the original design is known as *hardening* a design. Figure 16.10 shows the three basic steps of DFTT, which are explained later.

16.4.1 Step I: Code Evaluation

DFTT coding rules are first developed through which the original Hardware Description Language (HDL) code is converted to DFTT-compliant code.

16.4.2 Step II: Sensitive Path Selection

An assumption is made here that the attacker's purpose is to insert an additional structure in the original design to reliably steal internal sensitive information. For this purpose they would attempt to evaluate the relative merit of internal signals (such as the encryption key in a cryptographic chip) before inserting Trojans. Hence, the DFTT tool (developed in parallel with the DFTT methodology to automate the whole DFTT procedure) isolates paths in which sensitive signals, or other signals which are auxiliary to sensitive signals, flow from primary inputs to primary outputs.

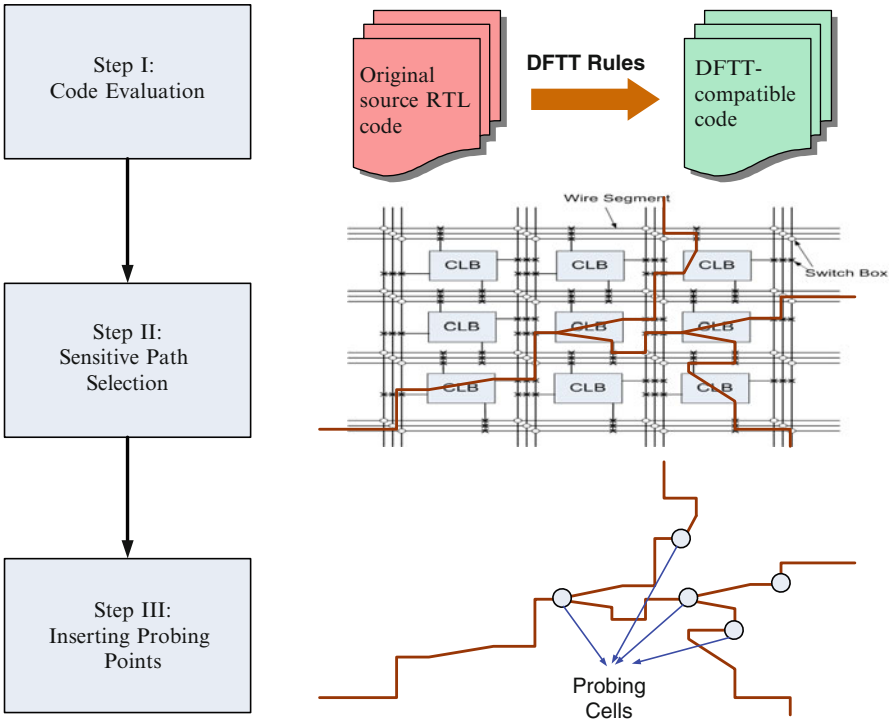


Fig. 16.10 The basic procedure of hardening a design via the DFTT methodology [5]

16.4.3 Step III: Inserting Probing Points

Based on the sensitive paths chosen in Step II, probe cells are inserted into the DFTT-compliant code. This step is similar to the insertion of scan flip-flops (SFFs) when performing DFT, but the probe cell is slightly different from a normal SFF because of the emphasis on two key characteristics: *genuineness* and *integrity* [5].

After the design is hardened using the DFTT methodology, the subsequent testing process is similar to that used in DFT. DFT-style trigger-response pairs generated by DFTT tools are loaded into the CUT. Assuming that there are no manufacturing faults within the design, any mismatch between the CUT's response sequence and the genuine response sequence reveals that internal logic has been modified. Reverse engineering or other related testing methods can then be performed to further analyze the suspicious chips.

16.5 Proof-Carrying Hardware

In [20], the authors raised a limitation of existing Trojan detection methods, namely the fact that they all try to scrutinize ICs for the presence of Trojans in the post-silicon stage. Far less is known or has been researched regarding this problem in pre-silicon stages. Unfortunately, the same problem exists in the design for hardware trust domain. All the previously introduced Trojan prevention methodologies are based on the assumption that any inserted Trojans will be detected relatively easily in the post-silicon stage if it is simple for designers to measure side-channel signals or hard for attackers to find true probability of internal events. None of these methods have tried to protect third-party hardware IP from hardware Trojan threats.

Recently, research on how to combat the threats of hardware Trojan on third party IP has emerged. Pertinent efforts attempt to exploit a well-developed body of work in the software domain, known as PCC. Originally developed by Necula et al. [21] in 1996, PCC provided a new way of determining whether code from a potentially untrusted source is safe to execute. This was accomplished by establishing a formal, automatically verifiable proof that the questionable code obeys a set of formalized properties. Such a proof may demonstrate, for example, that a given set of machine instructions follows the predefined type-safety rules of a particular programming language. The proof is then combined with the code, which allows the recipient to automatically check the code against the proof. Only if the check process finishes with no errors can the recipient know that the code is safe to run.

The idea of expanding this methodology to the hardware trust domain first appeared in [6]. The authors made a case for the necessity of PCH based on the increasing prominence of FPGAs and reconfigurable devices; if hardware itself becomes just as instantaneously reprogrammable as software, then the capability to establish the trustworthiness of unknown circuitry is inherently desirable. The authors further elucidate the novelty of their approach by contrasting it with other, more common security practices such as formal verification or model-checking. They argued that PCH, in contrast to these, is unique in that it requires very little of end users, who need only perform a straightforward validation check of the proof. The burden of demonstrating security instead falls on the producers, who must construct the proof when they provide the original IP core. This new approach was also differentiated from simple checksum-hashing of FPGA bitstreams because the latter does not take into account the actual functionality of the code being transmitted.

As a first step toward provable hardware security, the authors proposed a technique for presenting proofs of combinational equivalence for digital logic functions implemented in FPGA bitstreams. This approach required an agreed-upon specification function $S(x)$ for each logic function and an implementation $I(x)$ extracted from the FPGA netlist. From these two inputs, a proof would automatically be generated to show that the implementation $I(x)$ was combinationally equivalent to the specification $S(x)$. The consumer could then check this proof against $I(x)$ and $S(x)$ to quickly see that the implemented function agrees with its specification.

The specific technologies employed to achieve this result include standard FPGA CAD tools, a satisfiability (SAT) solver, and a SAT trace checker. From the netlist output of the CAD synthesis tools, each logic function must be proven against its specification. For each such function, then, the authors propose to create a structure called a "miter" which is formed by taking the XOR of $S(x)$ and $I(x)$ as $M(S(x), I(x))$. As the output of the miter can only be *true* if there exists some boolean vector x for which $S(x) \neq I(x)$, a proof that $M(S(x), I(x))$ is unsatisfiable therefore demonstrates that $S(x)$ and $I(x)$ are equivalent.

When the SAT solver finds that $M(S(x), I(x))$ is indeed unsatisfiable, a trace is output to show how this unsatisfiability result was achieved. It is precisely this trace that constitutes the correctness proof under the PCH system. When traces for all relevant functions have been extracted, they are sent along with the bitstream to the consumer. The consumer may then regenerate a miter for each logic function from its netlist implementation and check this against the corresponding proof trace. If all functions check against their traces, then the hardware is accepted as safe.

This work represented a first step toward proof-based security. It is clear that many types of Trojans could be prevented under such a system because modifications to the combinational behavior of an FPGA bitstream's logic functions would be immediately detectable. Nevertheless, the expressiveness of this approach is limited by the need to specify exact Boolean functionality. In software PCC, security policies have generally specified a broader definition of "safe" behavior without necessarily stipulating precisely what a program must compute.

In response to this limitation, other researchers have sought ways of expanding PCH to encompass a more abstract notion of security-related properties. The authors in [7] present Proof-Carrying Hardware Intellectual Property (PCHIP) to guarantee proofs about a circuit's HDL representation, rather than the FPGA bitstream. PCHIP bears a superficial resemblance to more traditional formal verification methods in that it uses a domain-specific temporal logic to codify properties, but its ultimate goal is the transmission and efficient validation of proofs of these properties, which standard formal verification cannot accomplish.

PCHIP introduces a new protocol, shown in Fig. 16.11, for the design and acquisition of hardware intellectual property IP cores. Under this system, a hardware IP consumer commissions a vendor to construct a module according to both a standard functional specification and a set of formal security properties stated in a temporal logic. These properties are markedly different from the specification in that they do not necessarily describe the functional behavior of the module; rather, they delimit the acceptable boundaries of this behavior so that the consumer can rest assured no undesired functionality exists. It is then the IP vendor's task to construct a formal proof that his module complies with these properties. Just as in PCC and PCH, the resulting proof is then transmitted back to the consumer along with the IP cores.

Unlike PCH, however, the properties allowed in PCHIP are much more abstract. In [7], the authors described a formal semantics for a carefully codified syntax of an HDL using the Coq proof assistant [22]. In the context of this semantic model they were able to then craft a temporal logic describing the behavior of signals in

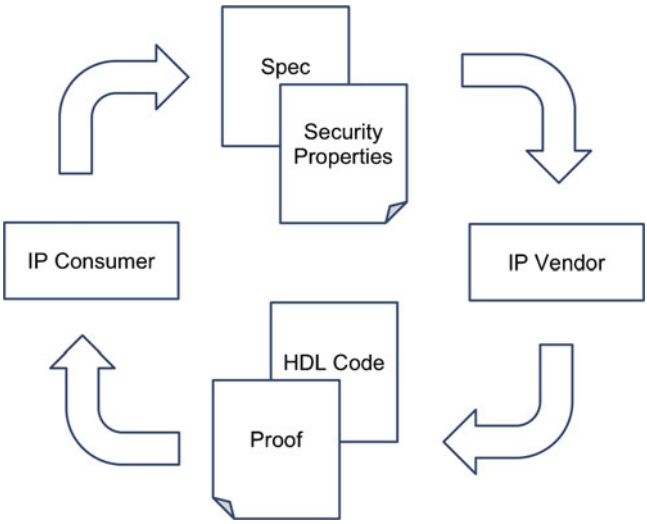


Fig. 16.11 IP core design and aciqusition protocol [7]

synchronous circuits. Security-related properties could be specified in this logic as trees of complex predicates and quantifiers, as opposed to simple Boolean function specifications allowed in PCH. PCHIP provides a set of rules to be applied to HDL code in order to generate a set of propositions in Coq that represent that code's behavior in the already established semantic model. These propositions are then referred to in the proofs that must be constructed for each security-related property.

Figure 16.12 shows how proof-checking proceeds when the consumer receives a circuit's IP code and its corresponding Coq proofs. The code is first passed through a "verification generator" to regenerate all the propositions describing the circuit's behavior, since it is not known whether those included in the received proof actually match the vendor's untrusted code. However, because they are generated according to the same set of rules on both the vendor's side and the client's, it is guaranteed that the resulting propositions will also be the same. Were this not so, then proofs would not be able to refer to them consistently. The regenerated semantic description is then recombined with proofs and security properties, and is checked by the Coq interpreter. If the proofs are found to be valid, then the module is accepted as trustworthy.

The authors of PCHIP argue that the security-related properties expressible in their system can effectively prevent certain types of Trojans by prohibiting the kinds of malicious behaviors Trojans might engage in. Nevertheless, their work leaves a number of important questions that future research might address, especially with regard to the generation of security-related properties. These questions also ask how many of these properties need be standardized, whether there is significant overlap in the kinds of guarantees that consumers of different modules would like to have proven, to what extent proof construction and management can be automated, etc.

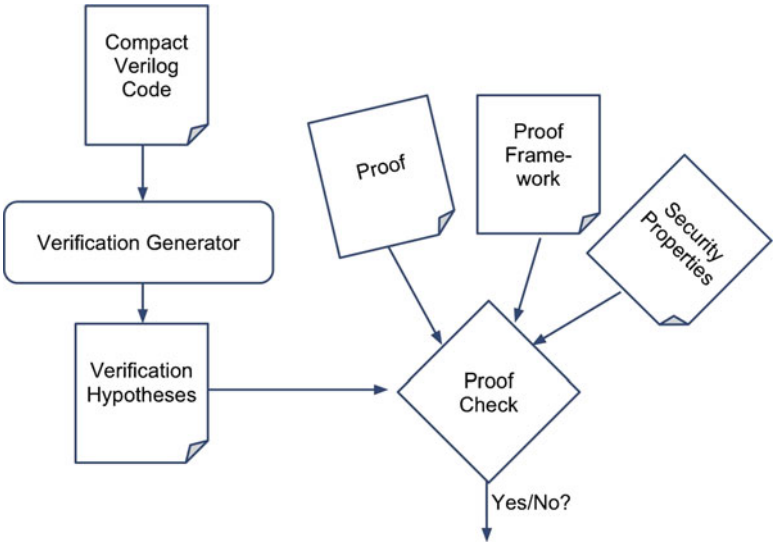


Fig. 16.12 Automated design verification [7]

16.6 Summary

This chapter introduced Trojan prevention methodologies which have been proposed in recent years in the field of *design for hardware trust*. While these methods have proven effective either in simplifying detection of inserted hardware Trojans or in preventing the insertion of Trojans themselves, they do have limitations which one should also be aware of and which point the directions of interest for future research in this area. For side-channel fingerprint-based Trojan prevention methods, area overhead is a key concern because of the complex control logic needed for propagating internal measurements to primary outputs. Other methods, including design obfuscation, dummy flip-flops, and inverted voltage, can only be used if the fundamental assumption that attackers will only choose rare events to trigger the Trojan is valid. Furthermore, a concern with all these methods is the security of the hardening scheme itself, as it is likely that attackers will first compromise the hardening scheme before tampering with the original circuit. In this direction, DFTT is the first method that provides an excellent mechanism for protecting the hardening scheme.

Formal methods such as the PCH and PCHIP paradigms constitute an excellent starting point for future research in the field of pre-silicon Trojan prevention in third party IP. Both methods move the burden of demonstrating IP core security onto the producer in order to accelerate the IP security validation process during its implementation. In the foreseeable future, it is likely that such protocols will become an industry standard in the area of trusted IP acquisition.

Finally, in the domain of hardware trust, contemporary research has almost exclusively focused on digital circuits. However, the extensive use of analog electronics in sensors and actuators, as well as Radio-Frequency (RF) electronics in telecommunications, will certainly attract the interest of both potential attackers and the hardware security and trust community. The first signs of activity in this area have appeared in [23], where the problem of Trojans in wireless cryptographic ICs was studied, so it is very likely that Trojan prevention methods for analog/RF circuits will also be necessitated in the near future.

