12-23-2011

Exploitation of Unintentional Information Leakage from Integrated Circuits

William E. Cobb

Exploitation of Unintentional Information Leakage From Integrated Circuits

DISSERTATION

William E. Cobb, Major, USAF

AFIT/DEE/ENG/11-06

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright Patterson Air Force Base, Ohio

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/DEE/ENG/11-06

EXPLOITATION OF UNINTENTIONAL INFORMATION LEAKAGE FROM INTEGRATED CIRCUITS

DISSERTATION

Presented to the Faculty of the
Graduate School of Engineering and Management
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

William E. Cobb, B.S.C.S., M.S.E.E.
Major, USAF

December 2011

AFIT/DEE/ENG/11-06

EXPLOITATION OF THE UNINTENTIONAL INFORMATION LEAKAGE OF INTEGRATED CIRCUITS

William E. Cobb, B.S.C.S., M.S.E.E.

Major, USAF

Approved:

Rusty O. Baldwin, PhD (Chairman)	7 Sep 11 Date
Yong C. Kim, PhD (Member)	7 Sep 11 Date
Mark E. Oxley, PhD (Member)	7 Sep 2011 Date
Michael A. Temple, PhD (Member)	7 Sep 2011 Date

Accepted:

M. U. THOMAS Dean, Graduate School of Engineering and Management	7 Oct 2011 Date

Acknowledgements

First, and most importantly, thanks to my family—my wife, children, and parents—for their love and support throughout this endeavor. Knowing they believed in me kept me going through the most difficult of times.

I would also like to express my sincere gratitude to my advisor, Dr. Rusty Baldwin, for his guidance and encouragement thoughout this process. Without his support I never would have had the courage to embark on my journey into the unknown, nor the resources to succeed. Additionally, I owe much to my committee members, Dr. Michael Temple, Dr. Yong Kim, and Dr. Mark Oxley, for their always fruitful advice and suggestions. Finally, I would like to thank everyone on the CCR center staff, and in particular Mr. Eric Laspe and Dr. Timothy Lacey, for the many aspects of their support that have helped make my journey here at AFIT a successful one.

William E. Cobb

Table of Contents

Page

Acknowledgements iii

List of Figures ix

List of Tables xi

Abstract xii

1. Introduction 1

1.1 Problem Addressed 1

1.2 Security Implications of Information Leakage 3

1.3 Research Objective 4

1.4 Dissertation Organization 5

2. Background 7

2.1 Overview 7

2.2 Emission Sources 8

2.2.1 Variations in Computation Time 8

2.2.2 Variations in Power Consumption 9

2.2.3 Electromagnetic Emissions 11

2.2.4 Acoustic Emissions 14

2.3 Applications 14

2.3.1 SCA Attacks 15

2.3.2 Reverse Engineering 20

2.3.3 Covert Channel Engineering 21

2.3.4 Trojan Detection 22

2.4 Adversary Models 22
2.5 Techniques 24
2.5.1 Timing Analysis 25
2.5.2 Simple Analysis 26
2.5.3 Differential Analysis 30
2.5.4 Profiling Techniques 34
2.5.5 Advanced Techniques 36
2.6 Countermeasures 42
2.7 Practical Considerations 44
2.8 Summary 48

3. Intrinsic Physical Layer Authentication of Integrated Circuits . 49
3.1 Abstract 49
3.2 Introduction 49
3.3 Problem Definition 51
3.4 Notional Physical Layer Device Authentication System Design 52
3.5 Unintentional RF Emissions of ICs 55
3.6 Related Work 56
3.6.1 Physical Unclonable Functions (PUF) 56
3.6.2 RF Certificates of Authenticity (RF-COA) . . 57
3.6.3 RF-DNA Fingerprinting 57
3.7 RF-DNA Fingerprint Generation and Classification . . 58
3.7.1 RF-DNA Feature Extraction and Statistical Fingerprint Generation 58
3.7.2 Classifier Training 63
3.7.3 Fingerprint Classification 65
3.8 Experimental Methodology 68

Page

3.8.1 Experimental Setup 68
3.8.2 Signal Collection 69
3.8.3 K-Fold Cross Validation 71
3.8.4 Noise Simulation 72
3.9 Results 72
3.10 Conclusion 79
3.11 Supplementary Discussion 80
3.11.1 Experimental Setup 80

4. Leakage Mapping: A Systematic Methodology for Assessing the Side Channel Information Leakage of Cryptographic Implementations 82
4.1 Abstract 82
4.2 Introduction 82
4.3 Background 84
4.3.1 Side Channel Emissions of ICs 84
4.3.2 Information Leakage and Side Channel Analysis 85
4.3.3 Structure of Symmetric Block Ciphers 86
4.3.4 Advanced Encryption Standard (AES) 87
4.3.5 Modeling Leakage 92
4.3.6 Measures of Information Leakage 97
4.4 Leakage Mapping 99
4.5 Experimental Methodology 109
4.5.1 Implementation A 109
4.5.2 Implementation B 110
4.5.3 Data Alignment 111
4.5.4 Resource Requirements 112
4.6 Results 112

4.6.1 Intermediate Cipher Leakages 113
4.6.2 Leakage Maps 114
4.6.3 Attack Validation 124
4.6.4 Suitability for Protected Implementations . . 126
4.7 Conclusion 127
4.8 Supplementary Discussion 128
4.8.1 Correlation-Based DSCA 128

5. Conclusion 132
5.1 Research Summary 132
5.1.1 RF-DNA Fingerprinting of Integrated Circuits 132
5.1.2 Leakage Mapping 134
5.2 Recommendations for Future Research 135
5.2.1 RF-DNA Fingerprinting of ICs 135
5.2.2 Leakage Mapping 139
5.2.3 Related Future Research Recommendations . 141

Appendix A. Side Channel Analysis Countermeasures 143
A.1 Constant Time 143
A.2 Constant Power Countermeasures 143
A.3 Temporal Desyncronization 148
A.4 Masking countermeasures 149
A.4.1 Masking of complex ciphers 150
A.4.2 Implementation levels for masking 151
A.4.3 Practical considerations 151
A.5 Power Supply Shielding 152
A.6 EM Shielding 152
A.7 Leakage-Resistant Arithmetic 153
A.8 Dynamic Reconfiguration 153

Page

Appendix B. Kocher's Timing Attack on RSA 155
B.1 Overview 155
B.2 Overview of RSA 155
B.2.1 Implementation of RSA 156
B.2.2 Information Leakage Modular Exponentiation 156
B.3 Key Assumptions of Kocher's Attack 157
B.4 The Attack 158

Appendix C. Kocher's SPA Attack on DES 161

Appendix D. RSA Message Blinding 165

Appendix E. AES Object Source Code 166

Appendix F. MemMapTRS Source Code 180

Bibliography 190

List of Figures

Figure		Page
1.1	Side-Channel Leakage from Physical Systems.	2
2.1	Dynamic power dissipation of CMOS inverter	9
2.2	Rohatgi's SPA on DES Key Parity Check	28
2.3	Hamming distance power leakage of 8-bit micro-controller	29
2.4	Typical DPA result	33
2.5	Typical power analysis setup	45
2.6	Typical SCA experimental setup	45
3.1	RF-DNA statistical fingerprint generation process.	58
3.2	Mean amplitude response	61
3.3	Average RF-DNA fingerprints	62
3.4	Signal collection and pre-classification processing process.	70
3.5	Average RF-DNA fingerprinting identification performance	73
3.6	Verification ROC curves (full range)	77
3.7	Worst case ROC curves (high SNR)	78
3.8	Riscure Inspector system	81
4.1	AES state matrix to byte-vector mapping	89
4.2	Comparison of alternate AES round definition	92
4.3	Byte-oriented depiction of AES encryption round	93
4.4	`AESComputeIntermediates` pseudo code	102
4.5	Construction of intermediate cipher state matrix	103
4.6	Known-key correlation pseudo code	107
4.7	Temporal leakage map (Imp. A)	116
4.8	Temporal leakage map (Imp. B)	117
4.9	Typical correlation plot (Imp. B)	118
4.10	Summary leakage map (Imp. A)	120

Figure Page

4.11 Summary leakage map (Imp. B) 123

4.12 Guessing Entropy (Imp. B) 126

4.13 DSCA Block Diagram 129

A.1 Dual Rail Domino Logic 145

A.2 Dual Rail Domino XOR 146

C.1 Power consumption of conditional branches 161

List of Tables

Table		Page
3.1	Verification Error Types.	68
3.2	Tested PIC micro-controller device classes.	69
4.1	Number of intermediate result states computed during AES encryption and key scheduling (inclusive of input and output states).	90
4.2	Maximum tolerable correlation (ρ_{max}) to achieve N_t-trace DSCA-resistance as a function of α, predicted using (4.16).	109
4.3	Summary of characteristics for evaluated implementations. . .	111
4.4	Summary of correlation results for all considered *byte-oriented* leakage models.	113
4.5	Comparison of number of traces required for successful attack as predicted by (4.15) vs. actual results of a correlation-based DPA attack on *Imp. B* using a $\mathcal{HD}^3(\mathbf{s}^{9,1}, \mathbf{s}^{10,0})$ leakage model.	125
A.1	Dual-rail domino signal encoding [WH05]	145

Abstract

The information leakage of electronic devices, especially those used in cryptographic or other vital applications, represents a serious practical threat to secure systems. While physical implementation attacks have evolved rapidly over the last decade, relatively little work has been done to allow system designers to effectively counter the identified threats. This work addresses the technology gap between identified problems and potential solutions, and significantly advances the study of information leakage in two primary areas of investigation:

1. *Radio-Frequency "Distinct Native Attribute" (RF-DNA) fingerprinting* of integrated circuits (ICs) for device authentication, and
2. *Leakage mapping* to assess the information leakage of arbitrary cryptographic implementations.

First, the *RF-DNA fingerprinting* technique is used to recognize unique ICs based on fabrication process-induced variations in unintentional electromagnetic (EM) emissions in a manner analogous to biometric human identification. The effectiveness of the technique is demonstrated through an extensive empirical study, indicating the technique scales well for both identification and verification tasks. Empirical results are presented for 40 near-identical devices, with correct device identification success rates of greater than 99.5%, and average verification equal error rates (EERs) of less than 0.05%. Correct identification success rates exceeding 90% were maintained under analysis conditions of SNR $\geq$ 15 dB.

Whereas all previously known techniques require device hardware or software modifications, RF-DNA fingerprinting permits opportunistic passive authentication using unintentional RF emissions during pre-existing processes and protocols. This characteristic makes the approach suitable for security applications involving com-

modity commercial ICs, with substantial cost and scalability advantages over previous approaches.

Second, a systematic *leakage mapping* methodology is developed and demonstrated to comprehensively assess the information leakage of arbitrary block cipher implementations. The proposed framework provides a comprehensive approach to assess the information leakage of all algorithmically specified key-dependent intermediate computations for implementations of symmetric block ciphers. The resulting leakage assessment quantitatively bounds the resistance of an implementation to the general class of differential side channel analysis techniques, and provides system designers and evaluators with a tool to objectively assess whether countermeasures implemented are justified given the added cost in time, space, and energy compared to the obtained reduction in exploitable information leakage. Furthermore, the systematic approach enables evaluators to quickly and efficiently repeat the assessment process for different variations of implementations, which helps to ensure the addition of countermeasures does not inadvertently introduce new unexpected sources of information leakage.

The *leakage mapping* framework is demonstrated using the well-known Hamming Weight and Hamming Distance leakage models, with recommendations to extend the technique using more accurate models. The approach effectiveness is demonstrated through empirical assessment of two typical unprotected implementations of the Advanced Encryption Standard, and the assessment results are empirically validated against correlation-based differential power and electromagnetic analysis attacks.

EXPLOITATION OF UNINTENTIONAL INFORMATION LEAKAGE FROM INTEGRATED CIRCUITS

1. Introduction

It is common knowledge that electronic equipment radiates electromagnetic (EM) energy that can interfere with other nearby devices. It is for this reason that airline passengers are required to "turn off all portable electronic devices", and consumer electronics sold in the U.S. are required to undergo certification testing for compliance with Federal Communications Commission (FCC) regulations [FCC09]. Digital devices that incorporate clocks, oscillators, or other high frequency pulses are specifically regulated as known unintentional emitters because they produce radio frequency (RF) radiation.

Over the past decade there has been a growing realization that the *content* of unintentional emissions, in addition to being a source of interference, can also be a source of information about the emission producing device's internal state. This realization has profound implications for the physical security of sensitive electronic systems since in many instances the "leaked" state information is sufficient to infer precise details about the operations the device is performing and the data it is processing. Such details can include extremely sensitive private information such as cryptographic key material. This research studies the limits of how much information can be gained by exploiting the unintentional information leaked from secure systems.

1.1 Problem Addressed

When viewed externally, all physical systems produce both intended and unintended outputs. The unintended outputs are quantifiable, physically observable phenomena produced as a side-effect of normal operation. When an unintended

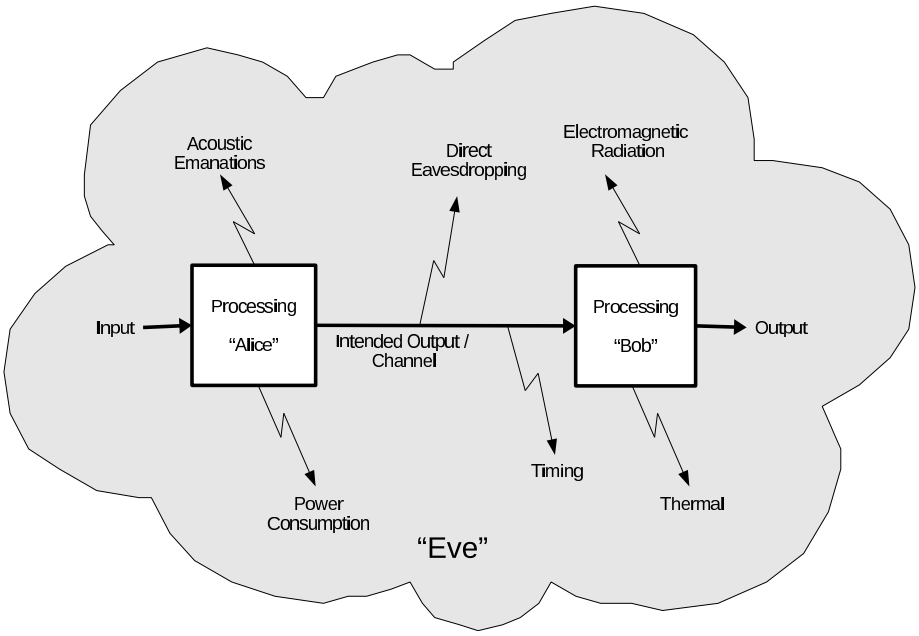

Figure 1.1 Side-Channel Leakage from Physical Systems.

observable outcome is correlated to some aspect of the internal state or system operation, a *side-channel* is said to *leak* information. The term *side-channel* was first introduced by Kelsey et al. [KSWH00]. However, military and government agencies have been aware of information leakage due to unintentional emissions for decades prior [Boa73, vL85]—as early as 1914 by some reports [And01].

Definition 1 (Side-Channel) *Any* ***unintended*** *physically observable time-varying phenomenon that is correlated to the internal state, operations, or data being processed by a device of interest.*

EM radiation, including RF radiation and other regions of the EM spectrum, is just one of many different side-channels through which electronic devices may leak information. Other channels include variations in power consumption, variations in computation time, and even acoustic and thermal emissions. The various known sources of side-channel emissions are depicted in the block diagram of a notional two-party communications system shown in Figure 1.1. The underlying phenomena that cause these emissions are described in further detail in Section 2.2.

1.2 Security Implications of Information Leakage

The side-channel information leakage from electronic devices is of intense interest to security researchers since it presents attack vectors through which unauthorized parties may gain access to information that is intended to be kept private. Information leakage may inadvertently reveal the contents of protected data such as passwords, cryptographic keys, or PIN numbers. Further, it may allow an adversary to reverse engineer critical aspects of intellectual property or critical technologies such as proprietary algorithms, circuit designs, or protocols. Therefore, unintended leakage of information poses a serious *practical* security threat to electronic systems—particularly cryptographic systems—when naive designers assume a closed, secure environment where only the intended outputs are visible [ZF05].

Over the past decade, a significant body of research has been dedicated to applying side-channel analysis (SCA) as an attack mechanism to defeat the security of cryptographic devices. Typically, the earlier research has focused on the vulnerability of systems to *key recovery attacks*—i.e., the extraction of *private* or *secret key* material from a cryptographic system. The existence of SCA techniques means the security of sensitive information that relies on the secrecy of a cryptographic key as a primary mode of protection can be greatly reduced in practice if access to side-channel emissions is realistically possible. The type of information that is typically protected varies widely, but includes things such as the content of secure communications, software or firmware code which reveals critical technology or intellectual property details, and tokens permitting access to secure networks or systems.

The protection of sensitive proprietary algorithms or techniques (i.e., *critical technology* or *intellectual property*) is of particular importance in both military and commercial applications. Reverse engineering poses a serious threat since it can enable competitors or adversaries to bypass years of research and development through counterfeiting or theft of intellectual property. Commercially, the total global loss in revenue due to counterfeiting and pirated products is predicted to reach $1.7 trillion

by 2015 [Fro11]. This estimate does not account for intangible losses such as brand deterioration. Likewise, in military applications, compromise of critical technology degrades weapon system combat effectiveness and useful life expectancy.

Whereas 'traditional' techniques for reverse engineering of integrated circuits (ICs) still require very expensive, specialized equipment (e.g., scanning electron microscopes) [TJ09], side-channel attacks can be performed using widely available (and relatively inexpensive) commercial tools. Commercial SCA systems are available for sale on the open market, to include the Riscure Inspector system used in this research [Ris09], and source code for carrying out a variety of attacks is readily available on the Internet [Par10, OT11]. Since it is increasingly common for critical technologies or intellectual property to be in software or firmware protected by a cryptographic system, the existence of SCA key extraction techniques must be a consideration when assessing the security level of any such systems.

1.3 Research Objective

The overall research objective is to investigate various aspects of the information leakage of ICs. The study of information leakage from electronic devices, particularly ICs, is a relatively new area of investigation and researchers are only beginning to formalize the problem and develop rigorous methods for assessing the vulnerability of physical devices to SCA techniques. Most research to date has focused on the exploitation of cryptographic systems, with a particular emphasis on key recovery attacks. Thus far, very little work has been done that enables system designers to effectively counter the threat of side-channel attacks or to investigate alternative, constructive uses of the phenomena. Investigating alternative aspects of information leakage is the focus of this work, and includes two main thrusts.

First, unintentional emissions are investigated as a source of information to recognize or verify the identity of a unique IC. The problem of IC authentication has numerous practical applications, including providing enhanced security for secure

access mechanisms (e.g., anti-cloning), detection of unauthorized modifications to circuit designs (e.g., hardware Trojan detection), or forensic attribution of electronic evidence in criminal or other cases. Whereas all previously known authentication techniques require either device hardware or operational software modifications, unintentional emissions can be passively analyzed. Thus, an effective authentication approach based on the unintentional emissions of a device would result in a more cost-effective and scalable approach to the problem than existing solutions.

The second thrust is to investigate techniques that enable system designers to effectively and systematically assess the vulnerability of a particular cryptographic implementation to known side-channel attack techniques. From a cryptographic system designer or engineer's perspective, designing a system that is secure against the plethora of rapidly evolving physical implementation attacks is daunting. While new or enhanced attack techniques continue to be published at a rapid pace, very little work has been done to aid system designers in practically addressing the resulting security risks. This research investigates the development of a systematic *leakage mapping* framework to guide system designers in making sound decisions during the development process to obtain, with some degree of certainty, a desired level of resistance to side-channel attacks.

1.4 Dissertation Organization

This document is divided into five chapters. Chapter 2 provides an historical overview of the relevant work done to date, and introduces fundamental concepts necessary for the study of side-channel emissions and information leakage. Additional relevant, but not critical, background information is included as appendices.

The main document body is composed of two chapters containing scholarly articles prepared during this research: *Intrinsic Physical Layer Authentication of Integrated Circuits* (Chapter 3) [CLB+11] and *Leakage Mapping: A Systematic Methodology for Assessing the Side-Channel Information Leakage of Cryptographic Imple-*

mentations (Chapter 4). Each chapter contains the aricle text, including relevant background and methodology, as submitted for publication with some editing as required for incorporation into the format of this document. Because the submitted versions of each article were constrained by mandatory page limitations, some additional supplementary explanation is provided at the end of each chapter to expand on the methodology and results in more detail where appropriate.

Chapter 5 concludes the main document and summarizes the key findings of this work, and provides several recommendations for future research.

Finally, selected source code developed during this research is included in the appendices. A full archive of the source code and data sets used to produce the results are available separately on electronic media.

2. Background

2.1 Overview

This chapter introduces the key concepts and techniques investigated in this research. Side channel information leakage by electronic systems is a multi-disciplinary field that synthesizes a variety of sub-domains of knowledge to include:

- Cryptography,
- Computer architecture and engineering,
- Microelectronic devices,
- Digital and VLSI systems design,
- Electromagnetic theory,
- Probability theory,
- Signal processing and pattern recognition,
- Communications and information theory, and
- Software design.

Where required, the relevant aspects of each field are covered to describe needed concepts as they are introduced. The remainder of this chapter is laid out as follows. Section 2.2 describes the principle sources of side channel information leakage and their underlying causes. Section 2.3 describes the main applications of side channel analysis to date, with an emphasis on cryptanalytic key recovery attacks. This section also describes the basic aspects of cryptographic systems which make SCA analysis techniques feasible in practice. Section 2.5 reviews the primary classes of cryptanalytic SCA techniques, including so-called *simple*, *differential*, and *profiling* techniques. Section 2.6 describes a variety of proposed countermeasures to improve the resistance of physical systems to SCA techniques. Finally, Section 2.7 discusses important practical aspects of SCA attacks.

2.2 Emission Sources

Electronic devices create a variety of side channels during the normal course of operation. Each channel can leak information when the characteristics of the physically observable behavior are correlated to the electronic circuit's internal operation or state. This section reviews the common side channels of electronic devices—including timing, power, electromagnetic and acoustic (cf. Fig. 1.1).

2.2.1 Variations in Computation Time. The time for a microprocessor or other electronic device to complete tasks is, in general, variable and depends on the data processed [Koc96]. The reasons for variations in processing time are unique to a particular implementation, but may include dependencies on conditional branches or other conditional execution mechanisms, cache misses, pipeline stalls, waiting on queries to external devices such as RAM, and so on [Koc96]. For combinational logic devices, the time required for an output to reach a steady, glitch-free state may vary depending on the inputs, circuit layout, or other factors [WH05].

A notional example of a simple algorithm with data-dependent computation time in most implementations is shown in Algorithm 1.

Algorithm 1 Data-Dependence Example

Require: Integers x and y, Control bit b.

1: **if** $b = 1$ **then return** x^y
2: **elsereturn** $x + y$
3: **end if**

The value, `b`, is some control bit and `x` and `y` are operands. If the control bit is a '`1`' an exponentiation is performed; if not, the operands are added. Assuming an exponentiation operation is much more costly in terms of execution time than an addition (which it would be in any typical implementation) the algorithm will take longer when `b` is '`1`' vs. '`0`'. Thus, an outside observer can infer the value of bit `b` from the amount of time the operation takes to complete. The implementation, therefore, is said to *leak* the value of the data bit `b` through a timing side channel.

Kocher discovered that such *unintentional* variations in execution time can be significantly correlated to the data processed by a device [Koc96]. In many cases the correlation of the unintentional variations leak sufficient information to recover an entire key from a cryptographic device. The *cryptanalytic* applications of timing and other SCA techniques are discussed in Section 2.3.1.

2.2.2 Variations in Power Consumption. Most modern integrated circuits, including general purpose microprocessors, are based on complementary metal oxide semiconductor (CMOS) transistor technology.[1]

The power consumed by CMOS devices has both static and dynamic components. Since the static component is nearly constant, it can be neglected for the purposes of SCA since it is not data or operation dependent and therefore does not leak information about the internal system state. Dynamic power consumption, on the other hand, is a function of internal switching activity of individual transistors. Since the switching activity depends on the operations performed and data manipulated, the resulting variations in dynamic power consumption are a source of side channel information leakage [MOP07].

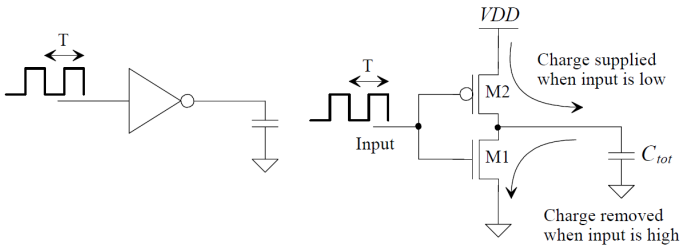

Figure 2.1 Dynamic power dissipation of the CMOS inverter [Bak07]

The total current drawn from a constant voltage power supply at any point in time is the sum of the current drawn by all the individual logic cells. There are two

[1]Strictly speaking, modern transistor technology is no longer necessarily metal- or oxide-based since those material layers can be replaced by alternative materials. However, the term CMOS is overwhelmingly used in the literature and general practice to refer to both true CMOS and other technologies with similar behavior [WH05]. The differences are irrelevant to this work, and the term is used to generically refer to all CMOS-like devices.

primary sources of dynamic power consumption in a CMOS circuit. The first is the current drawn and dissipated when individual transistors are switched `on` and `off` respectively. Fig. 2.1 illustrates the concept using a simple CMOS inverter. When the input transitions from `1` to `0`, $M1$ is switched `off` and $M2$ is switched `on`. The cell draws a charging current from a constant voltage power supply to charge the intrinsic and extrinsic capacitances (C_{tot}). When the input transitions from `0` to `1`, $M1$ is switched `on` and $M2$ is switched `off`, and the stored energy is discharged through the ground line. When the input remains the same, the transistors' states do not change and there is no dynamic current flow.

The second primary source of dynamic power consumption in CMOS logic is the short circuit path between the power rail and ground created during the short period of time during each transition when both the pMOS and nMOS networks are partially `on`. This short-circuit current contributes significantly to the dynamic power consumption of a circuit.

Combinational CMOS circuits also experience transient signal behavior, known as glitches or dynamic hazards, as inputs propagate and arrive at different times through the circuit. In large or complex combinational circuits, dynamic power consumption is influenced to a large degree by the glitches encountered before the circuit settles into its intended steady-state output—to the point that glitches may actually become the dominant source of dynamic power consumption [MOP07].

For microprocessor devices, power consumption is affected by the instruction being executed, the content of data being manipulated, and the address or location of memory or data registers being accessed [QS02]. For pipelined architectures, there may be two or more instructions in the pipeline during a clock cycle, all of which contribute to power consumption during that cycle [QS02].

The statistical relationship between the activity of a circuit and its total power consumption can be used to infer information about the data being processed—including sensitive information such as cryptographic keys used in a ciphering opera-

tions [KJJ99]. Kocher introduced two *key recovery* attacks based on this observation—simple power analysis (SPA) and differential power analysis (DPA). Both passively extract the entire secret key from implementations of the Data Encryption Standard (DES) cryptographic algorithm. Kocher's SPA and DPA techniques are described in Sections 2.5.2 and 2.5.3.

2.2.3 Electromagnetic Emissions. The electromagnetic emissions (both radiated and conducted) produced by electronic devices during normal operations are physically caused by the time-varying current drawn by the circuit as the transistors switch `on` and `off`. As current flow varies, it induces many tiny time-varying electromagnetic fields. These fields combine through complex interactions and propagate as time-varying EM waves via both radiation and conduction through the power supply and ground lines and other conductive materials [Boa73, AARR02, AARR07]. The fundamental nature of these effects is well understood as described by Maxwell's equations [AARR02, AARR07]. The idea that EM emanations leak information has been known for several decades, although it has only recently become the focus of significant academic research [Boa73, And01].

Most early EM emissions research was related to the eavesdropping risks of peripheral devices such as video display units (see Section 2.3.1.1). Van Eck and Laborato's (1985) eavesdropping attack, described in Section 2.3.1.1, was the first known research on the topic [vL85]. The first published cryptanalytic applications were [QS02, GMO01], both of which were extensions of Kocher's timing and power attacks.[2]

Quisquater and Samyde were the first to extend Kocher's power and timing attacks to the EM side channel [QS00, QS01]. By observing the EM side channel data they achieved more precise measurements of circuit activity than would be possible with power analysis short of physically modifying the circuit. Achieving

[2]The authors of [QS00] refer to their EM variations of Kocher's attacks as Simple EM Analysis (SEMA) and Differential EM Analysis (DEMA).

the same precision of measurements with power analysis would require an invasive attack to bypass the filtering effects of the power distribution network and peripheral components.

In 2001, Gandolfi et al. published the first concrete examples of EM cryptanalysis by successfully extracting keys from CMOS hardware implementations of DES, COMP128, and RSA [GMO01]. Gandolfi's results reinforced Quisquater and Samyde's claim that EM emanations allow more efficient DSCA attacks, resulting in successful key recovery with fewer observations.

Agrawal et al. more recently published and updated an extensive study of EM information leakage [AARR02, AARR07]. One of their key findings was that single wide-band EM sensors capture many unique information leakage *signals*, each of which carry distinct information. They also noted that a substantial source of information leakage is the very weak amplitude-modulated EM information conducted by surfaces or lines attached to the device. The conducted EM leakage propagates through attached or nearby conductive surfaces to include the power and ground lines, implying that side channel data collected for power analysis attacks is likely to contain weaker and higher frequency EM leakage signals due to unintentional power line modulation.

EM emanations caused by a circuit's operation fall into two primary categories—direct and indirect emanations [AARR07]. *Direct emanations* are created by current flow through a circuit's intended current path and the resulting switching activity in the circuit's transistors or other devices. *Indirect emanations* are created when small couplings between densely packed electronic components modulate existing carrier signals emitted by the device, and can modulate both unintentional and intentional EM signals transmitted by a device. Both types of emanations can lead to side channel information leakage.

Through the analysis of leaked EM information, Agrawal et al. claimed successful attacks against various cryptographic implementations. Additionally, they

posit the possibility of more powerful attacks using multiple complementary electromagnetic sensors with different characteristics to simultaneously collect side channel information leakage from multiple signals.

Agrawal's most interesting experimental result is that strong information bearing signals (particularly carriers at multiples of the system's clock frequency) propagate well beyond the near-field—beyond 50 feet for an unprotected commercial secure socket layer (SSL) accelerator card operating inside an unmodified, closed computer server [AARR02, AARR07, Roh06]. Practical template attacks on such a device with no countermeasures were possible at a distance of approximately 15 feet [AARR02, AARR07]. Mangard reported similar results and demonstrated extraction of a cryptographic key from a smart card at a distance of more than 5 m in an unshielded environment with an unoptimized measurement setup [Man03]. The threat from these types of attacks may actually be greater because an adversary could potentially place an inconspicuous receiver close to the target device (within 5 ft.), which could relay the side channel data of interest to a remote processing station [Roh06]. The signal strength at that distance may be sufficient to enable single-observation template attacks and advanced signal processing techniques capable of defeating some SCA countermeasures.

A potentially even more powerful technique was published by Burnside, et al. on the illumination of an integrated circuit by an external carrier [BEA08]. The targeted integrated circuit is illuminated with a low-power external RF source, and the data-dependent behavior is modulated on the reflected external signal which is then captured. The authors speculate that this technique may result in effective side channel attacks at greater distances than those based on the native emissions produced by the target device, but no experimental results were provided to support that hypothesis.

2.2.4 Acoustic Emissions. The acoustic emissions of devices are also a source of side-channel information leakage. Recently declassified lectures on the History of Communications Security reveals that intelligence agencies were concerned with acoustic information leakage from mechanical cryptographic devices as early as the 1940's—even over noisy telephone lines when a phone receiver was operated in the same vicinity as the sensitive equipment [Boa73].

Various peripheral electronic devices also reportedly leak information. For example, the sound of keyboard (or other keypad) presses is correlated with what a user is typing [AA04, ZZT05]. However, few other peer-reviewed publications exist on the subject. Recently, a similar attack was demonstrated using the sound of a dot matrix printer to identify the characters printed [Gib09]. Dot matrix printers are still used widely in financial and medical applications which makes the existence of such a vulnerability a relevant practical concern.

Most surprising, however, is that electronic devices such as general purpose microprocessors actually leak information through their acoustic emanations. Shamir and Tromer have performed some initial, unpublished, experiments indicating that the acoustic emissions of commercial microprocessors are sufficient to determine with good precision when specific operations begin and end. Further analysis could lead to effective timing attacks on cryptographic devices [ST04].

2.3 Applications

The vast majority of side channel research to date deals with the *security vulnerabilities* introduced by unintentional information leakage. Side channel vulnerabilities range from simple eavesdropping risks of peripherals, such as video displays or keyboards, to the inadvertent disclosure of the cryptographic keys critical to the security of entire systems.

Beyond the various cryptanalytic attack modes, side channel analysis has also been investigated for several other applications including reverse engineering, hard-

ware covert channel circuitry, and the detection of unauthorized modifications or additions to circuits (Trojans) during outsourced chip fabrication.

This section provides an overview of the different applications of side channel analysis investigated over the past three decades of research in this area.

2.3.1 SCA Attacks. The term *SCA attack* (often just SCA in the literature[3]) is commonly used to describe any use of side channel information to either:

1. Bypass or compromise the security mechanisms of a system, or
2. Infer information about the internal state, data or operations of a device that is not intended to be accessible.

SCA attacks are a subset of the more general class of *implementation attacks* which exploit vulnerabilities in a device's physical implementation rather than attacking, for instance, the mathematical strength of a cryptographic algorithm. Other common implementation attacks are *fault attacks* and physical tampering techniques [ZF05]. SCA attacks differ from many other implementation attacks in that they are generally passive—meaning they can be implemented without revealing the attacker's presence and without damaging or otherwise physically affecting the system of interest [And01]. This is in contrast to many other known implementation attacks that are either active—risking disclosure of the attacker's presence or intent, or invasive—risking damage or triggering of a circuit's tamper-resistant features.

The two most prevalent SCA attacks—eavesdropping and key recovery attacks—are described below.

2.3.1.1 Eavesdropping. Eavesdropping attacks on electronic devices are analogous to eavesdropping on human conversations. Both involve an adversary,

[3]The acronym SCA is used interchangeably throughout the literature to refer to both side channel analysis and side channel attacks. For clarity, the term SCA is used to describe the more general *side channel analysis*. Side channel attacks are referred to as *SCA attacks*.

Eve, secretly listening to what is intended to be a private conversation between an originator (Alice) and a recipient (Bob). A typical traditional eavesdropping attack might involve Eve tapping into video display cabling for a computer monitor or television display to see the same picture Bob is seeing.

SCA eavesdropping attacks differ from classical eavesdropping in that they do not target the primary or intended communications channel. For example, returning to the scenario of a video display signal, in the SCA eavesdropping attack Eve would not directly tap into the primary video display signal. Rather, she would attempt to reconstruct the video signal from an alternate, unintended side channel of information such as EM emanations produced by the system's cabling. Eve may prefer a side channel attack to the direct eavesdropping approach for a number of reasons such as restricted access to the system or tamper-resistant cabling.

The first published eavesdropping attack to exploit unintentional emissions was by Wim van Eck [vL85]. Cathode ray tube (CRT) video terminals produce electromagnetic emissions that can be remotely intercepted and reconstructed by an eavesdropper at a substantial distance—even through walls. The attack is carried out using inexpensive commercially available receiver technology to passively view a real-time reproduction of what is being displayed on a remote target computer. More recently, van Eck's attack has been extended to show it is still relevant to modern-day flat panel technology [Kuh04].

Eavesdropping SCA attacks typically target peripherals or human interface devices. Attacks have been published that target the electromagnetic emanations of computer displays and keyboards [VP09], the flashing of light-emitting diode (LED) activity indicators on computer peripherals such as modems [LU02], and the acoustic activity of keyboards and printers [AA04, ZZT05], among others.

Eavesdropping attacks are not the principal focus of this research, and the details of these attacks are not covered further here. Additional details can be found in [vL85, Kuh04, VP09].

2.3.1.2 Cryptanalysis and Key Recovery Attacks. The term *side channel cryptanalysis* refers to the practical application of SCA to break or bypass cryptographic implementations—with a typical objective being the recovery of a secret key from a device [MOP07, ZF05, LR09]. Whereas SCA eavesdropping attacks target side channel information leakage that reveals the same information as an intended input or output, side channel cryptanalysis attacks attempt to gain knowledge of the internal state of a device or the data it is processing by analyzing the information leakage from the implementation.

SCA cryptanalysis relies on the fact that cryptographic systems are not manifested in physical form as a pure mathematical function. Mathematically, a encryption operation is a function $E_K[P] \mapsto C$ where K is the key, P is the plain-text or data being encrypted, and C is the intended output or cipher-text [Sch96]. However, in reality, a physical system implements the encryption algorithm as $E_K[P] \mapsto (C, S_1, S_2, \ldots, S_n)$ where S_i is any additional information unintentionally leaked by the system through one of n side channels.