References

1. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Design Test Comput 27: 10–25
2. Chakraborty RS, Bhunia S (2009) Security against hardware Trojan through a novel application of design obfuscation. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design pp 113–116
3. Salmani H, Tehranipoor M, Plusquellic J (2009) New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 66–73
4. Banga M, Hsiao M (2009) VITAMIN: voltage inversion technique to ascertain malicious insertion in ICs. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 104–107
5. Jin Y, Kupp N, Makris Y (2010) DFTT: design for Trojan test. In: Proceedings of the IEEE International Conference on Electronics Circuits and Systems, pp 1175–1178
6. Drzevitzky S, Kastens U, Platzner M (2009) Proof-carrying hardware: towards runtime verification of reconfigurable modules. In: Proceedings of the Reconfigurable Computing and FPGAs, pp 189–194
7. Love E, Jin Y, Makris Y (2011) Enhancing security via provably trustworthy hardware intellectual property. In: Proceedings of the IEEE Symposium on Hardware-Oriented Security and Trust, pp 12–17
8. Li J, Lach J (2008) At-speed delay characterization for IC authentication and Trojan horse detection. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 8–14
9. Rai D, Lach J (2009) Performance of delay-based Trojan detection techniques under parameter variations. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 58–65
10. Jin Y, Makris Y (2008) Hardware Trojan detection using path delay fingerprint. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, pp 51–57
11. Reece T (2009) Implementation of a ring oscillator to detect Trojans (Vanderbilt University). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Vanderbilt.pdf. Accessed 17 July 2011
12. Rajendran J, Kanuparthi AK, Somasekharan R, Dhandapani A, Xu X (2009) Securing FPGA design using PUF-chain and exploitation of other Trojan detection circuits (Polytechnic Institute of NYU). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Polytechnic_JV.pdf. Accessed 17 July 2011
13. Embedded Systems Challenge (CSAW – Cyber Security Awareness Week) (2009) Polytechnic Institute of New York University.http://www.poly.edu/csaw-embedded
14. Guo R, Kannan S, Liu W, Wang X (2009) CSAW 2009 team report (Polytechnic Institute of NYU). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Polytechnic_Rex.pdf. Accessed 17 July 2011

15. Narasimhan S, Du D, Wang X, Chakraborty RS (2009) CSAW 2009 team report (Case Western Reserve University) CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/CaseWestern.df. Accessed 17 July 2011
16. Yin C-ED, Gu J, Qu G (2009) Hardware Trojan attack and hardening (University of Maryland. College Park). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Maryland.pdf. Accessed 17 July 2011
17. Jin Y, Kupp N (2009) CSAW 2009 team report (Yale University). CSAW Embedded System Challenge. http://isis.poly.edu/~kurt/s/esc09_submissions/reports/Yale.pdf. Accessed 17 July 2011
18. Wolff F, Papachristou C, Bhunia S, Chakraborty RS (2008) Towards Trojan-free trusted ICs: problem analysis and detection scheme. In: Proceedings of the IEEE Design Automation and Test in Europe, pp 1362–1365
19. Abramovici M, Bradley P (2009) Integrated circuit security: new threats and solutions. In: Proceedings of the 5th Annual Workshop Cyber Security and Information Intelligence Research: Cyber Security and Information Challenges and Strategies, article 55
20. Hsiao MS, Tehranipoor M (2010) On trust in third-party hardware IPs. Trust-HUB.org, 2010. http://trust-hub.org/resources/133/download/trust_hub_sept2010-v03.pdf. Accessed 17 July 2011
21. Necula GC (1997) Proof-carrying code. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp 106–119
22. INRIA (2010) The coq proof assistant. http://coq.inria.fr/. Accessed 17 July 2011
23. Jin Y, Makris Y (2010) Hardware Trojans in wireless cryptographic ICs. IEEE Design Test Comput 27: 26–36

Chapter 17
Security and Testing

Kurt Rosenfeld and Ramesh Karri

17.1 Introduction

Test interfaces are present in nearly all digital hardware. In many cases, the security of the system depends on the security of the test interfaces. Systems have been hacked in the field using test interfaces as an avenue for attack. Researchers in industry and academia have developed defenses over the past 20 years. A diligent designer can significantly reduce the chance of system exploitation by understanding known threats and applying known defenses.

17.1.1 Development of Test Interfaces

Test interfaces have been part of man-made systems for at least 100 years. They address a need for applying a stimulus and/or making an observation via a path other than the primary functional path. For example, large tanks holding liquids or gases usually have inspection ports. These ports allow the internal condition of the tank to be visually evaluated to avoid unexpected failure due to corrosion. Otherwise, it would be difficult to predict failure. Brake systems on cars often have inspection holes in the calipers. This allows the condition of the brake pads to be assessed without disassembling the brakes. More than just the *functional* question of "Do the brakes work?", the inspection hole allows the mechanic to answer the deeper question of "How much more life is left in the brake pads?". In areas where

K. Rosenfeld (✉)
Google Inc., NY, USA
e-mail: kurt@isis.poly.edu

R. Karri
Polytechnic Institute of New York University, Brooklyn, NY, USA
e-mail: rkarri@poly.edu

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_17,

operational reliability and efficiency are valued, features are added to products to make them testable, to let their maintainers probe their internal condition.

As electronic devices grew more complex in the mid 20th century, it became difficult to tune them or diagnose problems with only an input–output view of the system. Take, for example, a 1960s radio receiver. These receivers contain several filters and mixers cascaded to form the desired frequency response. There are dozens of adjustments, many of which interact, and all of which affect the output. Optimal receiver performance is achieved for a specific vector of settings. Applying a signal to the input while observing the output, it is almost impossible for the technician to infer which adjustment to change to bring the receiver closer to correct alignment. To make their equipment maintainable, manufacturers provided test points in their circuits, where signals could be measured or injected. This allowed the circuit to be *structurally decomposed* to make maintenance straightforward. Each section can be independently aligned, a process involving only a small number of adjustments.

When electronic computers were first developed in the 1940s and 1950s, it was customary to write "test" or "checkout" programs that could be run on the system to verify correct functionality of the hardware. Test programs were designed so that if it failed, it would provide the technician with an indication of where it failed, speeding diagnosis and repair. The method of running programs on the computer to test the computer is really just functional testing, and since there isn't enough time for the tests to cover all possible states and transitions of the hardware, this testing paradigm can never provide rigorous assurance of the hardware even if all of the tests pass.

As the complexity of computers grew in the 1960s, designers sought stronger assurance from testing, and faster fault isolation. From an operational standpoint in the field, designers wanted to minimize the mean time between when a fault is detected and when the system is back up. As computers began to play crucial roles in real-time operations, high availability became a goal, in addition to the traditional performance goals. All of these factors led major computer developers such as IBM to develop techniques for testing the structural blocks independently.

17.1.2 Example: Testing a Two-bit State Machine

As an illustration of a digital circuit testing problem, consider testing a 2-bit synchronous circuit with the state diagram shown in Fig. 17.1. The circuit has two inputs: clock and reset. It has one output signal, which is high if and only if the state is S3. On every rising edge of the clock signal, the next state is determined as follows:

- If the reset signal is asserted, the next state is the all-zeros state.
- Otherwise, the next state is `oldstate+1` *mod* `4`.

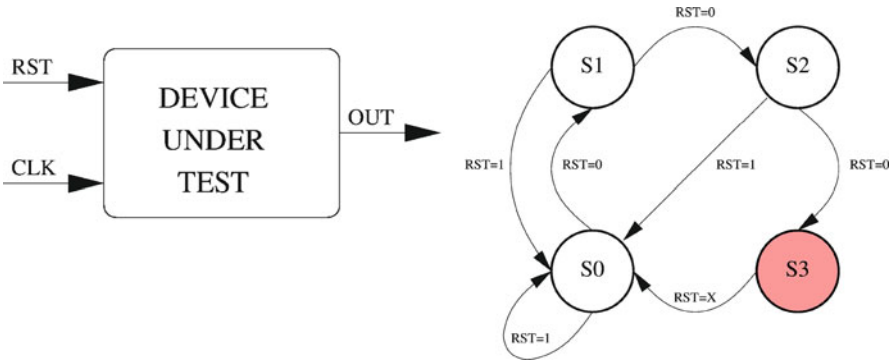


Fig. 17.1 The state machine is specified as having four states. If RST is asserted, the next state is S0. Otherwise, the state machine counts forward. The output is high when the state is S3

The process of testing a practical state machine implementation is affected by three main considerations:

1. What are we trying to determine?
2. What can we assume about the device under test?
3. How is our testing constrained?

17.1.2.1 What are we Trying to Determine?

For the first question, one possible answer is that we are trying to determine whether the device under test is equivalent to a reference implementation or model. Equivalence, in this case, means that for all possible input sequences, the outputs are the same. Another possibility is that we aim to determine whether the device is equivalent to the reference implementation for some restricted set of inputs. Yet another possibility is that we are checking whether an invariant rule is satisfied for some set of input sequences.

17.1.2.2 What can we Assume?

Assumption: Number of States

The second question we must ask when testing a circuit is what we can assume. A state machine is composed of a set of states, a set of edges (transitions), a set of inputs, a set of outputs, and an initial state. It is profoundly helpful to know how many states the system has, how many flip-flops are in the circuit. If we know how many states it has, we can, for example, apply the pumping lemma for regular languages [1], to place limits on the state machine's behavior. The pumping lemma states that if the number of states is finite, there must be an infinite set of input sequences that result in the same final state. This has practical ramifications for

testing. If the number of states is bounded, it is possible, at least in theory, to completely test the circuit using a finite set of test vectors.

Under adverse security circumstances, we are not able to safely assume that the device under test has the number of states it is specified to have. A Trojan horse might be inserted into the circuit during design or fabrication. A typical Trojan horse waits for a *trigger*, which is a special pattern or a sequence. The trigger occurs in, for example, the input data stream, the Trojan activates its *payload*. A payload carries out a malicious action which can be as complex as executing an embedded program, or as simple as halting or resetting the system. Trojans are designed to pass normal testing, so they typically contain all of the benign specified logic, plus extra logic for trigger detection and payload execution. Consequently, from a testing standpoint, the Trojan is more an extra feature than a defect.

Testing for the presence of extra state variables is exceedingly difficult. Consider testing a system that is intended to implement the four-state state machine shown in Fig. 17.1. There are two inputs: clock and reset. There is one output. The output is "1" when the state is S3. The system could faithfully implement the intended state machine (Fig. 17.1), or it might, for example, implement the state machine shown in Fig. 17.2, where the output is "1" when the state is S03, S13, S23, or S33.

For normal inputs, the machine shown in Fig. 17.2 might be indistinguishable from the machine in Fig. 17.1, but it has more states than the intended design. Certain rare sequences of inputs cause the two machines to differ in their outputs. Black-box testing would be much more likely to conclude that the state machines are the same than to find the difference. The rare sequence that causes them to differ can be the trigger for an embedded Trojan, and the way in which the they differ can be the payload of that Trojan.

As we cannot force the counter directly into an arbitrary state, we must sequentially visit the states and test each of the edges while observing the functional output.

However, consider the effect of that test sequence on the machine shown in Fig. 17.2. The machine goes through the following sequence of states: S00, S01, S02, S03, S00, S00, S01, S02, S00, S01, S10, S11, S12, S13, S10. At no point during the test sequence does its externally observable behavior differ from the intended behavior, that shown in Fig. 17.1, although the final state is not the initial state. In the case of this example, running the test sequence repeatedly will not uncover any differences between the actual state machine and the specified one. Although the Fig. 17.1 system and Fig. 17.2 system are the same for the Table 17.1 test sequence, they behave quite differently for other test sequences, specifically, any sequence that puts the Fig. 17.2 system into the S23 state, where it locks up. In summary, when testing a state machine, we make an assumption about the number of states. If our assumption is wrong, we are likely to make invalid inferences about the system under test.

Assumption: Determinism or Randomness

A pivotal assumption is that the device under test is deterministic. If it is not, then we must characterize its behavior statistically instead of logically. That completely

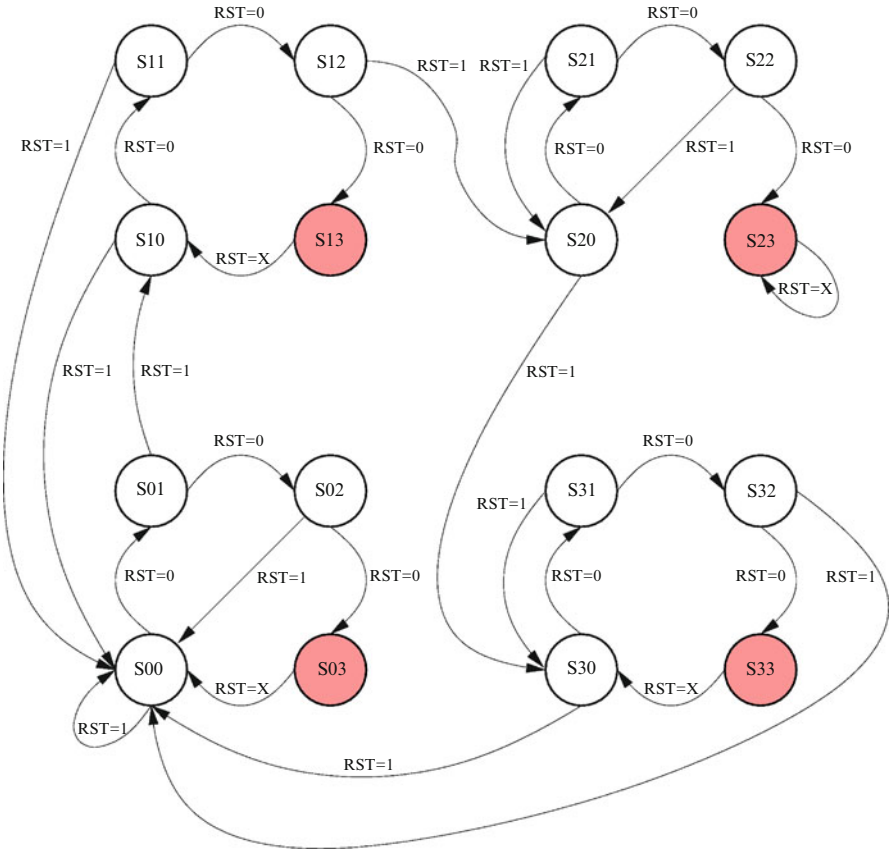


Fig. 17.2 The machine behaves similarly to the machine shown in Fig. 17.1, but deviates for certain rare inputs. Starting in initial state S00, if RST is given the sequence 0,1,0,0,1,0,0,0 the machine enters state S23, at which point the behavior of the system deviates from that shown in Fig. 17.1. S23 is a terminal state. The only was to exit S23 is to reinitialize the system (e.g., cycle the power)

Table 17.1 Test routine for the original two-bit counter shown in Fig. 17.1

RST	Current output	Transition	RST	Current output	Transition
1	X	X − >S0	1	0	S2 − >S0
0	0	S0 − >S1	0	0	S0 − >S1
0	0	S1 − >S2	1	0	S1 − >S0
0	0	S2 − >S3	0	0	S0 − >S1
0	1	S3 − >S0	0	1	S1 − >S2
1	0	S0 − >S0	0	0	S2 − >S3
0	0	S0 − >S1	1	1	S3 − >S0
0	0	S1 − >S2			

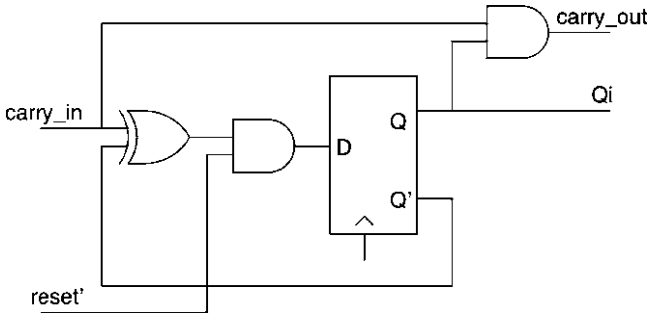


Fig. 17.3 A cascadable section of a synchronous binary counter with synchronous reset

changes the nature of the testing procedure. One example is testing a pseudorandom bit sequence generator. The output is expected to satisfy certain statistical requirements. There are standard requirements, such as Golomb's randomness postulates. A single test is not sufficient to establish the randomness of a sequence. Standard suites of tests have been developed for the purpose of comparatively evaluating cryptographic systems [2]. Related to the testing pseudorandom sequence generators is the testing of "true" randomness sources, which derive their randomness from a physical source. Typically a diode is used as a noise source, which is then digitized to produce random bits. Testing such devices for their fitness in security-critical applications involves several special criteria such as their immunity to external factors influencing their output.

How is our testing constrained?

Systems that only allow interaction in their normal functional use pattern demand black-box testing. This constraint can appear for a variety of reasons, and has wide-ranging implications. One reason for the absence of test features is their perceived cost, either in engineering effort or in production cost. The implications of being forced to resort to black-box testing are an exponential increase in the time required to test a system, and decreased confidence that it is thoroughly tested.

When we are not constrained to black-box testing, the practical approach to testing, for example, a counter, is to structurally decompose it and test the components and the interconnections, and then argue that if the components are good, and the interconnections are good, then the system is good. When we decompose the circuit, we break it into small islands of logic that are easily testable, avoiding the exponential explosion of test complexity seen in the previous paragraph. A 2-bit counter can be implemented as two 1-bit sections cascaded. A 128-bit counter can be implemented as 128 cascaded sections, each containing a flip flop and three logic gates, as shown in Fig. 17.3.

If suitable test structures are provided, the 128 sections can be tested independently. The number of tests necessary for one isolated section is

$$\#\mathtt{tests} = 2^{\mathtt{F}}2^{\mathtt{I}},$$

where F is the number of flip-flops in each section and I is the number of inputs to each section. We have $F = 1$ and $I = 2$, so eight tests a required per section of the counter. If the stages of a B-bit counter are tested sequentially, the number of tests is

$$\#\mathtt{tests} = 2^{\mathtt{F}}2^{\mathtt{I}}\mathtt{B} = 8\mathtt{B}.$$

Without structural decomposition, we have to visit each state and test the circuit's response to the RESET $= 0$ input and the RESET $= 1$ input. This requires two tests per state, so

$$\#\mathtt{tests} = 2 * 2^{\mathtt{B}} = 2^{\mathtt{B}+1}.$$

Structural testing is not just somewhat more efficient than black-box testing. It is profoundly more efficient. The number of tests required for structural testing is of linear complexity, $O(\mathrm{B})$, while black-box testing is of exponential complexity, $\mathrm{O}(2^{\mathrm{B}})$. Similar general results apply to circuits other than counters. The total complexity of the pieces of a logic circuit is almost always less than the complexity of the circuit as a whole. Consequently, black-box testing of whole circuits is avoided for all but the simplest systems.

17.1.3 Fault Testing vs. Trojan Detection

The standard goals of test engineering are to detect flaws that occur naturally during fabrication and isolate the exact location of the flaw. Any significant deviation from the behavior intended by the designers is considered a fault. By this definition, a piece of malicious logic that is added to the design by an attacker before fabrication is a fault. Although it might seem elegant, and it is certainly correct, to group malicious modifications with manufacturing faults, it is not practical to do so. The fault models assumed when testing for manufacturing faults do not include the changes that would be made by a malicious party who adds some kind of Trojan horse to the design.

17.1.4 VLSI Testing: Goals and Metrics

VLSI testing is always done in terms of a prescribed *fault model*. For example, a common fault model for logic circuitry is that each node can be correct, stuck at 1, or stuck at 0. If this *stuck-at fault model* is used, the goal of testing is to determine, for each node, which of the three conditions describes it. A set of tests is said to *cover* a fault if the test would detect the fault if the fault were present in the device being tested. *Test coverage* is the percentage of possible faults that are covered by a

given set of tests. In many cases, it is practical to create a set of tests that has 100% coverage. In some cases, 100% coverage is not reachable for a practical number of test vectors.

The number of test vectors that are needed for 100% test coverage is an indication of the *testability* of the circuit. Two factors are important in determining the testability of the circuit:

- Controllability
- Observability

Certain topologies are known to result in poor testability. One example is *reconvergent fanout*. This is when a signal fans out from a single node and follows multiple parallel paths, and then reconverges into a single node. This topology exhibits poor testability because the signals along the parallel paths are not independently controllable. Logic gates with a large number of inputs, particularly XOR gates, are also problematic.

Design for Test (DFT) is a design process that runs in parallel with the functional design process. The goal of DFT is to ensure that the final design meets testability goals while minimizing the costs associated with testing. There are four main costs that relate to testing.

- Die area cost
- Time required to apply tests
- Cost of the required testing station
- Computation required to generate test vectors

17.1.5 Conflict Between Testability and Security

Conventional security best practices are to conceptualize the system as a set of modules which expose simple interfaces and hide their internal implementation. This type of control limits the complexity of the interactions in the system. However, black-box testing is profoundly inefficient. Providing controllability and observability of the internal elements within a module makes that module testable, but the module then loses the security that comes from having a single, restricted interface. Thus, there is an apparent conflict between security and testability.

17.2 Scan-Based Testing

As we discussed in the previous section, an important approach for achieving testability in a VLSI chip is to include elements in the design that allow it to be structurally decomposed for testing. Most designs are synchronous, meaning that they are composed of logic and flip-flops, and the flip-flops only change state during

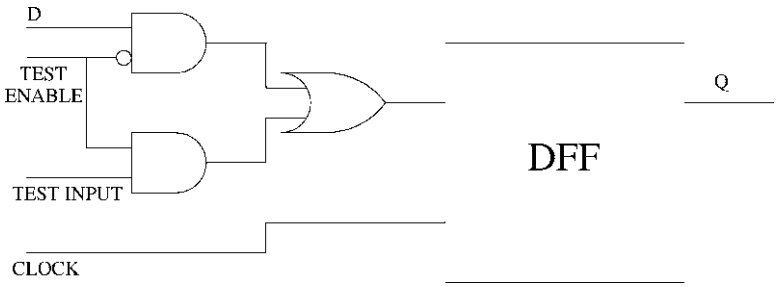


Fig. 17.4 The simplest scan flip-flop cell is simply composed of a multiplexer and a regular D flip-flop. The Q output of one scan cell can be connected to the TEST INPUT of another scan cell, enabling a chain configuration

the rising or falling edges of the clock. Synchronous designs can be separated into a set of logic gates, which can be tested by verifying their truth table, and a set of flip-flops, which can be tested by chaining them to form a shift register.

The scan-based testing paradigm replaces the regular flip-flops in the design with "scan flip-flops" as shown in Fig. 17.4. These are flip-flops with input multiplexers which select between the regular "functional" input and the test-mode input. Typically, the test-mode input comes from the output of another flip-flop, thus forming a scan chain. The operation of a scan chain can be thought of as having three phases:

- *Assert test mode*. All flip-flops are configured into a distributed shift register. Test data is shifted in. This data is applied to the inputs of the logic.
- *Deassert test mode*. Flip-flops are configured to get their data from the outputs of the logic. The flip-flops are clocked, thus latching in the output value of the logic.
- *Reassert test mode*. All flip-flops are configured into a distributed shift register. Test data is shifted out. This data is returned to the tester for analysis and comparison with the expected output.

Using this testing method, the tester only tests combinational logic, instead of testing state machines. This amounts to a profound reduction in the testing time required to achieve a given level of test coverage.

17.2.1 Scan-Based Attacks

Systems that support scan-based testing are potentially vulnerable to attacks that use the scan chains as vectors for reading or modifying sensitive data contained in the device. Cryptographic keys are common targets for this kind of attack.

The most basic scan attack applies to chips that contain a key register that can be scanned. In this attack the attacker typically connects to the JTAG port of the victim chip, selects the scan chain that contains the key register, and shifts out the key.

Less naive chips avoid having the key directly scannable. However, simply excluding the key register from the scan chain is not sufficient to prevent a skilled attacker from extracting the key. Yang et al. [3] present a practical attack against an AES hardware implementation with a nonscannable key register. They exploit the scan chains as an information leakage path, allowing recovery of the crypto key. They assume that the key is not directly scannable, and their attack uses indirect information to reconstruct the key. They also assume that the attacker does not have knowledge of the internal scan chain structure of the crypto chip. The attack begins with discovering the scan chain structure, determining which bit positions in the scan chain correspond to the bits of the intermediate result register. Next, using the observability of the intermediate result, the attacker recovers the round key. The attack shows that even if the key register is not exposed directly though any scan chain, the byproducts of the key contain enough information for the attacker to infer the key bits.

17.2.2 Countermeasures for Scan Attacks

Countermeasures are available for protecting against scan attacks. There is a tradeoff between security and testability. Effective countermeasures for scan attacks must simultaneously provide acceptable levels of security and testability.

Hely et al. [4] observe that a static assignment of functional registers to positions in the scan chain is risky because an attacker can infer the assignment and then use the scan chain to extract secret information from the chip. To mitigate this threat, they introduce *Scan Chain Scrambling*. For the authorized user, a *test key* is provided to the chip and assignment of registers to scan chain positions is static. For an attacker without the ability to authenticate, the assignment is semirandom. Chunks of the set of scannable flip-flops are mapped to chunks of the scan chain sequence, but the order will be unknown to the attacker. The permutation changes periodically while the chip is operating.

Yang et al. [3] propose the "Secure Scan" scheme for protecting embedded keys from being leaked via scan chains. To allow the crypto hardware to be fully tested without exposing the key or its byproducts, Secure Scan introduces a second embedded key, the *mirror key* for use during testing.

At any moment, as shown in Fig. 17.5, the chip is either in mission mode ("Secure mode") or test mode ("Insecure mode"). When the chip is in secure mode, the mission key is used but scanning is disallowed. When the chip is in insecure mode, scanning is allowed, but only the mirror key is used. It is impossible to get from secure mode to insecure mode without powering off the chip. Secure Scan thus allows almost complete structural testability without exposing the mission key. As the mission key is not used while the chip is in insecure mode, the mission key is

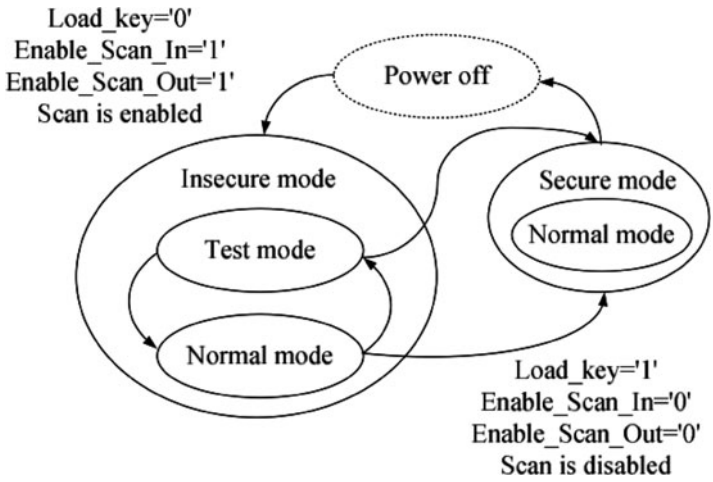


Fig. 17.5 Secure Scan state diagram. The only way to get from secure mode, where the mission key is loaded, to insecure mode, where the chip is testable, is to go through a power cycle reset, which wipes all volatile state

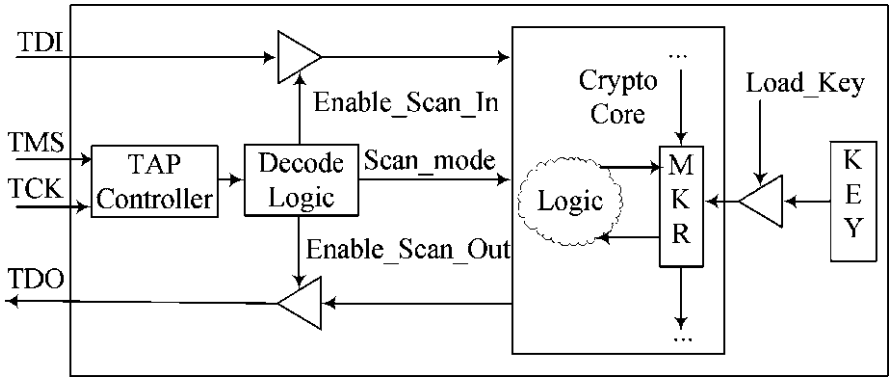


Fig. 17.6 Secure Scan architecture. The mirror key register (MKR) is loaded only when the Load Key signal is asserted, which is controlled by the state machine shown in Fig. 17.5

obviously untestable. This is, however, not a serious drawback in practice because the correctness of the key can be quickly verified byfunctional testing.