For electronic systems, a fundamental reason that *information leakage* occurs is that circuits must perform a variety of intermediate steps to produce a desired final output value from a particular input. When viewed at the transistor level, even fundamental logic cells such as an XOR gate produce intermediate signals in the propagation path to the output.

In general, systems and components are designed in such a manner that only the final output is intended to be externally visible. However, although the results of each intermediate computation are not normally observable, some information about their activity and content is leaked through the side channels they produce.

Definition 2 (Intermediate value) *Any non-final result produced during the intermediate steps of a computation.*

Consider a system based on Algorithm 2, below. This algorithm can be viewed as a simple, albeit very weak cipher that encrypts data by adding it to the *Hamming weight* of the key, K. The system's security depends on keeping the contents of the key, K, a secret from outside observers or malicious agents. Assume K is stored in some internal memory location inaccessible to outside observers.

Algorithm 2 Example of Intermediate Value Leakage

Require: Input X

1: $T \leftarrow X$
2: **for** $i = 0$ to 31 **do**
3: **if** $K_i = 1$ **then**
4: $T \leftarrow T + 1$
5: **end if**
6: **end forreturn** T

There are at least three potential sources of information leakage in this algorithm:

1. The comparison operation at line 3 directly accesses a key bit during each loop iteration.
2. The algorithm's execution path during each loop iteration is conditional on a key bit.
3. The intermediate value T is manipulated at line 4 whenever the key bit is a '1'.

Due to a variety of phenomena described in Sections 2.2 and 2.5, each of these actions may cause variations in the physically observable characteristics of the device that are statistically related to the operation or data being manipulated. Analysis of this side channel data may reveal K_i directly, or more subtly the value or change in state of T which indirectly reveals one bit of K. As a result, the secret key K may be revealed—compromising the security of the system on which it depends.

For cryptographic or other secure systems, the implications are far-reaching. Since almost all security mechanisms in electronic and computer systems rely on

preserving the secrecy of some sensitive information, any attack that can bypass those mechanisms is of great concern. Whereas many traditional cryptanalytic attacks are of theoretic interest but are not practical threats, recent research has shown side channel leakage to be of great *practical* concern.

In 1996, Kocher proposed the first known attack capable of using side channel leakage (timing information) to extract an entire key from *implementations* of several cryptographic algorithms [Koc96]. Since the publication of Kocher's original paper, numerous new cryptanalytic side channel attacks and improvements have been published targeting a wide range of cryptographic algorithms and protocols. Successful practical attacks have been mounted against hardware and software implementations of block, stream, and public-private type algorithms including DES, AES, Rivest-Shamir-Adleman (RSA), and various elliptic curve schemes [Koc96,KJJ99,Roh06,And01]. Mangard, et al. published an entire text dedicated to SCA attacks of AES implementations [MOP07]. Brumley and Boneh demonstrated a practical remote attack against an OpenSSL-based webserver across a local area network [BB05].

Recently, a complete practical attack on the KeeLoq encryption scheme has been published capable of extracting the master key from a device in a single observation [KKMP09]. KeeLoq is a ubiquitous remote keyless entry protection algorithm used in a wide variety of commercial keyless entry systems for vehicles and garage door openers. The publication of a simple and efficient attack against such a widely used system amplifies the practical threat of SCA.

Section 2.5 reviews the major classes of SCA cryptanalytic techniques developed to date. In response to these discoveries, a variety of countermeasures have been introduced to reduce the vulnerability of electronic systems to each new type of attack. Section 2.6 reviews known countermeasures.

2.3.2 Reverse Engineering. The idea of using SCA for reverse engineering applications dates to Kocher et al.'s original (1998) DPA paper, in which the authors claim to have used side channel analysis techniques to reverse unknown algorithms and protocols. The authors posited that automation of reverse engineering of unknown systems might be possible, although no details of such an approach were given. More recently, some success has been realized using side channel leakage to manually facilitate reverse engineering and extraction of keys from unknown algorithms. This application of SCA has been termed side channel analysis for reverse engineering (SCARE) [Nov03].

The first explicit use of SCA for reverse engineering in the academic literature was by Quisquater et al. [QS02]. The authors show that it is possible to sufficiently characterize the electromagnetic side channel emissions of a microprocessor to create an instruction template dictionary. Future observations of the EM emissions from an identical microprocessor can be automatically classified to determine what operation was being performed.

Novak (2003) proposed an attack, with later improvements by Clavier (2004, 2007) to reverse engineer non-trivial portions of an unknown A3/A8 cryptographic algorithm [Nov03]. The targeted algorithm is the COMP128-2 cipher used in cellular phone SIM cards to authenticate and generate keys in GSM networks [Roh06]. Clavier improved on Novak's original technique and demonstrated a more practical reverse engineering approach capable of retrieving lookup tables of an unknown cryptographic algorithm without requiring any prior knowledge of the secret key [Cla04, Cla07]. Daudigny, et al. (2005) used SCARE to recover unknown details of a DES implementation [DLMV05].

Early reverse engineering attempts were applied specifically to software-based microprocessor implementations. The first extension of the technique to an unknown hardware implementation was in 2008 [RDG+08].

In all of these cases domain knowledge is used extensively, but the passive reverse engineering capability is nevertheless interesting. In many scenarios, the non-invasive nature of side channel analysis provides clear advantages over more invasive reverse engineering techniques. Reverse engineering applications of SCA are still a relatively undeveloped area of investigation, and it is currently unknown whether it is possible to generalize and automate the process as envisioned by Kocher, et al. [KJJ99, Roh06]. However, the technique's existence is another reason system designers should not rely on *security by obscurity*.

2.3.3 Covert Channel Engineering. Researchers in high-assurance computing realized early on—at least by 1973—that timing variations can be used to *intentionally* pass information across a controlled perimeter [Lam73,van90,Wra91,And01]. For example, a malicious Trojan program can be placed inside the controlled perimeter of a secure computing environment where it captures data and re-transmits it via an encoded communications channel [Lam73, van90, Wra91, And01]. Such a communications channel can be created by intentionally varying some system characteristic visible to an observer outside the controlled perimeter. For example, the Trojan process can cause page faults or variations in CPU demand or disk cache loading. A receiving process outside the perimeter can monitor the timing variations and decode the message. This type of scheme is known as a *covert timing channel* [And01].

Circuit design or embedded software codes can be modified to surreptitiously broadcast secret information over a covert wireless channel (such as the EM side-channel) to a nearby receiver [DS06, Dyr07]. There are no concrete examples of such a hardware-based malware attack having actually occurred in the literature to date, but there are numerous examples of commercial hardware shipping with software-based malware. Several recent papers have been published with proof of concept designs that illustrate the feasibility of such an attack [KTC+08, LKG+09].

2.3.4 Trojan Detection. The economic realities of semiconductor and microchip manufacturing have resulted in a large-scale migration of most integrated circuit (IC) fabrication to facilities outside the United States. In 2005, a special U.S. Defense Science Board Task Force examined the effects and risks of outsourcing high performance microchip production to foreign countries [Off05] and concluded this practice introduces risks that are unacceptable for domestic defense and intelligence applications. The Defense Science Board determined that existing diagnostic capabilities are insufficient to assure that chips produced by foreign foundries are unmodified. One particular risk cited is the opportunity for malicious foreign agents to introduce unauthorized design changes such as Trojan horses into the chips, as described above, to establish covert channels of communication.

Agrawal, et al. proposed extending side-channel analysis techniques to detect the presence of hidden or modified circuitry in integrated circuits [ABK+07]. The approach constructs SCA fingerprints by profiling the power, temperature and electromagnetic side channel characteristics of an integrated circuit. After fingerprinting, destructive techniques are used to verify the authenticity of the characterized chips; i.e., they have not been modified from their original intended design. Remaining ICs are statistically tested against the fingerprints of the known good chips. The approach clearly requires that the destructive verification techniques be extremely reliable in detecting modifications to assure that the baseline is truly an unmodified chip. This may not be practical in light of the Defense Science Board findings.

2.4 Adversary Models

When discussing SCA attacks, various assumptions must be made about the capabilities of an adversary. It is common to refer to how *powerful* a hypothetical adversary must be to carry out a particular attack in practice. In real world situations, an adversary's capabilities can range from having very restricted access to

the device at one extreme, to having complete control of the device or system being attacked at the other.

A *powerful adversary* is one that has complete control of the device or system being attacked. In this scenario, the adversary can arbitrarily choose the number and content of inputs the device will process, can capture both the inputs and outputs of the device, and is assumed to be in an environment (such as a well-equipped laboratory) where the quality of the side channel data being collected may be optimized. In the most extreme case, the adversary may even have the capability to load new keys into the device. The only significant limitation is that stored keys cannot simply be retrieved as that is prohibited by most cryptographic devices.

A *weak adversary* has very limited capabilities to control and observe the device being analyzed. In the context of SCA attacks, Eve still has the capability to collect side channel data in some form (possibly noisy and otherwise suboptimal). Most attacks also require that she be able to capture the contents of either the input or output from the device (typically the *plain-text* or *ciphertext* in the case of a cryptographic device).

Many SCA attacks assume a *powerful adversary*, which implies some reduction in practicality under certain conditions. However, this does not negate the real-world threat since there are many scenarios where such a *powerful adversary* truly exists, and the attacks are easily within the reach of a moderately equipped malicious agent. The publication of actual SPA and DPA attacks on the KeeLoq remote keyless entry system [KKMP09] and the Xilinx FPGA bitstream protection [MBKP11, MKP11] amplify this point. Furthermore, many attacks such as profiling techniques (see Sec. 2.5) have very weak assumptions about the adversary from the perspective of access to the device being attacked.

Over the past decade, dozens of distinct SCA attacks and countermeasures have been developed. Most, if not all, are derived from Paul Kocher's original timing and power analysis attacks [Koc96, KJJ99]. Because of the large number of distinct attacks that have been developed, it is impractical to describe each in detail. However, most of the techniques are similar and can be generally described according to their basic principles of operation. Two of the most important distinguishing characteristics used to categorize SCA techniques are the analysis approach and the number of stages involved in the analysis.

The following definitions are used:

Simple SCA (SSCA). Any technique that directly interprets side channel measurements, typically through visual analysis of a captured signal in the time or frequency domain [KJJ99]. Generally applied to one or only a small number of observations. SSCA is a generalization of Kocher's SPA technique to any source of information leakage.

Differential SCA (DSCA). Any technique that uses statistical or other mathematical techniques to look for small differences in side channel data that may be correlated to the data or operations of interest. DSCA is a generalization of Kocher's DPA technique to other sources of information leakage and statistical analysis techniques [KJJ99]. DSCA is usually applied to a relatively large number of observations—varying from a few dozen to a million or more [MOP07, Roh06].

Profiling SCA. Any SCA technique that requires multiple data-collection or analysis steps, such as a profiling stage using a training device followed by an attack stage on a separate target device.

This section reviews the fundamental SCA techniques developed to date: timing attacks, simple SCA attacks, differential SCA attacks, and profiling attacks.

Each published SCA attack is unique in its approach, and most are specific to a particular algorithm or implementation platform. Timing attacks are presented first and separately because they provide the initial foundation for the application of SCA to cryptanalysis, and Kocher's original attack does not fit neatly into the definition of the other fundamental techniques that are the subject of widespread research today. A brief overview of more advanced variations is also provided to include profiling, higher-order DSCA, and advanced signal processing techniques that can be applied to aid in pre- or post-processing of captured side channel emissions data.

2.5.1 Timing Analysis. Timing analysis attacks exploit the small variations in the computation time of a cryptographic implementation described in Section 2.2.1. Asymmetric cryptographic algorithms in particular are known to have non-constant execution times due to their conditional execution of time-consuming operations such as multiplications [Koc96]. These non-constant times, together with knowledge of the underlying algorithm or implementation, allow an adversary to infer information about the sensitive data being processed. Although the idea of timing attacks is generally applicable to any cryptographic implementation that operates in non-constant time, the details of any particular attack are implementation specific and there is no generalized process for constructing such an attack.

Kocher's timing attack [Koc96] on RSA and other asymmetric public-key algorithms is widely considered to be the first side-channel cryptanalysis in the scientific literature. Kocher's attack targets the non-constant execution time of a microprocessor-based RSA algorithm implementation. The attack extracts the private key from a cryptographic device by analyzing the small variations in computation time of a large number of decryption operations. Kocher validated the attack against RSA Laboratories' reference implementation [Koc96, Lab94]. Kocher's timing attack on asymmetric ciphers is described in detail in Appendix B.

Several practical attacks have also been published to demonstrate the feasibility of extracting a cryptographic key *remotely* across a computer network based on the observed differences in computation or response times, including those variations introduced by cache misses (cf. [BB05, PH98, Ber05, KSWH98, KSWH00, Pag02]). It is noteworthy that even lookup-table-based approaches to cryptography, which, on the surface would appear to take constant time, can be vulnerable to timing attacks due to variations related to cache misses.

2.5.1.1 Final Notes. Kocher's original analysis targeted cryptographic algorithms based on modular exponentiation. However, he asserts that *any* cryptographic implementation that runs in non-constant time is likely to be vulnerable to timing attacks. The plethora of attacks that have been published over the last decade reinforces this claim, and investigators continue to identify new examples.

2.5.2 Simple Analysis. In the SSCA class of techniques, an analyst examines (typically visually) measured side channel data looking for distinct features that reveal information about the operations being performed or data being processed. The analyst is, in effect, performing a high-level reverse engineering based on knowledge of the algorithm that produced the side channel leakage. The concept is not limited to the power side channel (e.g., Kocher's DES attack [KJJ99]), and can potentially be applied to the data obtained from any of the side channels described in Section 2.2.

When side channel data is visually analyzed in the time domain, features like shape and magnitude can look very different for different operations, or for the same operation performed with different data or outcomes. A well-known example is the power consumption for conditional branch instructions in micro-controllers, which often depends on whether the branch is taken or not [KJJ99]. The power trace of a device that takes a branch may look significantly different than the trace of the same device when it does not take the branch. By analyzing these differences and

applying knowledge of the underlying algorithm, it is possible to deduce the content of the data being processed—generally some sensitive private data of interest such as a portion of a cryptographic key.

2.5.2.1 Simple Attack on a DES Parity Check. The level of difficulty involved in applying SSCA to extract data from a system varies greatly. An extreme example of a trivial attack on a cryptographic key focuses on the key loading mechanism rather than the cryptographic algorithm itself [Roh06]. When a key is loaded into memory from a storage location, it is desirable to verify the key's integrity before it is used. One way to do this is to store a parity bit with each 7 data bits. The data parity is verified when the key is initially loaded.

Algorithm 3 DES parity-check algorithm [Roh06]

Require: Key K (8 bytes — ea. 7 data bits, 1 parity bit).

```
1: for i = 8 downto 1 do
2:     parity ← 0
3:     for j = 8 downto 1 do
4:         if bit j of Key[i] = 1 then
5:             parity ← parity + 1
6:         end if
7:     end for
8:     if Even(parity) then
9:         ParityError();
10:    end if
11: end forreturn ()
```

Algorithm 3 illustrates an implementation of such a parity checking algorithm. The algorithm examines the value of each data bit and adds one to the parity if the bit is a `'1'`. Line four of the algorithm executes a conditional branch decision based on each data bit.

Figure 2.2 shows how trivial an attack on this type of implementation can be. The inner and outer loop structure is immediately apparent from the peaks. The data shown here includes the execution of three outer loops, each of which is composed of

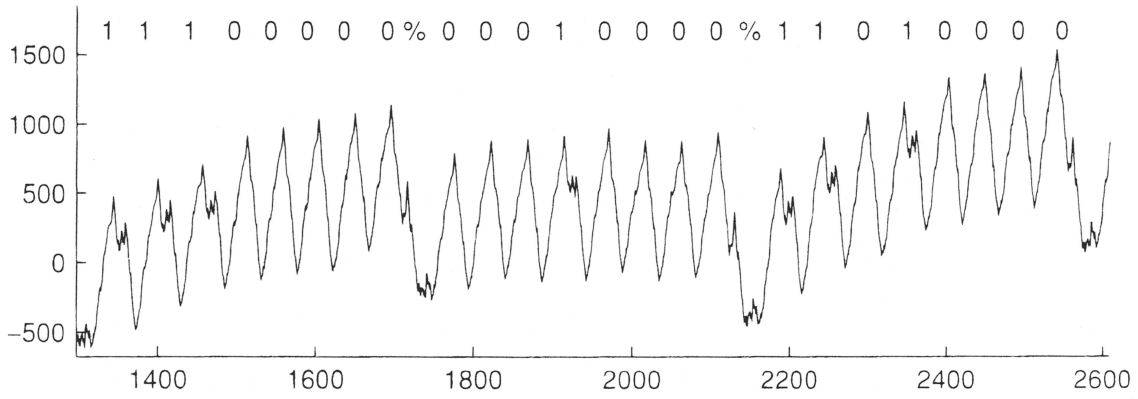

Figure 2.2 Rohatgi's SPA Attack Against DES Key Parity Check [Roh06].

eight inner loops. The differences between the inner loop iterations where the data bit is a `'1'` and those where it is a `'0'` are also clearly apparent. When the data bit is a `'0'`, the addition step on Line 5 is skipped, and the current drawn by the circuit peaks and drops off rapidly. When the data bit is a `'1'`, however, the current drawn first drops off and then increases again for a short time. Thus, it is trivial to "read" the entire secret key by directly examining the current trace and assigning each peak a value of `'0'` or `'1'`. It is noteworthy that a timing attack is also possible since the inner loop appears to take slightly longer when the data bit is a `'1'` than when it is a `'0'`.

Despite the *simple* label, most SSCA attacks are significantly more subtle and complex than the parity example. Kocher's DES attack, for instance, requires substantial insight into the operation of the underlying cryptographic algorithm to extract the key from a device.

Information may leak from a wide variety of operations and can vary based on the data being manipulated as well. For example, on some microprocessor-based devices a `load register` instruction reveals the *Hamming weight* or *Hamming distance* of the data register [MDS99]. For such devices, the power consumed is proportional to the number of `'1'` bits being processed, or transferred across a bus.

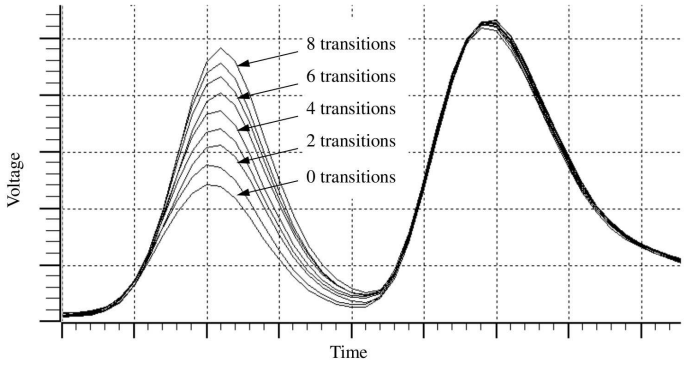

Figure 2.3 *Hamming distance* power leakage from an 8-bit smart-card microcontroller performing a `load register` operation [MDS99]

Definition 3 (Hamming weight) *is the number of non-zero bits in a binary vector of finite length [MOP07].*

Definition 4 (Hamming distance) *between two binary vectors of the same length is the number of coordinates in which the two vectors differ [Mac03].*

Figure 2.3 is the result of an SCA experiment on a vulnerable 8-bit smart-card that clearly leaks the *Hamming distance* of the data being loaded into a register. Such leakage is significant because for some cryptographic implementations, knowledge of the Hamming weight or distance of a portion of the cryptographic key is sufficient to make a brute-force key search or mathematical cryptanalysis of the remaining possibilities computationally practical [MDS99].

2.5.2.2 Practicality of SSCA Attacks. A significant practical consideration for SSCA techniques is they require substantial knowledge of *how* a particular cryptographic implementation is implemented. Although the need for implementation details appears to be a severe limitation, numerous researchers assert that SSCA techniques are still quite effective in practice unless the system implements specific countermeasures to prevent them [KJJ99, Roh06, MOP07]. Careful inspection of the side channel leakage may actually reveal sufficient information about the imple-

mentation to allow further refinement of an attack, or sufficient information for the attacker to infer important details about how the system is implemented.

SSCA attacks are most effective against sequential implementations of algorithms. Use of SSCA to attack a fully pipelined hardware implementation of DES, for instance, would be difficult because the parallel circuit activity would significantly hinder direct visual interpretation [MOP07]. However, even if SSCA techniques are insufficient to extract information from a side channel signal directly, they are frequently useful to help guide more advanced techniques and to infer information about the underlying implementation or algorithm being executed.

2.5.3 Differential Analysis. The second power analysis technique introduced in [Koc96] is known as differential power analysis (DPA). DPA is a statistical technique that correlates the effects of the data being manipulated to the power consumption trace rather than inferring the data being manipulated from knowledge of what operations were performed as in SPA. Whereas SPA can be performed using only a single power trace for an encryption device, DPA requires multiple power traces to make statistical inferences. DPA is capable of extracting information due to variations that are too subtle to be identified through direct examination of the data by SSCA. The technique is not unique to the power side channel, and can be generalized to the physical leakage due to other phenomena as *differential SCA* (DSCA) [KJJ99, ZF05, MOP07, Roh06]. A key assumption of most DPA attacks is that the attacker has essentially unfettered access to the physical device under attack and can cause it to perform encryption or decryption operations at will.

DSCA techniques use statistical properties of multiple samples, which reduces *noise* from measurement error or non-relevant circuit activity (either intentional or unintentional) while amplifying the effects of the circuit performing the operation or manipulating the data of interest. With a sufficiently large number of observations, it is possible to identify very small correlations between the internal device state and

the leaked side channel information. Various mathematical techniques are used to identify these correlations, as described in Section 2.5.3.1.

The common characteristic of all DSCA techniques is they use statistical methods to find small differences in the leaked side channel information due to variations in processed data or operations over a relatively large number of operations. The number of observations required to successfully apply DSCA techniques range from dozens to millions depending on the implementation, technique, and environmental factors, and any countermeasures present [MOP07].

A key advantage of DSCA techniques over SSCA techniques, besides their more robust information recovery capability, is they do not require detailed implementation knowledge. Rather, basic knowledge of the underlying algorithm is sufficient to carry out the statistical analysis. In fact, DSCA techniques have the interesting property (as pointed out in [KJJ99]) that they "automatically" identify the points in time where the side channel leakage is correlated to the value of some piece of data that is used by the cryptographic algorithm since the chosen metric indicating the presence of correlation will be maximized at the relative times in the trace when the relationship is strongest. The *leakage mapping* approach developed in Chapter 4 provides a methodology for systematically identifying all such potential points of key-dependent leakage throughout an entire encryption operation.

Mangard, et al. and other researchers have noted that even if the details of a particular implementation are unknown, the combination of DSCA and SSCA techniques can be used together to learn sufficient details about an implementation to allow a successful SCA attack [MOP07].

The more accurately the *leakage model* employed characterizes the true relationship of a particular intermediate value to the side channel leakage from the device, the better the results will be in the final step of the DSCA attack. Various techniques for modeling the leakage associated with an intermediate computation are discussed in more detail in Chapter 4.

2.5.3.1 Statistical Techniques. There is no consensus on an *optimal* mathematical technique for the DSCA procedure, although various proposals have been put forward with claims of optimality. Kocher's original DPA attack was based on a *difference of means* technique [KJJ99, MOP07]. Later works introduced other approaches including the Pearson correlation coefficient, Distance of Means (a variation of Kocher's difference of means), and Bayesian estimation procedures. The correlation coefficient and Bayesian techniques have received the most attention in recent research. Mangard, et al. claim the correlation coefficient is the most general because it can more easily handle multiple-bit leakage models than some other techniques [MOP07]. Recent experimental data supports the claim that all of the commonly used statistical techniques produce, roughly, equivalent results [MOS09].

The basic strategy behind all of these techniques is to identify linear relationships between the predicted leakage under some leakage model and one or more columns of the actual observed data matrix—where each column corresponds to a particular sampled instant in time relative to the start of the encryption operation. When the key hypothesis is correct, there should be a linear relationship between the hypothesized leakage value and the observed leakage value at the times when that value was actually manipulated by the circuit. If Pearson's correlation coefficient is used, the highest observed correlation indicates the corresponding key hypothesis most likely to be correct. Furthermore, the relative position of the samples with the highest correlation coefficients indicate at what relative time(s) the targeted intermediate value is manipulated. The DSCA procedure is described in detail in Chapter 4.

The largest correlation coefficients indicate the sub-key most likely to have produced the observed results. Plotting the rows of the correlation matrix will result in noticeable peaks in the plot that corresponds to the true key, while incorrect keys have the appearance of noise within a solid band. A typical DPA result for three key hypotheses (and reference current trace) is plotted in Fig. 2.4. The trace showing

a large peak indicates the correct (sub)key hypothesis has likely been found. In practice, keys similar to the true key may also produce high correlations, leaving some ambiguity as to what the correct key is—or even resulting in an incorrect key having the highest correlation coefficient at some point in time. This can sometimes be overcome directly through additional post-processing or by collecting additional observations. Furthermore, even if a DSCA attack leaves many possible candidate sub-keys, the reduction in *guessing entropy* may be sufficient to allow brute force computation of the full key.

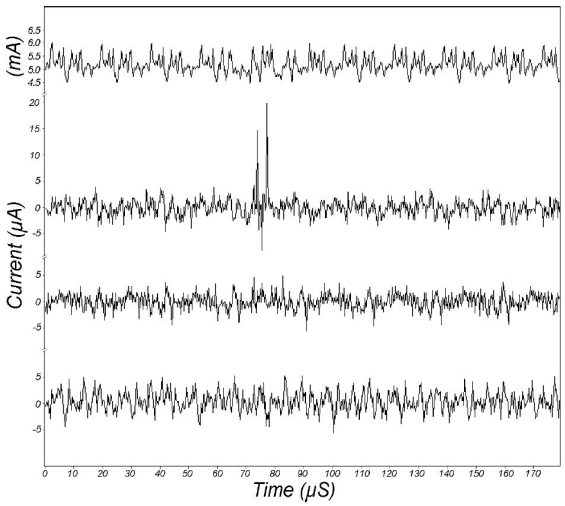

Figure 2.4 Typical DPA traces, one correct and two incorrect, with power reference [KJJ99].

2.5.3.2 Practical Limitations of DSCA. Because DSCA techniques require less implementation knowledge and are typically better than SSCA at extracting information, they are generally considered to be a more powerful class of attacks. However, if significant noise is present the statistical techniques used can require a very large number of observations to be successful. Collecting a large number of observations in practice may require an attacker to have dedicated access and control of the device being analyzed. Thus, in practice, the security vulnerabilities of a system to DSCA techniques may be less of a concern to the system designer than SSCA vulnerabilities.

Another practical limitation of standard DSCA techniques is an adversary must generally have access to obtain either known inputs or outputs from the device, although some advanced differential techniques (see Section 2.5.5.3) can be applied in situations when this is not possible. AES, when operating in counter-mode, is also susceptible to DPA techniques with knowledge of side channel data only (no knowledge of inputs or outputs is necessary) [Jaf07].

2.5.4 Profiling Techniques. Profiling, or multi-stage attacks were first introduced by Chari et al. in 2002 [CRR02] in the context of side channel cryptanalysis. In general, profiling techniques are specific applications of the general problem of pattern recognition, in which a *classifier* is presented with a *pattern* or signal (e.g., the EM leakage from a target device), and must then classify it as belonging to one of several possible classes [TK09].

These attacks postulate a powerful adversary that possesses a separate identical (or nearly identical) device over which they have full control. In the variation known as a *template attack*, this training device is used for a profiling step, in which an attacker creates a precise multivariate probability distribution of the device's side channel leakage while it operates on known sub-keys [GLRP06]. This phase of the attack is sometimes referred to as the *offline* phase.

The results of the profiling stage are used to classify future observations from a target device over which the adversary does not have full control, but can observe the side channel leakage. By classifying the new observed trace according to the distribution they are likely to come from, the most likely sub-key is revealed. Profiling techniques are considered very powerful because they require only a few (sometimes just one) observations to extract a key. Additionally, the attack phase does not require that the adversary control or have knowledge of the device inputs or outputs. The attack phase is sometimes referred to as the *online* phase.

Two specific variations of multi-stage attacks that have been proposed are the *template attack* [CRR02] and the *stochastic model* attack [SLP05]. *Template attacks* characterize the side channel leakage from the device by creating a template for each possible sub-key. The noise distribution is therefore assumed to be key-dependent. Thus, the profiling stage of a template attack creates covariance matrices for each possible (sub)key. Schindler et al.'s *stochastic model*, on the other hand, presumes the side-channel noise is independent of the (sub)key.

Gierlichs, et al. showed that while stochastic models have a lower up-front computational cost during profiling, they are less effective at classifying future signals [GLRP06]. Template attacks are the more effective tool if there are no limitations on the number of observations that can be collected during profiling and if the workload for template building is computationally feasible. Schindler's stochastic model may be more effective in cases where there is a bound on the number of observations that can be collected during profiling, or if other considerations mean constructing a full set of templates is not practical.

In 2005, Agrawal et al. extended template attacks to combine traditional DPA techniques with template attacks—termed a *template-enhanced DPA attack*. This technique is effective at defeating standard masking countermeasures against power analysis attacks on smart-card implementations of AES and DES produced by two separate manufacturers [ARRS05]. Recent work has shown in detail how template attacks can be used in practice to extract a cryptographic key without knowledge of the system inputs or outputs (plain- or cipher-texts) [HTM09] with only a few (< 200) observed operations.

A practical limitation of template attacks is the computational cost due to their reliance on multivariate statistics to characterize the dependencies among the various temporal locations in the leakage traces. For traces containing a large number of sample points, this step is very computationally intensive, and it may not be practical to consider all the sampled data. In practice, most of the published techniques

overcome this limitation by selecting a subset of the data points for use in the profiling / template building stage. Typically, some heuristic is used to select a subset of *points of interest* that correspond to the same relative times in each observation [MOP07, APSQ06].

Several rules of thumb have been used including selecting the subset of samples that correspond to the relative time where the observations show the maximal variance [MOP07] or maximal difference between mean traces when observations are partitioned according to a classical Kocher-style DSCA attack [CRR02]. Additional criteria are sometimes applied such as limiting the number of selected points to one per clock cycle to eliminate redundancy of the selected data points [APSQ06]. A primary objective is to compress the data set that must be manipulated while maintaining the most important information—which then takes on the role of characterizing the features of different signal classes. Recent research to find more optimal techniques for identifying leakage points is discussed below.

2.5.5 Advanced Techniques.

2.5.5.1 Principal Component Analysis. Principal component analysis (PCA) is a linear transformation that reduces the dimensionality of a data set while retaining the majority of the important information from the original data.[4] A well-known application of PCA is automatic facial recognition. The PCA transformation is such that each coordinate (component) is orthogonal to the others and is a linear combination of many individual samples. By using PCA to select the *points of interest* in profiling attacks, the n dimensions (components) that account for a largest percentage of the overall variance between the sample classes are identified. In template attacks, PCA maximizes the inter-class variance between possible operations (typically the same operation performed using each possible sub-key). The

[4]PCA and its variations are also known as the empirical/discrete Karhunen-Loève transform, the Hotelling transform, and proper orthogonal decomposition (POD) in various academic circles.

magnitude of the resulting eigenvalues associated with each component indicates the relative importance of the component, and normally a small number of principal components are sufficient to capture the majority (80-90%) of the inter-class variance.

Archambeault et al. introduced a systematic technique using PCA for selection of leakage points of interest that can subsequently be used to build templates as an improvement on previous heuristic techniques [APSQ06]. Archambeault experimentally demonstrated a case where 7 components chosen by PCA produce better classification results (93.3% vs. 91.8% correct) than a template built from 42 points using a difference of means heuristic. The paper does not, however, assess the computational efficiency of performing the combined PCA and template-building procedure to previous techniques using larger numbers of sample points for the template-building phase.

Archambeault's PCA technique has two notable advantages over heuristic techniques for selecting points of interest. First, it provides superior classification results with a smaller computational effort during the template-building phase although the computational load of the PCA transformation itself may negate this benefit. Secondly, PCA provides a quantitative measure of the number of components (transformed data points) to adequately capture the majority of the data's variability. A key assumption that has thus far resulted in effective attacks in practice is that the side-channel variability is a good indicator of the temporal location of information leakage.

A weakness of the PCA technique is that although the resulting subspace maximizes the *inter*-class variance between possible classes (sub-keys in most template attacks), the technique neglects the effect of *intra*-class variance on classifier performance. This is because the principal components are computed from the *mean* traces for each possible signal class (generally an operation performed for a given

sub-key in most SCA template attacks).[5] As a result, if there is a large degree of variance in the side-channel leakage for a single class (sub-key or intermediate value), it could result in degraded classifier performance.

2.5.5.2 Fisher's Linear Discriminant Analysis (LDA). Standaert and Archambeau proposed applying an alternate technique known as Linear Discriminant Analysis (LDA) to address the possible effects of intra-class variance on classifier performance [SA08]. LDA seeks a linear transformation of sample points into a subspace that maximizes the ratio between the inter-class distance and the intra-class variance *after* projection. Experimental data shows that LDA, as intuitively expected, provides a more optimal characterization of the target signal classes than does PCA. However, computation of the LDA solution is substantially more expensive than PCA, and thus limits the number of samples per observation that can be handled. Although more expensive, the authors demonstrated that LDA is an effective tool for dimensionality reduction for use in device profiling. In this work, an $n-$class variant of LDA known as Multiple Discriminant Analysis (MDA) is employed for dimensionality reduction in Chapter 3.

2.5.5.3 Higher-Order DSCA. The DSCA techniques described to this point are first-order DSCA attacks. The attack *order* refers to the number of samples from the trace that are simultaneously considered [MOP07]. If more than one sample is considered, then the attack is known as a *higher-order DSCA attack (HO-DSCA)*. A d^{th}-order DSCA attack is one which simultaneously considers d samples of the side-channel trace. HO-DSCA attacks were first proposed by Kocher in his original DPA paper, and have been investigated extensively since [KJJ99, Roh06, MOP07].

[5] PCA could be performed directly on the raw data, but instead of capturing the components of maximal variance between classes, it would result in components that capture the components of maximal variance across all data (irrespective of the classes).

The motivation for HO-DSCA attacks is to overcome *masking countermeasure* described in Section A.4. To be effective against masking, HO-DSCA considers d separate points that correspond to d different sensitive intermediate values, each of which is protected by the same mask. Theoretically, a d^{th}-order DSCA attack is capable of bypassing $(d-1)^{th}$-order masking scheme [CJRR99].

The precise time when the d sensitive intermediates are manipulated is generally unknown *a priori* by the adversary, although most of the available literature on HO-DSCA attacks assumes these locations are known [PRB09]. However, for practical implementation reasons, masking implementations usually generate two sensitive intermediates using the same mask in the first and last rounds. Therefore, in practice, attacking intermediates in the first and last round has been shown to be effective against many masking implementations [OMPR05, MOP07]. Frequency-domain signal processing techniques have also been proposed as a technique to identify the manipulation times [WW04]. In the context of profiling techniques, an adversary has control of a training device and can identify relative times by carefully profiling the device's behavior.

Once the times of the sensitive data manipulations are identified, the data is pre-processed to combine the multiple sensitive variables through some *combining function* (typically additive or multiplicative), which maps the multi-variate problem to a uni-variate problem [MOP07]. After pre-processing, the DSCA attack proceeds as normal.

Because of the practical issues involved in locating the sensitive manipulations and the complexity of carrying out such an attack is believed to increase exponentially with the attack order, HO-DSCA attacks are sometimes dismissed as impractical [MOP07, PRB09]. Oswald et al. published a practical attack on an 8-bit smart-card micro-controller protected by a 2^{nd}-order masking scheme with under 400 observations, without any *a priori* knowledge of the times when sensitive data would be manipulated [OMPR05]. Other recent work indicates sub-exponential complex-

ity growth attacks are possible [GBPV10]. Additionally, Mangard et al. found that hardware masking implementations are frequently easier to attack than software implementations because of the unintentional effects of parallel implementations that simultaneously process both random masks and masked values [MOP07].

2.5.5.4 Multi-channel Attacks. Agrawal et al. proposed combining information leakage from multiple simultaneous side channels [ARR03]. Their approach is a generalization of standard DSCA techniques and requires leakages from the combined channels be very "similar" at the times when the side channel signal is correlated to the underlying data or operations of interest. For the particular DES implementation studied, a multi-channel EM and power attack is significantly more efficient than either single-channel attack alone. The number of traces required to successfully extract the DES key using a maximum-likelihood based DPA attack is substantially reduced—in some extreme cases by more than 80%. The initial approach does *not* allow for combination of leakages that occur at different relative times in the side channel trace.