Lee et al. [5] point out that the intellectual property contained in an integrated circuit is also at risk because of scan chains. An attacker that can control and observe signals inside a chip may be able to infer the logic of the chip, and thereby learn the design. To prevent this, the authors introduce a technique they call *Low-Cost Secure Scan*, that blocks unauthorized access to the scan chains. The technique requires modification to the algorithms used for inserting scan chains in designs,

but the scope of the protection includes the intellectual property of the design, not just embedded keys. To use the Low-Cost Secure Scan system, the test vectors contain key bits in addition to the test stimulus. If the key bits are correct, the chip produces the precalculated output. Otherwise, the test response is pseudorandom. The pseudorandom response is intended to raise the difficulty for an attacker who wishes to launch a guessing attack, by not giving an immediate explicit indication of whether the attacker's guess is correct.

17.3 BIST

Built-In Self Test (BIST) is a popular technique for testing hardware without requiring external test equipment. There are many varieties of BIST designed to test different kinds of circuits and to detect different classes of faults. A common feature of all BIST systems is that a large volume of test data moves between the on-chip BIST logic and the circuit under test, while a minimal amount of data moves between the chip and it surrounding system. In the extreme, the chip can simply produce a one-bit status indication of whether it is working correctly, or there is an error.

Two of the most common varieties of BIST are memory BIST and logic BIST. In principle, differentiating between memory and logic is not necessary. In practice, testing techniques that target a specific type of circuit are much more efficient in terms of test time required to get adequate fault coverage. Typical memory BIST uses a state machine or a small embedded microprocessor to generate memory access signals that carry out a standard memory test algorithm such as the March series of tests [6]. In a typical logic BIST setup, pseudorandom test vectors are generated on-chip and applied to the circuit under test. The responses are compacted and aggregated during many test cycles, using a Multiple-Input Shift Register (MISR). This produces a fixed-length final value that is compared with the expected value, which is hard-coded into the chip.

From a security standpoint, BIST has many ramifications. In both logic and memory testing, the BIST controller can act as a trusted proxy between the tester and the chip's core logic. This architecture can raise security by enforcing a limited set of actions that can be taken by a tester. The enforcement of limited interfaces is consistent with good security design (e.g., principle of least privilege). The tester of cryptographic logic needs assurance that the hardware is working correctly, but shouldn't necessarily have access to secret data in the chip, such as an embedded key. Similarly, BIST can provide assurance that a memory is defect-free while eliminating the possibility of misuse of memory "debugging" functionality for tampering with security-critical data.

Despite BIST's advantages of running at full functional clock speed and improving security, it has two problems. First, it typically provides fault detection, but not fault isolation. Second, it adds area to the chip. A BIST implementation contains a test pattern generator and an output response analyzer. Both of these hardware modules occupy area. However in certain applications this area cost can

be eliminated. A technique called Crypto-BIST [7] uses a symmetric cipher core (AES) to "test itself". By looping the output of the AES core back into the input, the AES core functions as both the test pattern generator and output response analyzer. Crypto-BIST achieves 100% stuck-at fault coverage of the AES core in 120 clock cycles of test time. The essence of the technique is the observation that strict avalanche criterion of the cryptographic algorithm causes the AES core to act as both a diverse source of test patterns and a sensitive output response analyzer, leading to high test coverage in few cycles.

17.4 JTAG

In the 1970s and early 1980s, a common way of testing printed circuit boards was to add test points, and to probe these test points with a bed-of-nails test fixture, as shown in Fig. 17.7.

This approach could not keep up with the increases in component density and pin spacing. Consequently, other methods of testing were applied. As always, cost was a

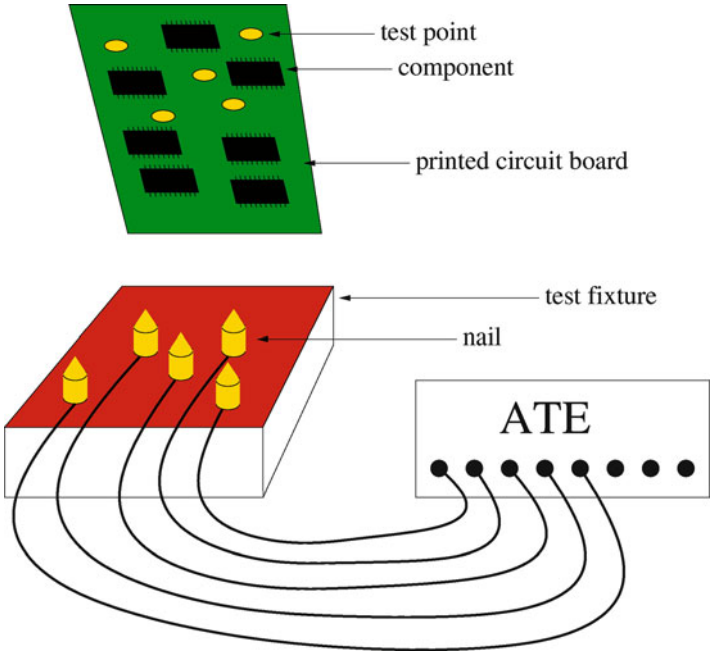


Fig. 17.7 Bed of nails test fixture. Automated test equipment (ATE) generates stimulus signals and measures responses. The ATE is connected to the test fixture, which contains one nail per test channel. Each nail is spring-loaded so it maintains a controlled pressure when contacting the test points on the printed circuit board being tested

major factor affecting the choice of test method. Another factor was interoperability, since components from many manufacturers coexist on large printed circuit boards. Having a single *test interface* from which all components could be tested was desired. The solution was developed by a working group known as the Joint Test Access Group in the 1980s. This became IEEE Standard 1149.1 and is widely referred to simply as JTAG.

IEEE 1149.1 standardizes the set of signals used to access test logic inside chips. The standard specifies the use of scan-based testing for the internal logic of the chip and also for the inter-chip wiring. JTAG uses synchronous serial communication with separate signals for data and control. Using 1149.1, a tester can force signals on pins, read signals on pins, apply signals to the core logic, read signals from the core logic, and invoke arbitrary custom test functions that might exist in certain chips. However complicated the testing task might be, the test protocol always uses the same states (Fig. 17.8) and signals:

- TCK – test clock, while in test mode, all events happen on edges if TCK
- TMS – test mode select, determines the next state of the JTAG port
- TDI – test data in, test vectors and JTAG instructions are applied via TDI
- TDO – test data out, test responses or data that loops through
- TRST – test reset, optional hardware reset signal for the test logic

An important feature of JTAG is its support for daisy chaining. Devices can be wired in a chain where the TDO (output) of one device is applied to the TDI (input) of the next device in the chain, as shown in Fig. 17.9. The TCK, TMS, and TRST signals can be simply bussed to all chips in the chain, within fan-out limits. Otherwise, buffers can be used. Each chip in the JTAG chain has a state machine implementing the protocol.

One of the state variables controlled by the state machine is the Instruction Register (IR). The instruction register is typically between 4 and 16 bits. Some instructions are mandated by the JTAG standard, while implementers are free to define as many of their own instructions as they like. One of the most important instructions is the required instruction, BYPASS. When the IR contains the BYPASS opcode, a JTAG-compliant chip places a single flip-flop in the path from its TDI input to its TDO output. Therefore, a chain of chips in the BYPASS state behaves like a shift register (Fig. 17.10).

17.4.1 JTAG Hacks

JTAG has played a part in many attacks on the security of digital hardware. Attackers have used it to copy cryptographic keys out of satellite boxes for the purpose of pirating satellite TV service [8]. The JTAG port in Microsoft's Xbox 360 has been exploited to circumvent the DRM policies of the device [9]. Powerful low-level capabilities are often exposed through the JTAG interfaces of systems. Attackers have learned this, and when they attack a device, they look for a JTAG port, among other things.

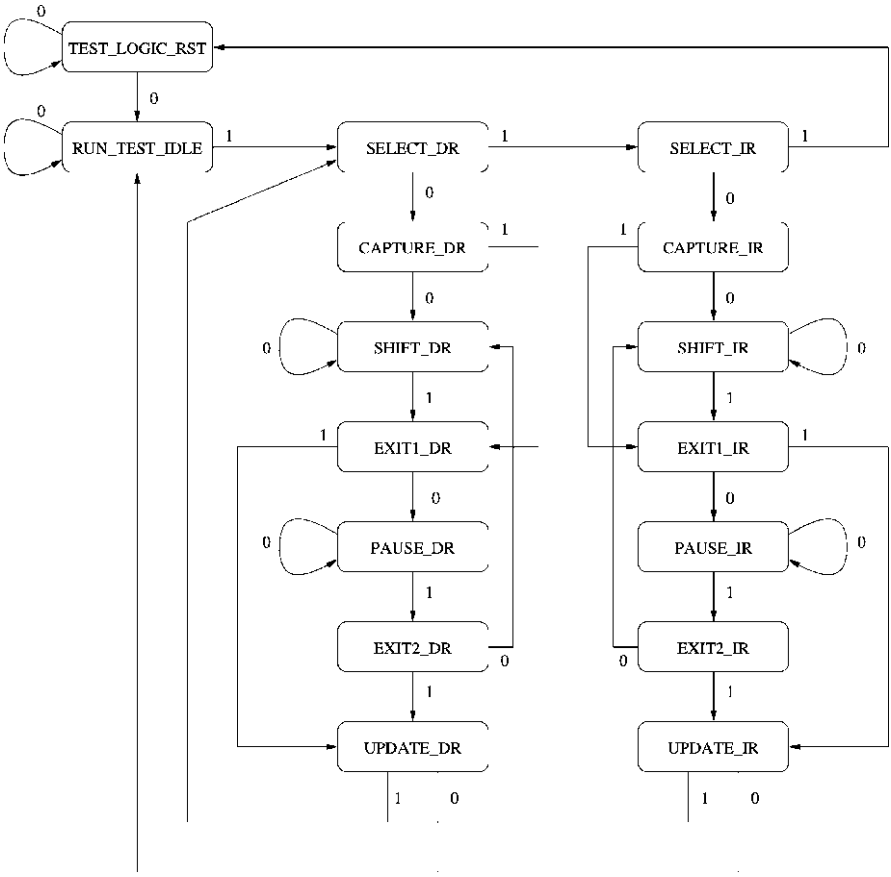


Fig. 17.8 The JTAG state machine. There are 16 states. The TMS signal determines the next state. The SHIFT_DR state is used for applying stimuli and collecting responses. From any state, the TEST_LOGIC_RESET state can be reached by holding TMS high for five clock cycles

Rosenfeld and Karri [10] examine the threat of JTAG-level hacking. Specific attention is given to the vulnerabilities that result from the common daisy-chain topology of JTAG wiring. They consider the possibility of one malicious node in a JTAG chain attacking other nodes or deceiving the tester. They examine the threat of a malicious node hijacking the bus by forcing the control signals. With two nodes (the tester and the attacker) both driving a bus wire at the same time, it becomes an analog question, who will win. The research showed that it was often possible for the attacking node to hijack the bus when the JTAG bus wires are short, and always possible to hijack when the bus wires are long, due to pulse properties of transmission lines.

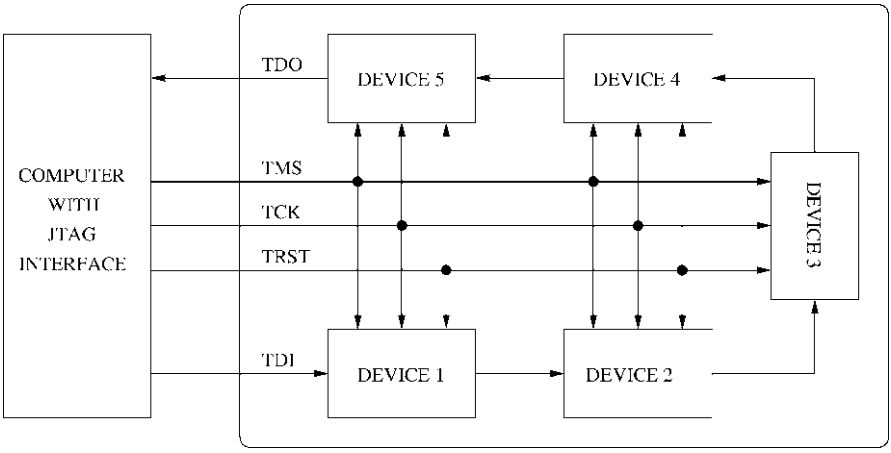


Fig. 17.9 A typical JTAG system. TMS, TCK, and TRST are bussed to all of the devices. TDO of each component is connected to TDI of the next component, thereby forming a daisy-chain topology

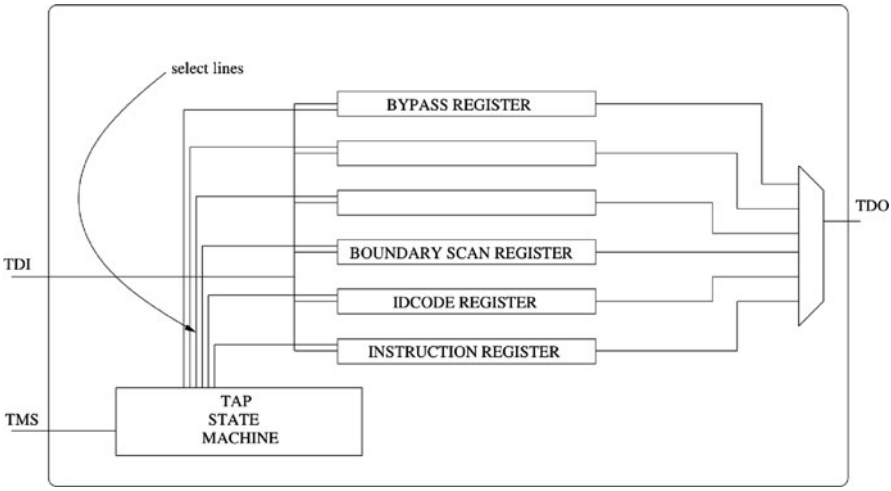


Fig. 17.10 The essential components of a basic JTAG implementation include a test access port state machine, an instruction register, one or more data registers, and an output multiplexer. Each chain of scan flip-flop cells (internal or boundary) appears to JTAG as a data register that can be selected with the appropriate instruction

17.4.2 JTAG Defenses

Several defenses for JTAG have been proposed over the years. When considering JTAG defenses, it is important to keep in mind the many constraints and requirements that affect the design process. For example, flexibility to provide in-field firmware updates is often valuable, but for this to be secure, some sort of authentication mechanism is required. Some applications have tight requirements on cost and cannot tolerate the extra circuitry required for authentication. As always in engineering, there are trade-offs, and making the best choice requires a detailed understanding of the application.