Standaert and Archambeau later extended the concept of a multi-channel attack to the creation of multi-channel templates [SA08]. In their technique, the power and EM side channel measurements (taken simultaneously during a single operation) are simply concatenated together to create a combined power / EM feature vector. Templates are then built normally, as described in Section 2.5.4. The authors found that the combined power and EM template attack performed significantly better than either technique alone (and that EM alone performed significantly better than power alone) for the 8-bit micro-controller evaluated.

2.5.5.5 Combination of SCA with Other Techniques. Although generally outside the scope of this research (with the exception of the algebraic cryptanalysis technique described below), a related research area combines SCA techniques with other attacks, including invasive or semi-invasive attacks [SA03, Roh06, Sko06]

and mathematical cryptanalysis techniques to augment SCA attacks (or vice versa) [Roh06]. For example, side channel information can be used to precisely time the injection of faults [Roh06].

For secure cryptographic algorithms, mathematical cryptanalysis is not a credible threat in itself since the required computational effort makes it impractical assuming the mathematical basis has no flaws. Implementation attacks such as side-channel attacks, on the other hand, exploit the vulnerabilities introduced by the physical realization of the cryptographic algorithm—which is not a pure black-box representation of the cipher. Even if an SCA attack is unsuccessful at extracting a full cryptographic key from a physical system, it may be successful at identifying a portion of the key or eliminating some class of possibilities. In such a case, the problem complexity may be reduced enough to make a brute force or other direct cryptanalytic attack practical [Roh06].

One of the most interesting and effective techniques published to date is the recent work by Renauld and Standaert to combine profiling SCA attacks with algebraic cryptanalysis [RSVC09]. Rather than focusing on a single intermediate value, algebraic SCA techniques use extensive profiling of a device to exploit the information leakage of as much information about the various intermediate values as possible. Standard SCA template-based techniques are used for the profiling and intermediate value extraction from the target device. The extracted intermediates become known values in an over-defined system of boolean equations that can be solved for the unknown key using an automated boolean satisfiability (SAT) solver. The results achieved using this technique are compelling, in that this appears to be the first technique to claim the ability to extract an AES key with a single target trace and no knowledge of plain- or cipher-text. The technique currently depends on very accurate extraction of the intermediate values used in the solution, although the authors suggest several ways this dependency can be reduced in practice.

2.6 Countermeasures

All SCA countermeasures have a common objective—the elimination of discernible linkages between sensitive internal circuit behavior (data manipulation and operations) and one or more external, physically observable phenomena. In practice, completely eliminating information leakage from a single side channel is an extremely difficult problem. The problem becomes more difficult when all possible side channels are considered. Various researchers have asserted that it is fundamentally impossible to make the side channel emissions of a device completely independent of the underlying computations [MOP07, Roh06].

As is the case when security measures are implemented for any type of system, all of the countermeasures discussed in this section carry with them associated implementation costs. Typical costs are slower performance, increased circuit size, increased power consumption, and/or increased design time and expense. For this reason, in the complex trade-space of real systems, designers may focus on the most sensitive or critical circuit areas, and attempt to prevent information leakage only from those subcomponents. In practice, system designers and architects balance side-channel leakage resistance and other security measures with required system cost and performance. Rather than attempting to build an impenetrable system, a suitable goal is a design secure enough to deter would-be adversaries from applying the necessary resources to defeat it. For a rational adversary, this is achieved in practice if the costs of carrying out an attack outweigh the perceived benefits of defeating the system's protections.

At the highest level, countermeasures are classified as either *procedural* or *physical design* countermeasures. *Procedural* countermeasures include security practices put in place to improve the security of a system. For example, a system architect may understand that a device is vulnerable to various SCA attacks. Knowing this, the architect may impose a restriction on the number of times a unique key can be

used before it must be changed so that if a particular key is compromised, it will not negatively effect the security of future transactions [And01, CJRR99].

Physical design countermeasures can be sub-divided into two primary approaches: *reduce the signal* and *increase the noise*. Both approaches degrade the signal-to-noise ratio of information leakage during intermediate computations to the point where it becomes impractical (in terms of number of required observations or pre- and post-processing time) to extract information of concern from the physically observable phenomenon. The actual implementation of either of these approaches can take place at various levels of system design such as the protocol, system architecture, operating system, software (algorithm) or hardware levels [RO04a].

For each of the two general approaches, a large number of specific techniques have been proposed. Most counter a specific side-channel threat, e.g., power or timing information leakage since they have been the subject of the majority of the SCA research to date. Notably, no perfect countermeasure exists to date, and most of the proposed countermeasures still have significant SCA vulnerabilities. Some of the most popular and promising countermeasures in theory have been shown to be inadequate in practice soon after their introduction. Likewise, some widely used countermeasures such as masking can be broken almost completely in the context of profiling scenarios where the adversary has access and full control of a separate training device [OM07]. However, in a well-designed system the countermeasures may be sufficient as part of an integrated *defense-in-depth* strategy to deter less determined adversaries—or at least be sufficient to protect a cryptographic key or secure device for its useful life [CJRR99].

The principle SCA countermeasures that have been proposed to date are described in Appendix A.

2.7 Practical Considerations

Numerous publications have documented the nuances of implementing side channel attacks in practice, with the book by Mangard et al. being the most thorough treatment [MOP07]. This section briefly reviews techniques for experimentally collecting side channel data and pre- and post-processing of the data to align traces. Each side channel attack is unique and dependent on the specific device implementation being targeted. However, the approach outlined below is mostly generic and should apply to most situations.

There are several ways side channel data can be experimentally collected. The experimental setup used in this research to collect EM data is described in Section 3.11.1. The most common technique for capturing power consumption data is for an attacker or security evaluator to insert a small resistor (1-100 Ohms) in series with either the primary power supply line or the ground line, and sample the voltage difference across the resistor using a digital oscilloscope [MOP07]. If the resistor size is too small, it makes measuring the voltage drop difficult, and if it is too large it may cause the circuit to malfunction. The time-varying voltage is proportional to the current drawn from the power supply, and to the power consumed by the circuit (thus the name power analysis). The procedure is semi-invasive since it requires circuit or power supply modifications. A typical power analysis setup is shown in Fig. 2.5.

An alternate technique preferred by some is to use a contact-less current probe to collect the power consumption data. This technique is less invasive, and only requires passing the power supply line through the measurement device. However, Mangard et al. state that such a setup will have lower sensitivity than the former direct measurement technique.

A block diagram of a typical experimental setup, representative of the setups used for this research, is shown in Fig. 2.6. The steps involved in this experimental setup are [MOP07]:

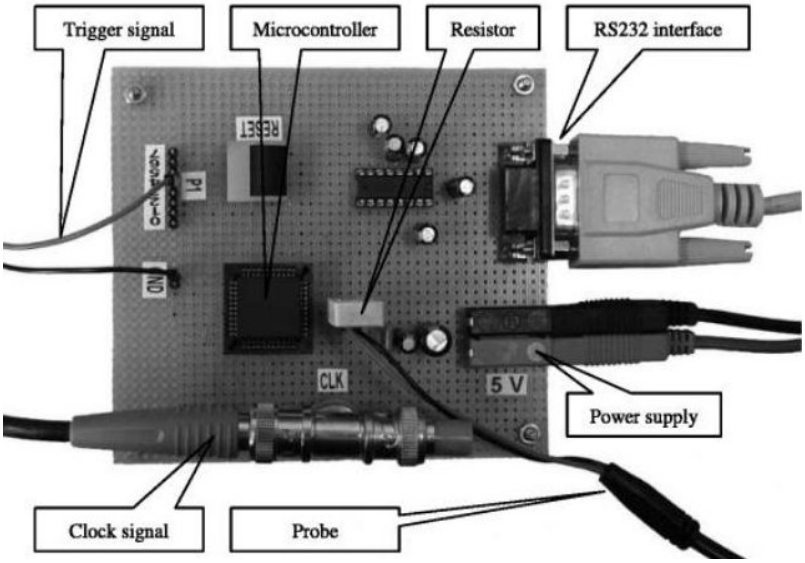

Figure 2.5 A typical power analysis experimental setup [MOP07]

1. Supply the cryptographic device with power and a clock signal.
2. Configure and arm the oscilloscope from the controlling PC.
3. Command the target cryptographic device to begin execution and trigger data collection by the oscilloscope.
4. Digitally sample the voltage variations across the resistor.
5. Collect the output of the cryptographic operation.
6. Retrieve the recorded side-channel data.

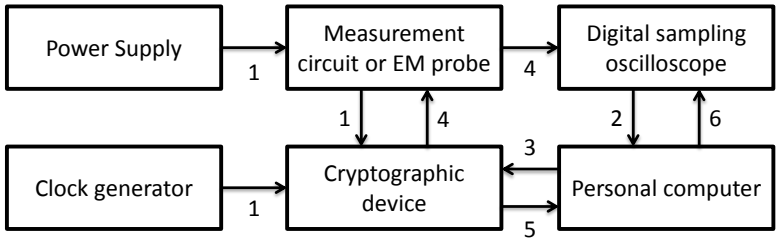

Figure 2.6 Block diagram of typical experimental setup for side-channel analysis (adapted from [MOP07]).

Even if the target system has its own source, it may be more effective to substitute more precise components to provide the power and clock signal. The data collection technique may be adapted by the attacker to meet the unique objectives of a particular SCA attack against the system of interest—which may be driven by, for example, what countermeasures the system has implemented and the accessibility of any internal circuitry. Modern chips often have several power and ground pins as well as complex power distribution networks with filtering capacitors that reduce the effectiveness of power analysis attacks. To achieve better results, it may be necessary to target a specific pin rather than attempting to perform power analysis on the noisy global power source.

Quisquater and Samyde were the first to report application of DPA-like experiments on EM emissions [QS01]. Their initial collection were taken from a circuit using a small (less than 2 cm) diameter flat coil of copper wire attached to a digital oscilloscope as the measuring device. The original experiments were conducted inside a Faraday cage to minimize received noise from external sources of electromagnetic energy. However, most recent EM attacks have reported minimal noise as a result of conducting EM measurements even without such a device. The work of Quisquater is also the first to report using a motorized table to precisely position the EM probe over the device under test. Using this technique, they demonstrated the ability to create a three-dimensional map of the EM leakage (magnitude) produced by the device. Similar capabilities, including the Riscure Inspector system used for this work [Ris09], are now commercially available.

2.7.0.6 Signal Processing Techniques. In practical SCA attacks, signal alignment and countermeasures can make straightforward application of the DSCA techniques impossible or at least very time consuming. In some cases, the collected signals must be pre- or post-processed to make an attack possible. Signal (mis-)alignment can severely impact the results of a DSCA attack, and several proposed countermeasures intentionally induce temporal misalignment. If all of the

collected side channel traces do not begin at precisely the same relative time, noise is introduced into the process, which requires many more measurements to mount a successful attack. In experimental setups, initial alignment is achieved by asserting a signal that tells the digital oscilloscope to begin collecting data at the instant the ciphering operation begins. In real-world applications of SCA techniques, there is obviously no trigger signal present. Certain commercial systems (e.g., the icWaves subsystem of the Riscure Inspector side channel analysis system) can recognize, in real-time, the signal features that indicate an encryption operation of interest is beginning [Ris09]. These systems obtain traces that are nominally aligned at the start of the encryption operation. Mangard et al. discuss a number of techniques that can be used to statically align trace data when the digital sampling does not begin at the precise same time [MOP07].

Several techniques can overcome temporal desynchronization countermeasures. Mangard et al. discuss a number of techniques to align trace data [MOP07]. Akkar et al. suggested pre-processing signals where temporal desynchronization is present and normalizing the individual traces by stretching or compressing the samples within each clock period [ABDM00]. After pre-processing, DSCA techniques are applied as normal. One technique for accomplishing this was introduced by Woudenberg under the name *elastic alignment* [vWWB11]. Some investigators have also indicated that simply pre-processing the captured traces to convert time-domain signals to the frequency domain (via Fast Fourier Transform) and performing all subsequent analysis in the frequency domain will significantly reduce any impact of misalignment [RO04b, GHT05, PHF09].

Clavier introduced an attack variation that overcomes the randomization of operation duration, which he termed *sliding window* DPA [CCD00]. The attack works by conducting a DPA attack as normal, and post-processing the resulting statistical measure (e.g., correlation coefficients) over a fixed window of time, effectively

restoring the amplitude of the characteristic peak values which indicate a correct key hypothesis.

Numerous other applications of signal processing have been investigated. Since SCA is, in essence, a signal detection/estimation/classification problem, it is likely that well-known signal processing techniques from other fields (communications systems, biomedical signal processing, etc.) can be adapted for SCA.

2.8 Summary

This chapter summarized the relevant work and technical background necessary to the study of side channel analysis and information leakage from integrated circuits. This information supports the subsequent material presented in the remainder of this document.

3. *Intrinsic Physical Layer Authentication of Integrated Circuits*

This chapter contains the text of an article that was submitted and accepted for publication to the Institute of Electrical and Electronic Engineers (IEEE) *Transactions on Information Forensics and Security* [CLB+11]. This article was co-authored by Mr. Eric Laspe, Dr. Rusty Baldwin, Dr. Michael Temple, and Dr. Yong Kim.

3.1 Abstract

RF distinct native attribute (RF-DNA) fingerprinting is adapted as a physical layer technique to improve the security of integrated circuit (IC)-based multi-factor authentication systems. Device recognition tasks (both identification and verification) are accomplished by passively monitoring and exploiting the intrinsic features of an IC's unintentional RF emissions without requiring any modification to the device being analyzed. Device discrimination is achieved using RF-DNA fingerprints comprised of higher-order statistical features based on instantaneous amplitude, phase and frequency responses as a device executes a sequence of operations. The recognition system is trained using Multiple Discriminant Analysis to reduce data dimensionality while retaining class separability, and the resultant fingerprints are classified using a linear Bayesian classifier. Demonstrated identification and verification performance includes average identification accuracy of greater than 99.5% and equal error rates of less than 0.05% for 40 near-identical devices. Depending on the level of required classification accuracy, RF-DNA fingerprint based authentication is well-suited for implementation as a countermeasure to device cloning, and is promising for use in a wide variety of related security problems.

3.2 Introduction

Physical implementation attacks on secure electronic systems have evolved rapidly in the past few years making it increasingly difficult for new countermea-

sures and security practices to keep pace [BMV05, TJ09]. In contrast to mathematical cryptanalytic attacks which are typically hypothetical in nature, implementation attacks present a serious and immediate threat since the strength of the underlying algorithms and protocols is rendered largely irrelevant. Examples of implementation attacks range from complex techniques requiring expensive and highly specialized equipment (e.g., laser fault injection or focused ion beam manipulation) to surprisingly simple, low-cost attacks targeting the unintentional information leakage produced by devices during normal operation (e.g., simple power analysis) [BMV05, TJ09].

An extensive body of academic and commercial research has been dedicated to examining the physical security of cryptographic and other secure devices. This work has emerged in the last decade under the titles *side channel analysis* and *fault analysis* (cf. [BMV05, AARR02, KJJ99, ZF05]). Given that many implementation attacks are well within the reach of even modestly funded and minimally equipped individuals, they should be given serious practical consideration when designing modern systems. A prudent design approach is to 1) assume that secure tokens or other essential system components are subject to counterfeiting, cloning, or sensitive data extraction, and 2) take appropriate steps to mitigate the associated risks as part of an integrated, multi-tiered system security architecture.

RF "distinct native attribute" (RF-DNA) fingerprinting (cf. [SITMM08,KTM09, RTM10,RPT11,HBK06,WMTM10,WTR10,CGB$^+$10]) is adopted herein as a way to augment existing multi-factor authentication schemes via physical layer authentication at the device level to counter cloning and related threats. The term RF-DNA is used to embody the coloration of RF emissions (both intentional and unintentional) induced by the intrinsic physical attributes of a unique device. Only RF emissions produced by *unintentional* emitters such as integrated circuits (ICs) are considered in this study.

Using the RF-DNA approach, semiconductor-based IC devices are passively recognized based on discriminating features (*RF-DNA fingerprints*) extracted from their intrinsic physical properties in a manner analogous to biometric human identification. Because this technique exploits emissions caused by intrinsic inter-device variability, it is suitable for a variety of security applications involving commodity commercial ICs, and does not require any physical device modifications. Moreover, our initial results indicate the technique can be adapted to work with existing processes and protocols, and is likely suitable for use with a wide variety of IC devices, e.g., general purpose microcontrollers, programmable logic devices such as FPGAs, and custom ASICs. To our knowledge, the work presented here and in [CGB+10] is the first to propose using the intrinsic DNA of *unintentional* emissions for IC recognition.

This work makes a number of distinct contributions while expanding on the initial proof-of-concept results in [CGB+10] with an extensive empirical evaluation. In particular, previous RF-DNA work has predominantly considered device *identification* tasks. However, the primary use case envisioned for IC fingerprinting is to counter cloning and related threats, which requires identity *verification*. Herein, a systematic approach is developed and introduced to evaluate RF-DNA fingerprinting effectiveness in the context of both identification and verification tasks for arbitrary n-class problems. Performance of the proposed technique is evaluated under a wide range of simulated noise conditions, and empirical results are presented indicating the RF-DNA technique performance scales well for identification and verification tasks involving 40 near-identical devices.

3.3 Problem Definition

This work assesses the suitability of RF-DNA fingerprinting for two distinct, but closely related device recognition tasks: *identification* and *verification*. These

tasks are analogous to human recognition tasks for which biometric-based pattern recognition systems are frequently used [JRP04]:

1. *Device identification.* The recognition system *determines* a device's identity by comparing a captured device fingerprint with reference fingerprint templates for all known devices. Identification requires a *one-to-many* comparison and is considered more difficult than verification.
2. *Device verification.* The recognition system *checks* the authenticity of a device's claimed identity (by virtue digital credentials presented) using a *one-to-one* comparison. As with biometric verification, the objective of physical layer device verification is to prevent two devices from using the same identity.

Previous RF-DNA fingerprinting work predominantly focused on the *one-to-many* identification task in the context of wireless network security, where a device entering a network needs to be verified as belonging to a pool of authorized devices. However, detection of cloned security tokens such as smart-card based identification cards or payment devices, requires *one-to-one* verification that the claimed identity of the device matches its physical fingerprint. Herein, the suitability of RF-DNA fingerprints for use by physical layer device recognition systems is assessed for both identification and verification tasks.

3.4 Notional Physical Layer Device Authentication System Design

This section describes a system design for applying RF-DNA fingerprinting to the device identification and verification problems described above. The basic design is modeled after a typical biometric system [JRP04], and includes five modules:

- *Sensor module.* The sensor module captures unintentional RF emissions, and is composed of an RF receiver and a near-field probe. Details of the particular sensor module used to obtain the experimental data are described in Sec. 3.8.2.

- *Feature extraction module.* Features are generated based on the statistical behavior of one or more instantaneous response(s) within pre-defined fixed signal regions. Both the device enrollment and the feature matching processes use the feature extraction module to generate a statistical fingerprint for each captured signal as described in Sec. 3.7.
- *Classifier training module.* Extracted features are post-processed using dimensionality reduction and probability density estimation techniques to generate a *reference template* for each enrolled device.
- *System database module.* As each device is enrolled, the set of training fingerprints and associated reference template are stored in a verification database. As each device is issued or associated with a particular digital identity, the database is updated to reflect the pairing (e.g., *device A1 belongs to John Smith*).
- *Classification / feature matching module.* For recognition tasks, one or more fingerprints are extracted from the presented device. The extracted fingerprint(s) are compared to the stored reference templates in the system database to either identify the device or verify its presented identity.

The basic steps involved in fingerprinting each device at enrollment are:

1. Command the device to execute a short pre-defined sequence of operations (the *challenge*).
2. Capture the unintentional near-field RF emissions produced by the device as it executes the operation sequence (the *response*).
3. Extract discriminating features from the captured emissions to produce an *RF-DNA fingerprint.*
4. Repeat the above steps N_{FP} times to obtain a set of *training* fingerprints.
5. Process the set of extracted training fingerprints to generate a *reference template* for each enrolled device.

6. Repeat the above steps for each of N_{CR} challenge-response sequences.
7. Store each set of training fingerprints and generated reference template in a database.
8. When the physical device is issued to an individual or paired with a set of digital credentials, update the database to associate the stored reference template with those digital credentials.

In the context of a cryptographic *challenge-response* system, the command to execute a particular operation sequence is the *challenge* and the unintentional RF emissions produced by the device while executing the sequence is the *response*.

The operation sequence used can be composed of any fixed, repeatable process. For general purpose microcontrollers, the operation sequence would be a series of microcode instructions (e.g., some combination of loop, branch, control, or arithmetic commands) on known (fixed) data. For programmable logic devices or ASICs, the sequence could be composed of several clock cycles where different combinational logic paths are activated, starting from a known fixed configuration. When possible, device configuration (clock rate, on-chip peripheral status, etc.) should be held or reset to a known state prior to beginning the operation sequence.

After enrollment, subsequent device recognition tasks are performed by repeating the challenge-response protocol to obtain an authentication fingerprint. The authentication fingerprint is processed by the classification / feature matching module for identification or verification. Although the experiments conducted herein are limited to a single challenge-response sequence ($N_{\mathrm{CR}} = 1$), extension of the approach to an arbitrary number of challenge-response pairs is straightforward.

An explicit RF-DNA challenge-response procedure is unnecessary for many envisioned applications because the RF-DNA fingerprinting procedure can be be piggybacked on existing protocols. For example, it is typical to initiate communications with smart-cards by issuing a reset command to the card and parsing the resulting

answer to reset (ATR) response. A straightforward RF-DNA implementation might generate a fingerprint from a small portion of the ATR response. Other opportunistic responses for fingerprinting of smart-card authentication include PIN verification or credential retrieval. Similar opportunistic approaches can be envisoned for identification and verification tasks based on microcontrollers, programmable logic devices such as FPGAs, and ASIC-based devices.

3.5 Unintentional RF Emissions of ICs

It is common knowledge that electronic equipment radiates electromagnetic (EM) energy that can interfere with nearby devices. It is for this reason that airline passengers are required to "turn off all portable electronic devices," and consumer electronics undergo certification testing for compliance with Federal Communications Commission (FCC) [FCC09] or other regulating standards. Digital devices that incorporate high frequency clocks, oscillators, etc. are specifically regulated as known unintentional emitters and require strict testing to ensure emissions do not exceed tolerable levels.

Variations in current flow through a device due to clock distribution, transistor switching, and other IC component activity produce EM fields that combine through complex interactions and propagate via both radiation and conduction in the form of time-varying EM waves. The fundamental nature of these effects is well understood and is described by Maxwell's equations [AARR02].

In the past decade there has been a growing realization that unintentional emissions are not only a source of interference, but also a useful source of information about the internal state of the emission producing device [KJJ99, AARR02, BMV05, ZF05, PEK+09]. This has had profound implications for the physical security of sensitive electronic systems since in many instances the *leaked* state information is sufficient to infer precise details about the operations the device is performing and/or the data it is processing. More recently, it was shown in [CGB+10] that in addition

to data and operation-dependent characteristics, the unintentional near-field RF emissions of individual ICs also exhibit significant device-dependent characteristics.

The most likely source of inter-device emission variability is the random process variations introduced during die fabrication and packaging [Ver10]. Although IC fabrication processes are necessarily precise, structural variations are still introduced in the final device structure on a very small scale (deep sub-micron in modern IC technology). As a result, no two chips are exactly alike. As long as process-induced variations are within acceptable tolerances, the device will operate correctly from a black-box functional perspective.

For this study, the hypothesis is that the fabrication process-induced variations in each individual chip's electrical properties color the unintentional RF emissions from the circuit, and that the resultant coloration is sufficient to uniquely identify the emissions' source. Although only unintentional *RF emissions* are studied herein, the approach used is believed to be applicable to other side channel emissions such as variations in the power consumption of the device.

3.6 Related Work

Various methods have been proposed to use the uniqueness of inter-device process variations to enhance security. Previously proposed physical layer device recognition techniques include physical unclonable functions (PUFs) [Ver10, PRTG02], RF certificates of authenticity (RF-COAs) [DK07], and the exploitation (i.e, RF fingerprinting) of unique signal coloration within *intentional* emissions produced by wireless networking [SITMM08, KTM09, RTM10, HBK06, RPT11, DC09] and RFID-based devices [DHBČ09].

3.6.1 Physical Unclonable Functions (PUF). PUF techniques refer to two distinct approaches for device authentication. The first augments an IC with specialized internal measurement circuitry that computes a one-way function from glitch

counts, propagation delays, or other electrical properties that vary randomly with intrinsic process variations of the IC [Ver10, MKP09]. A second approach combines a grid of capacitive sensors integrated into the top metal layer of the IC with a conformal coating doped with randomly distributed dielectric particles applied on top of the IC's passivation layer. The conformal coating requires active interrogation (i.e., application of a specified voltage with known amplitude and frequency) and internal measurement of the response by the circuit [Ver10].

3.6.2 RF Certificates of Authenticity (RF-COA). The RF-COA technique attaches a three dimensional constellation of small randomly shaped conductive or dielectric objects to an RFID device. This is similar to a PUF coating except that both the interrogation and response measurement are carried out by an external RFID reader. The RFID reader is modified to include a dense matrix of patch antennae to transmit and receive high-frequency RF signals. The reader interrogates a modified RFID object and extracts a fingerprint to compute a COA [DK07].

3.6.3 RF-DNA Fingerprinting. RF fingerprinting has been proposed as a physical layer technique to enhance the security of various wireless communications devices (e.g., RFID [DHBČ09], 802.11 WiFi [RPT11], 802.15 WPAN [HBK06,DC09], 802.16 WiMAX [WMTM10], GSM [WTR10]. The device fingerprinting methodology developed herein is specifically based on previous RF-DNA work in [SITMM08, KTM09, RTM10]. The preliminary results in [CGB$^+$10] confirmed that the general approach was promising for recognition of ICs based on their *unintentional* emissions.

A significant advantage of RF-DNA fingerprinting compared to PUF and RF-COA techniques is its applicability to the authentication of any commodity IC without modifications to the internal circuitry or application of an external coating. Additionally, measurement of the device *response* is passive and does not require a transmitter.

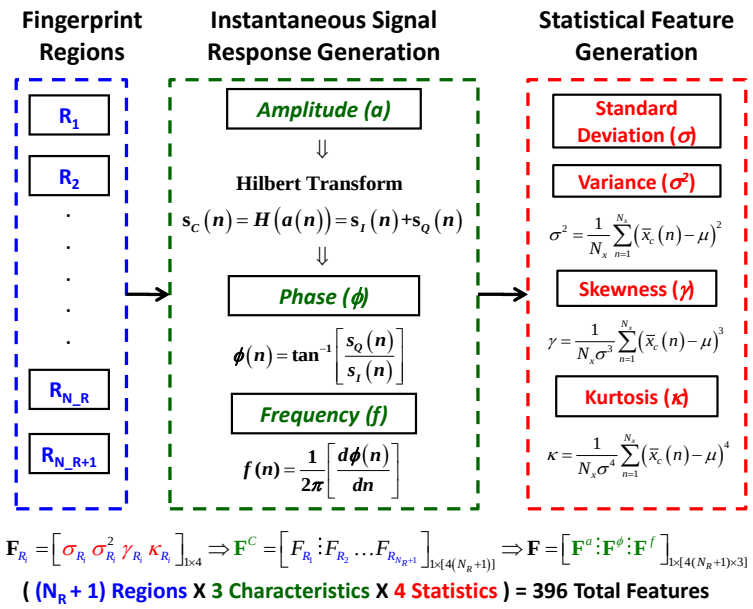

Figure 3.1 RF-DNA statistical fingerprint generation process.

To implement RF-DNA fingerprinting, the sensor module described in Sec. 3.4 would be integrated into the terminal or device reader. This approach is believed to be reasonable given that space and power constraints are generally less restrictive in a reader (e.g., smart-card reader or ATM machine) than in the secure token or device itself.

3.7 RF-DNA Fingerprint Generation and Classification

3.7.1 RF-DNA Feature Extraction and Statistical Fingerprint Generation.

The statistical fingerprint generation methodology used herein is based on [SITMM08] with modifications made to 1) extend the process from the limited 3-class to general N-class problems, and 2) to enable both identification and verification device recognition tasks. RF-DNA statistical fingerprint feature vectors, $\mathbf{F}$, are extracted from real-valued time-domain samples based on the statistical behavior of instantaneous signal response(s) within pre-selected response regions. The complete fingerprint extraction and generation process is shown in Fig. 3.1.

Three instantaneous signal responses are generated from the real-valued time domain samples: instantaneous amplitude (IA) given by $a(n)$, instantaneous phase (IP) given by $\phi(n)$, and instantaneous frequency (IF) given by $f(n)$. To calculate $\phi(n)$ and $f(n)$, the real-valued signal samples are first converted to I–Q samples, $s_{C(n)} = s_{I(n)} + s_{Q(n)}$, using a Hilbert transform [Lyo04]. The IP samples are calculated using

$$\phi(n) = tan^{-1}\left[\frac{s_{Q(n)}}{s_{I(n)}}\right], \tag{3.1}$$

with the corresponding IF (Hz) given by

$$f(n) = \frac{1}{2\pi}\left[\frac{d\phi(n)}{dt}\right]. \tag{3.2}$$

The resultant IA and IF responses are "centered" using the amplitude (μ_a) and frequency (μ_f) means to remove potential collection biases

$$a_c(n) = a(n) - \mu_a, \tag{3.3}$$

$$f_c(n) = f(n) - \mu_f. \tag{3.4}$$

Finally, the centered responses in (3.3) and (3.4) are normalized by their respective maximum magnitudes to compensate for power variation.

3.7.1.1 Statistical Fingerprint Generation. After centering and normalization, $N_f = 4$ statistical features are generated within each selected region of each instantaneous response: standard deviation (σ), variance (σ^2), skewness (γ)

and kurtosis (κ) [SITMM08]. For an arbitrary centered and normalized sequence $\{\bar{x}_c(n)\}$ having N_x samples, these features are

$$\sigma^2 = \frac{1}{N_x}\sum_{n=1}^{N_x}\left(\bar{x}_c(n) - \mu\right)^2, \tag{3.5}$$

$$\gamma = \frac{1}{N_x\sigma^3}\sum_{n=1}^{N_x}\left(\bar{x}_c(n) - \mu\right)^3, \text{ and} \tag{3.6}$$

$$\kappa = \frac{1}{N_x\sigma^4}\sum_{n=1}^{N_x}\left(\bar{x}_c(n) - \mu\right)^4, \tag{3.7}$$

where standard deviation (σ) is $\sqrt{\sigma^2}$.

For all results presented in Sec. 4.6, each statistic was calculated over $N_R = 32$ equal length, contiguous sub-regions spanning a total region of interest (ROI). The ROI was empirically selected from the collected signal and contains $N_{CL} = 32$ clock cycles of device operations. Extensive pilot studies confirmed that for the parameter combinations studied, partitioning samples into sub-regions corresponding to integer multiples of the number of clock cycles in the ROI yielded statistically superior results relative to partitioning based on fractional clock cycles. The full ROI encompassing all $N_{CL} = 32$ clock cycles was used as an additional "total" region giving $(N_R + 1) = 33$ total regional contributions for each device. Fig. 3.2 illustrates the sub-region allocation process used herein.

For each subregion and the "total" region, the four statistics are concatenated to form a regional *RF-DNA marker vector*

$$F_{R_i} = \left[\sigma_{R_i}\ \sigma^2_{R_i}\ \gamma_{R_i}\ \kappa_{R_i}\right]_{1\times 4}, \tag{3.8}$$

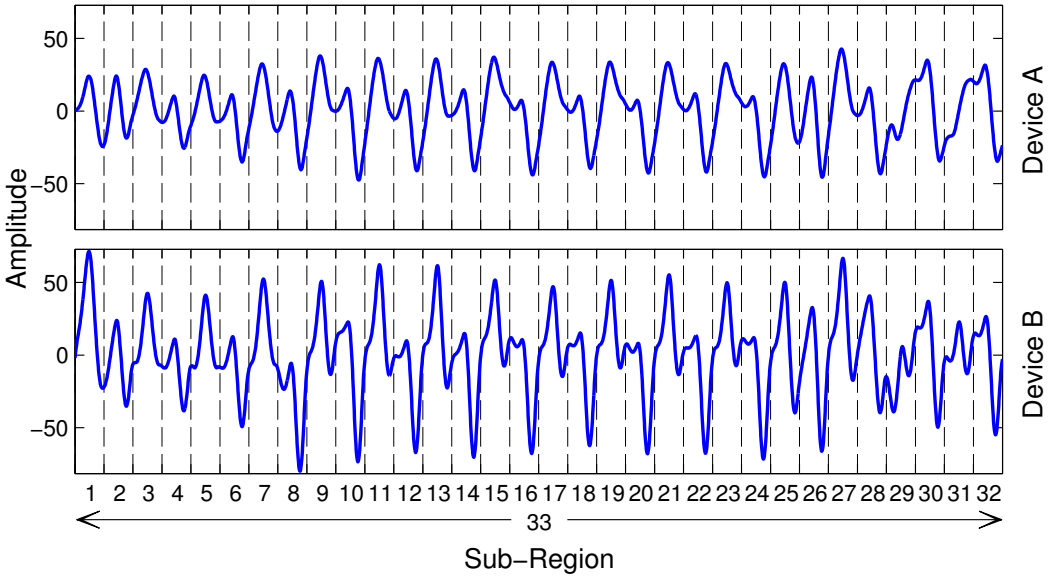

Figure 3.2 Mean amplitude response of signals collected for all devices from part numbers A and B (see Tab. 3.2). The x-axis shows sub-region allocation using $N_R = (N_{\text{CLK}} + 1) = 33$ sub-regions for calculation of RF-DNA statistical fingerprints as described in Sec. 3.7.1.1.

where $i \in \{1, 2, \ldots, (N_R + 1)\}$. The RF-DNA marker vectors (3.8) are concatenated to form a *composite characteristic vector* for each selected characteristic

$$\mathbf{F}^C = \left[F_{R_1} \vdots F_{R_2} \vdots F_{R_3} \ldots \; F_{R_{N_R+1}} \right]_{1\times[4(N_R+1)]}, \tag{3.9}$$

where the superscripted C denotes a specific characteristic response, i.e., a, ϕ, or f. Considering IA, IP, and IF the final statistical fingerprint for each signal is a vector of $N_f \cdot (N_R + 1) \cdot N_i = 4 \cdot 33 \cdot 3 = 396$ total elements, or

$$\mathbf{F} = \left[\mathbf{F}^a \vdots \mathbf{F}^\phi \vdots \mathbf{F}^f \right]_{1\times 396}. \tag{3.10}$$

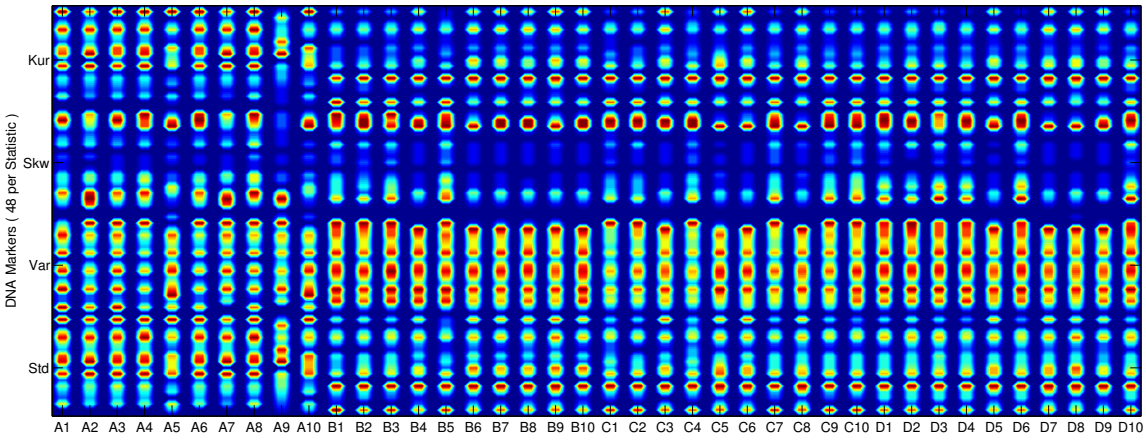

Figure 3.3 Average of 500 RF-DNA fingerprints for each of the 40 tested devices. Fingerprints are composed of statistical features extracted from $N_{CL} = 32$ clock cycles (at collected SNR). The x-axis is the alpha-numeric designator for each chip (see Table 3.2).

Finally, the *training matrix* composed of N_{FP} separately collected statistical fingerprints is

$$\mathbb{F}_T = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_{N_{\text{FP}}} \end{bmatrix}_{N_{FP} \times 396}, \tag{3.11}$$

which is input to the classification training process described in Sec. 3.7.2.

The RF-DNA fingerprints shown in Fig. 3.3 illustrate *intra-* and *inter*-part number variability in the statistical features generated for 40 unique devices. For visual clarity, the RF-DNA markers in $\mathbf{F}$ are scaled, compressed and/or expanded, and quantized such that the plotted data spans the interval $[0, 1]$ within each statistic. The quantized markers are stacked vertically to create an electrophoresis-like plot.

The RF-DNA plot is helpful in developing an intuitive understanding of which statistical features exhibit the most variance both within and across part numbers. It

is expected that the classifier will have the greatest difficulty distinguishing between parts that have visually similar RF-DNA fingerprints. For example, the RF-DNA plots for devices $A4$, $A6$, and $A8$ appear to be very similar across all features. These devices are expected to be confused more frequently than devices that look substantially different across one or more statistical features. Device $A9$, on the other hand, appears quite unique and would generally be expected to be less confused with other devices. Likewise, the large apparent differences between the Class A chips and the chips in all other classes implies that a classifier should be able to distinguish Class A chips from the other classes more easily.

3.7.2 Classifier Training. Consistent with previous RF-DNA fingerprinting work, training of the classification system is accomplished using multiple discriminant analysis (MDA) to reduce feature dimensionality and improve class separability. MDA is an extension of Fisher's (two-class) linear discriminant analysis (LDA) to N-classes that linearly transforms the sample points into an $(N-1)$-dimensional subspace without reducing the class separability power [TK09]. The MDA projection maximizes the ratio between inter-class distance and intra-class variance. For all results presented herein, input fingerprint data is projected from the original 396-dimensional data into a compressed $(N_D - 1) = 39$-dimensional space.

Given input data matrix $\mathbb{X}$, the MDA transformation first finds the within (intra-) ($\mathbb{S}_w$) and between (inter-) ($\mathbb{S}_b$) class scatter matrices [TK09]

$$\mathbb{S}_w = \sum_{i=1}^{N_D} P_i \Sigma_i, \tag{3.12}$$

$$\mathbb{S}_b = \sum_{i=1}^{N_D} P_i \left(\boldsymbol{\mu_i} - \boldsymbol{\mu_0}\right) \left(\boldsymbol{\mu_i} - \boldsymbol{\mu_0}\right)^T. \tag{3.13}$$

where Σ_i is the covariance matrix for class C_i, and P_i is the prior probability of class C_i (assumed to be equal for all classes).

Projection matrix $\mathbb{W}$ is formed from the $(N-1)$ eigenvectors of $\mathbb{S}_w^{-1}\mathbb{S}_b$. The formation of $\mathbb{W}$ optimally maximizes the ratio of inter-class distance to intra-class variance [TK09]. Individual fingerprints are projected onto the $(N-1)$-dimensional MDA space by

$$\mathbf{F}_i^{\mathbb{W}} = \mathbb{W}^T\mathbf{F}. \tag{3.14}$$

For all results presented herein, the full MDA-projected fingerprint training matrix, $\mathbb{F}_T^{\mathbb{W}}$ is a combination of N_{FP} MDA-projected training fingerprints, each with $(N_D - 1) = 39$ elements

$$\mathbb{F}_T^{\mathbb{W}} = \begin{bmatrix} \mathbf{F}_1^{\mathbb{W}} \\ \mathbf{F}_2^{\mathbb{W}} \\ \vdots \\ \mathbf{F}_{N_{\text{FP}}}^{\mathbb{W}} \end{bmatrix}_{N_{FP}\times 39}. \tag{3.15}$$

Classifier training is completed by fitting a multivariate normal distribution to the MDA-projected data and saving the estimated distribution parameters (mean vector, $\widehat{\boldsymbol{\mu}}_{\boldsymbol{i}}^{\mathbb{W}}$, and covariance matrix, $\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{i}}^{\mathbb{W}}$) for each class. For the *linear* Bayesian classifier used herein, a pooled estimate of the covariance matrix is used in lieu of individual covariance matrices for each class [TK09]. Note these parameters are for the distribution fit to the training data *after* MDA projection, as indicated by the superscripted $\mathbb{W}$.

The complete training process produces the following:

- MDA projection matrix ($\mathbb{W}$)
- N_D sets of MDA-projected training fingerprints ($\mathbb{F}_{c_i}^{\mathbb{W}}$) (one set for each device)
- N_D estimated mean vectors ($\widehat{\boldsymbol{\mu}}_{\boldsymbol{c_i}}^{\mathbb{W}}$)
- A pooled estimate of the covariance matrix ($\widehat{\boldsymbol{\Sigma}}^{\mathbb{W}}$)

The pair of parameters composed of the mean vector and covariance matrix for each device is the *reference template* for that device.

3.7.3 Fingerprint Classification. As described in Sec. 3.3, RF-DNA statistical fingerprints can be used for both identification and verification tasks. For both applications, the unintentional RF emissions of a target device are collected and the relevant statistical features are extracted and projected into the MDA space using (3.14) to generate a projected device fingerprint ($\mathbf{F}^{\mathbb{W}}$). The classification technique and performance measures used to evaluate system performance are different for each application and are described below.

3.7.3.1 Device Identification. For device *identification*, a captured fingerprint is compared to the reference templates of all devices in a verification database (one-to-many comparison) using some similarity criterion. The identity of the target device is determined by computing the similarity score for each comparison and selecting the best match. The similarity measure used herein is the Bayesian posterior probability under assumptions of equal prior probabilities and equal costs. This approach is optimal for the minimization of classification error probability [TK09].

Stated formally, for the case with N_D devices, a device fingerprint $\mathbf{F}^{\mathbb{W}}$ is assigned to class ω_i, where $i \in \{1, 2, \ldots, N_D\}$ if

$$P\left(\omega_i \,|\, \mathbf{F}^{\mathbb{W}}\right) > P\left(\omega_j \,|\, \mathbf{F}^{\mathbb{W}}\right) \quad \forall j \neq i, \tag{3.16}$$

where $P\left(\omega_i \,|\, \mathbf{F}^{\mathbb{W}}\right)$ is the conditional *posterior probability* that $\mathbf{F}^{\mathbb{W}}$ belongs to class ω_i. According to *Bayes' rule*, the conditional probability is [Mac03]

$$P\left(\omega_i \,|\, \mathbf{F}^{\mathbb{W}}\right) = \frac{P\left(\mathbf{F}^{\mathbb{W}} \,|\, \omega_i\right) P\left(\omega_i\right)}{P\left(\mathbf{F}^{\mathbb{W}}\right)}. \tag{3.17}$$

For a given fingerprint $\mathbf{F}^{\mathbb{W}}$ the denominator is constant across all ω_i and can be neglected when evaluating the relative probabilities in (3.16). Assuming equal prior probabilities for all classes, $P(\omega_i) = 1/N_D$ is likewise constant, and the decision criteria in (3.16) reduces to maximizing the *likelihood* $P\left(\mathbf{F}^{\mathbb{W}} \,|\, \omega_i\right)$ for all ω_i. The likelihood is estimated from the multi-variate Gaussian distribution defined by each device reference template [TK09]

$$P\left(\mathbf{F}^{\mathbb{W}} \,|\, \omega_i\right) = \frac{1}{(2\pi)^{(N_D-1)/2}\left|\widehat{\mathbf{\Sigma}}\right|^{1/2}} \cdot \exp\left(\mathcal{F}_e\right), \tag{3.18}$$

where

$$\mathcal{F}_e = -\frac{1}{2}\left(\mathbf{F}^{\mathbb{W}} - \widehat{\boldsymbol{\mu}}_{\boldsymbol{i}}\right)^T \widehat{\mathbf{\Sigma}}^{-1}\left(\mathbf{F}^{\mathbb{W}} - \widehat{\boldsymbol{\mu}}_{\boldsymbol{i}}\right). \tag{3.19}$$

System identification performance is evaluated based on the overall correct identification percentage, calculated as the percentage of time the classifier correctly identifies the true class of an observed fingerprint over all trials. Confusion matrices are generated to facilitate analysis of identification errors, but are not presented in the interest of space.

3.7.3.2 Device Verification. For device *verification*, the captured fingerprint need only be compared with the specific template corresponding to the device's claimed identity to determine authenticity. Again, the similarity measure used is the Bayesian posterior probability assuming equal priors and equal costs. The decision for device verification is a binary result, where the device is deemed authentic when the posterior probability exceeds a pre-determined threshold; otherwise, it is

classified as an impostor. Stated formally, a newly observed device fingerprint $\mathbf{F}^{\mathbb{W}}$ is considered authentic when

$$P\left(\omega_c \,|\, \mathbf{F}^{\mathbb{W}}\right) \geq t, \tag{3.20}$$

where ω_c is the class to which the device claims to belong, and t is the decision threshold.

For each verification decision the system can make two types of errors (see Table 3.1) [JRP04]:

1. incorrectly accepting an impostor as authentic (*false accept*)
2. incorrectly rejecting an authentic device as an impostor (*false reject*)

The threshold, t, used for accept/reject decisions can be adjusted to tune system performance to increase security (reduce false accept errors) or increase accessibility (reduce false reject errors).

Verification performance is assessed herein by plotting the receiver operating characteristic (ROC) curve and corresponding equal error rate (EER) for each curve. The ROC curve is generated by plotting the true accept rate (TAR) vs. false accept rate (FAR) as t is varied across the interval $[0, 1]$ [JRP04]. EER corresponds to the point on the ROC curve where the false reject rate ($\text{FRR} = 1 - \text{TAR}$) and FAR are equal, and is frequently used as a summary statistic to compare the performance of various classification systems. In general, lower EERs indicate better system classification performance. The EER achieved for each plotted ROC curve is provided herein to facilitate future comparisons.

Table 3.1 Verification Error Types.

	System Decision	
Reality	Authentic	Imposter
Authentic	True Accept	False Reject
Imposter	False Accept	True Reject

3.8 Experimental Methodology

All results were obtained by analyzing features of empirically collected unintentional RF emissions from a given number of tested devices. A description of the experimental setup and analysis methodology used herein is provided below.

3.8.1 Experimental Setup. The unintentional RF emissions of a total of $N_D = 40$ individual 16-bit PIC24F micro-controllers are evaluated [Inc10]. These devices include ten unique chips from each of four different part numbers as shown in Table 3.2. The part numbers were intentionally selected to provide varying degrees of similarity in device architecture. All ten chips within each part number are from the same manufacturing lot.

One of the PIC parts (Part A) shares the same basic architecture as the others but has several on-chip peripherals not present on the other three parts. The other three PIC parts (B, C, and D) have minimal architectural differences and are identical except for the amount of on-board flash RAM (64, 48, and 32 kbit respectively). Hereafter, individual chips are referenced by an alphanumeric designator corresponding to their respective part number and unique chip number, i.e., $A1, A2, ..., D9, D10$ as in Table 3.2.

The PIC devices are representative of the low cost micro-controllers used in a variety of real-world commercial embedded systems, including various security applications [PEK+09] and are easy to obtain through normal commercial channels. All of the chips were fabricated using an unspecified 180 nm process. Since all chips within a part number are from the same manufacturing lot, layout and architectural

Table 3.2 Tested PIC micro-controller device classes.

Part Class	Device Numbers	PIC Part Number
A	A1-A10	PIC24FJ64GA102 I/SP
B	B1-B10	PIC24FJ64GA002 I/SP
C	C1-C10	PIC24FJ48GA002 I/SP
D	D1-D10	PIC24FJ32GA002 I/SP

features are identical; the only anticipated physical differences between chips with the same part number are those resulting from random (uncontrolled) variations in the die fabrication and packaging processes.

For device control and measurement, the micro-controllers were mounted on a single evaluation board and programmed to respond to commands sent over an RS-232 serial interface. The circuit board was fixed in place on a measurement table using a custom jig to minimize any lateral movement during or between collections, and was powered from a standard lab DC power supply (Agilent E3631A) to reduce effects due to uncontrolled supply voltage fluctuations. A detailed description of the experimental setup used can be found in [CGB+10].

The average amplitude response of the RF signals captured from all chips in part classes A and B are shown in Fig. 3.2. As might be intuitively expected given the architectural differences, the average signal response produced by the Class A chips is noticably different from the response produced by Class B. The mean amplitude responses produced by classes C and D are omitted from the plot because they were visually indistinguishable from the average response of class B at the scale shown.

3.8.2 Signal Collection. The emissions from each micro-controller were collected using a near-field probe connected to a Lecroy 104-Xi-A oscilloscope as shown in Fig. 3.4. The probe acts as an antenna to receive the unintentional emissions from the device under test and does not directly contact the chip. All

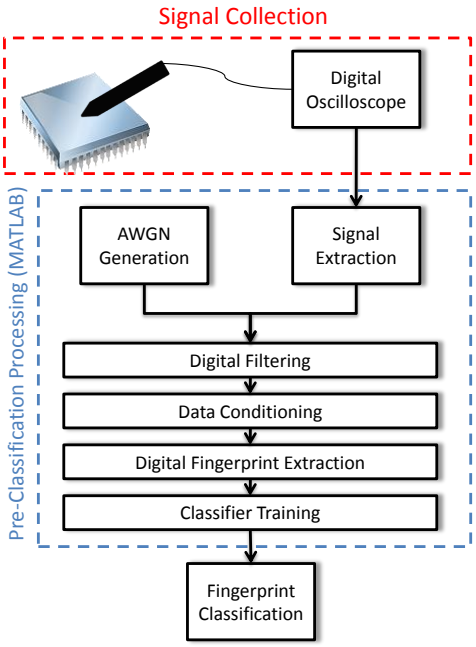

Figure 3.4 Signal collection and pre-classification processing process.

data was collected at a sample rate of $f_s = 2.5$ GSa/sec with a $W_{\text{LP}} = 1$ GHz low pass anti-aliasing filter inserted between the probe and the oscilloscope.

The high sampling rate used is excessive for the devices under test, which operate at $F_{\text{CLK}} = 29.48$ MHz, but allows post-collection simulation of various receiver configurations. All results are based on post-processed signals with an effective sample rate of 208 MSa/sec, which was achieved by passing all collected signals through a low-pass Butterworth filter having a -3dB bandwidth of $W_{\text{BB}} = 100$ MHz, and decimating them by a factor of 12 using proper decimation (i.e., every twelfth sample is retained, all others are discarded).

To simulate the enrollment process in Section 3.4, each micro-controller is repeatedly commanded to perform an identical sequence of operations on known (constant) data. At the start of the operation sequence, the micro-controller asserts a trigger signal to begin the data acquisition process. In practice, devices intended for authentication could easily be configured to generate the required trigger. Use of this technique for non-cooperative recognition tasks would require either precise

detection of the event to be fingerprinted or post-processing to extract the relevant portion of the captured signal.

For this study, environmental effects (e.g., temperature and supply voltage) were controlled by warming-up each device for approximately 20 minutes to stabilize the operating temperature and using a bench power supply to provide a stable supply voltage. After warm-up, $N_{FP} = 500$ signals were collected from each chip as the challenge-response sequence was repeatedly executed. For practical implementations, training over the range of expected operating temperatures and supply voltages has been shown to be an effective technique to deal with environmental fluctuations [TSU04].

For all collections, acquisition order for the chips was randomly generated to prevent any collection-order dependent variance. No measures were taken to isolate the data collection system from background environmental noise—all collections were performed in an office building environment with numerous co-located PCs and wireless devices.

3.8.3 K-Fold Cross Validation. Classification performance is evaluated using K-fold cross validation. Consistent with common practice [TK09], $K = 10$ is used such that each collection of $N_{\mathrm{FP}} = 500$ statistical fingerprints (one for each sequence of operations) per device is divided into ten blocks each having $N_K = 50$ fingerprints per block. Nine blocks from each device are used for training and one block is "held out" for classification. The training and classification process is repeated ten times until each of the ten blocks has been "held out" and classified. Thus, each block of statistical fingerprints is used once for classification and nine times for training. Final cross-validation performance statistics are calculated by averaging the results of all $K = 10$ folds and calculating 95% confidence intervals to determine statistical significance.

3.8.4 Noise Simulation. Signal collection occurred in a controlled office environment while limiting outside influences where possible. To assess performance under less ideal conditions (noisier integrated systems, poor probe positioning, increased separation between sensor and system under test, etc.), captured signals are power-scaled for analysis. To evaluate performance under lower SNR conditions the collected signals were combined with like-filtered additive white Gaussian noise (AWGN) to achieve desired analysis SNRs of $-50 \leq \text{SNR} \leq 50$ dB in 1 dB increments as shown in Fig. 3.4. During pre-classification processing, the baseband signal and AWGN were digitally filtered using the same filter, i.e., a low-pass Butterworth filter with a -3 dB bandwidth of $W_{\text{BB}} = 100$ MHz.

For each Monte Carlo iteration, a total of N_Z independent AWGN realizations are generated, filtered, scaled and added to the collected signals prior to fingerprint generation. After N_Z Monte Carlo iterations at each SNR, $K = 10$ fold cross-validation results are averaged to calculate summary classification performance statistics. All results are based on $N_{FP} = 500$ empirically collected signals per device and $N_Z = 100$ simulated AWGN realizations at each analysis SNR for a total of $(N_D \times N_Z \times N_{FP} = 2\,000\,000)$ classification decisions.

3.9 Results

Extensive pilot studies were performed using all $N_D = 40$ devices to assess 1) the influence of various system parameters on overall performance, 2) the ability of the classification approach to deal with the large number of devices, and 3) the sensitivity of fingerprint performance to various arbitrarily selected *response* sequences. Performance was evaluated by varying the system parameters over $N_{CL} = \{2, 3, ..., 64\}$, $N_R = \{N_{CL}/2, N_{CL}, N_{CL} \cdot 2\}$, and $W_{BB} = \{25, 50, 100, 250, 500\}$ (MHz). All pilot studies analyzed signals at the collected SNR (no noise scaling). The effect of including each of the instantaneous responses (IA, IP, and IF) was also assessed during the pilot studies. The results used $N_R = N_{CL} = 32$ and $W_{BB} = 100$ MHz

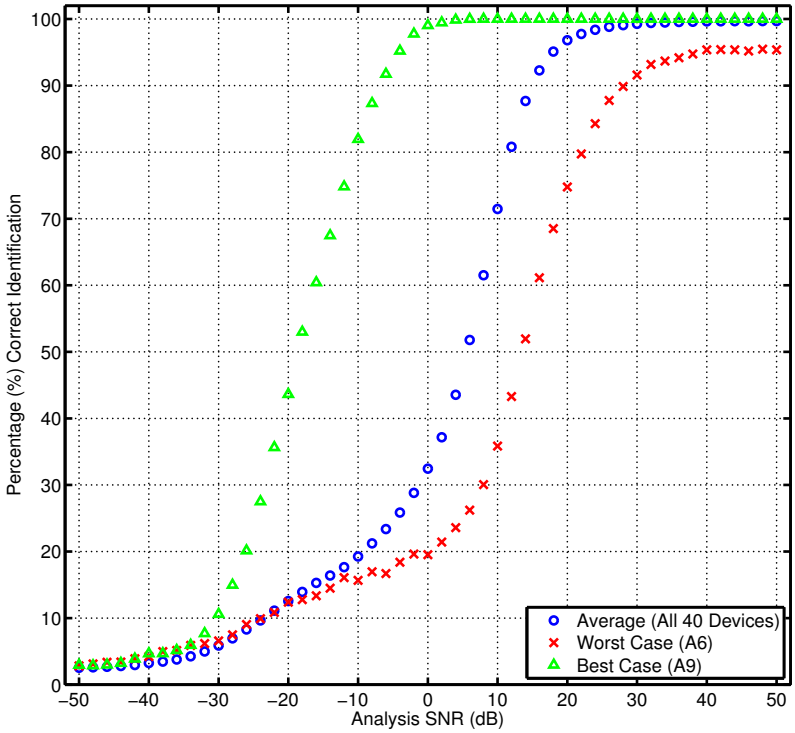

Figure 3.5 Average *identification* performance results (classification success rate) for K=10 fold cross-validation with $N_Z = 100$ random noise realizations at each analysis SNR.

for statistical fingerprints composed of all three instantaneous responses (IA, IP, and IF). This combination of parameters and signal responses provides a reasonable trade-off between system performance and computation time.

Fig. 3.5 shows overall average, best, and worst-case observed *identification* performance for the $N_D = 40$ tested devices in Table 3.2. The 95% CIs were calculated for the average performance data. However, they are omitted from the plot for visual clarity since they are approximately the width of the data markers. As indicated in Fig. 3.5, the classifier achieved an overall average identification success rate of of 99.7% ($\sigma = 0.892$) at the collected SNR (no added noise), and maintained average identification success rates of 90% or better for simulated SNRs $\geq$ 15 dB. Perfect identification (100% classification success rate) was achieved for 27 of the 40 tested devices when analyzed at the collected SNR. The worst observed identification

performance was exhibited by device $A6$, which was successfully identified 95.7% of the time at the collected SNR. The confusion matrices (not presented) show that the majority of identification errors for device $A6$ are associated with misclassifying device $A6$ as either $A8$ or $A4$. As discussed in Sec. 3.7.1.1, this is due to the similarity evident from visual inspection of the device statistical fingerprints depicted in Fig. 3.3. Likewise, the most unique looking fingerprint in Fig. 3.3 (device $A9$) yields the best identification performance of all tested devices across the range of simulated SNRs. As expected, identification performance degrades as the input signals are corrupted with noise, and approaches the accuracy of a random guess ($1/40 = 2.5\%$) at the lowest analysis SNRs.

While it is anticipated that the achievable SNR for the envisioned applications will be very high in practice since emissions are captured using a near-field probe, it is important to note that the identification accuracies are representative of what is achievable by extracting a *single* fingerprint from the device being identified. Extension of the Bayesian classification approach for multiple extracted fingerprints is straightforward, and should provide considerable improvement in identification accuracy. Such improvement may be required for specific applications not considered here.

To evaluate system *verification* performance, a total of 40 ROC curves were generated at each analysis SNR, again using K-fold cross-validation. The ROC curves were generated by sequentially treating each tested device as the *claimed identity* and testing the verification performance against the known *true identity* associated with each fingerprint. Each individual curve shows the achievable trade-space between system security and accessibility as a function of the selected decision threshold for a particular device. Figs. 3.6 and 3.7 show the ROC curves for the worst-performing device identity ($A6$) at selected analysis SNRs.

At the collected SNR, the system achieved an average EER across all tested identities of 0.044% ($\sigma = 0.12$); 32 of the 40 tested identities exhibited EERs of

$< 0.01\%$. The worst four EERs were 0.19, 0.33, 0.50, and 0.53%. As with the identification task, worst case *verification* performance was associated with the template for device $A6$, followed closely by device $A8$. Again, this is due to the similarity between the statistical fingerprints of the two devices as shown in Fig. 3.3. Fig. 3.6 shows the behavior of device verification performance as the input signal quality degraded, i.e. performance decreases significantly as the collected signal is corrupted with additional simulated noise, eventually reaching near "random" verification accuracy at SNR $= -40$ dB. Again, if higher performance is required for a specific application, verification accuracy can be improved by extracting multiple fingerprints from the device.

In this study, steps were taken to ensure that the device being authenticated is as isolated from external effects as practical. In operational systems, it is likely that a particular chip is permanently integrated into a larger system component or board. For instance, in embedded systems or smart-cards, the micro-controller is likely to be soldered or otherwise packaged internally to the authentication device. Since the process controls for integration and packaging may not be as stringent as those for fabrication of the IC itself, it is believed that fingerprinting of the integrated device may actually prove to be more effective than fingerprinting of isolated ICs as considered here.

While the raw data acquired for this study was obtained using a high speed digital sampling oscilloscope, sample rate and bandwidth were intentionally limited using the procedure described in Sec. 3.8.2 to simulate a less capable receiver setup (e.g., a low-cost data acquisition card). Furthermore, preliminary experiments using a simulated tuned receiver setup show that a substantial amount of the overall discriminatory information is carried in narrow sidebands around harmonics of the device clock frequency. Thus, it is believed that more practical, low-cost receiver architectures are likely to remain highly effective.

A final observation is that the results suggest acceptable performance may be achieved for some devices without the need for extensive optimization of the response sequence. The results herein were obtained by arbitrarily designating several clock cycles of an overall operation sequence as the *response* region. No statistical difference in performance was observed during limited trials when the designated response region was varied to include sub-regions containing very different microcode instruction sequences. This suggests that the technique can be implemented in an opportunistic manner, where the challenge-response sequence can be conveniently selected from a portion of other authentication procedures or protocol communications. Additional improvements in performance might be obtained by more carefully choosing or defining a response that emphasizes device sub-circuitry that exhibits high inter-device variability. However, it is believed that for many applications the opportunistic approach will provide sufficient performance without the need for further performance improvement through response optimization.

Although obtained under controlled conditions using a single sensor module, the results thus far are very promising and the proposed technique merits additional investigation. These results also suggest that RF-DNA fingerprinting is suitable for other applications such as Trojan or counterfeit device detection and forensic attribution for criminal or other investigations.

A considerable amount of work remains to fully understand the suitability of RF-DNA fingerprinting for practical security implementations. Intuitively, the nature of the intrinsic characteristics that induce inter-device variations suggests a fingerprint based on those variations will be extremely difficult to impersonate. However, further analysis and experimentation are needed to confirm this. Other areas for additional study include:

- Permanence and robustness of RF-DNA features under varying environmental conditions.

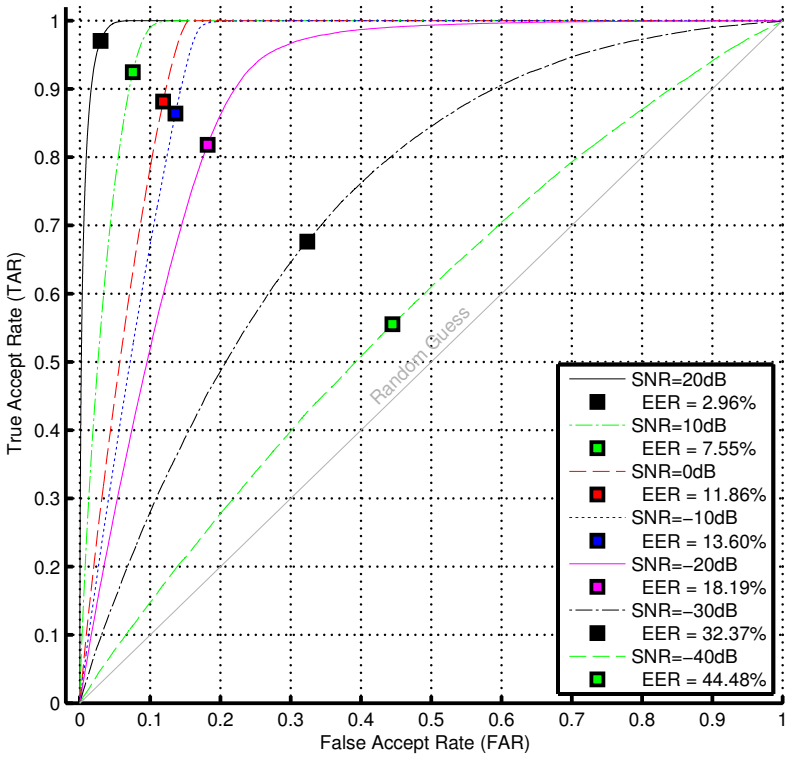

Figure 3.6 Worst case (device A6) observed ROC curves showing *verification* performance across the full range of simulated noise conditions.

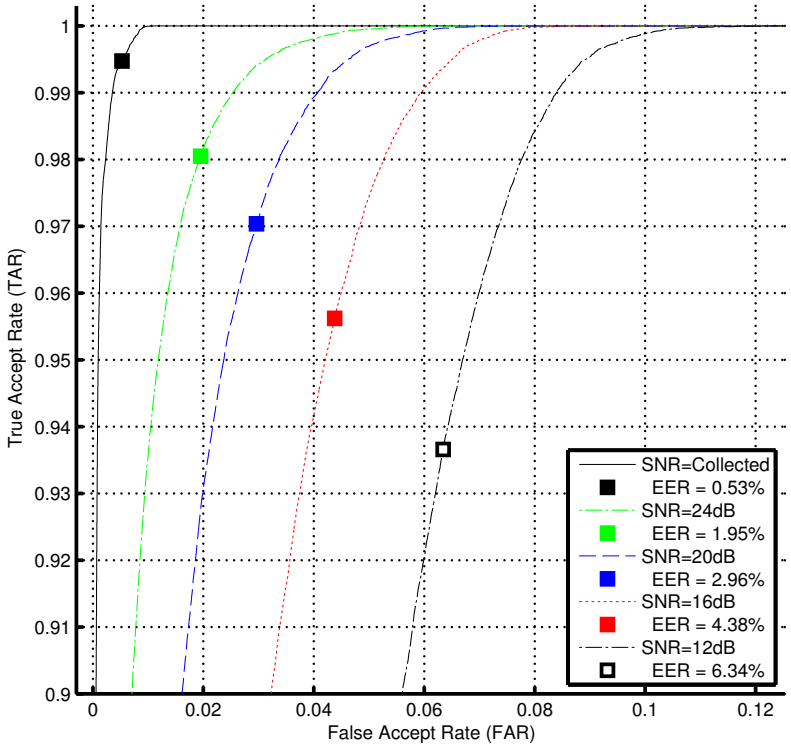

Figure 3.7 Worst case (device A6) observed ROC curves showing *verification* performance at collected SNR and under simulated high SNR conditions.

- Sensitivity of fingerprint performance to variations due to different sensor modules or sensor positioning.
- Scalability to very large databases.
- Optimization of the device response sequence in the challenge-response phase to maximize device discriminability.
- Suitability for low-cost implementations through bandwidth optimization and investigation of alternative sensor module architectures.
- Effectiveness for programmable or custom logic ICs such as FPGAs or ASICs.

Finally, recent research [KTM09, WTR10] has demonstrated significant performance gains using spectral Fourier-based and wavelet-based RF-DNA fingerprints for intentional emissions. Application of these approaches to recognition of *unintentional* IC emissions remains an area of future research.

3.10 Conclusion

RF "distinct native attributes" (RF-DNA) possessed by the unintentional emissions of ICs are a rich source of discriminatory information for device recognition. Empirical results demonstrate the suitability of RF-DNA fingerprinting for both identification and verification device recognition tasks. For experimentally collected emissions, the technique correctly identifies devices greater than 99.5% of the time, with average verification EERs of less than 0.05% using a single extracted fingerprint. Correct identification success rates of better than 90% were maintained under analysis conditions of SNR $\geq$ 15 dB. Thus, RF-DNA fingerprinting is promising for anti-cloning and related security applications requiring unique IC device recognition. Furthermore, the impressive performance indicates the technique may be adaptable to less ideal conditions while still providing acceptable performance. Finally, these results were obtained using a single extracted fingerprint, and a substantial improve-

ment in performance is believed to be realizable through a straightforward extension of the approach for multiple extracted fingerprints.

3.11 Supplementary Discussion

This section contains an expansion of the material submitted as part of this article that could not be included due to mandatory space limitations.

3.11.1 Experimental Setup. This section includes additional information on the experimental setup used to acquire the data for all experiments conducted during this research. The primary source of data used for this research (unintentional EM signals) was collected using AFIT's commercial Riscure Inspector side channel analysis system. The experimental setup closely follows the setup described in Sec. 2.7 as shown in Fig. 2.5.

The system is configured to collect unintentional RF emissions from the device under test using a near-field probe (1 GHz bandwidth) connected to a Lecroy 104-Xi-A oscilloscope. The probe acts as an antenna to receive the unintentional emissions from the device under test, and does not directly contact the chip. The oscilloscope has a 1 GHz bandwidth, a maximum sample rate of 10 GSa/sec, and four channels with 12.5 MBits of sample memory each. Depending on the device technology and clock rate of the device under test, hardware low-pass filters (either built-in to the oscilloscope, or in-line filters inserted between the H-field probe and the oscilloscope input) are used as necessary to prevent signal aliasing due to the analog-to-digital sampling process.

The near-field probe is mounted on a computer-controlled motorized XYZ table for consistent placement of the probe relative to the device under test. Initial probe position for measurements is established by performing a two-dimensional scan of the surface of the tested chip as it repeatedly executes the operation of interest. The results of the scan are processed with a digital bandpass filter and analyzed

to determine the location of maximal RF energy in the band corresponding to the known internal clock frequency. The probe and relative device positions remained fixed for all subsequent collections. Custom jigs are fabricated for each test board to help stabilize the device being measured and to ensure consistent placement of the probe for subsequent measurements.

Figure 3.8 Riscure Inspector Side Channel Analysis System [Ris09]

To improve collection efficiency and reduce post-processing for signal alignment, the training devices are controlled by a PC over an RS-232 serial interface. Devices are programmed to assert a trigger signal on one of the general purpose input/output (GPIO) pins at the start of the operation sequence. The oscilloscope is configured through a PC interface to collect the RF signal for a fixed time interval each time the trigger is asserted. This enables precise identification and alignment of the individually collected signals without the need for extensive post-processing. As described in Chap. 4, triggering based on known features that indicate the start of the encryption operation using real-time EM emissions is also feasible, but requires substantial post-processing of the acquired signals to achieve similar alignment.

4. Leakage Mapping: A Systematic Methodology for Assessing the Side Channel Information Leakage of Cryptographic Implementations

This chapter contains the text of an article that has been submitted for publication to the Association of Computing Machinery (ACM) *Transactions on Information and System Security* [CBL11]. This article was co-authored by Dr. Rusty Baldwin and Mr. Eric Laspe.

4.1 Abstract

We propose a generalized framework to quantify the side-channel information leakage from arbitrary cryptographic implementations. The framework provides a comprehensive methodology to assess the information leakage from all algorithmically-specified key-dependent intermediate computations for implementations of symmetric block ciphers. The leakage assessment quantitatively bounds the resistance of an implementation to the general class of differential side channel analysis techniques. The leakage mapping framework is demonstrated using the well-known Hamming Weight and Hamming Distance leakage models, with recommendations for extension of the technique to more accurate models. The approach is applied to two typical unprotected implementations of the Advanced Encryption Standard, and the assessment results are empirically validated against a correlation-based differential power analysis attack.

4.2 Introduction

Over the last several years, there has been an extensive effort in academia and industry to address physical implementation attacks on cryptographic systems, to include both side channel and fault attacks [KJJ99, MOP07, AARR02, ARRS05, BCO04, SLP05, Sko06, SMY09, GBTP08]. However, relatively little work has been

done to aid cryptographic system designers in practically addressing the identified security risks.

From a security perspective, it is naïve and dangerous to assess implementation security based on the results of limited testing. For instance, the failure of a differential power analysis (DPA) attack based on the Hamming Weight of an S-Box output should not be the basis for concluding the implementation is secure against DPA or other side channel analysis techniques in general.

It is also dangerous to assess the overall strength (or weakness) of an implementation's security based on analysis of a small subset of the overall implementation (e.g., one S-box), particularly for complex FPGA or ASIC circuits where various portions of the algorithm may be implemented in completely separate physical areas of a die, with very different layouts and routing in each area.

From a cryptographic system designer or engineer's perspective, designing a system that is secure against the plethora of rapidly evolving physical implementation attacks is daunting. Nevertheless, to sufficiently characterize the leakage of an implementation against the full spectrum of tools that may be employed by a real adversary, it is essential that a systematic evaluation of an implementation be conducted against the widest variety of attacks possible. In particular, the effectiveness of countermeasures cannot be adequately understood unless their effects are comprehensively assessed against the full spectrum of available attack techniques.

The work by Standaert, et al. [SMY09] is the first concerted effort to provide a practice-oriented framework for a fair comparative evaluation of side channel attacks and implementation leakage. The work provides a solid foundation for the comparison of attack effectiveness as well as a standard method for quantitatively bounding the leakage from a particular device.

However, to assess the worst-case leakage of an implementation, Standaert's methodology essentially relies on an evaluator's skill in the subtle art of building an

optimal template attack. Furthermore, the results are only applicable to the targeted portion of the overall cryptographic algorithm, which may neglect to identify other exploitable leakages. In light of techniques such as algebraic SCA [RSVC09, RS09], assessing the overall security of a system based on the leakage from a limited portion of an overall implementation is simply insufficient.

Herein, we expand on the objectives of [SMY09] by introducing a systematic approach to comprehensively quantify the leakage characteristics of a given implementation. We provide a practical tool set for making sound decisions during the system design process to achieve, with some certainty, a desired level of resistance against differential side channel attacks.

The contributions of this work include a generalized framework for leakage assessment of block ciphers, and a method to bound the resistance of an implementation against the general class of differential SCA techniques. The approach is applied to two typical unprotected implementations of the Advanced Encryption Standard (AES), and the assessment results are empirically validated against a correlation-based DPA attack.

4.3 Background

4.3.1 Side Channel Emissions of ICs. All physical systems, viewed externally, produce both intended and unintended outputs. The unintended outputs are quantifiable, physically observable phenomena produced as a side-effect of normal operation. When an unintended observable outcome is correlated to some aspect of the internal state or operation of the system, the resulting *side channel* is said to *leak* information. Herein, the term *information leakage* generically refers to any such phenomena that exhibits a statistical relationship to the underlying operations being performed or data being manipulated.