17.4.2.1 Elimination of JTAG

One way to eliminate the risks associated with JTAG is to eliminate JTAG from the design. There are several ways this can be done while maintaining low *escape rate*, the probability of a defective part being shipped to a customer.

One method is simply to use conservative design rules. Common sources of manufacturing faults are shorts and opens in the metal wiring of the chip. If wires are made wider, and spacing between wires is kept greater, many manufacturing faults are avoided. If transistors have nonminimum gate length, that eliminates another source of faults. This approach has a high cost in area and speed.

Another method, and one that is very popular, is to use BIST, discussed in the previous section. The result of running BIST can be as simple as a single bit indicating whether the chip passes the tests or fails. In this form, BIST provides security benefits because internal scan can be eliminated from the set of supported JTAG instructions, thus significantly reducing the chip's attack surface.

BIST, however, is not always a satisfactory replacement for scan-based testing. As BIST test vectors are generated pseudorandomly instead of deliberately, using an ATPG algorithm, it can be difficult to get full test coverage using BIST. This is partially offset by the fact that BIST is typically done *at-speed*, meaning that test vectors are applied at the same rate that functional data would normally be applied. In contrast, when test vectors are applied using external automated test equipment, the test clock is typically an order of magnitude slower than the functional clock. Another disadvantage of BIST is that it does not provide fault isolation. For the engineers developing a chip, it is essential to be able to quickly iterate toward a successful design that can be shipped. Without the ability to determine the location and type of the fault, designers are not able to fix the problem. For this reason, BIST is more useful for testing during full production and in the field, where a failure will simply cause the part to be discarded. Scan-based test infrastructure is often retained in the design, in case it is needed for engineering purposes.

17.4.2.2 Destruction of JTAG Hardware after Use

In many circumstances, an argument can be made that the test structures in a chip are only needed at the factory, and constitute nothing more than a security risk once the chip is shipped. In such cases designers sometimes elect to disable the JTAG circuitry on the chip after testing. A common way of implementing this is with fuses that can be electrically blown by the tester. For finer grained control over what capabilities remain enabled, the design can contain more than one fuse. A patent by Sourgen [11] in 1993 discusses these techniques.

The IMX31 microprocessor from Freescale Semiconductor is an ARM-based chip intended for mobile and media applications. This type of embedded processor is often required to protect the data that in processes, in the case of digital rights management, and the code that it runs, in cases where the system software contains valuable intellectual property. The IMX31 supports four JTAG security modes, selectable by blowing fuses. In mode 4, the JTAG logic allows all possible JTAG operations. In mode 1, only the JTAG operations necessary for interoperability are allowed. Blowing fuses is an irreversible operation.[1] Therefore, the security mode can be raised, but never lowered. This is fits well with a common use case of JTAG, where it is used at the factory for testing and perhaps by engineers for in-system debugging in their labs, but should not be used in the field by hackers.

17.4.2.3 Password Protection of JTAG

Buskey and Frosik developed a scheme they call Protected JTAG [12], which enhances the security of JTAG by requiring authentication and authorization to access particular features of the chip's test structures. The scheme makes use of a trusted server which uses a preshared elliptic curve key pair to prove to the chip that the user's JTAG access request has been authenticated. The authors anticipate a use case where the tester connects directly to the chip and connects to the trusted server via the Internet, using standard Internet communication security protocols. Once authentication is complete, the chip stays in the authenticated state for the duration of a test session. The benefits of having a separate trusted server for authentication and authorization are that these can be managed independently, after the chip is deployed. For example, a new user can be added any time, with access to an arbitrary set of test features. A disadvantage of the scheme is the reliance on the continued security and availability of the authentication server.

[1] If the threat model includes skillful, well-funded attackers who are willing to physically disassemble and modify the chip, then blowing fuses is not necessarily irreversible.

17.4.2.4 Hiding the JTAG Behind a System Controller

One approach to JTAG security is to use a system controller chip. In this architecture, the system controller acts as a proxy for test-related communication with one or more chips, typically on a printed circuit board. This scheme adds security to a system without requiring any modification to the chips themselves. The system controller can enforce sensible security policies such as:

- All accesses must be authenticated.
- Authenticated testers can only access the resources for which they are authorized.
- Only signed and verified firmware updates are permitted.
- "Backrev" (reverting to a previous version) of firmware is not permitted.
- All communication between the tester and the system controller is cryptographically protected against man-in-the-middle attacks.

The system controller can play an active role in the testing and maintenance of the system, beyond simply acting as a proxy [13]. The system controller can store test sequences and run the tests automatically at power-up time or when a specific test routine is externally invoked. This architecture provides the benefits of BIST as discussed in Sect. 17.3. A controller supporting this type of approach is commercially available under the name SystemBIST [14]. It also provides functionality for verifying the state of JTAG-connected devices, for example, to verify that the configuration bit file of an FPGA was correctly programmed. As with all practical security, it is not absolute. Successful practical approaches have to strike a balance between cost, functionality, and security. The value of the system controller approach is that it preserves the economy of scale of using commodity chips that don't directly support any JTAG security enhancements, while providing increased functionality in the form of BIST, while defeating some common classes of attacks.

17.4.2.5 Crypto JTAG with Embedded Keys

Rosenfeld and Karri [10] introduce a set of security enhancements for JTAG that are backward compatible and interoperable with the original IEEE standard. Compact cryptography modules are included in the security-enhanced chips, allowing the tester to authenticate the chip, preventing unauthorized testing, encrypting test data going to and from the chip, and protecting against modification of test data. Keys are programmed into each chip at the factory and delivered to the user through an out-of-band secure channel. This allows the customer of the chip (e.g., system integrator) to confirm that it came from the real factory, thus thwarting supply chain attacks.

17.5 System-on-chip Test Infrastructure

The system-on-chip (SoC) is an important paradigm for the integrated circuit industry. The essential idea is that modules can by designed independently, and integrated onto a single chip. An SoC is in many ways similar to a printed circuit board containing many chips from different manufacturers. Compared to a conventional fully in-house chip development process, SoC development presents new security challenges in how to test, configure, and debug the modules within the chip.

A large number of independent entities are involved in a typical SoC development cycle.

- SoC integrator
- Core designer
- CAD tool provider
- IC testing service
- IC fabrication service
- IC packaging service

An additional set of entities are affected by the security of the device after it is deployed.

- End users
- Service and infrastructure providers
- Content creators and other holders of intellectual property

SoCs are subject to a variety of threats at different stages of their life-cycle. The goals of the attacks that exploit the test mechanisms include grabbing cryptographic keys, changing system behavior, and learning secrets about the intellectual property contained in the SoC.

17.5.1 SoC Test Hacks

The test access mechanisms that are used in SoCs evolved from those that are used on printed circuit boards. In particular, JTAG has been used both for external interfacing of the chip to the test equipment as well as for internal test access to cores within the die. All of the threats that apply to the test interfaces of monolithic ICs also apply to SoCs. At the same time, a number of new threats affect SoCs, primarily due to the fragmented nature of their design, with different trust assumptions applying to the different modules.

17.5.1.1 Test Bus Snooping

SoC test signals can be routed on the chip several different ways. Several engineering considerations affect test signal routing. Traditionally, the trade-off has been

between speed and cost. Wide data paths and dedicated per-core wiring raise cost but give good testing speed. Narrow, often bit-wide, data paths and shared wiring lower cost, but reduce testing speed. The trade-offs in test bus design have been studied extensively [15]. In an SoC made of several cores, the optimal configuration often has multiple cores time-sharing the same test wiring. Typically, the intention is for test data to be communicated between the tester (master) and the test target (slave). Some implementations of shared test wiring allow malicious cores to *snoop* the test bus, receiving messages that go to or from another core. The actions that a malicious core can take with snooped test data depend on the system. If, for example, the test data contains cryptographic keys, a malicious core can leak the keys to an attacker through a side-channel.

17.5.1.2 Test Bus Hijacking

Another concern for the SoC designer is that on shared test wiring, a malicious core could actively interfere with communications between the tester and the target core. This type of attack is most threatening when test data actually passes through the untrustworthy test logic of the malicious core, as it does in daisy-chained architectures like JTAG.

17.5.2 Defenses for SoC Test Mechanisms

Several techniques have been applied by chip companies to address the problem of securing SoC test mechanisms. Successful techniques must, as always, balance security, cost, functionality, and performance goals. Additionally, human factors affect whether new mechanisms will succeed. Engineers are inundated with information during the design process, and often prefer to use mechanisms that are known to work and don't require learning. An example of this effect can be seen in the continued use of RS-232, even in brand new systems with no legacy. To succeed in the market, enhancements that add security to SoC test interfaces must meet security, cost, functionality, and performance goals and also should not burden the engineers that use them.

17.5.2.1 Elimination of Test Mechanisms

The most straightforward approach to securing the test interface is, as always, to eliminate it. Elimination of test interfaces is usually not a practical solution because they are often required, for example, for programming firmware into chips and for configuring platform-specific settings. For reasons of economy of scale, it is preferable for a single chip to support multiple use cases. Test interfaces provide a convenient mechanism for low-level configuration of complex chips. Eliminating

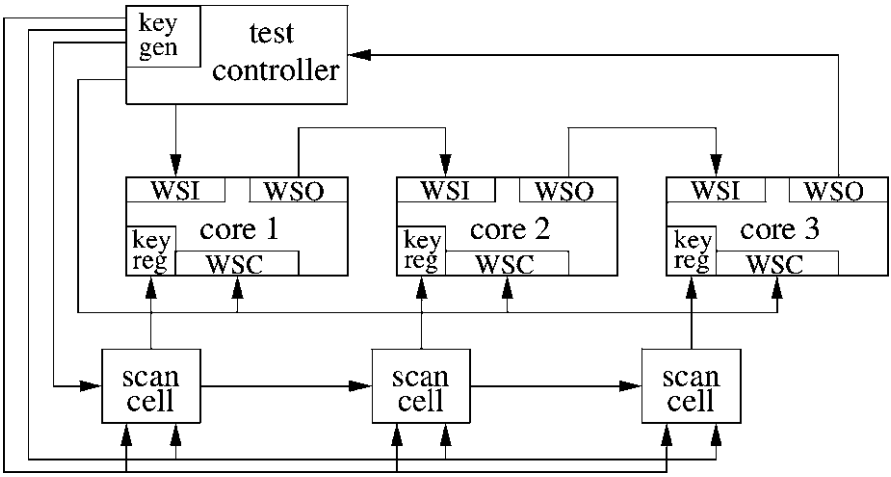


Fig. 17.11 A chain of scan cells is used for distributing keys to each of the cores. The scan cells are configured not to expose key bits at their outputs while they are being shifted

the test interface from a chip design would limit the market for the chip. The applications where security is a very high concern, such as electronics for defense systems, require in-field testability for maintaining high availability.

17.5.2.2 Elimination of Shared Test Wiring

SoC test infrastructures can be secured against attacks by hostile cores by giving each core its own test bus. This results in a star architecture, where the cores are the points of the star, and the test controller is the center. If the trust assumption is that the SoC integrator, CAD tools, and fabrication are trustworthy, but the third-party IP cores are not trustworthy, then the star architecture provides good security, since it minimizes the damage that can result from an untrustworthy core. However, it is the most expensive topology from a standpoint of wiring cost.

17.5.2.3 Cryptographic Protection for Shared Wiring

To capture the cost savings of shared wiring while restricting the security risks of untrustworthy cores on the shared test bus, cryptography can be used. Rosenfeld and Karri [16] developed a low-cost technique that adds security without breaking compatibility with existing test interface standards. They introduce a structure that is essentially a scan chain that snakes through the SoC and provides a trusted mechanism for delivering a crypto key to each core (Fig. 17.11). The technique is based on the assumption that the logic and wiring that is placed by the SoC integrator

is trustworthy, while the third-party cores are not trustworthy. After delivering separate crypto keys to each core during initialization, the tester can use shared test wiring for the actual testing and configuration. The test wiring data path can be arbitrarily wide while the key delivery wiring need only be one bit wide.

17.6 Emerging Areas of Test Security

As engineered systems become more complex year after year, they often acquire testability features. Sometimes standard solutions like JTAG are used, but sometimes new testing schemes are invented and deployed. Each testing scenario has its own security concerns, its own strengths, and its own weaknesses. A common mistake is to develop a new system without listing security in the design requirements. In fact, it is rare to find a nontrivial system where security is not in fact a requirement. Automobiles and medical implants are two areas where test and management interfaces have been deployed without security. Both of these areas have seen recent research results demonstrating the vulnerabilities of the actual systems in the field.

17.6.1 OBD-II for Automobile

Modern cars are heavily computerized. They contain multiple computers performing real-time tasks such as engine control, climate control, braking, traction control, navigation, entertainment, and drivetrain control. These subsystems are typically linked together on a communication bus. From a central point, a technician can check the status of all of the subsystems by communicating with them over the bus. This development is helpful for reducing the time needed to diagnose a problem with the car. The most common interface for electronic testing of cars is On-Board Diagnostics II (OBD-II). OBD-II is a connector, typically under the dashboard.

Like all test interfaces, OBD-II exposes the system to attacks that would not otherwise be possible. Koscher et al. [17] present an analysis of automobile security with a focus on OBD-II and the Controller-Area Network (CAN) that is used to interconnect the components in the car. They show that by plugging a malicious computer into the OBD-II test interface, they can severely undermine the security of the entire car, causing it to behave in ways that would be dangerous or even deadly for the occupants of the car and for others around them.

It is a common saying in the field of information security that security cannot be an afterthought; it must be considered from the beginning and built into the product. It is unlikely that the security flaws in existing cars will be fixed. Most likely, a new version of the standards will be required. OBD-II was mandated by the US government for new cars sold on the American market. Similarly, they could mandate the use of a secure successor to OBD-II and CAN.

17.6.2 Medical Implant Interface Security

Some medical implants have remote management features. This allows the physician or technician to interact with the device after it has been implanted, to test, configure, and tune the device. The noninvasive nature of these interactions is important for the patient's health and comfort. Some implant products use wireless radio frequency communication to link the programming console with the implanted device. As always, if implemented naively, radio links are subject to sniffing, unauthorized access, and jamming attacks. In the context of medical implants, attacks against the management interface could have very serious consequences. Unfortunately, many implants that have already been deployed are in fact vulnerable to remote attacks on their management interface.