Information leakage can result from a variety of side channel emission sources. Known sources include variations in power consumption, radiated electromagnetic

(EM) energy, computation time, and even acoustic or thermal emissions. The various known sources of side-channel emissions are shown in the block diagram of a notional two-party communication system in Fig. 1.1.

4.3.2 Information Leakage and Side Channel Analysis. The objective of *side channel analysis* is to infer details about internal data or operations based on the observation of a side channel while that data is being processed. Various mathematical techniques have been introduced [KJJ99, SLP05, BCO04, GBTP08] that permit an adversary to exploit relationships between the data manipulated internally to a device and the externally observable side channels. Even if the underlying data has a very small influence on the external observable, it is often possible to recover secret key material from a cryptographic system using these techniques.

Side channel attacks have profound implications for the physical security of sensitive electronic systems since cryptographic algorithms form the very foundation of virtually all modern secure systems—with applications ranging from remote keyless entry systems to secure payment technologies to the protection of various forms of intellectual property. Because cryptographic key secrecy plays such a central role in security, preserving that secrecy and understanding the practical implications of any vulnerabilities that may lead to the disclosure of key material is of critical importance. Thus, although side channel techniques are generic in the sense that they can be used to infer a wide variety of information about the internal activity of an integrated circuit, the majority of the research in this area has been on key recovery attacks (i.e., *side channel attacks*) of cryptographic systems.

We develop a framework of techniques that permit a cryptographic engineer or security evaluator to systematically assess, investigate, and counter the numerous sources of information leakage that can lead to unintentional disclosure of sensitive key material.

Hereafter, we focus on symmetric block ciphers, with particular attention to the Advanced Encryption Standard (AES) [Nat01]. Application of leakage mapping to other symmetric block ciphers is straightforward, and the general approach introduced herein can be adapted to most other classes of cryptographic algorithms (e.g., stream ciphers, asymmetric algorithms, etc.) with minimal modifications.

4.3.3 Structure of Symmetric Block Ciphers. Symmetric cryptographic algorithms are typically made up of repetitive sequences called *rounds*, where each round is composed of a sequence of *primitive operations*. A cryptographic operation is denoted $\mathrm{C}_k(m)$ where k is a fixed key drawn from the key space $\mathcal{K}$ and m is the input message (either plain-text or cipher-text depending on the cryptographic mode of operation) drawn from the message space $\mathcal{M}$.

To enable efficient implementation on a wide variety of hardware and software platforms, cryptographic algorithm designers often constrain the underlying primitive operations to function on small sub-blocks of data no larger than the data bus width of potential target platforms. In practice, this means that data sub-block sizes for many algorithms are limited to eight or fewer bits to permit implementation on low-cost micro-controllers or embedded processors. Throughout this work, the term data is used generically to include all inputs to an operation, including the key.

A practical implication of the constrained input size for each intermediate computation is that the *intermediate result* of each primitive operation step can be exhaustively predicted with low computational effort. The dependence of an implementation's behavior on these data sub-blocks is one of the key concepts exploited by many SCA attacks.

For example, consider a typical 8-bit substitution box (S-Box) such as the one used in the AES Rijndael algorithm. Each AES S-Box produces an output that depends only on an 8-bit input and an 8-bit portion of the round key. A common attack scenario involves a known input, i.e., a *known plain-text* for an encryption

operation. The output can, then, be exhaustively predicted for each of the $2^8 = 256$ possible round sub-keys. Thus, although both the AES state array and key are 128 bits resulting in $2^{128\times2} \approx 1.2 \times 10^{77}$ possible combinations[1], the modular structure of the algorithm enables prediction of all intermediate results after each S-Box stage with only $16 \times 2^8 = 4096$ computations.

The bit-wise exclusive-or (XOR) function commonly employed in cryptographic algorithms further exemplifies this point. An XOR operation effectively yields a data sub-block size of 1-bit since each bit of the output only depends on two input bits. Thus, to compute all possible hypothetical states given knowledge of half the input to an XOR stage (assuming the key bit is the only unknown) requires only $128\times2 = 256$ total computations.

4.3.4 Advanced Encryption Standard (AES). The AES is a public and thoroughly documented symmetric block cipher. The reader is referred to [DR01, Nat01] for an in depth discussion of the design and mathematics of the algorithm which are beyond the scope of this work. A brief overview of the algorithm and the specific notation employed herein are described below. The notation and algorithm description have been tailored to allow a more natural description in the context of SCA attacks and the leakage mapping technique introduced in Sec. 4.4.

4.3.4.1 Notation. Throughout this work, vector representations of data are denoted by an over-arrow and matrix forms are denoted by bold type. Except where explicitly specified, each element of a vector or matrix represents one byte (8 bits) of data. Various other representations of data (e.g., bit- or word-oriented) are sometimes employed to more naturally match the actual machine representation used for an arbitrary implementation. Such representations are distinguished by an

[1]AES can also be implemented with a 192 or 256 bit key. Individual round keys are 128 bits for all FIPS 197 approved variants of the Rijndael algorithm.

underset $_{(n)}$ where n is the number of bits represented by each vector or matrix element, i.e.,

- $\vec{x}$ A vector of *bytes* where $\vec{x}_i \in \{0, 1, \ldots, 255\}$ is the i^{th} element of the vector.
- $\underset{(n)}{\vec{x}}$ A vector of $n-$bit elements where $\underset{(n)}{\vec{x}_i} \in \{0, 1, \ldots, 2^n - 1\}$.
- $\mathbf{x}$ A matrix of *bytes* where $\mathbf{x}_{i,j} \in \{0, 1, \ldots, 255\}$ is the element at the i^{th} row and j^{th} column of the matrix.
- $\mathbb{X}$ An n-dimensional matrix, where $n \geq 3$. The notation used to denote individual elements of the matrix differ, but in general superscripts (e.g., $\mathbb{X}^i$) are reserved to denote *round* indices, and subscripts (e.g., $\mathbb{X}_{i,j}$) denote individual elements within a round.

4.3.4.2 AES State Representation. All FIPS 197 approved variants of the AES algorithm operate on 128-bit data blocks, represented algorithmically as a (4×4) *state array* composed of 16 bytes of data [Nat01]. Using the above notation, the bit-vector of the state is formed from the individual state bits is

$$\underset{(2)}{\vec{s}} = [b_0 \; b_1 \; b_2 \; \cdots \; b_{127}]_{[1\times 128]}\,, \tag{4.1}$$

where $b_i \in \{0, 1\}$ represents the i^{th} bit of data. Each byte of the state array is formed from 8 bits of the 128-bit data block or

$$\begin{aligned} s_0 &= b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \\ s_1 &= b_{15} b_{14} b_{13} b_{12} b_{11} b_{10} b_9 b_8 \\ &\vdots \\ s_{15} &= b_{127} b_{126} b_{125} b_{124} b_{123} b_{122} b_{121} b_{120}, \end{aligned}$$

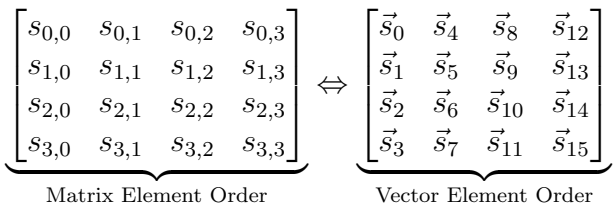

Figure 4.1 Illustration of mapping between AES state matrix (indexed by row and column) and corresponding elements of the equivalent byte-vector.

where $s_i \in \{0, 1, \ldots, 255\}$. The (byte-oriented) AES *state vector*, $\vec{s}$, is then formed from the 16 bytes of AES state information

$$\vec{s} = [s_0 \; s_1 \; \ldots \; s_{15}]_{[1\times 16]}\,, \tag{4.2}$$

where $\vec{s}_i$ denotes the i^{th} byte, and $i \in \{0, 1, \ldots, 15\}$. The equivalent AES *state matrix*[2] is formed from the bytes of the state vector in column order. The linear indices of the *state vector*, $\vec{s}$, can be translated to or from equivalent row and column indices of the *state matrix* $\mathbf{s}$, using the mapping in Fig. 4.1.

The AES master key (in byte-vector form) is denoted $\vec{\mathfrak{K}}$, and is formed from the N_B bytes of the key

$$\vec{\mathfrak{K}} = [k_0 \, k_1 \; \ldots \; k_{N_B-1}]_{[1\times N_B]}\,, \tag{4.3}$$

where k_i is the i^{th} byte for $i \in \{0, 1, \ldots, N_B - 1\}$, and N_B is determined from the key size as shown in Table 4.1. As with the AES state information, the equivalent *bit*-vector representation is denoted $\underset{(2)}{\vec{\mathfrak{K}}}$.

The master key is expanded into a full key schedule composed of $(N_r + 1)$ individual 128-bit (16-byte) *round keys*, where $N_r = 10, 12,$ or 14 for implementations

[2]Throughout this work, the term *state matrix* is used interchangeably with the *state array* defined in [Nat01].

Table 4.1 Number of intermediate result states computed during AES encryption and key scheduling (inclusive of input and output states).

				# Intermediate Steps	
Key Size	N_r	N_B	# Rnd Keys	Cipher	Key Sched.
128	10	16	11	41	70
192	12	24	13	49	70
256	14	32	15	57	79

using 128, 192, or 256-bit keys, respectively (see Tab. 4.1). The i^{th} round-key (in byte-vector form) is denoted $\vec{k}^i$, where $i \in \{0, 1, \ldots, N_r\}$.

Each of the $(N_r + 1)$ individual 16-byte round keys, $\mathbf{K}^i$, for round i (in byte-matrix form) is formed from the individual round key bytes

$$\mathbf{K}^i = \begin{bmatrix} \vec{k}^i_0 & \vec{k}^i_4 & \vec{k}^i_8 & \vec{k}^i_{12} \\ \vec{k}^i_1 & \vec{k}^i_5 & \vec{k}^i_9 & \vec{k}^i_{13} \\ \vec{k}^i_2 & \vec{k}^i_6 & \vec{k}^i_{10} & \vec{k}^i_{14} \\ \vec{k}^i_3 & \vec{k}^i_7 & \vec{k}^i_{11} & \vec{k}^i_{15} \end{bmatrix}_{4\times 4} , \tag{4.4}$$

for $i \in \{0, 1, \ldots, N_r\}$. The full key schedule is constructed as a three-dimensional matrix denoted $\mathbb{K}_{[4\times 4\times (N_r+1)]}$, where the individual matrix elements are

$\mathbb{K}^i_{r,c}$ The element (byte) of the full AES key schedule at row r, column c of the i^{th} round key, where $i \in \{0, 1, \ldots, N_r\}$.

4.3.4.3 AES Round Structure. Each round of the AES is composed of a sequence of four primitive operations that manipulate the byte-oriented state matrix, **s**. The AES primitive operations are

- `AddRoundKey` (`ARK`): Combines the input data with the round key using a bit-wise XOR operation.

- `SubBytes` (`SB`): Independently processes each byte of the state using a non-linear substitution (S-box).
- `ShiftRows` (`SR`): Cyclically shifts (byte-oriented rotation) the rows of $\mathbf{s}$ by incremental offsets.
- `MixColumns` (`MC`): Independently mixes the data within each column (4 bytes) of $\mathbf{s}$.

A detailed description of each AES primitive operation can be found in the AES specification [Nat01].

One full round of an AES encryption as defined herein is illustrated in Fig. 4.3. This definition is slightly different from that in [Nat01]. The two descriptions are semantically equivalent, as depicted in Fig. 4.2, but the round definition herein allows a more natural mathematical description of side channel attacks and clarifies the dependency between each individual round key and the locally dependent intermediate results.

The first $(N_R - 1)$ rounds of an AES encryption are identical and composed of a sequential application of the `ARK, SB, SR,` and `MC` primitive operations. The final round omits `MC`, instead applying an additional `ARK` operation to produce the cipher-text output, $\mathbf{c}$.

The AES state after completing the j^{th} intermediate computation step of round i is denoted $\mathbf{s}^{i,j}$, and the individual elements of the AES state matrix after each step are indexed by their matrix indices, i.e.

$\mathbf{s}^{i,j}$ AES state matrix after completing j^{th} intermediate computation step of round i,

$\mathbf{s}^{i,j}_{r,c}$ The element (byte) of the AES state matrix at row r and column c after completing j^{th} intermediate computation of round i.

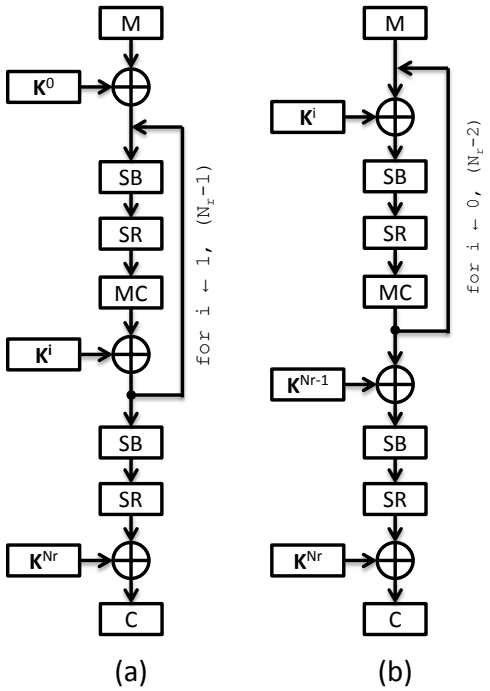

Figure 4.2 Comparison of (a) FIPS 197 round definition to (b) the round definition used herein. The two definitions are semantically equivalent, but the modified description in (b) allows a more natural mathematical description of intermediate result dependencies on round keys.

Thus, $\mathbf{s}^{0,0}$ and $\mathbf{s}^{N_r-1,4}$ represent the plain-text, m, and cipher-text, c, of an AES encryption operation, respectively. Since the output of each round is the input to the next, $\mathbf{s}^{i,4} \equiv \mathbf{s}^{i+1,0}$ for any two adjacent rounds i and $(i+1)$.

4.3.5 Modeling Leakage. The information leakage from a cryptographic implementation can be modeled at various levels of abstraction depending on how much is known about the implementation's internal design architecture (to include both hardware and software aspects).

Herein, a leakage model is defined to include a minimum of two components

- A *leakage function* $\mathcal{F}_L(\cdot)$ used to transform one or more intermediate results into hypothetical leakage values, and

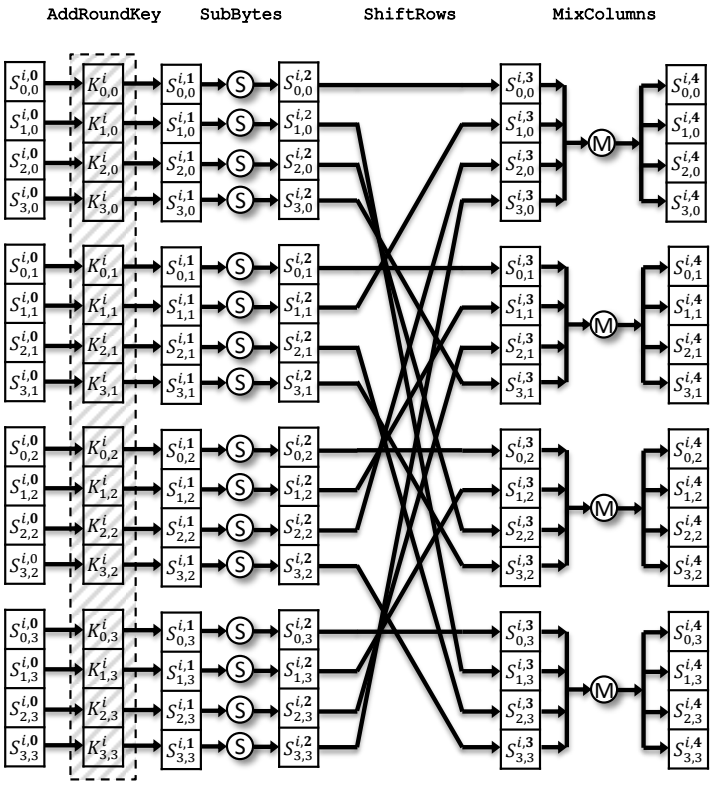

Figure 4.3 Byte-oriented depiction of one round of an AES encryption for round $i \in \{0 \dots N_{R-1}\}$, based on the round definition in Fig. 4.2b.

- A parameter, N_D, that defines the assumed data sub-block size used for model predictions.

The leakage function, $\mathcal{F}_L$, can operate on a single static state value or on two different states. The latter definition is used to create models based on the transistor switching activity or other underlying physical phenomena influenced by the change from one state to another.

The term leakage model, when considered in combination with the particular mathematical technique used in a differential side-channel analysis (DSCA) attack, is closely related to the concepts of *side channel distinguisher* and *selection function* employed in other works (e.g., [SGV08]). Herein, the leakage model is considered separately from the mathematical tools used to analyze the relationship between the modeled data and actual leakage data.

4.3.5.1 Algorithmic Leakage Models. In general, cryptographic system designers have a great deal of flexibility in how they implement an algorithmic specification. It is common, for example, for designs to combine primitive operations or other steps to efficiently use the resources available on the target platform, e.g. the *T-Table* implementation commonly used in 32-bit software implementations of the AES [DR01].

Theoretically, given sufficient resources, a symmetric algorithm with a 128-bit key and a 128-bit data block size could be implemented as a one-step lookup table with $2^{128\times2} \approx 1.16 \times 10^{77}$ entries. In practice, this is clearly infeasible, and designs can be expected to substantially follow the algorithmic specification—at a minimum producing the same outputs as the reference implementation after each round.

Because implementations of cryptographic algorithms generally follow the algorithmic specification to some degree, *algorithmic leakage models* can be constructed even if no details of the underlying implementation architecture are known.

Because generality of the systematic approach was an important objective of this research, such models are the basis of our work as they can be applied to any implementation with minimal modification. Supplementing these techniques with additional implementation knowledge is straightforward, and would lead to a more conservative leakage assessment relative to the worst-case scenario where an adversary is able to obtain substantial information about the underlying implementation.

4.3.5.2 Common Leakage Models. A variety of leakage models have been suggested [KJJ99,MOP07,BCO04] with varying degrees of complexity. In practice, even very simple algorithmic leakage models have been shown to be adequate to carry out a wide variety of side channel attacks. Such simple models also provide a solid basis for a systematic leakage assessment of most implementations while maintaining the generality of the technique. Two common leakage transformations proposed in previous work, Hamming Weight and Hamming Distance of intermediate results, are described below.

The Hamming Weight leakage transformation assumes a relationship between the number of non-zero bits in an intermediate result of interest and the resulting data-dependent variance leaked through a side channel. The Hamming Weight leakage transformation, $\mathcal{HW}(\cdot)$, for an arbitrary bit-vector, $\underset{(2)}{\vec{x}}$, with n-elements is

$$\mathcal{HW}\left(\underset{(2)}{\vec{x}}\right) = \sum_{i=0}^{n-1} \underset{(2)}{\vec{x}_i}, \tag{4.5}$$

where

$$0 \leq \mathcal{HW}\left(\cdot\right) \leq n.$$

For data representations other than bit-vectors, it is assumed that each grouping of N_D bits is computed by converting each individual element to an equivalent bit-vector representation before computing the Hamming Weight. In the interest

of conciseness, we omit the details of these conversions. Herein, the $\mathcal{HW}$ leakage transformation operates on vectors or matrices as well as individual elements, i.e.

$$\mathbf{s}' = \mathcal{HW}(\mathbf{s})$$

denotes the Hamming Weight leakage transformation of the state matrix $\mathbf{s}$, where each matrix element's Hamming Weight is computed independently. The Hamming Weight transformation results in an output matrix with the same dimensionality as the input.

Hamming Distance is related to the Hamming Weight, but measures the number of bits that differ between two bit-vectors of the same length. The Hamming Distance, denoted $\mathcal{HD}(\cdot,\cdot)$, between two n-element bit-vectors, $\underset{(2)}{\vec{x}}$ and $\underset{(2)}{\vec{y}}$, is

$$\mathcal{HD}\left(\underset{(2)}{\vec{x}}, \underset{(2)}{\vec{y}}\right) = \sum_{i=0}^{n-1} \left(\underset{(2)}{\vec{x}_i} \oplus \underset{(2)}{\vec{y}_i}\right) \tag{4.6}$$

where $\oplus$ denotes the bit-wise exclusive-or operator, and

$$0 \leq \mathcal{HD}(\cdot,\cdot) \leq n. \tag{4.7}$$

In practice both the Hamming Weight and Hamming Distance models effectively account for a significant portion of the variance in side channel emissions during certain operations. For example, the Hamming Weight model is normally highly accurate for operations involving a pre-charged data bus such as those commonly used on micro-controllers.

Likewise, the Hamming Distance model has been found quite accurate for operations that toggle the state of the bits stored in a hardware register or latch. An appropriately constructed Hamming Distance model can effectively account for

virtually all variance in the side channel emissions of even an iterative FPGA-based hardware implementation of the AES [SÖP04, SMPQ06].

4.3.5.3 Improved Leakage Models. In addition to the simple models described above, more complex (and accurate) models can also be constructed. For example, a straightforward improvement to the Hamming Distance model, known as a switching distance model, distinguishes between $0 \rightarrow 1$ and $1 \rightarrow 0$ bit transitions [PSQ07] by assigning different weights to each transition. Application of the switching distance model within the leakage mapping framework is straightforward.

If details of the hardware architecture (i.e., the net-list) are available, the switching activity of the device can be simulated at the transistor level. Although such models are highly accurate, their employment requires access to detailed design information that is difficult to obtain for commercial hardware devices. Furthermore, the time required to simulate the full circuit over a large number of random inputs makes such models inappropriate for our use.

A more promising technique for integration into a systematic leakage assessment methodology is linear regression based models [SLP05]. Although not considered in this initial study, such models can be constructed relatively efficiently. Furthermore, whereas the Hamming Weight and Distance models assume a *fixed* leakage transformation across all sampled instants in time, a regression-based approach could adapt the model for each sample. Models constructed using regression techniques might be considered leakage agnostic since they can adaptively capture the statistical relationship between the variance of the observed signal at each sample and the data of interest. It is believed that such an approach would be substantially more robust at the expense of significantly increased computational effort.

4.3.6 Measures of Information Leakage. Various approaches are used to quantify information leakage. The most frequent include Kocher's difference of means [KJJ99], Pearson's product-moment correlation [BCO04], and Shannon's

entropy and mutual information [GBTP08, SMY09, BGP+11]. Correlation is used herein based on its computational efficiency and effectiveness in practical attacks as demonstrated through numerous empirical studies [MOP07,SGV08]. Other measures such as the entropy or mutual information-based approach could also be used given sufficient computational resources, and may prove to be more suitable for implementations that exhibit leakages that do not conform to the Gaussian assumptions that underly the correlation approach.

Pearson's correlation coefficient measures the strength of a positive or negative linear relationship between two random variables. Although independent random variables will have a correlation of $\rho = 0$, the converse does not guarantee independence since the measure is only sensitive to linear relationships.

The population correlation coefficient, ρ_{XY}, between two real-valued random variables X and Y is

$$\rho_{XY} = \frac{\texttt{Cov}\,(X, Y)}{\sqrt{\texttt{Var}(X) \cdot \texttt{Var}(Y)}} \tag{4.8}$$

where $-1 \leq \rho \leq 1$ [RN88]. The *sample correlation coefficient*, $\mathrm{r}_{\mathbf{XY}}$, for N_t observations of the random variables, $\mathbf{X}$ and $\mathbf{Y}$, is

$$\mathrm{r}_{\mathbf{XY}} = \texttt{Corr}(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i=1}^{N_t}(\mathbf{X}_i - \overline{\mathbf{X}})(\mathbf{Y}_i - \overline{\mathbf{Y}})}{\sqrt{\sum_{i=1}^{N_t}(\mathbf{X}_i - \overline{\mathbf{X}})^2 \cdot \sum_{i=1}^{N_t}(\mathbf{Y}_i - \overline{\mathbf{Y}})^2}}, \tag{4.9}$$

where $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$ are the observed sample means over the N_t traces. This formulation has the practical advantage of allowing the correlation coefficient to be updated incrementally by adding new observations (rows) to each matrix. The sample correlation coefficient will approach its true value given a sufficiently large number of sample observations.

4.4 Leakage Mapping

Our *leakage mapping* procedure is a systematic approach which characterizes the information leakage from an implementation throughout an entire cryptographic operation, including intermediate results computed during 1) all rounds of the cipher, and 2) related activities such as key-scheduling. This is done by carrying out a *known key* SCA analysis procedure for all algorithmically specified intermediate results, inclusive of input and output, under the assumption of various selected leakage models.

As described above, Pearson's correlation coefficient quantifies the magnitude of the identified leakages. The overall procedure is composed of the following steps

1. Acquire a sufficiently large number of side channel leakage traces while randomizing key and input data,
2. Pre-compute all algorithmically specified intermediate results,
3. Model the data-dependent leakage variations based on the known intermediate results,
4. Use a known-key correlation procedure to identify and quantify all potentially exploitable leakages, and
5. Interpret the results to bound the number of traces required for key inference based on maximum observed correlations, and prepare summary statistics and visual representations of the data to aid in qualitative leakage assessment.

   Each step of the leakage mapping procedure is described in detail below.

Step 1. Acquire profiling side channel data. The first step of the leakage mapping procedure is to collect a suitable number of side channel traces while the evaluated device performs cipher operations on known (random) input/key pairs. The number of traces, N_t, required to sufficiently characterize side channel leakage is highly implementation dependent.

Ideally, the leakage from a device would be measured under replications of all possible input/key combinations to adequately characterize the noise induced by non-cipher related sources (e.g., ambient or environmental noise, unrelated circuit activity, nearby off-die electronic components, etc). Since this is clearly impractical (it would be quicker to brute force the key), a more reasonable approach is to choose the number of profiling traces, N_t, based on the desired SCA-resistance of the implementation. For example, the design objective might be to achieve DPA-resistance for up to $N_t \leq 1E6$ traces.

The side-channel signal observed during each of the N_t encryption operations is digitally sampled at a fixed rate, typically using a high-speed digital sampling oscilloscope (DSO) or similar data acquisition device. For each measured trace, N_s samples of side channel data are acquired. The vector $\vec{L}$ composed of the digital samples from a single observed cryptographic operation is referred to as a *trace*. The vector containing the samples from the i^{th} trace is then

$$\vec{L}_i = [L_0 \; L_1 \; \ldots \; L_{N_s}]_{1 \times N_s} . \tag{4.10}$$

For analysis, a *measurement matrix*, $\mathbf{L}$, composed of traces corresponding to the N_t individual (m, k) pairs is formed as

$$\mathbf{L} = \begin{bmatrix} \vec{L}_0 \\ \vec{L}_1 \\ \vdots \\ \vec{L}_{N_t-1} \end{bmatrix} = \begin{bmatrix} \vec{L}_{0,0} & \cdots & \vec{L}_{0,N_s-1} \\ \vec{L}_{1,0} & \cdots & \vec{L}_{1,N_s-1} \\ \vdots & \ddots & \vdots \\ \vec{L}_{N_t-1,0} & \cdots & \vec{L}_{N_t-1,N_s-1} \end{bmatrix}_{N_t \times N_s} , \tag{4.11}$$

where $\mathbf{L}_{i,j}$ corresponds to the j^{th} sample of trace i. Thus, each row of the measurement matrix corresponds to a trace taken for the i^{th} (m, k) pair, and each column corresponds to the sample taken at a particular instant in time relative to the start of each encryption operation, across all observed traces.

For each trace, the corresponding input and master key are also stored in byte-matrices for later use, i.e.,

$$\mathbf{M} = \begin{bmatrix} \vec{m}_0 \\ \vec{m}_1 \\ \vdots \\ \vec{m}_{N_t-1} \end{bmatrix} = \begin{bmatrix} \vec{m}_{0,0} & \cdots & \vec{m}_{0,16} \\ \vec{m}_{1,0} & \cdots & \vec{m}_{1,16} \\ \vdots & \ddots & \vdots \\ \vec{m}_{N_t-1,0} & \cdots & \vec{m}_{N_t-1,16} \end{bmatrix}_{[N_t \times 16]} . \tag{4.12}$$

and,

$$\mathfrak{K} = \begin{bmatrix} \vec{\mathfrak{K}}_0 \\ \vec{\mathfrak{K}}_1 \\ \vdots \\ \vec{\mathfrak{K}}_{N_t-1} \end{bmatrix} = \begin{bmatrix} \vec{k}_{0,0} & \cdots & \vec{k}_{0,N_B-1} \\ \vec{k}_{1,0} & \cdots & \vec{k}_{1,N_B-1} \\ \vdots & \ddots & \vdots \\ \vec{k}_{N_t-1,0} & \cdots & \vec{k}_{N_t-1,N_B-1} \end{bmatrix}_{[N_t \times N_B]} , \tag{4.13}$$

The result of the acquisition phase is composed of three sets of data:

1. The $(N_t \times N_s)$ *measurement matrix*, $\mathbf{L}$, containing the digitally sampled side channel data,
2. The $(N_t \times 16)$ *input matrix*, $\mathbf{M}$, containing the plain-text associated with each captured leakage trace, and
3. The $(N_t \times N_B)$ *master key matrix*, $\mathfrak{K}$, containing the master key associated with each captured leakage trace.

Step 2. Intermediate result computation. To characterize the security of a particular cryptographic implementation against the range of possible SCA attacks, it is necessary quantify information leakage for the duration of the full cryptographic operation. Most published SCA attacks target the outer rounds of the cipher since the intermediate results of those rounds can be exhaustively predicted if the input or

```
1: function AESComputeIntermediates(m,𝕂)
2:     s^{0,0} ← m
3:     for i ← 0, N_r − 2 do
4:         s^{i,1} ← AddRoundKey (s^{i,0}, 𝕂^i)
5:         s^{i,2} ← SubBytes (s^{i,1})
6:         s^{i,3} ← ShiftRows (s^{i,2})
7:         s^{i+1,0} ← MixColumns (s^{i,3})
8:     end for
9:     s^{N_r−1,1} ← AddRoundKey (s^{N_r−1,0}, 𝕂^{N_r−1})
10:    s^{N_r−1,2} ← SubBytes (s^{N_r−1,1})
11:    s^{N_r−1,3} ← ShiftRows (s^{N_r−1,2})
12:    s^{N_r,0} ← AddRoundKey (s^{N_r−1,3}, 𝕂^{N_r})
       return s
13: end function
```

Figure 4.4 Pseudo code to calculate and store all algorithmically specified intermediate results computed during the full AES encryption operation for a given input, $\mathbf{m}$, and key schedule, $\mathbb{K}$.

output is known. Standaert, et al. introduced the notion of *predictability* to describe intermediate results that meet this requirement [SÖP04].

However, more recent techniques such as *algebraic side channel analysis* and *chosen* or *adaptive input* techniques [RS09, RSVC09, LPdH10] have demonstrated the ability to directly target multiple leaked intermediate results from the middle rounds. Therefore, limiting an evaluation to better-known outer-round attacks neglects important exploitable leakages. This is particularly true for implementations that implement protective countermeasures based on the assumption that only outer rounds will be attacked, leaving the inner-rounds unprotected.

The `AESComputeIntermediates` algorithm in Fig. 4.4 executes an AES encryption algorithm as described in [Nat01] while preserving the state after each AES primitive operation. The retained intermediate state information is combined to construct a five-dimensional *intermediate cipher state matrix*, $\mathbb{S}_{[N_t \times 4 \times 4 \times 4 \times (4 \cdot N_r)+1]}$ (Fig. 4.5). The individual elements of the full matrix for the t^{th} trace are denoted $\mathbb{S}^{i,j}_{t,r,c}$.

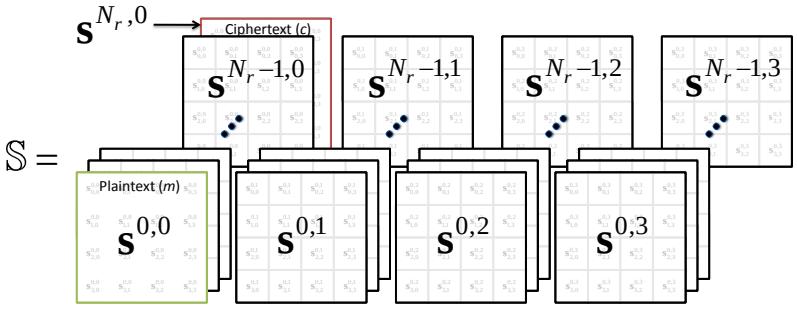

Figure 4.5 Construction of the *intermediate cipher state matrix*, $\mathbb{S}$, for a single AES encryption operation. For an AES encryption operation there are $N_s = (4 \cdot N_r) + 1$ separate states computed, inclusive of the plain-text and cipher-text.

The same basic process is repeated to construct the full *intermediate key schedule state matrix*, $\mathbb{T}_{[N_t \times N_{ks} \times 4]}$, based on the intermediate results computed during a straightforward implementation of the AES *key expansion* algorithm. Since a different (random) key is associated with each trace, one $\mathbb{T}$ matrix is computed and saved for each acquired profiling trace. Individual matrix elements are denoted $\mathbb{T}_t^{i,j}$ where t is the trace index, i is the loop counter as defined in the key expansion algorithm in [Nat01], and j is an index corresponding to the individual intermediate computational step within each loop iteration.

Step 3. Model data-dependent leakage variations. To discover all potentially exploitable data-dependent variability in side channel emissions, several different leakage models should be considered. Although some general observations have been made in previous work about which models are best suited for specific architectures, focusing too narrowly on a particular leakage model and corresponding attack is likely to neglect other important sources of information leakage. Rather than choosing one specific leakage model that may not be well-suited to the particular implementation being considered, a more systematic approach is taken to avoid any inadvertent oversights.

The results herein were prepared by considering leakage models based on Hamming Weight and the Hamming Distance of the states separated by 1 to 4 computational steps, for all algorithmically specified intermediate computation results. The Hamming Distance between the result of two states is denoted $\mathcal{HD}^n(\cdot,\cdot)$ where n is the number of intermediate computation steps separating the initial and final state values used in the computation. For example, the *hypothetical leakage matrix*, $\mathbb{H}$, modeled as the Hamming Distance between the AES plain-text and the output of round 0 (input to round 1) is

$$\mathbb{H} = \mathcal{HD}^4\left(\mathbb{S}^{0,0}, \mathbb{S}^{1,0}\right),$$

since the input to each round is separated by four primitive computational steps.

The leakage assessments conducted to validate the approach introduced herein considers each of the following leakage transformations

- $\mathcal{HW}$: $\mathcal{HW}\left(\mathbb{S}^{0,0}\right)$, ..., $\mathcal{HW}\left(\mathbb{S}^{N_r,0}\right)$
- $\mathcal{HD}^1$: $\mathcal{HD}^1\left(\mathbb{S}^{0,0}, \mathbb{S}^{0,1}\right)$, ..., $\mathcal{HD}^1\left(\mathbb{S}^{N_r-1,3}, \mathbb{S}^{N_r,0}\right)$
- $\mathcal{HD}^2$: $\mathcal{HD}^2\left(\mathbb{S}^{0,0}, \mathbb{S}^{0,2}\right)$, ..., $\mathcal{HD}^2\left(\mathbb{S}^{N_r-1,2}, \mathbb{S}^{N_r,0}\right)$
- $\mathcal{HD}^3$: $\mathcal{HD}^3\left(\mathbb{S}^{0,0}, \mathbb{S}^{0,3}\right)$, ..., $\mathcal{HD}^3\left(\mathbb{S}^{N_r-1,1}, \mathbb{S}^{N_r,0}\right)$
- $\mathcal{HD}^4$: $\mathcal{HD}^4\left(\mathbb{S}^{0,0}, \mathbb{S}^{1,0}\right)$, ..., $\mathcal{HD}^4\left(\mathbb{S}^{N_r-1,0}, \mathbb{S}^{N_r,0}\right)$

Finally, for each considered leakage transformation, the data sub-block size is varied over $N_D \in \{1, 8, 16, 32, 64, 128\}$ to account for differing natural machine representations that might be present on an arbitrary hardware or software AES implementation. Although it is impractical to directly exploit (predict) the larger ($N_D \geq 64$) data sub-block sizes using exhaustive techniques such as standard DPA attacks, the information gleaned from analysis of leakages based on models of the the larger data block sizes aid in understanding what aspects of an implementation are causing leakages.

This semi-comprehensive approach allows the technique to be applied to various implementations of the AES that optimize the algorithm for a particular hardware or software architecture (e.g., use of T-Tables) without loss of generality.

The result of this step is the hypothetical leakage matrices containing the predicted leakages under each assumed model, i.e.

$$\underset{(N_D)}{\mathbb{H}} = \mathcal{F}_L \left(\underset{(N_D)}{\mathbb{S}} \right) \tag{4.14}$$

for $\mathcal{HD}$ models, where N_D is the data sub-block size and $\mathcal{F}_L$ is the leakage transformation function. The individual elements of a byte-oriented ($N_D = 8$) hypothetical leakage matrix are denoted $\mathbb{H}^{i,j}_{t,r,c}$ where i is the AES round, j is the computational step within the round, t is the trace index, and (r, c) are the row and column of the AES state matrix. Note that for Hamming Distance models, (i, j) corresponds to the first argument in the $\mathcal{HD}(\cdot, \cdot)$ leakage transformation.

Leakages related to the key scheduling procedure are modeled in a similar manner. The details of the procedure are omitted here in the interest of space.

Step 4. Known-Key Correlation. The final step of the leakage mapping procedure is to quantify the leakage from an implementation under the assumptions of each considered leakage model. The objectives of this step are two-fold:

1. To identify potentially exploitable *leakages* (data-dependent variations) of all intermediate results, and
2. To quantify the magnitude of all identified leakages.

To identify and quantify leakages, we use a *known key correlation* procedure based on the correlation power analysis (CPA) technique [BCO04, MOP07]. The primary difference between a CPA attack and our technique is that herein the objective is not to infer a key (which is known *a priori* to the evaluator) but rather to

identify leakages and quantify their magnitudes. Therefore, only the correct (known) sub-keys need be considered.

Although the evaluator may have detailed knowledge of precisely when each intermediate computation occurs, this technique does not rely on any such information. This keeps the approach as general as possible while reducing the risk of an evaluator inadvertently overlooking leakages that occur outside of the expected times.

The basic procedure involves computing pairwise correlation coefficients between each column of the hypothetical leakage matrix, $\mathbb{H}$, and each column of measurement matrix, $\mathbf{L}$. The `AESKnownIVCorrelation` algorithm in Fig. 4.6 illustrates the procedure for a byte-oriented leakage model.

A variation of this known-key correlation procedure is exhaustively performed for the intermediate results of the AES encryption and key scheduling algorithms under each considered leakage model, adjusting the procedure as necessary to account for the varying data sub-block size. Correlation results are also computed directly for the full key schedule matrix, $\mathbb{K}$, to identify any leakages due to direct key manipulation not captured in modeling of the specified algorithm.

The output of this step is a correlation matrix, $\mathbb{R}$, for each considered leakage model. Each element of $\mathbb{R}$ represents the correlation between a) the hypothetical leakage modeled from a particular computational step or intermediate result of the cipher and b) the observed side-channel signal sampled at some instant in time (relative to the start of the cipher), computed across all traces.

The overall result is subjectively (through visual analysis of the graphical representation) and statistically analyzed to identify aspects of the implementation that exhibit problematic leakages.

It is assumed that the start of each trace is *temporally aligned*, i.e., the first sample of each trace corresponds to the same instant in time relative to the actual

```
1: function AESKnownIVCorrelation(ℍ,L)
2:     for r ← 0, 3 do
3:         for c ← 0, 3 do
4:             for i ← 0, N_r − 1 do
5:                 for j ← 0, 3 do
6:                     for s ← 0, N_s − 1 do
7:                         ℝ^{i,j}_{t,r,c} ← Corr(ℍ^{i,j}_{(),r,c}, L_{(),s})
8:                     end for
9:                 end for
10:            end for
11:        end for
12:    end for
       return ℝ
13: end function
```

Figure 4.6 Pseudo code to perform a known-key correlation between a byte-oriented hypothetical leakage matrix, $\mathbb{H}$ and the measurement matrix, $\mathbf{L}$.

start of the encryption operation. For evaluations conducted by a system designer this can easily be accomplished by designing the implementation to supply a trigger signal to the acquisition system at the appropriate time. For systems where precise triggering is not possible, alternate approaches are to use a real-time signal-monitoring device such as the commercially available Riscure *icWaves* device [Ris09] or temporal alignment through post-acquisition signal processing.

Step 5: Interpretation of results. For a given ρ, the number of traces needed to determine whether the correlation is statistically significant (different from zero) can be determined using [MOP07]

$$N = 3 + 8\left(\frac{Z_{(1-\alpha)}}{\log\frac{1+\rho}{1-\rho}}\right)^2. \tag{4.15}$$

where Z is the critical value for $1-\alpha$ statistical confidence, which can be computed using statistical software or looked up in most statistical textbooks. Mangard, et al. suggest using $\alpha = 0.0001$ (99.99% confidence) which gives $Z = 3.719$ to estimate the number of traces required to mount a successful DPA attack [MOP07].

This same relationship can estimate the maximum tolerable leakage (in terms of correlation coefficient) to achieve DSCA resistance for a specified number of traces. Solving (4.15) for ρ yields

$$\rho_{max} = \frac{\exp\left(\frac{Z_{(1-\alpha)}}{\sqrt{\frac{N-3}{8}}}\right) - 1}{\exp\left(\frac{Z_{(1-\alpha)}}{\sqrt{\frac{N-3}{8}}}\right) + 1}. \tag{4.16}$$

If the comprehensive leakage assessment reveals any correlations that exceed the maximum threshold, ρ_{max}, then there is a high likelihood that the implementation fails to meet the desired security objective. Table 4.2 illustrates the relationship between the maximum tolerable ρ for a given objective number of traces, N_t, as a function of the required statistical significance, α. For example, to achieve DSCA resistance for up to $N_t = 100,000$ traces using $\alpha = 0.0001$, the leakage should meet the condition of $\rho \leq 0.0166$ for all considered leakage models.

Our results during extensive pilot studies suggest that $\alpha = 0.0001$ provides a good guideline for estimating the number of traces necessary for a completely successful differential CPA attack, i.e., one that successfully eliminates *all* guessing entropy. However, in many scenarios, a CPA attack can be considered successful if the key candidates are sufficiently reduced to make brute forcing the remaining search space computationally feasible. Thus, we suggest using a more conservative $\alpha = 0.1$ when using (4.16) as a test of DSCA resistance. Returning to the previous example, under this more conservative guideline an implementation should not exhibit any correlations that exceed 0.0057 to claim DSCA resistance for up to $N_t = 100,000$ traces.

Note that any claim of SCA resistance under this test is valid only under the evaluated combinations of leakage model parameters. More accurate leakage models may enable an adversary to successfully extract key material with less traces than indicated by this test. Additionally, it is important to keep in mind that this

Table 4.2 Maximum tolerable correlation (ρ_{max}) to achieve N_t-trace DSCA-resistance as a function of α, predicted using (4.16).

	Conf. Level (α)			
N_t	0.1	0.01	0.001	0.0001
1,000	0.0573	0.1038	0.1375	0.1650
10,000	0.0181	0.0329	0.0437	0.0526
100,000	0.0057	0.0104	0.0138	0.0166
1,000,000	0.0018	0.0033	0.0044	0.0053
1,000,000,000	0.0006	0.0010	0.0014	0.0017
10,000,000,000	0.0002	0.0003	0.0004	0.0005

procedure does not directly evaluate resistance to advanced profiling techniques such as template attacks.

4.5 Experimental Methodology

To evaluate the effectiveness of our techniques, the full leakage mapping approach was carried out against several different hardware and software AES implementations. We present results from two representative implementations (*Imp. A* and *Imp. B*) are presented; the characteristics of the two studied implementations are summarized in Table 4.3.

4.5.1 Implementation A. *Imp. A* was implemented on a low-cost 16-bit PIC micro-controller unit (MCU) representative of the type used extensively in commercial security applications (e.g., smart-cards, garage door openers, remote key-less entry systems, etc.), and fabricated using an unspecified 180-nm process technology. For device control and measurement, the PIC was mounted on an evaluation board and controlled through a serial interface. The development board was powered from a standard lab DC power supply to reduce effects of uncontrolled supply voltage fluctuations.

The radiated EM data from *Imp. A* was collected using a specially designed RF near-field probe with a wide-band preamplifier connected to a Lecroy 104-XI-A high speed digital sampling oscilloscope. The EM data was collected from an unmodified PIC MCU. The chip was not specially prepared or decapsulated, and all EM measurements were taken by placing the probe directly over the area of the chip that exhibited maximal energy in a narrow band around the known clock rate.

All data was collected at a sample rate of $f_{S1} = 5$ GSa/sec with a $W_{LP} = 1$ GHz low pass anti-aliasing filter inserted between the probe and the oscilloscope. The acquired data was down-sampled to an analysis sample rate of $f_{S2} = 200$ MSa/sec using proper decimation (i.e., every 25^{th} sample is retained and all others are discarded). The higher sample rate was chosen to permit post-acquisition simulation of various receiver configurations using data collected under identical environmental conditions.

For leakage mapping, a total of $100,000$ traces were acquired using random key and plain-texts drawn from the uniformly distributed key and message spaces. The signals were analyzed in their raw-acquired form without any averaging or other noise reduction post-collection processing steps. For attack validation, additional measurements were taken from the same device under fixed key conditions for $N_k = 100$ random keys, and $N_t = 1000$ traces per key.

4.5.2 Implementation B. The data for *Imp. B* is the publicly available power consumption data from the second *DPA Contest* [Par10]. The DPA Contest data set was chosen to demonstrate the feasibility of exhaustive leakage mapping for a device fabricated using the more modern 65 nm process technology and to allow reproduction of the results presented herein.

The *Imp. B* data set was collected from a Virtex 5 FPGA, which is a modern re-programmable logic device fabricated using a 65 nm process technology [Xil11]. The Virtex 5 was mounted on the commercially available SASEBO-GII board [fISR11]

Table 4.3 Summary of characteristics for evaluated implementations.

	Implementation ID	
	Imp. A	*Imp. B*
Algorithm	AES	AES
Key Size	128	128
Mode	Encryption	Encryption
Type	Software	Hardware
Device	Microchip PIC24 MCU	Xilinx Virtex 5 FPGA
Clock Rate	29.5 MHz	24.0 MHz
Side Channel	EM (Near-field RF)	Power
Sample Rate	200 MSa/sec	5 GSa/sec
# Profiling Traces	$100,000$	$1,000,000$

which is specially designed to facilitate control and side channel measurements of cryptographic designs.

For leakage mapping, the DPA Contest *template* data set was used. The template data set includes $1,000,000$ traces collected using random keys and plaintexts. For attack validation, we used the DPA Contest *public* data set, composed of $N_t = 20,000$ traces for each of $N_k = 32$ random keys. According to the available documentation, each trace in these data sets was averaged 10 times during acquisition to reduce the effect of external noise sources.

Additional details on the experimental setup and acquisition procedures for *Imp. B* are available on the DPA Contest [Par10] and SASEBO [fISR11] websites.

4.5.3 Data Alignment. For experimental efficiency, both sets of data were acquired with the aid of a trigger signal asserted by the target device at the start of each encryption operation of interest. This allows for near perfect alignment without requiring extensive post-collection processing of the acquired data. No further

alignment was performed on the *Imp. A* data set prior to processing. The DPA Contest data sets used for *Imp. B* are also nearly perfectly aligned.

In practice, a trigger is not required, and we successfully demonstrated acquisition setups with real-time triggering using only the monitored side channel emissions of both device types. Thus, the lack of an available trigger signal does not prohibit employment of our techniques, but does increase the required data processing time. The data from each target implementation was collected using random plain-text / key combinations for the number of profiling traces indicated in Table 4.3.

4.5.4 Resource Requirements. All results were obtained on a standard scientific workstation equipped with dual quad-core Xeon processors, a two terabyte hard-drive, and 72 GB of RAM. A comparable workstation can be obtained or built for approx $5,000 U.S. in 2011. All code was written in Matlab and optimized for computation speed (vice memory efficiency).

Execution of the full leakage mapping procedure for either implementation, across all considered parameter combinations, takes approximately 12 hours on this workstation. Although the memory requirements are high, the same results should be obtainable using a workstation with significantly less available memory by optimizing the computations to work primarily from disk.

4.6 Results

This section contains a representative sample of the results obtained by applying the full leakage mapping procedure to the two AES implementations described in Sec. 4.5.

The technique introduced in Sec. 4.4 was applied to selected data sets to assess the overall leakage from each implementation. The resulting data is evaluated to identify the maximal exhibited leakages, and the resulting leakage maps were studied to determine how much detail about the implementations a naïve adversary would

Table 4.4 Summary of correlation results for all considered *byte-oriented* leakage models.

	Implementation A			Implementation B		
Leakage Model	Min	Max	Mean	Min	Max	Mean
$\mathcal{HW}$	0.77	0.91	**0.84**	0.00	0.08	0.01
$\mathcal{HD}^1$	0.63	0.87	**0.73**	0.01	0.11	0.04
$\mathcal{HD}^2$	0.01	0.63	0.17	0.00	0.07	0.01
$\mathcal{HD}^3$	0.01	0.02	0.02	0.04	0.07	**0.05**
$\mathcal{HD}^4$	0.01	0.07	0.03	0.04	0.07	0.05

be able to obtain from the application of typical differential attack techniques under common leakage models.

Some selected results of these analyses are presented below. Note that although the results obtained through analysis of non-byte oriented leakage models are highly informative, the results presented herein are restricted to the more common byte-oriented model in the interest of space.

4.6.1 Intermediate Cipher Leakages. The information leakage from each implementation was evaluated using the procedure in Sec. 4.4 for both cipher and key expansion intermediate computations under all considered leakage models. Table 4.4 summarizes the minimum, maximum, and mean correlation results observed for the considered byte-oriented ($N_D = 8$) leakage models.

As expected, the observed leakage from *Imp. A* (software) is much stronger than the leakage observed from *Imp. B* (FPGA). This can be attributed primarily to the architecture of the FPGA implementation, which processes all 16 bytes of the AES state in parallel. The uncorrelated parallel activity has the effect of acting as a noise generator and reducing the effective signal-to-noise ratio for the targeted intermediate result(s).

In contrast, the software implementation is constrained by the limited size of the data path, and all circuit activity (with the exception of other on-chip housekeeping or peripheral activity) within a particular clock cycle is thus dedicated to a single AES primitive operation.[3]

4.6.2 Leakage Maps. Although a tabular summary of the leakage assessment can capture the overall results, a visual presentation of the results permits a much more natural and efficient interpretation given the large number of considered parameter variations.

The visual presentation also aids the evaluator in drawing intuitive conclusions about what the most problematic leakages are, when and under what model parameters they occur, as well as quickly highlighting the presence of any unexpected leakages. Because the visual representation can be thought of as a mapping of the observed leakages in terms of time and intermediate results, we refer to them as *leakage maps*.

4.6.2.1 Temporal Leakage Maps. A *temporal leakage map* is constructed as a two-dimensional plot where the x-axis represents the sample time, and the y-axis corresponds to the intermediate step being modeled. Each correlation coefficient is represented as a single pixel, where the pixel intensity represents the relative strength of the identified correlation at a particular instant in time, normalized to the maximum observed correlation for the currently considered leakage model. One temporal leakage map is prepared for each data sub-block under each combination of model parameters, i.e., ($N_M = 128/N_D$) separate leakage maps are prepared for each leakage model, where N_D is the number of bits in each data sub-block.

[3]Because the PIC24 has a 16-bit data path, some operations are actually optimized to manipulate two bytes of state information or intermediate results at a time.

Because the leakage characteristics of sequential software and parallel hardware implementations are very different, the results of the assessments for *Imp. A* and *B* are each presented slightly differently.

For *Imp. A* the leakage due to each modeled intermediate step is sparse compared to the dimensionality of the correlation result matrices. As a result, it is difficult to visually discern the leakages if each correlation coefficient is simply mapped to a pixel intensity. Therefore, the correlation results for *Imp. A* are prepared using contour plots, which highlights the sparse leakages. A typical example of a temporal leakage map for *Imp. A* is shown in Fig. 4.7. This figure shows the magnitude of the correlation observed between the modeled leakages of all AES intermediate results under a byte-oriented Hamming Weight leakage model.

Several interesting observations can be made about the leakage from *Imp. A* based on a visual inspection of Fig. 4.7. First, the $N_r = 10$ encryption rounds and four primitive operations within each round are immediately obvious. Second, the EM side channel significantly leaks the Hamming Weight of *all* algorithmically specified intermediate results computed throughout the entire encryption operation. Thus, it can be inferred that this particular implementation closely follows the AES specification. Note that the similarity between the correlation of steps $2 - 3$ during each round is expected under a Hamming Weight model since the `SR` operation does not actually change any intermediate results.

For *Imp. B*, the observed leakages are no longer sparse in comparison to the dimensionality of the results matrices, and a direct mapping from the correlation matrix to the temporal leakage map is possible. A typical example of a temporal leakage map for *Imp. B* is shown in Fig. 4.8.

The example shown is the result of the leakage assessment for a byte-oriented $\mathcal{HD}^4$ leakage model. Although direct exploitation of the $\mathcal{HD}^4$ model has until recently been considered impractical for the standard DPA-class of attacks, analysis of the results from the $\mathcal{HD}^4$ leakage model reveals a great deal of information about

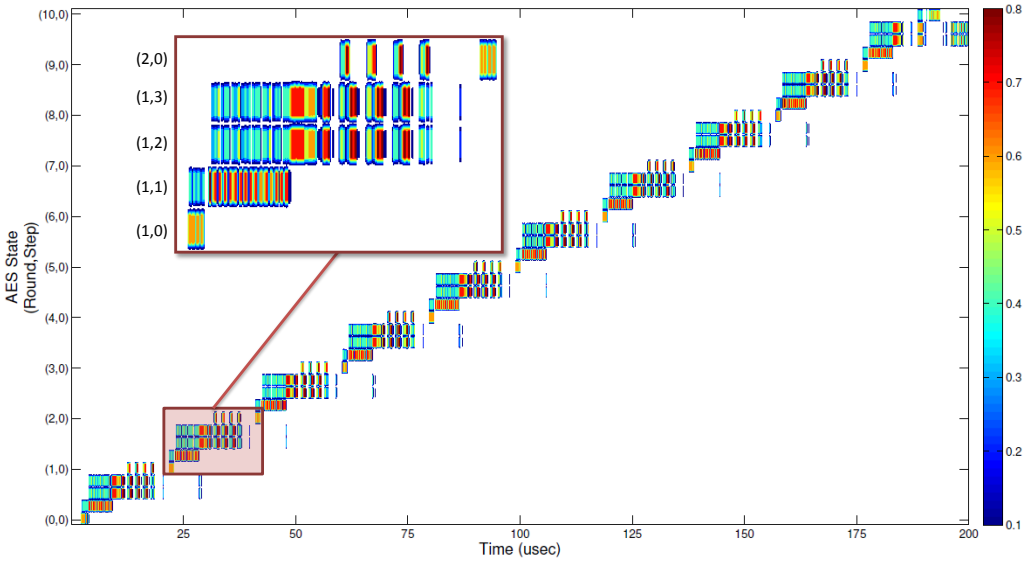

Figure 4.7 Temporal leakage map for *Imp. A*. under the byte-oriented $\mathcal{HW}$ model. The intensity represents the *maximum* observed magnitude of the leakage observed at each sampled instant across all 16 bytes of the AES state matrix.

what an adversary could discern about the underlying implementation architecture through the application of similar techniques. Furthermore, recent results indicate advances in computing technology are now making direct attacks on 32-bit key hypothesis feasible [MKP11], which makes consideration of such leakage models even more relevant.

The round structure of *Imp. B* is, again, immediately obvious from a cursory inspection of Fig. 4.8. However, in contrast to the leakage from *Imp. A*, *Imp. B* does not show high correlations for all computed intermediate results under the $\mathcal{HD}^4$ model. Rather, the higher relative correlations are isolated to the Hamming Distance between intermediate results following the `ARK` steps (i.e., $\mathcal{HD}(\mathbb{S}^{i,1}, \mathbb{S}^{i+1,1})$) in adjacent rounds. This strongly implies the use of registers at that point in the algorithm.

Examination of the source code of *Imp. B* confirms this, and provides experimental confirmation that the $\mathcal{HD}^4$ leakage transformation is a suitable abstrac-

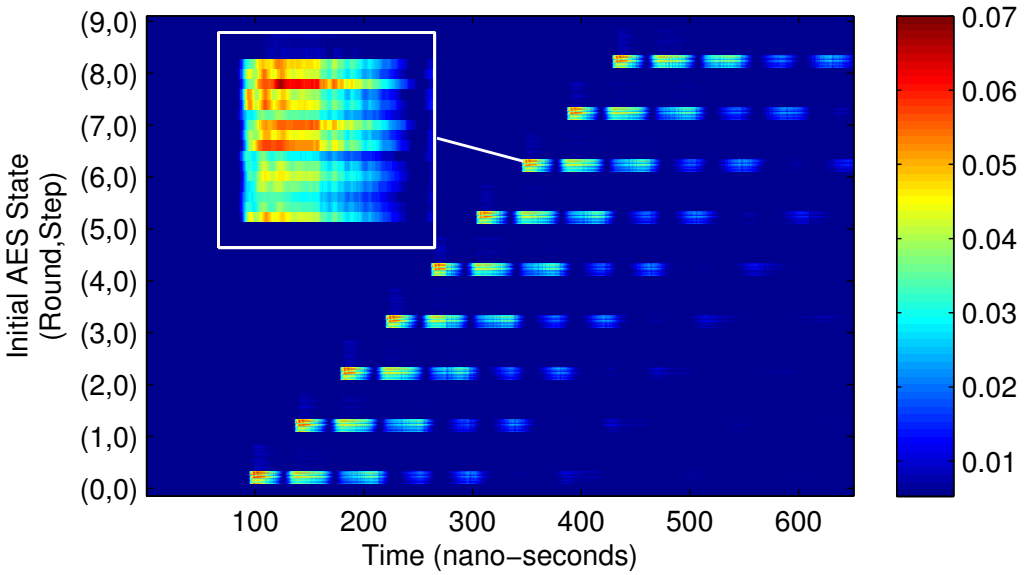

Figure 4.8 *Temporal Leakage Map* for *Imp. B* under a byte-oriented $\mathcal{HD}^4$ leakage model. The x-axis represents sample time, and each band along the y-axis represents one of the 37 *initial* states used in the Hamming Distance computation. Close examination reveals the variations between individual bytes are also visible within each band. The intensity represents the magnitude of the observed correlation at each instant in time, normalized to the maximum observed under this leakage model.

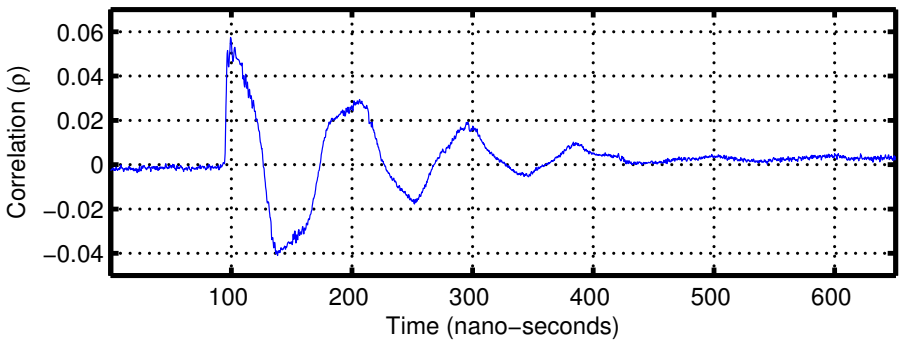

Figure 4.9 Typical correlation plot for *Imp. B* under a byte-oriented $\mathcal{HD}^4$ leakage model, illustrating the damped oscillatory nature of the correlation well beyond the originating clock cycle. The x-axis represents sample time, and the y-axis represents the observed correlation coefficient.

tion of the physical leakages produced by register updates for this implementation. These basic observations suggest that while the leakage of information under the $\mathcal{HD}^4$ model may not lead directly to successful key recovery attacks, the information leaked could be used by an adversary to craft more powerful techniques that take advantage of the particular implementation architecture.

Another interesting observation is that the leakages of *Imp. B* appear as a damped oscillatory response that continues well beyond the clock cycle in which the modeled computation is carried out. Fig. 4.9 shows a typical plot of the correlation between a modeled leakage and the power consumption of the circuit observed over the the duration of the encryption operation. It was confirmed through numerous pilot experiments that the leakages in the later clock cycles do lead to successful DPA attacks, and can be combined through various techniques to improve the effectiveness of standard DPA approaches.

4.6.2.2 Summary Leakage Maps. Whereas a temporal leakage map provides a visual summary of an implementation's leakage over time, a more useful tool for summarizing the leakage of each individual data sub-block is the *summary leakage map*. The summary leakage map is a concise graphical representation of the maximum leakages identified for each data sub-block, computational step, and

leakage model. The graph is constructed by determining the maximum observed (magnitude) correlation for each combination of parameters. The resulting tabular data is then graphically represented by mapping each correlation coefficient to a color intensity, normalized to the maximum observed correlation across all of the considered leakage models.

Summary leakage maps for *Imp. A* are depicted in Fig. 4.10. The summary maps for the $\mathcal{HD}^2$, $\mathcal{HD}^3$, and $\mathcal{HD}^4$ models are omitted since, as expected, the software implementation exhibits no statistically significant leakage under those models.

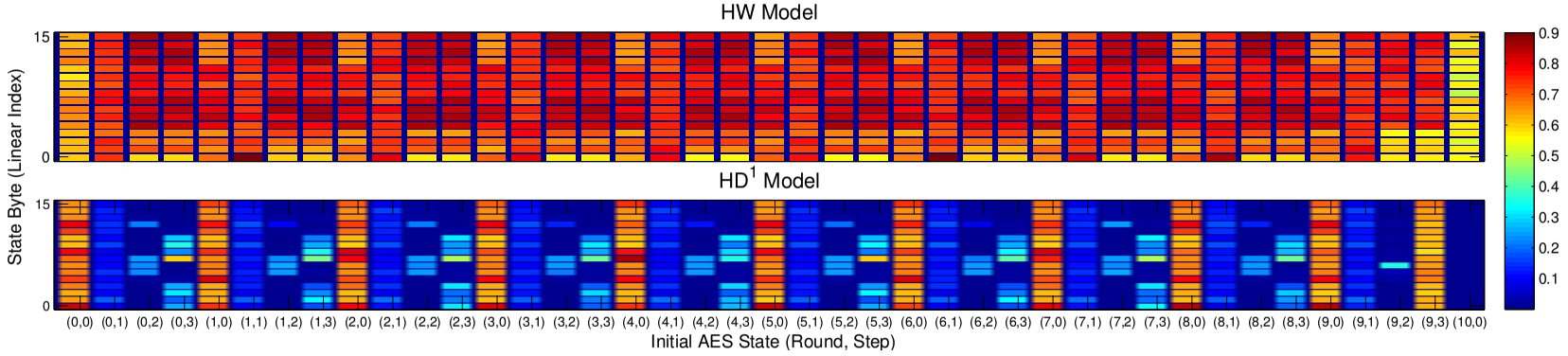

Figure 4.10 *Summary leakage map* showing maximum observed leakage from *Imp. A* across all algorithmically specified AES intermediate computation steps for the byte-oriented $\mathcal{HW}$ and $\mathcal{HD}^1$ leakage models. Each row (y-axis) corresponds to one byte of the AES state matrix indexed linearly, and each column (x-axis) corresponds to one intermediate computational step or result. The intensity represents the maximum magnitude of the correlation normalized to the global maximum correlation observed across all considered leakage models.

Inspection of Fig. 4.10 immediately reveals the extent of the problematic leakages associated with this implementation. As already noted, *Imp. A* significantly leaks the Hamming Weight of every intermediate value computed.

Additionally, there is also strong leakage associated with $\mathcal{HD}^1(\mathbb{S}^{i,0}, \mathbb{S}^{i+1,0})$, which corresponds to the `ARK` primitive operation in each round. Although this could be due to a register update, the observed leakage is more likely the result of the relationship between the $\mathcal{HD}^1$ model and the AES round keys. Note that the Hamming Distance across the `ARK` operation is determined by the number of bits toggled during the `XOR` operation with the associated portion of the round sub-key, which in turn is determined by the number of '1' bits in the key. Thus $\mathcal{HD}^1(\mathbb{S}^{i,0}, \mathbb{S}^{i+1,0})$ is equivalent to $\mathcal{HW}(\mathbb{K}^i)$.

Given this, it appears likely that this implementation leaks the Hamming Weight of the round keys. In fact, our results indicate that the key leakage exhibited is sufficient to permit key extraction from this particular implementation without knowledge of the plain-text or cipher-text.

One final note related to *Imp. A* is that although it is not evident from the $\mathcal{HD}^1$ summary leakage map, significant (exploitable) leakage is also caused by intermediate steps other than `ARK`. However, the relative strength of the `ARK` has the effect of masking the presence of these smaller leakages due to the normalization of the color intensity to the maximum observed correlation. Note that any masked leakages can be readily identified by a direct examination of the tabular results.

In general, it is assumed that system designers will address the strongest leakages first since they are presumably the easiest to exploit. It is recommended that the full leakage mapping procedure be repeated after implementing any SCA countermeasures, since any changes could incidentally introduce new sources of leakage. Once the stronger leakages have been eliminated or managed, any remaining leakages are more easily identified in the graphical leakage maps. In general, it is assumed that system designers will address the strongest leakages first since they are presumably

the easiest to exploit. It is recommended that the full leakage mapping procedure be repeated after implementing any SCA countermeasures, since any changes could incidentally introduce new sources of leakage. Once the stronger leakages have been eliminated or managed, any remaining leakages are more easily identified in the graphical leakage maps.

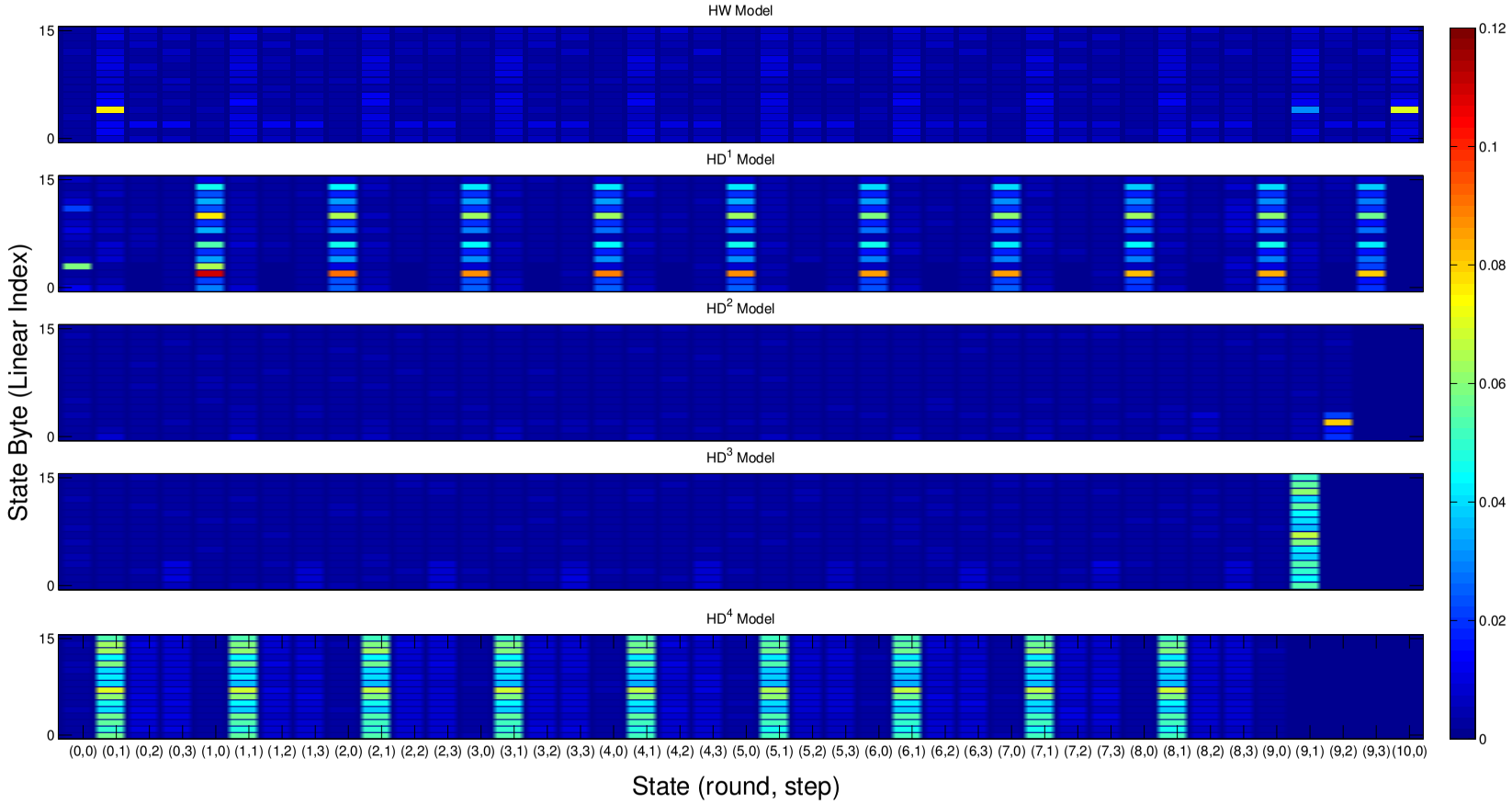

Figure 4.11 *Summary leakage map* showing maximum observed leakage of the AES intermediate computation steps under each considered *byte-oriented* leakage model for *Imp. B*. Each row (y-axis) corresponds to one byte of the AES state matrix, and each column (x-axis) corresponds to the starting step used in the Hamming Distance computation (or the precise step used in the case of the Hamming Weight model). The intensity represents the maximum magnitude of the correlation observed for each state byte under a particular leakage model, normalized to the maximum correlation observed under all leakage models.

Comparing the summary leakage map from *Imp. A* (Fig. 4.10) to that of *Imp. B* (Fig. 4.8) shows how different the leakage characteristics of the two implementations are. Whereas the PIC micro-controller shows strong leakage of the Hamming Weight across all intermediate computations, the FPGA implementation only exhibits strong leakage under the $\mathcal{HD}^1$, $\mathcal{HD}^3$, and $\mathcal{HD}^4$ models. Such leakages are characteristic of hardware implementations. Here, it is noteworthy that the $\mathcal{HD}^3$ leakage model only exhibits strong (relative) leakage for $\mathcal{HD}^3(\mathbf{s}^{9,1}, \mathbf{s}^{10,0})$. This is attributed to the replacement of the `MC` step in the final round with an additional `ARK`, as illustrated in Fig. 4.2. This bears additional scrutiny because the register update at this location meets the criteria of predictability using standard DPA techniques.

It is believed that the presence of strong $\mathcal{HD}^1$ leakage can be attributed to direct leakage of the round keys, $\mathbf{K}$, generated during real-time key scheduling—similar to the direct key leakage exhibited by *Imp A*. Again, examination of the source code supports this hypothesis since the generated round key is stored in a temporary register for each round. Additional experimentation would be necessary to confirm whether key expansion is the root cause of this leakage, and whether or not the leakage is vulnerable to direct key extraction.

4.6.3 Attack Validation. Using (4.16), the correlation results from each implementation are used to estimate the number of traces a standard correlation SCA attack would require to correctly extract the full master key. These predictions were compared to the number of traces actually required using standard correlation-based DSCA attacks. For *Imp. A*, the targeted intermediate for the DEMA attack was the output of the initial (Round 0) `SB` operation under a Hamming Weight model, i.e. $\mathcal{HW}(\mathbb{S}^{0,2})$. For *Imp. B*, the target was the Hamming Distance across the last three states, i.e., $\mathcal{HD}^3\left(\mathbb{S}^{9,1}, \mathbb{S}^{10,0}\right)$. Both attacks were carried out using a single sample taken at the instant found to exhibit the largest magnitude leakage under the corresponding leakage model.

Table 4.5 Comparison of number of traces required for successful attack as predicted by (4.15) vs. actual results of a correlation-based DPA attack on *Imp. B* using a $\mathcal{HD}^3(\mathbf{s}^{9,1}, \mathbf{s}^{10,0})$ leakage model.

	Predicted (α)			Actual		
	0.1	0.001	0.0001	Indiv. Byte	Full Key	GE < 32 bits
Min.	758	3,054	6,362	300	9,400	**2,200**
Mean	**1,469**	5,927	12,352	$6,861^4$	$17,272^4$	3,883
Max.	2,490	10,051	20,950	$> 20,000$	$> 20,000$	4,237

Table 4.5 shows the results of the attack validation for *Imp. B*.[4] The attack was attempted against each of the 32 available trace-sets in the DPA Contest *public* database, where each trace-set corresponds to a different secret key. Traces used for the attack were drawn from each available trace set in a randomly determined order. The minimum number of traces required to extract an individual key byte was 300. Out of the $32 \times 16 = 512$ byte extractions attempted, the attack failed to extract 11 of the key bytes after the available trace set ($N_t = 20,000$) was exhausted. Thus, the standard attack fails to achieve a first-order success rate of 1 [SMY09] given the limited number of traces.