Halperin [18] performed a security analysis of a commercial implantable cardioverter defibrillator. They reverse-engineered the communication protocol used by the management interface. The authors were able to sniff private information and to control the implanted device (potentially inducing fibrillation in the patient) and to cause it to exhaust its battery at an accelerated rate. This first phase of their work focused on assessing the level of security present. What they found was that the only obstacles to attack were security by obscurity, which they circumvented by capturing, analyzing, and emulating the link protocol. The second phase of their work was on suggesting practical measures for improving security. They offer three countermeasures against the attacks they discovered. One countermeasure is simply for the implant to make noise when communication is taking place. This alerts the patient and reduces the likelihood of an undetected attack. The other countermeasures focus on authentication and key exchange within the power and space constraints of medical implants.

17.7 Recapitulation and Projection

We have examined the way security and testability interact in various areas. Internal (structural) testability of systems is often required, particularly as systems reach a certain complexity. Testability features that provide controllability and observability of the internals of systems are major security risks if not designed with security in mind. Attackers have exploited unsecured test interfaces in the past and it is likely that this will continue. There is a significant and growing literature on how to provide testability without security vulnerability. The defenses that already exist in the literature are relevant to their intended target system. To some extent, the existing solutions can be adapted to new secure new testing scenarios as they emerge. Some testing scenarios will not be a good fit for existing solutions, and will require inventive minds to create new approaches for enabling internal testing without exploitation.

References

1. Sipser M (2006) Introduction to the Theory of Computation, 2nd edn. MIT, Cambridge
2. Rukhin A (2010) A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST, 2010
3. Bo Y, Kaijie W, Karri R (2006) Secure scan: a design-for-test architecture for crypto chips. IEEE Trans Comput Aided Des Integrated Circ Syst, 25(10): 2287–2293, doi:10.1109/TCAD.2005.862745
4. Hely D, Flottes M-L, Bancel F, Rouzeyre B, Berard N, Renovell M (2004) Scan design and secure chip [secure IC testing]. In: On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings of the 10th IEEE International, pp 219–224, 12–14 July 2004, doi:10.1109/OLT.2004.1319691
5. Lee J, Tehranipoor M, Plusquellic J (2006) A low-cost solution for protecting IPs against scan-based side-channel attacks. In: Proceedings of the 2006 IEEE VLSI Test Symposium. doi:10.1109/VTS.2006.7
6. Rajsuman R (2001) Design and test of large embedded memories: an overview. IEEE Des Test Comput 18(3): 16–27, doi: 10.1109/54.922800
7. Yang B (2009) Design and test for high speed cryptographic architectures. Doctoral Dissertation, Electrical and Computer Engineering Department, Polytechnic Institute of NYU
8. Dish Newbies JTAG Guide. http://www.geewizzy.com/geewizzysite/dish/jtagshtml/jtag.shtml.html. Accessed 19 July 2011
9. Free60 SMC Hack. http://www.free60.org/SMC_Hack. Accessed 19 July 2011
10. Rosenfeld K, Karri R (2010) Attacks and Defenses for JTAG. IEEE Des Test Comput 27(1): 36–47, doi:10.1109/MDT.2009.161
11. Laurent Sourgen (1993) Security locks for integrated circuit. US Patent 5264742
12. Buskey RF, Frosik BB (2006) Protected JTAG. In: Parallel Processing Workshops, International Conference on Parallel Processing Workshops, pp 405–414
13. Clark CJ, Ricchetti M (2004) A code-less BIST processor for embedded test and in-system configuration of boards and systems. In: Proceedings of the 2004 IEEE International Test Conference, pp 857–866, doi: 10.1109/TEST.2004.1387349
14. http://www.intellitech.com/pdf/FPGA-security-FPGA-bitstream-Built-in-Test.pdf. Accessed 19 July 2011
15. Iyengar V, Chakrabarty K, Marinissen EJ (2003) Efficient test access mechanism optimization for system-on-chip. IEEE Trans Comput Aided Des Integrated Circ Syst 22(5): 635–643, doi: 10.1109/TCAD.2003.810737
16. Rosenfeld K, Karri R (2011) Security-aware SoC test access mechanisms. In: Proceedings of the 2011 IEEE VLSI Test Symposium
17. Koscher K, Czeskis A, Roesner F, Patel S, Kohno T, Checkoway S, McCoy D, Kantor B, Anderson D, Shacham H, Savage S (2010) Experimental security analysis of a modern automobile. In: Proceedings of the 2010 IEEE Symposium on Security and Privacy, pp 447–462, doi:10.1109/SP.2010.34
18. Halperin D, Clark SS, Kevin F (2008) Pacemakers and implantable cardiac defibrillators: software radio attacks and zero-power defenses. In: Proceedings of the 2008 IEEE Symposium on Security and Privacy

Chapter 18
Protecting IPs Against Scan-Based Side-Channel Attacks

Mohammad Tehranipoor and Jeremy Lee

The need for on-chip security has been on the rise with the proliferation of cryptochips and other applications that contain intellectual property that must be protected. In order to test these chips, scan-based testing has been commonly used due to the ease of application and high coverage. However, once in the field, the test ports become a liability due to the amount of controllability and observability scan-based testing provides. This chapter presents a low-cost secure scan solution that allows the ease of testing using a scan while maintaining a high level of security that will protect the on-chip IP. The proposed solution authorizes users through the use of a test key that is integrated directly into the test pattern and will prevent unauthorized users from correctly analyzing the responses from the scan chain. The area overhead of the proposed solution is negligible, has no impact on performance, and adds several layers of security on top of the scan chain without modifying the standard test interface.

18.1 Introduction

Controllability and observability of circuits under test (CUT) have significantly reduced as chip design complexity continues to increase. This problem greatly affects test engineers' ability to create fast and reliable tests using the primary input and primary output alone, which negatively affect time to market and number of defective parts delivered to the customer. Design-for-testability (DFT) addresses this issue by considering manufacturing test during design.

M. Tehranipoor (✉)
UCONN Electrical and Computer Engineering, University of Connecticut, 371 Fairfield Way, Unit 2157 Storrs, CT 06269-2157, USA
e-mail: tehrani@engr.uconn.edu

J. Lee
DFT Engineer, Texas Instruments, Dallas, TX, USA
e-mail: jeremy-lee@ti.com

M. Tehranipoor and C. Wang (eds.), *Introduction to Hardware Security and Trust*,
DOI 10.1007/978-1-4419-8080-9_18, © Springer Science+Business Media, LLC 2012

Scan-based DFT is one commonly practiced technique that greatly improves controllability and observability by modifying flip-flops into a long chain essentially creating a shift register. This allows test engineers the ability to treat each flip-flop in the scan chain as a controllable input and observable output.

Unfortunately, the same properties that scan improves upon for testing also creates a severe security hazard. Scan chains become an easily exploitable side-channel for cryptanalysis [1, 2] due to the ability to place the CUT into any state (controllability) and stop the chip in any intermediate state for analysis (observability). Because of the widespread use of scan-based testing, this side-channel has become a major concern in industry [3, 4].

These concerns add to an already mounting problem of hardware security. Other side-channel attacks such as differential power analysis [5], timing analysis [6], and fault-injection [7, 8] attacks have also been shown to be potentially serious sources of security failures. Tamper-resistant designs [9, 10] propose to mend these leaks, however scan chains are necessary to expose any defects in the chip that may exist. While disabling the scan chains after manufacturing test, e.g., by blowing fuses, has become a common practice for applications such as smartcards [11], there are also applications that require in-field testing, which make it impossible to deliberately destroy access to the test ports.

As scan-based attacks require minimally invasive techniques to perform the attack [11], this exploit becomes accessible to attackers with a wide-variety of knowledge and resources [12]. A security measure that will prevent scan-based attacks requires the ability to scale depending on the desired level of security based on the application. It must also minimally affect the ability of a test engineer to efficiently test the chip after fabrication.

The latter point must be strongly considered as the entire purpose of using scan is for testing. Although the goals of security and testing appear to be contradictory, the security of the chip can easily fail if it is not properly tested.

18.1.1 Prior Work

A traditional method of scan security has been to place polysilicon fuses near the pin connections and blow them after manufacturing testing, but it has been shown that reconnecting these fuses can be performed with minimally invasive techniques [11]. An alternative option is to completely cut-off the test interface with a wafer saw [9]. However, both options eliminate any possibility for in-field testing. Many have gotten around the concern by using BIST to test the entire design [13] or a hybrid method that uses BIST on sensitive portions and scan on the remainder [14]. Although a viable solution, the fault coverage still does not reach the levels of scan and ATPG.

There has recently been an increased focus to secure scan designs without completely destroying access to the test interface. To prevent finding secret key information, a simple solution proposed is to use a second register, called the mirror

key register (MKR), to prevent any critical data from entering the scan chain when in test mode [2, 15]. Although effective, application of this solution is limited because this approach can only protect information and not the actual IP the chip may contain. Scan scrambling was presented in [16], which divides the scan chain into subchains and uses logic and a random number generator to randomly reconnect the subchains together again and internally scramble data. Using this strategy with a large number of subchains yield a high complexity, but also begins to create significant area overhead and requires modification to a standard test interface. Another technique checks a "golden signature" after a well defined consistent system reset to ensure secure switching between system and test modes [17]. A lock and key technique was developed in [12], which also divides the scan chain into subchains, but rather than connecting all of the subchains back together, all the subchains are independent of each other. By using a randomly seeded LFSR, one subchain at a time would be randomly enabled to scan-in and scan-out while all other subchains are left unaffected. This technique also has the disadvantage of scaling poorly for a very large number of subchains. A stream ciphering scheme based on a cyclic redundancy check message authorization code was presented in [18] to encode the contents of the scan chain, but this technique more than doubles the test time since concurrent scan-in of the next pattern and scan-out of the ciphered pattern is not possible. There have also been techniques to ensure that the scan-enable signal is properly controlled [17, 19]. Both techniques heavily rely on an additional security controller to monitor the inserted logic and create a consistent reset when it detects a security violation.

18.2 Scan-Based Attack Categorization

Developing a secure scan design is dependent on targeting both the type of attacker [12] and how they can potentially make the attack. The authors categorize the scan-based attacks into two types: *scan-based observability* and *scan-based controllability/observability* attacks. Each requires that a hacker has access to the test control (TC) pin. The type of attack depends on how a hacker decides to apply stimuli. The proposed low-cost secure scan design removes the hacker's ability to correlate test response data by creating a random response when an unauthorized user attempts access, which prevents hackers from exploiting the two attacks described in the remainder of this section.

18.2.1 Scan-Based Observability Attack

A *scan-based observability* attack relies on a hacker's 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 18.1a diagrams the necessary steps to perform a scan-based observability attack.

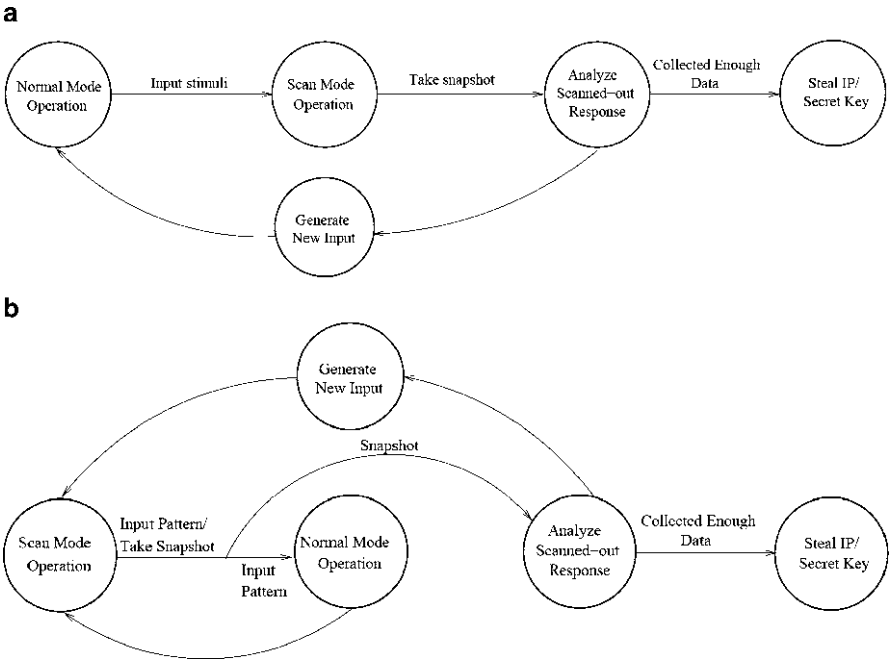


Fig. 18.1 Summary of the steps necessary to perform successful scan-based attacks: (**a**) Scan-based observability attack and (**b**) Scan-based controllability/observability attack

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 (or IP) for a particularly innovative chip.

18.2.2 Scan-Based Controllability/Observability Attack

Scan-based controllability/observability attacks take a different approach to applying stimuli to the CUT, which is shown in Fig. 18.1b. 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-in to 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 [7, 8], 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.