Unfortunately for cryptographic engineers, the failure of an attack to extract the correct key with 100% certainty does not imply that the key will not be found. A more practical measure of attack success in scenarios where one or more single plain-text-cipher-texts pair is available is the remaining Guessing Entropy [SMY09] which provides a measure of the remaining cost of a brute-force attack given the current ranking of the true key in the list of key hypotheses.

Fig. 4.12 shows the maximum, average, and mean guessing entropy for the attacks against all 32 keys as a function of the number of traces used in the CPA

[4]The mean values shown are estimates since all available $N_t = 20,000$ traces were exhausted before all key bytes were successfully extracted.

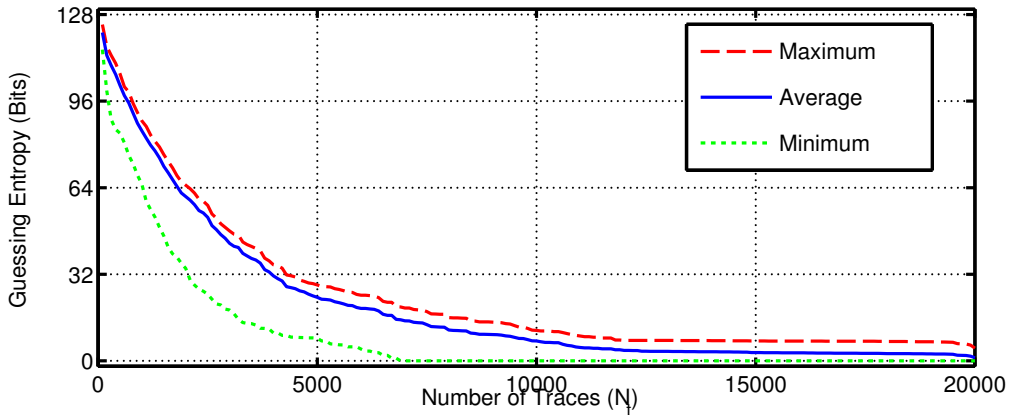

Figure 4.12 Guessing entropy of *Imp. B* as a function of the number of traces used in a correlation-based DPA attack on $\mathcal{HD}^3(\mathbf{s}^{9,1}, \mathbf{s}^{10,0})$.

attack. Clearly, although the attack occasionally *fails* on the basis of extracting the full key, the information leakage is sufficient to enable a practical brute force attack given a much smaller number of traces. If it is assumed, very conservatively, that it becomes practical to brute force the key when the Guessing Entropy is reduced to ≤ 32 bits, an average of approximately 4000 traces are required for an attack to succeed against this implementation.

Once again, it is extremely important to note that the abstract algorithmic model used in this particular attack is very likely to be sub-optimal, and more accurate models will probably succeed with fewer traces. Thus, when using (4.16) to predict DSCA-resistance based on the leakage model assessment, it is recommended to use a conservative $\alpha = 0.1$ or 0.2.

4.6.4 Suitability for Protected Implementations. Initial results indicate our approach is an effective tool for assessing the leakage from both protected and unprotected implementations. Several pilot studies, not presented herein, successfully used leakage mapping to assess the leakage from a software-based smart-card implementation using common countermeasures. We note that use of the leakage mapping methodology to effectively assess masked implementations would require

supplementing the simple Hamming Weight and Hamming Distance leakage models with more complex multi-variate leakage models, however the procedure required to do this is straightforward.

In general, the introduction of countermeasures significantly reduces the correlation at a particular instant in time by introducing various forms of noise or reducing the signal strength. Our systematic approach enables an evaluator to quickly and efficiently repeat the assessment process to identify how much more resistant an implementation is after adding a countermeasure. Perhaps more importantly, it also allows a designer to ensure that the implementation of one countermeasure does not introduce new unexpected sources of information leakage. The preliminary results indicate that the leakage mapping assessment methodology is well suited to assess whether a particular countermeasure is justified given the added cost in time, space, and energy.

4.7 Conclusion

We have developed a systematic, efficient process through which cryptographic engineers can assess the resistance of a particular implementation against differential SCA techniques. While our technique does not guarantee an implementation will be secure against all future side channel attacks, it does provide an efficient mechanism through which system designers and testers may gain substantial insight into the level of security of a particular implementation. Additionally, as illustrated throughout this paper, examination of the leakage maps permits insights that are not at all obvious if testing is carried out in a less thorough manner.

Our techniques can be adapted to incorporate more advanced statistical tools (e.g. mutual information or linear regression models)—in particular those capable of capturing non-linear leakages. However, the inclusion of any such techniques must be weighed in terms of the added benefit obtained compared to the additional computational effort required. Additionally, our studies to date have been

focused exclusively on signals in the time domain. Various other work has suggested that spectral-domain techniques may be even more effective, particularly against implementations which implement basic countermeasures such as random process interrupts.

For devices where the template attack scenario [CRR02, ARRS05] is deemed a legitimate threat, designers must also consider the multivariate leakage from the device in terms of profiled attacks. The methodology introduced by Standaert, et al. [SMY09] provides a solid foundation on which to evaluate whether a sound leakage model can be formed to attack the device using profiled attacks. However, an efficient methodology for formulating the leakage models to be tested in this context remains an open problem.

4.8 Supplementary Discussion

This section contains additional relevant material that could not be included in the submitted paper due to mandatory space limitations. It is assumed that most readers of this article will be familiar with the procedure of DSCA attacks in general, and correlation-based DSCA attack in particular. However, for completeness, the procedure employed in this research is described below.

4.8.1 Correlation-Based DSCA. The procedure used to carry out the correlation-based DSCA attacks is based on the correlation DPA described in [MOP07]. The general approach is shown in Fig. 4.13. Each step of the procedure is described below:

Step 1. Choose a target *intermediate computation* and corresponding *leakage model*. The chosen intermediate computation must depend on some small portion of the key and some observable or controllable input or output data t, and can be one or more bits. The results herein are based on byte-oriented leakage models. For Imp. A, the targeted intermediate was the

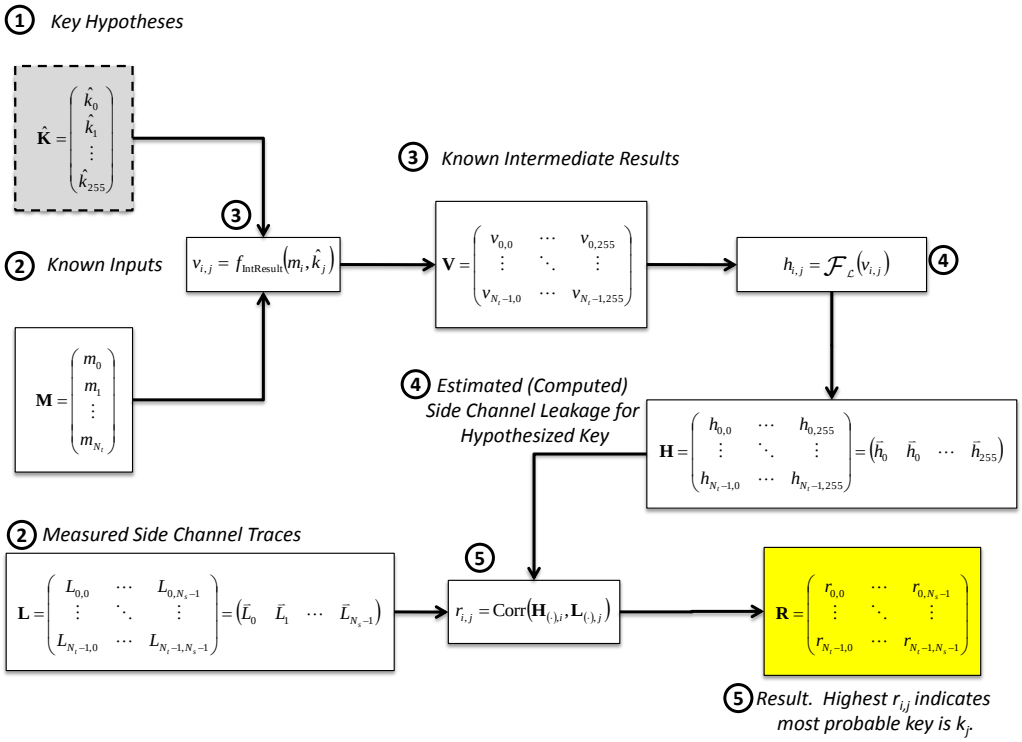

Figure 4.13 Block diagram of a generalized DSCA attack procedure for one byte of the key, assuming a $\mathcal{HW}$ leakage model. Note that steps 3-4 require minor modifications for use with leakage models that consider transitions between two states, e.g., $\mathcal{HD}^3(\cdot,\cdot)$.

output of the SB operation for Round 0 under a Hamming Weight model, i.e. $\mathcal{HW}(\mathbf{s}^{0,2})$. For Imp. B, the target was the Hamming Distance across the last three states, i.e., $\mathcal{HD}^3(\mathbf{s}^{9,1}, \mathbf{s}^{10,0})$. Note that the first attack assumes an observable or controllable plaintext, while the second attack assumes an observable ciphertext.

Step 2. Record side channel data. Record N_s samples of side channel data (e.g., strength of the EM field near the circuit) during each of N_t cryptographic operations performed using an unknown fixed key. Construct the *leakage matrix*, $\mathbf{L}$, and known-message matrix, $\mathbf{M}$ as described in Step 1 of Sec. 4.4. Note

that in the attack scenario, the adversary has no knowledge of the secret key, which is the objective of the attack.

Step 3. Considering one byte of the AES state at a time, compute the hypothetical intermediate state matrices for the states considered under the targeted leakage model for all possible (hypothetical) subkeys, $\hat{k}_i$, where $i \in \{0, 1, \ldots, 255\}$. The hypothetical results are computed by considering each possible subkey as the true subkey, and computing the intermediate computation results for each known byte of the observed message matrix, $\mathbf{M}$. For the byte-oriented attack, there are $2^8 = 256$ possible subkeys associated with each byte of the intermediate result. The result of this step is the $(256 \times N_t \times 4 \times 4)$ hypothetical intermediate result matrix, $\mathbb{V}$.

Step 4. Map each hypothetical *intermediate result* to a predicted side channel leakage value under the selected leakage model. In this step, each element of the matrix $\mathbb{V}$, from Step 3 is mapped to an estimated side channel leakage using the leakage model, i.e., $\mathcal{HW}(\mathbb{V})$ for *Imp. A*. The result of this step is the hypothetical leakage matrix, $\mathbb{H}_{[256 \times N_t \times 4 \times 4]}$.

Step 5. Compute the correlation between predicted hypothetical values and observed side channel data. In the final step, the hypothetical side channel leakages $\mathbb{H}$ computed in Step 4 are compared to the actual data $\mathbf{L}$ captured in Step 2 using (4.9), i.e., $\mathbb{R} = \texttt{Corr}\,(\mathbb{H}, \mathbf{L})$. Each column of the $\mathbf{H}_{[(),(),r,c]}$ sub-matrix corresponds to one of the 256 hypothesized subkeys for key byte $\mathbb{K}^0_{r,c}$ (for *Imp. A*) or $\mathbb{K}^{10}_{r,c}$ (for *Imp. B*). The columns of $\mathbf{L}$ represent the side channel leakage from the device at a particular point in time. At any particular point in time, it is not known in advance whether information leakage is present in the side channel signal due to the targeted intermediate value. Thus, the adversary must search over all sampled instants to find times when information about the intermediate value is leaked. The most probable correct key is the one that exhibits the maximum correlation coefficient between the

modelled leakages under all hypothetical keys and the observed side channel measurements at any of the considered instants in time. Alternatively, subkey candidates can be ranked in descending order of their maximum observed correlation. An adversary with access to a single known plaintext-ciphertext pair can then attempt to bruteforce the true key by trying the keys in order of likelihood. Note that an evaluator with knowledge of the true key can determine the remaining guessing entropy by computing the product of the rank of each of the 16 correct subkeys at the end of the attack procedure.

5. Conclusion

This chapter concludes the main document and provides an overall summary of the research activities and key findings, followed by several recommendations for future research.

5.1 Research Summary

The information leakage of electronic devices, especially those used in cryptographic or other vital applications, represents a serious practical threat to secure systems. While physical implementation attacks have evolved rapidly over the last decade, relatively little work has been done to allow system designers to effectively counter the identified threats. This work addresses the technology gap between the identified problems and potential solutions, and makes significant contributions to the study of information leakage in two primary areas of investigation:

1. *RF-DNA fingerprinting* of integrated circuits for device authentication, and
2. *Leakage mapping* to assess the information leakage from arbitrary cryptographic implementations.

The results and major contributions related to each area of investigation are described below.

5.1.1 RF-DNA Fingerprinting of Integrated Circuits. Unintentional electromagnetic (EM) emissions were investigated as a source of information to recognize or verify the identity of a unique integrated circuit (IC). The technique investigated, known as radio frequency distinct native attribute (RF-DNA) fingerprinting, was adapted from previous work (cf. [SITMM08, KTM09, RTM10, RPT11, HBK06, WMTM10, WTR10]) on intentional EM emissions. The technique was successfully adapted herein to recognize individual microchips based on fabrication process-

induced variations in each chip's unintentional RF emissions in a manner analogous to biometric human identification.

The problem of IC authentication has numerous practical applications, including 1) providing enhanced security for secure access mechanisms (e.g., anti-cloning), 2) detection of unauthorized modifications to circuit designs (e.g., hardware Trojan detection), and 3) forensic attribution of electronic evidence in criminal or other cases. It is believed that this work is the first to propose and demonstrate the feasibility of using the unintentional emissions for IC recognition.

Whereas all previously known IC recognition techniques require either hardware or software modifications to the device being recognized, the RF-DNA fingerprinting technique permits passive authentication based on analysis of the unintentional emissions produced during pre-existing processes and protocols. Because it is passive, it is suitable for security applications involving commodity commercial ICs without requiring any modifications to the device being authenticated. Thus, the proposed approach is very promising for security applications such as those requiring detection of cloned, copied, or counterfeited devices. Furthermore, because the technique does not require any modifications to the ICs, the approach is more cost-effective and scalable than other known techniques for applications involving commodity commercial ICs.

In addition to being the first application of RF-DNA fingerprinting techniques to the unintentional emissions of ICs, this research extends the previous work related to the intentional emissions of wireless networking equipment, in two new ways. Previous RF-DNA work has predominantly considered device *identification* tasks. However, the primary use case envisioned for IC fingerprinting is to counter cloning and related threats, which requires identity *verification*. A systematic approach was developed and introduced to evaluate the effectiveness of RF-DNA fingerprinting in the context of both identification and verifications tasks. Additionally, the RF-DNA

fingerprinting technique was extended from a fixed 3-class approach to one capable of identifying or verifying the fingerprints of devices for an arbitrary $N-$class problem.

An extensive empirical study was conducted and the performance of RF-DNA fingerprinting was evaluated under a wide range of simulated noise conditions. Empirical results indicate the technique scales well for both identification and verification tasks involving 40 near-identical devices. For experimentally collected emissions, the technique correctly identifies devices greater than 99.5% of the time, with average verification equal error rates (EERs) of less than 0.05% achieved using a single extracted fingerprint. Correct identification success rates of better than 90% were maintained under analysis conditions of SNR $\geq$ 15 dB.

The impressive performance indicates that RF-DNA fingerpinting is adaptable to less ideal conditions while still providing acceptable results. Finally, these results were obtained using a single extracted fingerprint. A substantial improvement in performance is believed to be realizable through a straightforward extension of the approach for multiple extracted fingerprints.

5.1.2 Leakage Mapping. The second major contribution of this work is the development and demonstration of a *leakage mapping* methodology for assessing the information leakage from arbitrary block cipher implementations. Prior to this work, [SMY09] provided the only proposed methodology to enable system evaluators to quantitatively bound the leakage from an evaluated implementation. However, this earlier work relies on the evaluator's ability to build an optimal template attack, and the end result is limited in focus to a small portion of the overall cryptographic algorithm.

The framework proposed here provides a comprehensive approach to assess the information leakage from all algorithmically specified key-dependent intermediate computations for implementations of symmetric block ciphers. The resulting leakage assessment quantitatively bounds the resistance of an implementation to

the general class of differential side-channel analysis (SCA) techniques, and provides system designers and evaluators with a tool that can be used to objectively assess whether countermeasures implemented are justified given the added cost in time, space, and energy compared to the obtained reduction in exploitable information leakage. Furthermore, the systematic approach enables evaluators to quickly and efficiently repeat the assessment process for different variations of implementations, which helps to ensure the additions of countermeasures does not inadvertently introduce new unexpected sources of information leakage.

While using this technique does not guarantee an implementation will be secure against all future side-channel attacks, it does provide an efficient mechanism through which system designers and testers may gain substantial insight into the level of security of a particular implementation. Examination of the leakage maps permits insights that are not at all obvious when testing is carried out in a less thorough manner.

The framework was demonstrated using the well-known Hamming Weight and Hamming Distance leakage models, with recommendations for extension of the technique to more accurate models. The approach was applied to two typical unprotected implementations of the Advanced Encryption Standard (AES), and the assessment results were empirically validated against correlation-based differential power and electromagnetic analysis (DPA/DEMA) attacks.

5.2 Recommendations for Future Research

A number of recommendations for future research were made in each article in the main body of this work. Those recommendations are revisited here with additional discussion.

5.2.1 RF-DNA Fingerprinting of ICs. Although the results of the empirical studies conducted thus far are very promising, a considerable amount of work

remains to fully understand the suitability of RF-DNA fingerprinting for practical security implementations. Intuitively, the nature of the intrinsic characteristics that induce inter-device variations suggests a fingerprint based on those variations will be extremely difficult to impersonate. However, further analysis and experimentation are needed to confirm this. Particular areas for additional study include:

5.2.1.1 Permanence and robustness of RF-DNA features under varying environmental conditions. It is likely that normal operation of an IC over its lifespan will affect the physical device structure and resulting RF-DNA fingerprints. More studies are necessary to assess the sensitivity of RF-DNA fingerprinting to such physical changes over long periods of time. If structural changes are found to cause the fingerprint to change significantly in a way that adversely affects fingerprint performance, one solution might be to design the authentication system to update the training database and reference fingerprints each time a device is successfully authenticated. In practical implementations, uncontrolled environmental fluctuations (e.g., temperature or supply voltage) are also expected to effect the fingerprints. Previous work has shown that environmentally-induced fingerprint variations for intentional emitters can be compensated for effectively by conducting the enrollment training procedure over the range of expected operating temperatures and voltages [TSU04]. Additional studies should be conducted to assess the suitability of this approach for application to the unintentional emissions of ICs.

5.2.1.2 Sensitivity of fingerprint performance to variations due to different sensor modules or sensor positioning. Another anticipated source of performance degradation is variations in the physical sensor characteristics, receiver components, and sensor positioning relative to the IC. The experiments conducted in this work used a single sensor and receiver module and controlled the sensor positioning over each IC. In practice, each device reader introduces its own unique effects on the fingerprint characteristics. Thus, the fingerprinting procedure must select fea-

tures that are insensitive to receiver and sensor induced variations. The sensitivity of RF-DNA fingerprints to such variations should be studied further to determine if performance is still acceptable for different sensor and receiver modules under more realistic operating conditions. A logical test case is smart-card based authentication tokens, where the amount of variation in the positioning of the smart-card and an embedded EM sensor in the reader could be controlled in practical applications. Variations due to card positioning in a contact smart-card reader could be simulated very realistically using a motorized XY stage to control probe positioning.

5.2.1.3 Scalability to larger databases. The RF-DNA fingerprinting technique performed very well for the 40 near-identical devices tested in this work for both verification and identification tasks. It is believed that the MDA approach employed is scalable to much large databases of devices, at least for the verification task. One limitation of the MDA technique employed herein is that it requires an initial number of features that exceeds the number of total potential classes. For identification, the straightforward MDA implementation will encounter practical limits as the number of classes exceeds the number of available (or computationally feasible) features. However, for identity verification applications of IC fingerprinting, many smaller databases can be used to ensure a single verification set does not violate the MDA requirements. Since in an authentication scenario the claimed device identity is known prior to attempting the classification, only the sub-database containing the claimed identity need be considered. Investigation of this solution could be tested using simulation to reduce the manual workload and cost associated with fingerprinting a large number of devices.

5.2.1.4 Challenge-response sequence optimization. The results herein were obtained by arbitrarily designating several clock cycles of an overall operation sequence as the *response* region. Although no statistical difference in performance was observed during limited trials when the designated response region was varied to

include different sub-regions, additional performance improvements may be obtained by more carefully choosing or defining the response. Future research into this area might focus on a rigorous investigation into the performance of fingerprints based on microcode instruction sequences that exercise different device subcircuitry.

5.2.1.5 Effectiveness for programmable or custom logic ICs such as FPGAs or ASICs. This work evaluated the effectiveness of the RF-DNA fingerprinting technique for low-cost commercial microcontrollers typical of those used in a wide-variety of modern security applications. However, the ICs tested were fabricated using a 180nm lithography process. An interesting extension of this work would be to conduct additional experiments to confirm the technique's suitability for other classes of devices such as FPGAs or custom ASICs, as well as those fabricated using more modern fabrication processes with much smaller features sizes.

5.2.1.6 Performance improvement through use of multiple fingerprints. Although it is anticipated that achievable SNRs for most applications of this technique will be very high in practice since the emissions are captured using a near-field probe, it is believed that identification and verification accuracies could be substantially improved over the results herein by considering multiple extracted fingerprints for the classification decision. It would be worthwhile to examine how much further performance could be increased, particularly for degraded SNR conditions, by either averaging multiple signals or by extending the Bayesian classification technique by iteratively classifying additional fingerprints until a desired confidence in the decision is reached.

5.2.1.7 Investigation of alternate side-channels. It is believed that the same techniques employed in Chapter 3 would be similarly effective if the RF emissions were replaced by other side-channel emissions such as variations in power consumption. If the power side-channel is found to have similar performance, it may

be more suitable in some situations such as those where precise probe positioning is difficult to achieve reliably. The experiments in Chapter 3 could easily be replicated for alternative side-channels by simply replacing the measured RF signal with the variations in power consumption. Power consumption variations can be obtained using a typical power-based side-channel analysis setup where the voltage drop is measured across a resistor placed in-line with the circuit's supply and/or ground line as illustrated in Sec. 2.7.

5.2.2 Leakage Mapping. Future work related to leakage mapping should focus on continuing to enhance the procedure to provide a more robust overall leakage assessment for arbitrary cryptographic implementations. A number of possible improvements to the procedure were suggested in Chapter 4, and are expanded on here.

5.2.2.1 Advanced Statistical Techniques. In particular, the techniques should be adapted to incorporate more advanced statistical tools such as mutual information or linear regression models rather than depending on the fixed Hamming Weight or Hamming Distance Models. The non-parametric mutual information approach is interesting because of the potential to capture non-linear information leakages, but the computational complexity is high. Thus, any efforts to employ such an approach would need to focus on identifying efficient techniques for computing the required non-parametric statistical distributions to make the systematic application of MIA feasible. This problem may be suitable for the application of parallel processing techniques using scientific high performance computing clusters or graphical processing unit (GPU) acceleration.

Linear regression based techniques (e.g., [SLP05]) are also highly promising because they can automatically adapt the leakage model to each considered sample or instant in time. Thus, models constructed using regression techniques might be considered to be leakage agnostic since they can adaptively capture the statisti-

cal relationship between the observed signal variance at each sample and the data of interest. It is believed that such an approach would be substantially more robust at the expense of significantly increased computational effort. An assessment conducted using leakage mapping augmented with regression-based models would be much more representative of the types of DSCA attacks a persistent, adaptive adversary might employ. Although the computational requirements would increase significantly, limited initial experiments suggest that a regression-based approach is feasible for the leakage mapping framework.

5.2.2.2 Suitability for Assessing Protected Implementations. Although some pilot studies were conducted to evaluate the suitability of leakage mapping against protected cryptographic implementations, the results so far are very limited due to the limited availability of protected implementations for study. To truly assess the suitability of the leakage mapping approach as a tool for making appropriate design decisions in the development of a protected implementation, it should be used directly in that enviornment. That is, it should be employed in an iterative fashion as a cryptographic implementation is designed and various countermeasures are applied to determine how well it guides the decision making process. At each step, it is also recommended that the leakage mapping results be validated against well-known attack techniques as in Chapter 4.

5.2.2.3 Frequency Domain Leakage Mapping. For this research, the leakage mapping approach was applied exclusively to signals in the time domain. However, various other work has suggested that spectral-domain techniques may be even more effective, particularly against some implementations that implement basic countermeasures such as random process interrupts [RO04b]. In general, the only change that would be necessary to apply leakage mapping to signals in the frequency or time-frequency domains would be input signal pre-processing and interpretation of the resulting correlation matrices. Spectrogram or similar pre-processing techniques

would be particularly interesting since the leakage assessment produced from time-frequency domain signals would identify both problematic instants in time as well as the frequency bands containing the majority of the leaked information.

5.2.3 Related Future Research Recommendations. In the course of this work, a large number of experiments were conducted that led to a variety of interesting observations that have not yet been fully investigated. One observation in particular—discovered during development of the leakage mapping procedure—merits further investigation.

It was emphasized in Section 4.4 that a significant advantage of the systematic leakage mapping approach is that it can help to prevent the inadvertent oversight of important leakages. Because leakage is assessed across all intermediate computations during the entire cryptographic algorithm, any unexpected leakages are highlighted when the full assessment results are reviewed.

The software-based AES implementation (*Imp. A*) studied in Chapter 4 exhibits such unexpected leakages. For this software implementation, direct key correlation would typically be expected twice in each round. This is because each round key is manipulated during the `ARK` operation and during on-the-fly key scheduling. Strangely, several bytes of the master key exhibit leakage during rounds other than the first round during which the master key also serves as the round key. Based on knowledge of the AES algorithm, this leakage is unexpected since the master key is not typically manipulated, directly or indirectly, at those times. Examination of the available source code for this implementation confirmed that the master key was not manipulated at the times when the unexpected leakage occurred.

The source of the key leakages was identified by estimating the clock cycle that corresponds to each unexpected leakage (using the MPLAB PIC simulator) and determining which instructions are being executed at those times. This analysis indicated that the source of the observed leakage is `ARK` operations that mix each

round's input with the associated round key. Further investigation revealed that the temporary round key is stored in a block of memory addresses that immediately preceeds the memory addresses where the master key is stored. This suggests that the content of the master key memory addresses is being leaked without ever explicitly accessing the addresses that contain the leaked information. The physical cause of this leakage is currently unknown, and additional experimentation is necessary to determine the root cause. Notably, even without identifying the cause, these leakages could easily be avoided by adding a *buffer* around the sensitive data in memory to prevent any accesses to adjacent memory locations.

Appendix A. Side Channel Analysis Countermeasures

A.1 Constant Time

Kocher, in his original paper on timing attacks, suggested a number of possible approaches system developers can take to reduce the vulnerability of their systems to similar attacks [Koc96]. The most obvious approach to counteract timing attacks is to make all operations take *constant time*.

Realizing this approach has proven to be difficult in practice, and some operations that were long thought to resist timing attacks (e.g., table lookups) have later proven to be vulnerable, as described in 2.5.1. When candidates algorithms were evaluated for selection as the new AES algorithm, vulnerability to SCA attacks was one of the criteria considered but table lookup based implementations were considered to be resistant to timing attacks. It has since been shown that, at least on general purpose microprocessor implementations, cache architectures make that an invalid assumption as described in Section 2.5.1 [Ber05].

Additionally, as Kocher originally noted, ensuring the intended output is produced in constant time is insufficient to prevent timing attacks since other measurable phenomenon can reveal the time taken by intermediate operations [Koc96]. Finally, *constant time* countermeasures have the undesirable effect of making all operations take the longest time, effectively de-optimizing the performance to the lowest common denominator [Roh06, MOP07].

A.2 Constant Power Countermeasures

One way to reduce the information signal produced through the power consumption side channel is for all operations to require constant power. If operations take precisely the same amount of power without regard for the data being processed, then power analysis attacks are no longer possible and the adversary must seek an alternate (hopefully less informative) source of information leakage. This approach,

unlike some of the others, must be implemented at the hardware level, and is therefore not applicable to software-based implementations that run on general purpose devices such as microprocessors. The term commonly used for constant-power logic design styles is *secure logic*.

Several *secure logic* variations have been proposed, which fall into two primary categories: transistor-level [TAV02,BGLT06], and gate-level logic styles [TV04, PM05]. Transistor-level logic styles typically perform better in the sense that they are more effective at reducing power consumption fluctuations. However, they typically can't be implemented using existing commercial design flows and standard cell libraries. Therefore, the cost and time required to develop systems based on these logic styles is much higher than for a standard CMOS design. In contrast, gate-level secure logic assembles compound constant-power logic cells from existing standard CMOS logic cells (e.g., AND and OR gates). These logic styles can generally be implemented using existing design flows, commercial tools, and standard cell libraries but are less effective at reducing variations in power consumption. In general, all of the proposed secure logic styles sacrifice power, size, and / or performance to reduce the side-channel information leakage of the circuit's power consumption.

Both the transistor and gate-level logic styles are based on the idea of dual or multi-rail logic. Figure A.1 illustrates a transistor-level pre-charged dual-rail domino style that achieves near constant power consumption profiles for all inputs and outputs.

The logic cell requires an input pair of both the *true* signal and its complement for each input. Likewise, it produces an output signal pair consisting of both *true* and *complementary* output values of the logic function, where Y_h represents the output and Y_l is its complement. While the the clock ϕ is low (or logical `'0'`), both output signals are being *pre-charged* to low voltage. When the clock ϕ goes high, the circuit evaluates and one (and only one) of the two output signals is asserted. During proper operation, both output signals should never be `'1'` simultaneously.

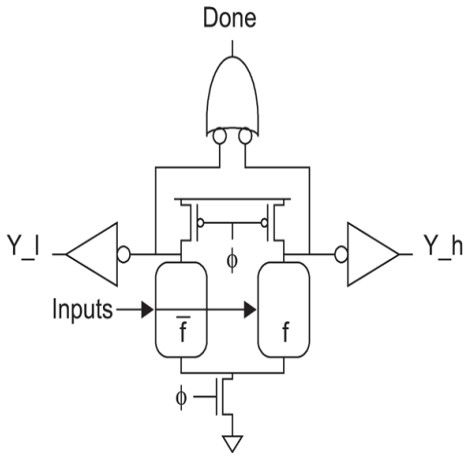

Figure A.1 Dual Rail Domino Logic Style with Completion Detection for Asynchronous Design. [WH05]

Table A.1 summarizes the states for a dual-rail domino logic cell. An example of an XOR gate implemented in dual-rail domino logic is shown in Figure A.2.

Table A.1 Dual-rail domino signal encoding [WH05]

sig_h	***sig_l***	***Meaning***
0	0	precharged
0	1	'0'
1	0	'1'
1	1	invalid

The idea behind the dual-rail design as a countermeasure is since the logic cell always calculates both the true and complementary outputs, either one half or the other of the complementary paths will be exercised during any particular operation. Thus, the power the logic cell draws should be basically independent of the inputs and output.

Note that the dual-rail logic style means that the logic can signal completion by tying both un-inverted outputs to a NAND gate as shown in Figure A.1. In this manner, domino dual-rail can be used for asynchronous logic, which has been proposed as a further countermeasure to DSCA attacks.

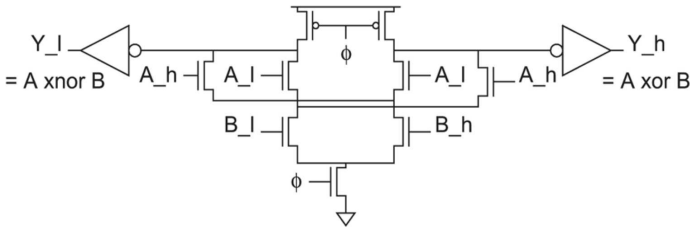

Figure A.2 Dual Rail Domino XOR/XNOR gate (without completion detection explicitly shown). [WH05]

A limitation of this approach is that routing can create capacitive imbalances in the cell, which can bias the power consumption. Several specific variations on the dual-rail logic concept have been proposed in the literature.

- *Sense amplifier based logic (SABL)* [TAV02]. A transistor-level custom synchronous logic style with a single switching event each clock cycle, during which it "...discharges and charges the sum of all the internal node capacitances together with one of the balanced output capacitance" [TV04]. SABL requires the design of an all-new custom cell library.
- *Wave dynamic differential logic (WDDL)* [TV04]. A gate-level semi-custom logic style that combines logic gates from a standard cell library into compound secure gates with constant power characteristics similar to SABL. WDDL can be implemented for both ASIC and FPGA-based designs. In general, WDDL is less effective than SABL at hiding power consumption variations due to internal computations and unbalanced routing of complementary wires can render the countermeasure less effective at reducing data-dependent power fluctuations. The technique can be integrated into an existing design flow using commercial EDA tools and commercially available complementary CMOS standard cell libraries with some modifications (semi-custom design flow) [TV06].
- *Masked dual-rail pre-charge logic (MDPL)* [PM05]. Another gate-level semi-custom logic style formed from the combination of standard cells that attempts

to overcome routing constraints. MDPL combines power equalization with intermediate value *masking* (see Section A.1).

- *Three-phase dual rail power (DRP) logic (TDPL)* [BGLT06]. A transistor-level enhancement to SABL intended to provide constant-power consumption without restrictive routing restraints. TDPL also requires the design of a custom cell library.

Another area of active research is the efficient integration of full- or semi-custom logic styles into existing design workflows. Baddam and Zwoliniski proposed a technique to effectively route circuits following the divided WDDL logic style using backend duplication [BZ08]. Their technique is interesting because it can reportedly be applied to FPGAs as well as custom ASIC designs. The authors presented experimental results from a 130nm technology FPGA demonstrating the technique's effectiveness.

Although hardware-based secure logic styles are promising in theory, achieving equalized power has proven to be very difficult to do in practice. Suzuki and Saeki showed that slight variations in gate input arrival times result in the complementary paths switching at different times, which results in exploitable power consumption leakage [SS06]. Mangard, Popp and Gammel showed that all of the fully complementary CMOS-based logic styles exhibit glitching behavior that exhibits similar characteristics [MPG05]. Schaumont and Tiri have recently shown that the combination of masking and dual-rail logic (e.g. MDPL), which was thought to be one of the most promising hardware countermeasures to prevent DPA, is fundamentally insecure [ST07]. Small differences in the routing of complementary wire pairs leads to imbalanced capacitances. Though these implementations are not directly vulnerable to traditional DSCA-type attacks, the routing imbalances induce a bias in the probability density function of the power consumption which can be filtered to remove the mask, resulting in a much lower DPA resistance than originally thought. The authors were able to extract the key from their AES implementation, originally

believed to be highly resistant to standard DPA attacks, with approximately 2,000 observations.

A.3 Temporal Desyncronization

Noise addition is an intuitive way to cover up information leakage of a circuit. The concept is that if noise can be increased enough, the attacker will be unable to identify and extract the underlying information signal. One way of increasing the noise in a side channel is to desynchronize the time when sensitive data manipulations take place from one operation to another. This can be accomplished by, for instance, randomizing the clock period, the order of operations, or the number of cycles taken for a microprocessor instruction [Koc96, Roh06, MOP07].

The theory behind these randomization techniques is that DSCA attacks are sensitive to the temporal alignment of the side channel traces. Variations in the start time of individual traces, or variations in the time when subsequent sensitive operations or data manipulations occur relative to that start will introduce noise into the corresponding side-channel signal. The effect on DSCA techniques is that the peaks in the correlation coefficients (or whatever statistical tool is used) which normally indicate a correct key are spread across a number of points in time. In digital signal processing this is known as *incoherent averaging* [CCD00]. If the start of traces are not temporally aligned, the adversary must either collect more samples or use sophisticated signal processing techniques to align them (see Section 2.7) [MOP07].

An alternate technique to achieve temporal descynchronization is to implement circuits using asynchronous or self-timed logic styles. Removing the clock as a source of information makes the task of side-channel analysis much more difficult and has the added benefit of countering clock-based fault attacks [FMP03]. In addition to the properties of constant power consumption that can be achieved through careful logic design, the dual-rail logic styles introduced above also lend themselves to

asynchronous design since each cell inherently signals the completion of an operation [FMP03].

A.4 Masking countermeasures

Masking countermeasures are a class of techniques that protect sensitive information from side channel attacks by making the physically observable phenomena *independent* of sensitive data. Masking of asymmetric ciphers is commonly referred to as *blinding* in the cryptographic literature [Koc96, MOP07].

Kocher first proposed applying *blinding* as a countermeasure to timing attacks [Koc96], but the technique is also effective at reducing vulnerability of systems other side channel attacks [CJRR99, Roh06, MOP07]. Kocher's blinding technique for protecting the private key is described in detail in Appendix D.