18.3 Low-Cost Secure Scan

The authors propose a low-cost secure scan (LCSS) solution that provides flexibility in design and integrates smoothly with current scan insertion flow. After performing scan insertion, the testing process is minimally affected while not affecting functional mode operation. As shown in Fig. 18.2, this is accomplished by inserting dummy flip-flops into the scan chains and including the key into the test patterns with respect to the position of the dummy flip-flops in the chains. By doing so, it verifies that all vectors scanned-in come from an authorized user and the correct response can be safely scanned-out after functional mode operation. If the correct key is not integrated into the vector, an unpredictable response will be scanned-out

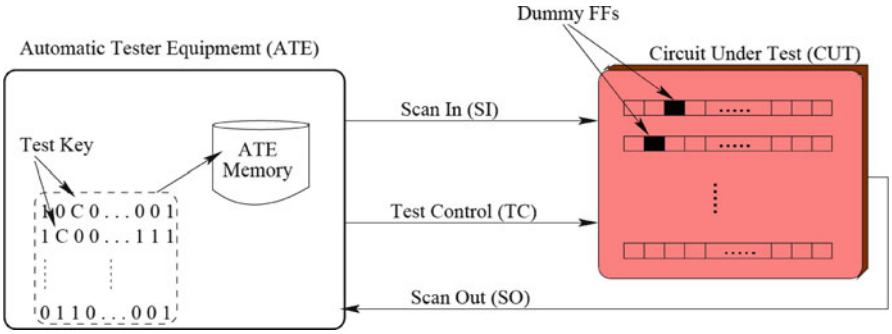


Fig. 18.2 Sample test pattern stored in ATE with test key bits located in pattern with respect to the location of the dummy flip-flops in the CUT

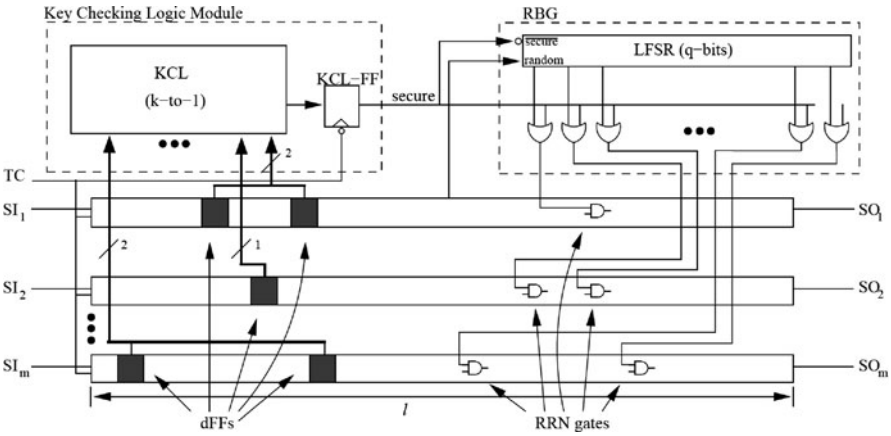


Fig. 18.3 General architecture for an LCSS design [27]

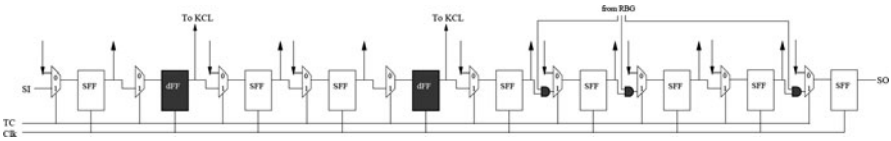


Fig. 18.4 An example of the low-cost secure scan chain with integrated dummy flip-flops and random response network

making analysis very difficult for an attack. By using an unpredictable response, a hacker would not be able to immediately realize that their intrusion has been detected as could be done if the CUT were to immediately reset [17]. Figure 18.2 demonstrates the interface between the tester and the CUT.

The state of the scan chain is dependent on the test key that is integrated into all test vectors. There are two possible states for the chain: *secure* and *insecure*. By integrating the key, all vectors scanned-in can be verified to be from a trustworthy source (secure). Without a correct key integrated into the test vector, when scanning in a new vector and out the response, the response will be randomly altered to prevent reverse engineering of critical data that is being stored in registers (insecure). By altering the response scanned out of the chain, both the scan-based observability and scan-based controllability/observability attacks are prevented as any attempt to correlate the responses from various inputs will prove unsuccessful due to the random altering of the data.

The LCSS architecture is shown in Fig. 18.3 with a more detailed look at a secure scan chain in Fig. 18.4. In order to use the same key for every test vector, dummy flip-flops (dFFs) are inserted and used as test key registers. Each dFF is designed similarly to a scan cell except that there is no connection to a combinational block. The number of dFFs included in the scan chain depends on the level of security the designer would like to include since the number of dFFs determines the size of the

test key. When implementing LCSS for multiple-scan design, the test key is inserted into the scan chain before it is broken into multiple scan chains. This ensures that the key can be randomly distributed throughout the many scan chains without needing to have a constant number of key registers in each chain.

All dFFs are concurrently checked by the Key Checking Logic (KCL), which is made of a block of combinational logic. The k-input block, where k is the total number of dFFs in the scan design (length of the test key), has a fan-out of a single FF (KCL-FF), which is negative edge sensitive to TC. As the CUT switches from test mode to functional mode (TC falls), the FF clocks in the output of the key checking logic. The KCL-FF is then used to inform the remainder of the secure design of the current secure or insecure state of the vector in the scan chain.

There is potential for the KCL to be implemented using a variety of more secure options since the KCL will essentially be the same for all chips fabricated using the same design. One such option would be to implement a postfabrication configurable KCL. This KCL implementation would allow different test keys from chip to chip and would prevent the possibility of determining a single key to compromise all chips of the same design. However, as each of the devices have a different test key, essentially a new test pattern set will need to be generated for each individual chip. This would either create a significant increase in test time or require a new secure test protocol that would need the tester to insert the test key into the pattern dynamically.

The third component of the LCSS solution ensures the random response in the scan chain when the test key fails to be verified by the KCL. The output of the KCL-FF fans out to an array of q 2-input OR gates. The second input of each OR gate comes from a q-bit LFSR that has been randomly seeded using one of a variety of options including, but not limited to, the value already present at reset, a *random* signal from a FF in the scan chain as shown in Fig. 18.3, or a *random* signal from the output of a separate random number generator [20]. The former option provides the least amount of overhead, but potentially the least secure, while the latter has the most security, but also the most overhead. By also using a *secure* signal to the LFSR, also shown in Fig. 18.3, the LFSR seed can continually be changed by an additional random source. Together, the LFSR and OR gate array make up the Random Bit Generator (RBG). The RBG output is used as input to the Random Response Network (RRN) that have also been inserted into the scan chain. The RRN can be made of both AND and OR gates to equalize the random transitions and prevent the random response from being all zeros or all ones. The optimal choice for randomness would be to use XOR gates, but as XORs add more delay, our design choice was to use AND and OR gates. As the dFFs are used to check the test key, dFFs must be placed before any gates of the RRN in the scan chain as shown in Fig. 18.4. If this property is not held, any key information that is trying to pass a gate of the RRN in the scan chain may potentially get altered either preventing the test key from ever being verified or even randomly changing a value to the correct key.

Normal mode operation of the CUT is unaffected by the addition of the LCSS design since the dFFs are only used for testing and securing purposes and are not connected to the original design.

18.3.1 LCSS Test Flow

Our low-cost secure scan design deviates very little from current scan test flow. As the security of the scan chain is ensured by integrating a test key into the test vectors themselves, no additional pins are necessary to use LCSS.

After a system reset and TC has been enabled for the first time, the secure scan design begins in an insecure state causing any data in the scan chain to be modified as it passes through each RRN gate in the chain. In order to begin the testing process, the secure scan chain(s) must be initialized with the test key in order to set the output of the KCL-FF to 1. Only the test key is required to be in this initialization vector since any other data beyond the first RRN gate will most likely be modified. During this time the KCL will constantly check the dFFs for a correct key. After the initialization vector has been scanned-in, the CUT must be switched to functional mode for one clock in order to allow the KCL-FF to capture the result from the KCL. If the KCL verifies the key stored in the dFFs, the KCL-FF is set to 1 and propagates the signal to the RRN, which becomes transparent for the next round of testing allowing the new vector to be scanned-in without alteration.

Testing can continue as normal once the initialization process has been finished. However, the chain can return to insecure mode at any time during scan testing if the correct test key is not present in all subsequent test vectors, requiring the k-bit key to be in all test patterns. Should that occur, the RRN will again affect the response in the scan chain and the initialization process must again be performed in order to resume a predictable testing process.

18.4 Automated LCSS Insertion Flow

Including LCSS scan minimally changes the current scan insertion flow. The authors were able to implement the LCSS insertion flow using Synopsys DFT Compiler [21] and small add-ons functions developed in C. The add-ons performed tasks like automating the KCL creation depending on the desired number of dFFs, creating a desired LFSR size for the RBG, performing RRN randomization and inserting RRN gates. In order to allow smooth RRN insertion, temporary placeholder FFs were instantiated in the scan chain that would later be replaced after the scan chain had been stitched together. The entire process became automated by a single script once the add-on functions were created. The LCSS insertion flow is summarized in the following steps.

18.4.1 Low-Cost Secure Scan Insertion Flow

1. Define # of dFFs (size of key and KCL).
2. Define KCL key (random or user defined).
3. Define # of LFSR bits.
4. Define # of RRN gates.
5. Load and Compile KCL and RBG.
6. Load and Compile Targeted Design (TD).
7. Set Current Design to TD.
8. Initialize KCL and RBG in TD.
9. Initialize dFFs in TD.
   a. Connect CLK of TD to dFFs.
   b. Connect Q of all dFFs to respective KCL port.
10. Initialize RRN placeholder FFs.
    - Conect CLK of TD to all placeholder FFs.
    - Conect D of all placeholder FFs to respective RBG port.
11. Reorder scan chain placing dFFs before all RRN placeholder FFs.
12. Perform scan insertion.
13. Replace RRN placeholder FFs with actual RRN gates.
14. Load and Compile new netlist with Low-Cost Secure Scan included in TD.

18.5 Analysis and Results

Overall, the proposed low-cost solution has little affect on important overhead aspects of chip design but significantly increases the security of the scan chain and still allows the testability of scan design. Analysis of the LCSS method and implementation results are explored in-depth later. When implementing the LCSS method, the authors aimed to affect area, test time, and performance as little as possible to maintain a low-cost strategy.

18.5.1 Overhead

18.5.1.1 Area

The size of the KCL for our designs remains fairly small, but is entirely dependent upon the size of the key and the value of key. The logic to perform the key verification at most will always be $(k - 1)$ gates plus the number of inverters. The number of inverters in the KCL is dependent on the value of the key and will never

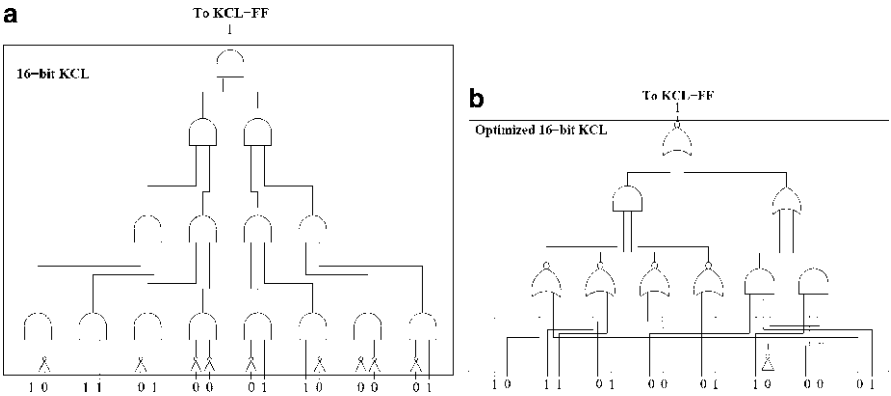


Fig. 18.5 (**a**)A 16-bit KCL designed with 2-input gates. The value of the valid key is determined by the inverters at the input. (**b**)The same 16-bit KCL after library mapping and optimization

Table 18.1 Area overhead of conventional scan and proposed LCSS on ISCAS'89 benchmarks

Benchmark name	#of gates	#of FFs	Size of bench (μm^2)	Bench w/scan (μm^2)	Bench w/LCSS (μm^2)	Scan/bench overhd (%)	LCSS/bench overhd (%)	LCSS – scan overhd (%)
s1423	323	74	22,639	33,499	35,926	39.13	58.70	19.57
s1488	381	6	13,062	13,846	18,274	6.00	39.90	33.90
s9234	507	145	40,526	58,293	62,721	43.84	54.77	10.93
s13207	1,328	625	146,105	223,755	228,182	53.15	56.18	3.03
s15850	1,824	513	144,1244	206,814	211,241	43.50	46.57	3.07
s35932	3,998	1,728	441,667	649,416	653,844	47.03	48.04	*1.05*
s38417	4,715	1,564	416,989	605,283	609,710	45.16	46.22	*1.06*
s38584	5,747	1,274	400,429	557,344	561,772	39.19	40.20	*1.10*

be greater than k. As k inverters would translate to a key of all zeroes or all ones depending on the KCL implementation, an actual key should have approximately the same number of ones and zeroes ($k/2$inverters). So, the size of the entire KCL will be $\frac{3k}{2} - 1$ gates when 2-input logic is used. An example of a 16-bit KCL is shown in Fig. 18.5a. Using 2-input logic, the key, x'B461', can be decoded using 24 gates. However, this logic can be optimized during synthesis. In Fig. 18.5b, the logic has been mapped and optimized to a standard cell library using the same key in Fig. 18.5a but only requires ten gates.

Table 18.1 shows the implementation results of LCSS and conventional scan on several ISCAS'89 benchmarks [22] using the Cadence Generic Standard Cell Library [23]. When including LCSS, the authors used a test key size of 10-bits, a 4-bit LFSR, and 10 RRN gates. The second and third columns of the table show the number of gates and FFs in the design after synthesis, respectively. Column 4 shows the total area of benchmarks before scan insertion in μm^2, which includes the area of both combinational logic and FFs. Columns 5 and 6 list the sizes of the

Table 18.2 Area overhead of LCSS on ISCAS'89 benchmarks without additional dummy flip-flops

Benchmark name	LCSS w/o dFFs/bench overhd (%)	LCSS w/o dFFs – scan overhd (%)
s1423	46.22	7.09
s1488	18.29	12.29
s9234	47.80	3.96
s13207	54.24	1.09
s15850	44.61	1.11
s35932	47.40	0.37
s38417	45.54	0.38
s38584	39.59	0.40

benchmarks after conventional scan and LCSS insertion, respectively. In columns 7 and 8, the ratio of the overhead created by conventional scan and LCSS is shown.

As the size of the benchmarks are small the area overhead of the LCSS design is fairly significant over the size of the benchmark, but this is also true for conventional scan. If one considers the overhead of our technique over conventional scan, which is shown in column 9, the area overhead is quite minimal for the larger designs. LCSS does pose a significant area impact when implemented in a small design, but as the benchmarks become larger, the overhead becomes less significant since the LCSS size is fixed and independent of the design. This is expected to further decrease for modern designs due to their immense size.

The LCSS sizes reported (Table 18.1, column six) include dFFs that have been added to the ISCAS'89 benchmark scan chains. However, as designs grow larger, usually dummy flip-flops are inserted into scan chains for testing purposes [24], e.g., event up scan chains, increase delay fault coverage, etc. It may be possible to use these dFFs as key registers. By using the already present dFFs, the impact of LCSS can be reduced further. Table 18.2 shows the overhead if dFFs did not have to be added. By considering this, the authors reduce the overhead of secure scan over conventional scan by more than half as shown by comparing columns seven and nine.

18.5.1.2 Test Time

For modern designs with a very large number of scan cells, including additional test key registers will not affect scan test time by a significant amount. The total LCSS test time (T_{LCSS}) is shown in (18.2) which is derived from (18.1), the test time for conventional multiple-scan design (T_{conv}).

$$T_{\text{conv}} = (n_{\text{comb}} + 2) \cdot \left\lceil \frac{n_{\text{ff}}}{m} \right\rceil + n_{\text{comb}} + 3 \tag{18.1}$$

$$T_{\text{LCSS}} = (n_{\text{comb}} + 3) \cdot \left\lceil \frac{n_{\text{ff}} + k}{m} \right\rceil + n_{\text{comb}} + 4. \tag{18.2}$$

Table 18.3 Percentage of test time increase (T_{inc}) of LCSS on ISCAS'89 benchmarks

Benchmark name	# of patterns	# of chains	# of FFs	LCSS $k = 10$-bits	LCSS $k = 40$-bits	LCSS $k = 80$-bits
s13207	100	10	625	2.5	7.3	13.6
s15850	90	10	513	3.0	8.7	16.3
s35932	16	10	1,728	6.1	7.9	10.4
s38417	90	10	1,564	1.7	3.6	6.2
s38584	118	10	1,275	1.6	4.0	7.1

The variable n_{ff} is the total number of flip-flops in the design without including dFFs, k is the number of dFFs (key size), m is total number of scan chains, and n_{comb} is the number of combinational vectors. Equation (18.2) accounts for the initialization process that places the scan chain into secure mode, the chain test, and all test application sequences. The increase to three scan procedures is due to the additional initialization sequence before the chain test starts. The increase by k accounts for the addition of the key directly inserted into the test pattern.

The total increase due to LCSS becomes the difference between (18.1) and (18.2). This results in (18.3).

$$\Delta T = T_{LCSS} - T_{conv} = \left\lceil \frac{k(n_{comb} + 3) + n_{nff}}{m} \right\rceil. \tag{18.3}$$

T_{LCSS} could be reduced further, subsequently reducing ΔT, if the initialization process and chain test occurred concurrently, but this is dependent upon the location of the dFFs. As modern designs continue to have more scan cells and k remains fairly small in comparison, the test time increase due to LCSS will become less significant.

The actual test time increase percentages (T_{inc}) are included in Table 18.3 for conventional scan and secure scan using a key size of 10, 40, and 80 bits on several ISCAS'89 benchmarks. The number of patterns in the benchmark test set is listed in Column 2. Each design was implemented using multiple scan design with $m = 10$ scan chains, which is shown in Column 3. In column 4, the number of FFs in the design before LCSS insertion is shown. Columns 5–7 show the values of T_{inc} for key sizes of 10, 40, and 80, respectively. The results show that T_{inc} can be negligible for large designs when k is much smaller than n_{ff}. As the size of the test key increases, the test time will incur some overhead while significantly increasing chip security.

From Table 18.3, a trend can be observed to indicate the percentage of test time increase (T_{inc}) by LCSS over conventional scan is directly proportional to the size of the key compared to the number of flip-flops in the design. So, T_{inc} can be summarized as shown in (18.4).

As modern designs increase in complexity, the number of FFs will become significantly greater than the number of scan chains ($n_{ff} \gg m$) and the number of test patterns to achieve a fault coverage will continue to increase ($n_{comb} \gg 0$).

So, as these two values grow larger, the ratio will become more dependent upon the number of dFFs inserted and the number of FFs originally in the design, as approximated by (18.5).

$$T_{\text{inc}} = \frac{\Delta T}{T_{\text{conv}}} \times 100 = \frac{k(n_{\text{comb}} + 3) + n_{\text{ff}}}{n_{\text{ff}}(n_{\text{comb}} + 2) + m(n_{\text{comb}} + 4)} \times 100 \tag{18.4}$$

$$T_{\text{inc}} \approx \frac{k}{n_{\text{ff}}} \times 100. \tag{18.5}$$

As multiple-scan design often do not have chains of equal length, the test time is dependent upon the length of the longest chain. If the number of dFFs added does not increase the length of the longest scan chain, only the initialization pattern would create test time overhead. However, the addition of the dFFs would more realistically cause the length of the longest scan chain to increase.

18.5.1.3 Performance

Performance of a design is not affected by LCSS any more than it would by conventional scan test. The speed of the design remains only affected by the inclusion of the scan chain itself and not further hindered by the inclusion of dummy flip-flops or RRN gates between the scan cells.

Power consumption during normal mode operation is affected very little. However, consumption during test mode only becomes a concern should the designer use a very large key with many RRN gates as this could considerably increase switching activity during scan operation. But as long as the key length and number of RRN gates remain small in comparison to the total number of scan cells, additional power consumption will be minimal.

18.5.2 Affect on Security and Testability

While maintaining a low-cost overhead, our secure scan solution is able to greatly increase the security of the scan chain while still providing the high controllability and observability conventional scan usually provides.

18.5.2.1 Security

If a hacker attempts to perform an attack, any vector scanned-in or any critical data scanned-out will be randomly altered by the RRN throughout the chain. With each additional RRN gate used in a scan chain, the probability of determining the original value drops.

Table 18.4 Percentage of test time increase of LCSS on ISCAS'89 benchmarks

Benchmark name	LCSS $k = 10$-bits	LCSS $k = 40$-bits	LCSS $k = 80$-bits
s35932	$2^{19}\binom{1159}{10}\binom{580}{10}$	$2^{49}\binom{1179}{40}\binom{590}{10}$	$2^{89}\binom{1206}{80}\binom{603}{10}$
s38584	$2^{19}\binom{958}{10}\binom{479}{10}$	$2^{49}\binom{978}{40}\binom{489}{10}$	$2^{89}\binom{1004}{80}\binom{502}{10}$

If the attacker would like to perform a scan-based side-channel attack without being hindered by the security measures in place, the hacker must bypass seven security elements:

1. The test key.
2. The location of each test key register in the scan chain (or test vector).
3. The number of RRN gates in each scan chain.
4. The type of RRN gate being used.
5. The random LFSR seed.
6. The LFSR polynomial.
7. In the case of multiple-scan designs, the hacker must also determine the configuration of the decoder/encoder on the input/output of the scan chain.

The last five security elements are very difficult to determine without disassembling the chip and using expensive equipment.

Each level of security adds to the complexity of determining the contents of the scan chain. The k-bit key alone provides 2^k potential combinations. This implementation placed all dFFs in the first 2/3of each scan chain. Choosing k-bits out of the first $2n/3$ scan cells in the chain creates an additional level of complexity to the security strategy used in low-cost secure scan, where n is the total scan length including dFFs. There are $\binom{\frac{2n}{3}}{k}$ possible key location combinations. The authors placed the RRN gates in the latter 1/3 of each scan chain creating a complexity of $\binom{\frac{n}{3}}{r}$ where r is the total number of RRN gates used in the design. Determining the type of RRN gate doubles the combinations. Also, the LFSR seed and LFSR polynomial each provide 2^q potential combinations. Combining all of these components together result in

$$2^{k+2q+1} \cdot \binom{\frac{2n}{3}}{k} \cdot \binom{\frac{n}{3}}{r} \tag{18.6}$$

total combinations. When considering multiple-scan chain decoding/encoding, (18.6) shows the minimum number of possible combinations. The authors have summarized the complexity of low-cost secure scan in Table 18.4 on two ISCAS'89 benchmarks using various key sizes. For simplicity, the LFSR size and number of

RRN gates has been fixed at 4-bits ($q = 4$) and 10 gates ($r = 10$), respectively. Table 18.4 shows that as the key increases and as the number of scan cells increase, the complexity dramatically increases.

18.5.2.2 Scan-Based Observability Attack Prevention

With the LCSS design in place, if the hacker attempts to run a system in functional mode and immediately takes a snapshot with access to the test control pin, any response on SO will result in randomized data due the lack of a correct test key and the RRN integrated into the chain. A hacker is given a false hope of a successful attack since a response is still scanned-out, but attempting to correlate the responses would fail.

18.5.2.3 Scan-Based Controllability/Observability Attack Prevention

If the hacker attempts to scan-in a vector, unless the hacker knows the test key, it will be randomly altered by the RRN essentially creating a new vector unknown to the hacker. When the hacker cycles TC, similar to the scan-based observability attack, the hacker will not be able to easily correlate a particular portion of the response to any critical registers needed to perform an effective attack.

This security design is also effective against a fault-injection attack. The hacker may choose a vector at random to potentially force the CUT into a fault condition. However, due to the random stimuli alterations of the integrated RRN, the hacker will have difficulty reproducing the same fault with the same vector.

18.5.2.4 DFT

In a multiple-scan design the test key can be distributed throughout all of the scan chains given that all of the chains are equal in length and the dFFs are placed before the RRN gates. Given that there may be hundreds of chains in the design, some chains may have one or two key registers while other chains have none at all. Still, only one KCL is necessary to check the key no matter how many chains are in the design. Placement of the RRN gates only occurs after the scan chain has been divided into multiple chains to prevent the potential of some chains being left unsecured by the RRN. Each of the scan chains can use all q-bits of the LFSR to randomize the data or use a different subset combination. Due to the flexibility of LCSS, there are a wide variety of customizable arrangements for the dFFs and RRN in multiple-scan designs while maintaining a conventional multiple-scan design interface.

Enhanced-scan techniques like test compression can still be used without any alterations since no changes are made to the interface of the scan chains. Even launch-off shift (LOS) [25] and launch-off capture (LOC) [26] can be implemented

as normal. It may be necessary to add an additional dead cycle between the capture and the next initialization cycles for LOS and an additional dead cycle between initialization and launch cycles for LOC, but the end result still successfully captured.

Including chips with LCSS in SoC designs or chips that need in-field testing do not need any special cases since the scan interface has not been changed beyond that of conventional scan design. By this level of abstraction, no additional pins are required by our LCSS design and map to a JTAG interface in the same fashion as conventional scan.

18.6 Summary

This chapter presented a low-overhead secure scan solution that can be used to prevent scan-based side-channel attacks. By integrating the proposed technique into the scan insertion flow, the complexity of determining secret information significantly increases since a hacker must bypass up to seven levels of security until being able to access the scan chain. The hacker also may waste valuable time performing an attack without realizing a security strategy is being used since a random response is still output by the scan chain as opposed to resetting the chip or setting all values to zero/one. This strategy is flexible and can be extended to a variety of security levels depending on the needs and preference of the designer. The goal of adding security as opposed to removing testability allows in-field testing that could later prove invaluable depending on the application.

References

1. Yang B, Wu K, Karri R (2004) Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: IEEE International Test Conference (ITC), pp 339–344
2. Yang B, Wu K, Karri R (2005) Secure scan: a design-for-test architecture for crypto chips. In: 42nd Annual Conference on Design Automation, June 2005, pp 135–140
3. Goering R (2004) Scan design called portal for hackers. http://www.eetimes.com/news/design/showArticle-.jhtml?articleID=51200154. Accessed Oct 2004
4. Scheiber S (2005) The Best-Laid Boards. http://www.reedelectronics.com/tmworld/article/CA513261.html. Accessed April 2005
5. Kocher P, Jaffe J, Jun B (1999) Differential power analysis. In: 19th Annual International Cryptology Conference on Advances in Cryptology, pp 388–397
6. Kocher PC (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: 16th Annual International Cryptology Conference on Advances in Cryptology, pp 104–113
7. Boneh D, Demillo RA, Lipton RJ (1997) On the importance of checking cryptographic protocols for faults. In: Eurocrypt'97, pp 37–51
8. Biham E, Shamir A (1997) Differential fault analysis of secret key cryptosystems. In: 17th Annual International Crytology Conference on Advances in Cryptology, pp 513–527

9. Kömmerling O, Kuhn MG (1999) Design principles for tamper-resistant smartcard processors. In: USENIX Workshop on Smartcard Technology, pp 9–20
10. Renaudin M, Bouesse F, Proust P, Tual J, Sourgen L, Germain F (2004) High security smartcards. In: Design, Automation and Test in Europe Conference
11. Skorobogatov SP (2005) Semi-invasive attacks – a new approach to hardware security analysis. Ph.D. dissertation, University of Cambridge
12. Lee J, Tehranipoor M, Patel C, Plusquellic J (2005) Securing scan design using lock & key technique. In: International Symposium on Defect and Fault Tolerance in VLSI Systems
13. 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 Design Test Comput 6–17
14. 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)
15. Yang B, Wu K, Karri R (2006) Secure scan: a design-for-test architecture for crypto chips. IEEE Trans Comput Aided Design Integr Circuits Syst 25(10)
16. H'ely D, Flottes M-L, Bancel F, Rouzeyre B, B'erard N, Renovell M (2004) Scan design and secure chip. In: 10th IEEE International On-Line Testing Symposium
17. H'ely D, Bancel F, Flottes M-L, Rouzeyre B (2005) Test control for secure scan designs. In: European Test Symposium, pp 190–195
18. Gomulkiewicz M, Nikodem M, Tomczak T (2006) Low-cost and universal secure scan: a design-for-test architecture for crypto chips. In: International Conference on Dependability of Computer Systems, May 2006, pp 282–288
19. D H'ely, Flottes M-L, Bancel F, Rouzeyre B (2006) Secure scan techniques: a comparison. In: IEEE International On-Line Testing Symposium, July 2006, pp 6–11
20. Jun B, Kocher P (1999) The intel random number generator. Cryptography Research, Inc., Technical Report
21. Synopsys Toolset Documentation (2006) User Manual for Synopsys Toolset Version 2006.06, Synopsys Inc.
22. ISCAS'89 Benchmarks (1989) http://www.fm.vslib.cz/kes/asic/iscas
23. GSCLib 3.0, "0.18 μm standard cell GSCLib library version 3.0," Cadence, Inc. (2005) http://crete.cadence.com
24. Bushnell ML, Agrawal VD (2000) Essentials of Electronic Testing. Kluwer Academic Publishers, Dordrecht (Hingham, MA)
25. Savir J (1992) Skewed-load transition test: Part I, calculus. In: International Test Conference, pp 705–713
26. Savir J, Patil S (1994) On broad-side delay test. In: VLSI Test Symposium, pp 284–290
27. Lee J, Tehranipoor M, Plusquellic J (2006) A low-cost solution for protecting IPs against scan-based side-channel attacks. In: VLSI Test Symposium, May 2006