The generalized application of masking to side channel attacks was proposed by Chari, et al. as a derivative of the already known *secret sharing* technique from the cryptographic literature [CJRR99]. Masking, in general, is accomplished by randomizing sensitive intermediate values processed by the device. The idea is to randomly *split* sensitive intermediate values into a number of shares, each of which is independent of the original sensitive data. Each sensitive unmasked intermediate value, v, is split into d shares where the relation

$$m_1 \star m_2 \star \cdots \star m_d = v \tag{A.1}$$

holds for some group operation $\star$. The shares $m_1 \ldots m_{d-1}$ are the *random masks*, and m_d is the *masked intermediate value*. The shares $m_1 \ldots m_{d-1}$ are assumed to be mutually independent random variables uniformly distributed over v. Typical choices for the masking operation, $\star$, are the logical XOR (*boolean masking*) and modular addition or multiplication (*arithmetic masking*) [MOP07, PRB09].

Masking does not make physically observable phenomena independent of the device operations, but rather randomizes the data on which the device is operating. Thus, the device still leaks information about the intermediate values being processed, but those intermediate values are independent of the secret or private key. An adversary with no knowledge that the masking countermeasure has been implemented will be unable to attack the device using standard SCA techniques. Even with knowledge that masking is used the difficulty of mounting an attack is significantly increased.

Most of the masking schemes proposed use a single random mask to split the sensitive intermediate value into just two shares, although higher-order masking schemes have also been proposed to provide further security against HO-DSCA techniques [CJRR99,MOP07,PRB09]. The complexity of both implementing higher-order masking schemes and attacking grows quickly with the order, so most research has focused on 2^{nd} order masking and 2O-DSCA attacks [PRB09].

Masking countermeasures are sometimes combined with constant-power logic styles, but that combination has been found to be fundamentally insecure as described in A.2.

A.4.1 Masking of complex ciphers. A key characteristic of *masking* countermeasures is they require changes to the underlying algorithm to handle the masked intermediate values properly. For RSA, the changes are straightforward due to the properties of modular arithmetic, but masking schemes can be substantially less straightforward for more complicated ciphers. For example, masking S-box lookups introduces significant overhead (performance and memory) since the masked equivalent of the entire table would have to be calculated for each new mask [MOP07]. More efficient implementations are possible if the S-box functions are computed dynamically using finite field arithmetic since the entire table doesn't have to be recomputed for each new mask value [OMPR05].

A.4.2 Implementation levels for masking. In addition to masking intermediate values at the algorithmic level (in either hardware or software), it is also possible to mask operations at higher system levels. Mangard et al. describe several techniques for masking hardware multipliers, combinational logic, and data buses [MOP07]. Various proposals have been made for masked logic styles to be implemented at the custom hardware design level for both ASIC and FPGA implementations.

A.4.3 Practical considerations. The security benefits of masking countermeasures can be easily lost if the technique is implemented carelessly. Mangard, et al. warns of several possible pitfalls when implementing masking schemes in practice. Consecutively storing a masked value and its corresponding mask in the same register or inadvertently consecutively transferring them across a data bus may leak the Hamming distance between the two and reveal the unmasked intermediate value. Likewise, intermediate values that are concealed by the same mask and processed consecutively can leak the Hamming distance between the two unmasked intermediate values. Simultaneously processing a masked value intermediate value and the mask used to protect it can unintentionally occur in parallel hardware implementations, effectively leaking the un-masked intermediate value. Finally, implementation tools (compilers, hardware synthesis tools, etc.) may inadvertently optimize away countermeasures if care is not taken [MOP07].

To be effective random masks should be changed frequently enough to prevent the masks themselves from introducing exploitable side-channel leakage. However, every mask change reduces performance of the system, and thus the frequency of changes must be balanced to achieve acceptable performance and security [MOP07].

A.5 Power Supply Shielding

A countermeasure to power analysis techniques is to isolate the internal time-varying requirements of a circuit from easily probed terminals. Shamir was the first to propose physically isolating a circuit's power supply from a circuit by using large shielding capacitors [Sha00]. Others have noted that capacitors used in power distribution networks of complex integrated circuits such as FPGAs filter some of the higher-frequency activity of the internal switching activity [MOP07]. Shamir's approach used two capacitors in tandem so that one could supply the circuit while the other recharged.

Ratanpal, Williams and Blalock pointed out that Shamir's technique can be bypassed by probing the power supplied by the capacitors instead of the global power supply (assuming the capacitors are off-chip or accessible), and reconstructing a full power consumption trace by combining the power traces from the two capacitors and applying standard DPA attack techniques against the combined trace [RWB04]. The authors proposed a more effective active signal suppression circuit. A key disadvantage of all power supply shielding techniques is that the are only effective against power analysis and do not protect against attacks on the EM or other side channels.

A.6 EM Shielding

There are few specific countermeasures that specifically address the problem of EM leakage. Quisquater et al. suggest several possible ways to reduce the vulnerability of systems to EM analysis, including reducing the magnitude of the radiated electromagnetic field through shielding (such as thick metal packaging), *imprisoning* the circuitry inside a Faraday cage to prevent EM radiation from escaping, reducing the power consumption of the device (and thus the resulting EM radiation), asynchronous logic, or using one of the various dual-rail power logic styles proposed elsewhere as a countermeasure to power analysis attacks [QS00, QS01]. Introducing temporal or spatial jitter with techniques such as the ones described in Sections

A.3 and A.8 may also improve resistance to EM-based DSCA attacks. The Faraday cage approach is likely to be effective, but may not be realistic for practical implementations of many systems.

A.7 Leakage-Resistant Arithmetic

Bajard, et al. proposed a leakage-resistant arithmetic (LRA) style based residue number system (RNS) as a countermeasure to timing, SSCA and DSCA attacks [BILT04]. In the RNS representation, large integers are represented by a series of smaller integers. The system is defined by a set of relatively prime *moduli*, $\{m_1, m_2, \ldots, m_N\}$. Any arbitrary integer, X, is represented as a set of N smaller integers $\{x_1, x_2, \ldots, x_N\}$ such that $x_i = X \bmod m_i$. The LRA system randomizes data, order of computations, and the function of individual logic cells by randomly selecting the initial set of moduli used and/or randomly changing the base moduli before and during an arithmetic operation.

The countermeasure is most suitable for public-key ciphers that operate over large finite fields such as RSA or ECC. The authors developed a reference implementation of a reconfigurable logic design to implement LRA for the RSA encryption algorithm. The resulting implementation takes approximately 5-7 times more gates than a standard modular exponentiation implementation, but the inherently parallel nature of the arithmetic operations offers potential performance improvements at larger key sizes ($\geq$ 2048 bits). Although the approach presents some intuitive degree of protection against SCA attacks, no experimental results were presented to validate the claim of improved SCA resistance.

A.8 Dynamic Reconfiguration

One interesting new area of countermeasures research is the idea of dynamically reconfiguring FPGAs or other devices in real-time, as introduced by Mentens et al. [MGV08]. The countermeasure creates temporal and spatial jitter by ran-

domly inserting delay registers into the computation path and changing the location where the logic that processes sensitive operations is instantiated. The temporal jitter descynchronizes the timing of sensitive operations as previously discussed in Section A.3. Spatial jitter increase resistance to EM attacks, which typically attempt to localize the area of maximal EM leakage. Fixed configuration spatial jitter was also previously introduced by Bajard, et al. and Ciet et al., but requires multiple dedicated instantiations of the elementary computational cells, which increases overhead [BILT04, CNPQ03]. Dynamic reconfiguration may achieve similar spatial protection with improved area and resource utilization efficiency.

Appendix B. Kocher's Timing Attack on RSA

B.1 Overview

The seminal paper on cryptanalytic SCA attacks, published by Kocher in 1996, targets the timing variations of several asymmetric ciphers including the popular RSA algorithm. This section describes Kocher's attack on RSA.

B.2 Overview of RSA

RSA use is widespread because of its strong computational security and ease of implementation [Sch96]. Asymmetric ciphers such as RSA are typically much slower than block ciphers such as DES or AES, but the key lengths are typically much larger—1024 bits or higher for RSA vs. 256 bits maximum for AES. Because they are relatively slow, asymmetric ciphers are not generally used for sustained transfers of large amounts of data, but are commonly used for applications such establishing a secure session between two parties over a computer network (e.g., the Secure Sockets Layer (SSL) protocol used for secure Internet browser sessions) or for encrypting or signing emails.

In RSA and other asymmetric ciphers, the ciphering process is based on the mathematical properties of modular exponentiation:

$$R = d^k \mod m \tag{B.1}$$

where d is the data being operated on, m is a publicly known modulus, and k is the cryptographic key.

In RSA and other asymmetric ciphers, cryptographic keys are generated in pairs—a public key k_{pub} which is used to encrypt data and a corresponding private key k_{pri} which can decrypt the data that was encrypted using k_{pub}. A public key is published as the set (k_{pub}, m) and is shared freely with anyone from whom the key's

owner would like to be able to receive secure messages. Asymmetric ciphers used in this manner are typically referred to as public-key encryption schemes [Sch96].

The mathematical details of how RSA keys are generated how/why the ciphering process works are irrelevant to Kocher's timing attack and therefore are not discussed here, but are described in detail by a number of references [Sch96]. It is noteworthy that many of the RSA implementations in use today are based largely on the 'C' source code from the original RSA laboratories reference implementation [Lab94].

B.2.1 Implementation of RSA. In practice, the modular exponentiation operations used in RSA and other similar ciphers are generally implemented using one of several well-known techniques. One common algorithm, which is used as an example in Kocher's paper, is the *binary square and multiply* method (also known as *addition chaining*) shown in Algorithm 4 [Koc96, Sch96].

Algorithm 4 Square and Multiply Modular Exponentiation—Adapted from [Koc96]

1: $s_0 \leftarrow 1$
2: **for** $i = 0$ to $w - 1$ **do**
3: **if** $k_i = 1$ **then**
4: $R_i \leftarrow (s_i \times y) \mod n$
5: **else**
6: $R_i \leftarrow s_i$
7: **end if**
8: $s_{i+1} \leftarrow R_i^2 \mod n$
9: **end forreturn** (R_{w-1})

B.2.2 Information Leakage Modular Exponentiation. There are at least two sources of potential information leakage in this algorithm [Koc96]. The first, which is the focus of Kocher's timing attack, is the conditional execution of the the modular multiplication on Line 4. The multiplication is executed only if $k_i = 1$. This data dependency results in variations in computation time—the effect of which is to leak information about the value of the key. Kocher's attack is based on analysis of

this first source of information leakage. A second source of information leakage is due to variations within the modular multiplication step itself.

B.3 Key Assumptions of Kocher's Attack

With all side channel attacks, it is important to be aware of the underlying assumptions that can limit the attack's practicality. Kocher's timing attack makes a number of assumptions about the attacker's capabilities, which are described here:

1. The attacker (Eve) has the ability to monitor the decryption operations of her target (Bob), and to accurately record the computation time for each decryption.
2. Eve is able to eavesdrop on and capture the encrypted messages that are sent to Bob. (Alternatively, in some scenarios an active adversary could actually be the source of the encrypted messages.)
3. Eve is able to do the above for a large number of decryption operations where Bob is using the same private key.
4. Eve must have knowledge of the amount of time required by Bob to perform the modular multiplication that occurs when a key bit is '1'. One way for Eve to accomplish this is to create a simulator using the same source code as the targeted implementation, running on a similar hardware platform. She can then precisely simulate the amount of time each partial calculation should have taken on her target platform.[1]

[1]Realistically, this is not an impractical constraint from the perspective of an attacker. A scenario where Eve may have this capability is if she is attacking one of the many widely used RSA implementations that are based on the original RSAREF reference implementation (which is freely available) or use open-source implementations of other cryptographic libraries.

5. Measurement error, loop overhead, and other computation time contributions not directly attributable to the mathematical operations of modular exponentiation are negligible.[2]

6. Modular multiplication computation times are approximately normally distributed. Kocher illustrates that this is the case for his targeted implementation (RSAREF toolkit running on 120MHz Pentium computer, MSDOS operating system) [Koc96].

7. Modular multiplication computations times are independently distributed. In fact, it has been shown that there is some correlation between multiplication times in a straightforward square and multiply implementation, but the approximations used for this attack are still useful in practice [Roh06].

B.4 The Attack

The actual attack is an iterative process whereby Eve attempts to guess the bits k_i of the full w-bit private key in the order they are used by the modular exponentiation algorithm. The basic steps of Kocher's timing attack are:

1. Eve observes Bob's decryption of n cipher-texts.

[2] If these factors are not negligible, it does not render Kocher's attack ineffective, but does increase the *noise*. The result is that the attacker must gather more samples in order to average out the underlying noise.

2. For each decryption operation $j = 1, 2, ..., n$, Eve records the total computation time, T_j. The total computation time for each individual observation can be decomposed as:

$$\mathbf{T}_j = \sum_{i=0}^{w-1} t_{i,j} + e_j = \begin{pmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{pmatrix} = \begin{pmatrix} t_{0,1} + t_{1,1} + \cdots + t_{w-1,1} + e_1 \\ t_{0,2} + t_{1,2} + \cdots + t_{w-1,2} + e_2 \\ \vdots \\ t_{0,n} + t_{1,n} + \cdots + t_{w-1,n} + e_n \end{pmatrix} \tag{B.2}$$

where $t_{i,j}$ is the time attributable to pass i through the square and multiply loop for observation j and e represents the remaining timing and error components including loop overhead and measurement error. Each $t_{i,j}$ depends on the value of the key bit k_i and the value of the cipher-text d_j. Initially, all $t_{i,j}$ are unknown.

3. Beginning with the LSB (k_0), Eve iteratively considers each bit k_i of the key. For subsequent iterations, it is assumed that all previous key bits ($k_{i-1}..k_0$) are known. Eve makes two hypotheses about k_i, namely

$$H_0 : k_i = 0, \quad H_1 : k_i = 1$$

Given her knowledge of the implementation, Eve simulates the partial computation for the first i key bits and determines the time $\sum_{q=0}^{i-1} t_{q,j}$ attributable to that portion of the modular exponentiation for each observation. The remaining time, which is attributable to the unknown key bits for which Eve cannot yet simulate the computation is:

$$\mathbf{T}_j^{H0|H1} = e + \sum_{p=0}^{w-1} t_{p,j} - \sum_{q=0}^{i-1} t_{q,j}. \tag{B.3}$$

The result of this step is two new lists of adjusted timings $\mathbf{T}_j^{H0}$ and $\mathbf{T}_j^{H1}$ for all observations ($j = 1, 2, ..., n$).

4. Eve then calculates the variance for $\mathbf{T}_j^{H0}$ and $\mathbf{T}_j^{H1}$ from the previous step. Whichever hypothesis ($H_0 : k_i = 0$ or $H_1 : k_i = 1$) resulted in the largest reduction in the variance over all adjusted computation times is chosen as the next key bit.

5. Eve repeats steps 3-4 for each bit of the key until the entire key is known.

Under the assumptions of independence and normal distribution of multiplication times, Kocher derived the probability that a wrong hypothesis will result in a greater reduction in variance of the adjusted computation times. When this does occur, the variance calculated in subsequent steps will actually begin to grow, indicating that an incorrect key bit was selected and that some backtracking is required to correct the error [Koc96,Roh06]. Kocher refers to this as an *error-detection property* of the timing attack.

In the general case, the probability at each step of selecting the correct hypothesis for the current key bit (given all previous key bits were chosen correctly) is given:

$$\begin{aligned} &= P\left[K_i = 0 | \left(Var\left(T_j^{\mathrm{H0}}\right) < Var\left(T_j^{\mathrm{H1}}\right)\right)\right] \\ &= P\left[K_i = 1 | \left(Var\left(T_j^{\mathrm{H1}}\right) < Var\left(T_j^{\mathrm{H0}}\right)\right)\right] \\ &= \Phi\left(\sqrt{\frac{n(i-c)}{2\,(w-i)}}\right) \end{aligned}$$

where c is the index of the first incorrectly guessed key bit.

Thus, Eve can make trades between the number of observations and the post-processing time (including backtracking for wrong guesses) to optimize the efficiency of her attack.

Appendix C. Kocher's SPA Attack on DES

Kocher's original SPA attack illustrates a typical application of SSCA. The principle of the analysis is that the power consumption of a micro-controller looks different for different operations. For instance, the side channel leakage may look significantly different when a conditional branch is taken compared to when the branch is not taken. Figure C.1 shows the current drawn by a micro-controller-based DES implementation for two sets of operations over a period of seven clock cycles [KJJ99]. Close examination of the traces reveals that they have very similar (nearly identical) current profiles until they reach clock cycles 6-7. Clock cycles 6-7 correspond to the micro-controller's execution of a conditional branch instruction. In one of the traces the branch is taken, and in the other trace the branch is not taken.

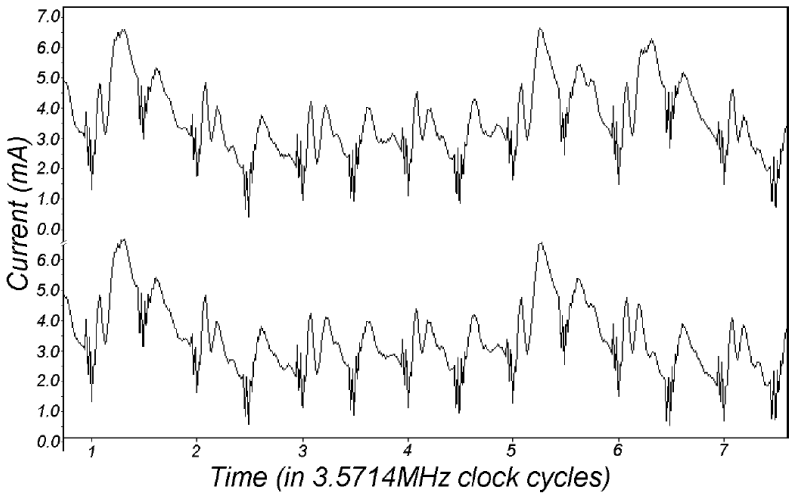

Figure C.1 Variations in Current Drawn Due to Different Conditional Branch Decisions (Adapted from [KJJ99])

In this example, the device being analyzed is a micro-controller running DES code, and the differences in current drawn by the device occur at times when the micro-controller is making a conditional branch decisions based on a portion (one bit in this case) of the cryptographic key. By applying publicly available knowledge of the DES algorithm and domain knowledge of how the algorithm would be imple-

mented in practice, it is possible to deduce the value of the cryptographic key from these differences.

In Kocher's example, the target is a (software, microprocessor-based) smart-card implementation of the Data Encryption Standard (DES).

Listing C.1 Assembler Code for DES Conditional Branch [**?**]

```
;=================================
M_SHIFT_C       MACRO
;=================================

        CLR     C
        MOV     A, PB_4_C
        RLC     A
        MOV     PB_4_C, A
        MOV     A, PB_3_C
        RLC     A
        MOV     PB_3_C, A
        MOV     A, PB_2_C
        RLC     A
        MOV     PB_2_C, A
        MOV     A, PB_1_C
        RLC     A
        MOV     PB_1_C, A
        JC      M_SHIFTC1
        CLR     PB_4_C.4
        JMP     M_SHIFTC2
M_SHIFTC1:
        SETB    PB_4_C.4, #1
M_SHIFTC2:
```

ENDM

The attack begins by analyzing the DES algorithm to identify some intermediate value that, if known, would reveal a portion of the key. For DES, examining the publicly available specification reveals that the generation of the sub-key used in each round of ciphering requires the rotation of two 28-bit key registers. Applying domain knowledge of how a rotation is likely to be implemented for a micro-controller-based implementation, it is expected that most implementations will optimize the 28-bit rotation operation and only copy the most-significant bit (MSB) to the least-significant bit (LSB) if it is a `'1'`. Otherwise, the code will probably perform a less costly shift operation—which would automatically fill the LSB with a `'0'` value. Thus, it is anticipated that straightforward implementations of DES will make a conditional branch decision based on the MSB of the key register before each rotation.

For the micro-controller-based smart-card device targeted, the power trace data looks significantly different for operations where a conditional branch is taken vs. operations where it is not taken (Figure C.1). Therefore, it is possible to tell whether the MSB of the key register is a `'1'` or a `'0'` by examining the power trace for the moment when the conditional decision is made.

Furthermore, according to the Federal Information Processing Standard 46-3 [Nat99], a total of 28 left rotations of each 28-bit key register are used in the DES algorithm. Therefore, every bit in the 56-bit DES key is subject to the above analysis over the course of a full 16 round ciphering process, permitting extraction of the full secret key by analyzing whether or not the jumps were taken.

A key point of SPA is that an attacker requires substantial knowledge of the algorithm being implemented in order to effectively use the technique. Since the detailed specifications of many encryption algorithms are publicly available, and in most cases the manufacturers of encryption devices advertise the type of encryption being used, this information is frequently easy to obtain. Kocher's second type

of attack, DPA, and improvements made later are not as dependent on detailed knowledge of the implementation.

The series of operations for both traces include an instruction that rotates the contents of a register left by one bit. If the most significant bit in the register (before the rotation) is a `'1'` then a special carry register is set to `'1'`. If the MSB is a `'0'` then the carry register is set to `'0'`.

A branch decision is then made based on the value of the carry register. If the carry register is set, the branch is taken. Likewise, if the carry register is not set, the branch is not taken. The two traces depict the current drawn by the device for each case (branch taken vs. branch not taken). Close examination of the traces reveals that both have almost identical current profiles until they reach clock cycles 6-7, which corresponds to the location of the conditional branch operation. Thus, by visually examining the power consumption side channel data, it is possible to determine whether or not a conditional branch was taken or not—and in doing so, to infer the value of the manipulated register's MSB.

Appendix D. RSA Message Blinding

To protect the private key k_{pri} during RSA decryption one option (known as *message blinding*) is to choose a random pair of numbers (m_i, m_f) such that

$$m_f{}^{-1} = m_i^{k_{pri}} \mod n \tag{D.1}$$

where n is used to represent the public modulus to prevent confusion with the mask value m. To do this, m_f can initially be chosen as a random number that is relatively prime to n. Then,

$$m_i = (m_f{}^{-1})^{k_{pub}} \mod n. \tag{D.2}$$

The input message (the cipher-text, C) is then *masked* prior to computing the RSA modular exponentiation by first multiplying it by $m_i \mod n$ or

$$C_m = C \times m_i \mod n. \tag{D.3}$$

Decryption is performed as normal to recover the plain-text

$$P_m = (C_m)^{k_{pri}} \mod n. \tag{D.4}$$

However, since the decryption was performed on the masked cipher-text, the resulting plain-text is also masked. To recover the original plain-text, the masked plain-text is multiplied by m_f

$$P = P_m \times m_f. \tag{D.5}$$

Appendix E. AES Object Source Code

This appendix contains the source code for a Matlab AES implementation that will pre-compute and preserve all FIPS 197 specified algorithmically specified intermediate results. This source code supports all FIPS 197 approved variants of AES, including all key sizes (128, 192, and 256-bit) and modes (encryption and decryption). The supported parameters are documented in `AESObject.m`.

Listing E.1 AESObject.m

```
% ========================================================================
% ========                      AES Object                        =========
% ========================================================================
%
%   Created: Jun 2010
%       By:  Maj Will Cobb
%   Last Modified: 28 Jul 2011
%       By:  Cobb

%  Implements FIPS 197 Advanced Encryption Standard (AES).  Supports all
%  FIPS 197 specified key sizes and operation modes.
%
%   This class will compute the output and all algorithmicly specified (per
%   FIPS 197 specification) intermediate values of the AES encryption /
%   decryption given a specific input (plain or ciphertext depending on
%   mode) and key.
%
%  Modified:
%
%       19 Jul 10 -- made drastic improvements to speed by
%       going to a lookup-table based approach for AES polynomial
%       multiplication.
%
%       14 Sep 10 -- Updated comments, prettied up hex output display
%
%       28 Jul 11 -- Moved lookup tables to separate .mat file to improve
%       code readability.
%
%       18 Aug 11 -- Added get.HexKeySchedule & get.BinKeySchedule.
%
%   Examples using Test Vectors from FIPS 197 Appendix C:
```

```
%
%           myAES = AESObject;
%           myAES.HexInput = '00112233445566778899aabbccddeeff';
%           myAES.HexKey = '000102030405060708090a0b0c0d0e0f';
%           myAES.Encrypt;
%           myAES.HexOutput
%
%           ans =
%
%               69C4E0D86A7B0430D8CDB78070B4C55A
%
%           myAES.HexInput = myAES.HexOutput;
%           myAES.Decrypt;
%           myAES.HexOutput
%
%           ans =
%
%               00112233445566778899AABBCCDDEEFF
%
%           % Get Key Schedule:
%           myAES.w
%
%           ans =
%
%   [0,1,2,3;4,5,6,7;8,9,10,11;12,13,14,15;214,170,116,253;210,175,114,250;
%    218,166,120,241;214,171,118,254;182,146,207,11;100,61,189,241;
%    190,155,197,0;104,48,179,254;182,255,116,78;210,194,201,191;
%    108,89,12,191;4,105,191,65;71,247,247,188;149,53,62,3;249,108,50,188;
%    253,5,141,253;60,170,163,232;169,159,157,235;80,243,175,87;
%    173,246,34,170;94,57,15,125;247,166,146,150;167,85,61,193;
%    10,163,31,107;20,249,112,26;227,95,226,140;68,10,223,77;78,169,192,38;
%    71,67,135,53;164,28,101,185;224,22,186,244;174,191,122,210;
%    84,153,50,209;240,133,87,104;16,147,237,156;190,44,151,78;%
%    19,17,29,127;227,148,74,23;243,7,167,139;77,43,48,197;]
%
%           % Get hex representation of key schedule (1 round key / row)
%           myAES.HexKeySchedule
%
%             ans =
%
%             000102030405060708090A0B0C0D0E0F
%             D6AA74FDD2AF72FADAA678F1D6AB76FE
%             B692CF0B643DBDF1BE9BC5006830B3FE
```

```
%              B6FF744ED2C2C9BF6C590CBF0469BF41
%              47F7F7BC95353E03F96C32BCFD058DFD
%              3CAAA3E8A99F9DEB50F3AF57ADF622AA
%              5E390F7DF7A69296A7553DC10AA31F6B
%              14F9701AE35FE28C440ADF4D4EA9C026
%              47438735A41C65B9E016BAF4AEBF7AD2
%              549932D1F08557681093ED9CBE2C974E
%              13111D7FE3944A17F307A78B4D2B30C5
%
%           % Get intermediate values:
%           AES_Intermediates = myAES.IV
%
%           ans =
%
%               temp: [70x4 double]
%                S: [41x4x4 double]
%
%           % Note that temp is the intermediates calculated in key
%           % schedule generation
%           % S is the 4x4 AES state array at each intermediate point in
%           % the calculation.  The first dimension is the designator
%           % for the intermediate value in order of computation.  Size of
%           % this dimension will determine on AES key size (128,192,256)

classdef AESObject < hgsetget

    properties(Constant)

        cENCRYPT_MODE = 1;
        cDECRYPT_MODE = 0;

        Nb = 4;

        binLookup = de2bi(0:255, 'left-msb');

    end

    properties

        KeyObjectHandle;
        IV = {};

        %  *** Default input / keys are FIPS 197 test vectors ***
```

```
118         HexInput =  {'32' '88' '31' 'e0'; ...
119                      '43' '5a' '31' '37'; ...
120                      'f6' '30' '98' '07'; ...
121                      'a8' '8d' 'a2' '34'};
122
123         HexKey =    {'2b' '7e' '15' '16' '28' 'ae' 'd2' 'a6'        ...
124                      'ab' 'f7' '15' '88' '09' 'cf' '4f' '3c'};
125 %         HexKey = {'8e' '73' 'b0' 'f7' 'da' '0e' '64' '52' ...
126 %                  'c8' '10' 'f3' '2b' '80' '90' '79' 'e5' ...
127 %                  '62' 'f8' 'ea' 'd2' '52' '2c' '6b' '7b'};
128 %         HexKey = {'60' '3d' 'eb' '10' '15' 'ca' '71' 'be'            ...
129 %                   '2b' '73' 'ae' 'f0' '85' '7d' '77' '81'            ...
130 %                   '1f' '35' '2c' '07' '3b' '61' '08' 'd7'            ...
131 %                   '2d' '98' '10' 'a3' '09' '14' 'df' 'f4'};
132
133         BinOutput = logical(zeros(1, 64));
134         DecOutput = uint8(zeros(1, 8));
135
136         % S is current AES State matrix
137         S = zeros( 4, 4);
138
139         % These lookup tables are loaded from a pre-computed .mat file
140         RotWord = [];
141         P = [];
142         ShiftRows = [];
143         InvShiftRows = [];
144         SBox = [];
145         InvSBox = [];
146         RCon = [];
147         PM = [];
148
149     end;
150
151     properties(Dependent = true)
152         Nk; % Valid values are 4, 6, 8 for AES-128, 192, 256
153         Nr; % Valid values are 10, 12, 14 for AES-128, 192, 256
154         HexOutput;
155         IntInput;
156         IntOutput;
157         IntKey;
158         HexKeySchedule;
159         BinKeySchedule;
160         Mode;
```

```
161         w;
162     end;
163 
164     properties(Access = private)
165         priMode = 1;
166         priKeyScheduleValid = 0;
167         priKeySize = 128;
168         priKeySchedule;
169     end;
170 
171 
172     methods
173 
174         function UpdateOutput(a)
175             if (a.Mode == a.cENCRYPT_MODE)
176                 a.Encrypt;
177             elseif (a.Mode == a.cDECRYPT_MODE)
178                 a.Decrypt;
179             else
180                 error('Invalid AES Mode Specified.  Must be 1 (encrypt) or 0 (↩
                        decrypt)');
181             end;
182         end;
183 
184         function Nk = get.Nk(a)
185             Nk = a.priKeySize / 32;
186         end;
187 
188         function Nr = get.Nr(a)
189             Nr = (a.priKeySize / 32) + 6;
190         end;
191 
192         function HexOutput = get.HexOutput(a)
193             HexOutput = reshape((dec2hex(a.S, 2))', 1, []);
194         end
195 
196         function IntKey = get.IntKey(a)
197             IntKey = hex2dec(a.HexKey)';
198         end
199 
200         function IntInput = get.IntInput(a)
201             IntInput = hex2dec(a.HexInput);
202         end
```

```
203
204         function IntOutput = get.IntOutput(a)
205             IntOutput = reshape(a.S, 1, []);
206         end
207
208         function HexKeySchedule = get.HexKeySchedule(a)
209
210             HexKeySchedule = [];
211             for i = 1:a.Nr+1
212                HexKeySchedule = [HexKeySchedule; reshape(dec2hex(reshape(a.w(4*(i↩
                       -1)+1:4*(i-1)+4,:)', 1, []), 2)', 1, 32)];
213             end
214
215         end
216
217         function BinKeySchedule = get.BinKeySchedule(a)
218
219             BinKeySchedule = [];
220             for iRnd = 1:a.Nr+1
221                 rnd_key = reshape(a.w(4*(iRnd-1)+1:4*(iRnd-1)+4,:)', 1, []);
222                 tmp = [];
223                 for iByte = 1:16
224                     if iByte > 1
225                         tmp = [tmp ' '];
226                     end
227                     tmp = [tmp dec2bin(rnd_key(iByte), 8)];
228                 end
229                 BinKeySchedule = [BinKeySchedule; tmp];
230             end
231
232         end
233
234         % AESObject constructor.
235         function a = AESObject(hexinput, hexkey, mode)
236
237             load AESObject_lookup_tables;
238
239             a.InvSBox = InvSBox; clear InvSBox;
240             a.PM = PM; clear PM;
241             a.RCon = RCon; clear RCon;
242             a.SBox = SBox; clear SBox;
243             a.ShiftRows = ShiftRows; clear ShiftRows;
244             a.InvShiftRows = InvShiftRows; clear InvShiftRows;
```

```
245             a.P = P; clear P;
246             a.RotWord = RotWord; clear RotWord;
247
248             if nargin
249                 a.HexInput = hexinput;
250                 a.HexKey = hexkey;
251             end
252
253             if nargin > 2
254                 a.Mode = mode;
255             else
256                 a.Mode = a.cENCRYPT_MODE;   % Default is 'encrypt' mode
257             end
258
259         end;
260
261         function set.HexInput(a, hexinput)
262
263             if iscellstr(hexinput)
264                 if ( length(hexinput) ~= 16 )
265                     error('Invalid hexadecimal input representation.  Must be 16 ↩
                            bytes ');
266                 end;
267                 a.HexInput = hexinput;
268             else
269                 if ( length(hexinput) ~= 32 )
270                     error('Invalid hexadecimal input representation.  Must be 16 ↩
                            bytes ');
271                 end;
272                 for idx = 1:(length(hexinput)/2)
273                     a.HexInput(idx) = cellstr(hexinput(2*idx-1:2*idx));
274                 end
275             end;
276
277         end;
278
279         function set.HexKey(a, hexkey)
280
281             if iscellstr(hexkey)
282                 if isempty(find([16 24 32] == length(hexkey), 1))
283                     error('Invalid hexadecimal key representation.  Must be 16, ↩
                            24, or 32 bytes long.');
284                 end;
```

```
285                 a.HexKey = hexkey;
286             else
287                 if isempty(find([32 48 64] == length(hexkey), 1))
288                     error('Invalid hexadecimal key representation.  Must be 16, ↩
                            24, or 32 bytes long.');
289                 end;
290                 for idx = 1:(length(hexkey)/2)
291                     a.HexKey(idx) = cellstr(hexkey(2*idx-1:2*idx));
292                 end
293             end;
294
295             a.priKeySize = length(a.HexKey) * 8;
296             a.priKeyScheduleValid = 0;
297
298         end;
299
300         function mode = get.Mode(a)
301
302             mode = a.priMode;
303
304         end;
305
306         function set.Mode(a, mode)
307
308             if (a.Mode == a.cENCRYPT_MODE)
309                 a.priMode = mode;
310             elseif (a.Mode == a.cDECRYPT_MODE)
311                 a.priMode = mode;
312             else
313                 error('Invalid AES Mode Specified.  Must be 0 or 1');
314             end;
315
316             a.UpdateOutput;
317
318         end;
319
320         %% Implement the AES Key Expansion algorithm per FIPS 197.
321         %   Verified working for FIPS 197 test keys (all sizes)
322         function keyschedule = get.w(a) %
323
324             if a.priKeyScheduleValid
325
326                 keyschedule = a.priKeySchedule;
```

```
327
328             else
329
330                 keyschedule = zeros(4*(a.Nr+1),4);  % 44, 52, or 60 rows for ↩
                        128/192/256 bit
331                 keyschedule(1:a.Nk,:) = (reshape (a.IntKey, 4, a.Nk))';
332
333                 % Pre-allocate memory to store intermediate values from key ↩
                        scheduling
334                 if (a.priKeySize == 128) || (a.priKeySize == 192)
335                     a.IV.temp = zeros(70,4);
336                 else
337                     a.IV.temp = zeros(79,4);
338                 end
339
340                 i = a.Nk;
341                 hist_idx = 1;
342
343                 while i < ( a.Nb * (a.Nr + 1) )
344                     temp = keyschedule(i,:); % Remember...base 1 not base 0
345                     a.IV.temp(hist_idx, :) = temp;
346                     hist_idx = hist_idx + 1;
347
348                     if (mod(i, a.Nk) == 0)
349                         temp = temp(a.RotWord);
350                         a.IV.temp(hist_idx, :) = temp;
351                         hist_idx = hist_idx + 1;
352
353                         temp = a.SBox(temp + 1);
354                         a.IV.temp(hist_idx, :) = temp;
355                         hist_idx = hist_idx + 1;
356
357                         temp = bitxor( temp, a.RCon((i / a.Nk), : ) );
358                         a.IV.temp(hist_idx, :) = temp;
359                         hist_idx = hist_idx + 1;
360
361                     elseif (a.Nk > 6 && (mod(i, a.Nk) == 4))
362
363                         temp = a.SBox(temp + 1);
364                         a.IV.temp(hist_idx, :) = temp;
365                         hist_idx = hist_idx + 1;
366
367                     end % if
```

```
368
369                     keyschedule(i+1,:) = bitxor(keyschedule((i+1) - a.Nk,:), temp)↩
                            ;
370
371                     i = i + 1;
372
373                 end; % while
374
375                 a.priKeySchedule = keyschedule;
376                 a.priKeyScheduleValid = 1;
377
378             end; % if
379         end
380
381     end;
382 end
```

Listing E.2 Encrypt.m

```
1  function out = Encrypt(a)
2
3      Nr = a.Nr;
4      Nb = a.Nb;
5      w = a.w;
6
7      IV_S = zeros(5+4*(Nr-1),4,4);
8      MCIVs = zeros(Nr-1,4,4,4,2);
9
10     S = reshape(a.IntInput, 4, 4);
11     a.S = S;
12     IV_S(1,:,:) = S;
13
14     S = bitxor(S, w(1:4,:)'); % AddRoundKey
15     IV_S(2,:,:) = S;
16
17     for round = 1:(Nr-1)
18         S = a.SBox(S + 1);   % SubBytes
19         IV_S(3+4*(round-1),:,:) = S;
20         S = S(a.ShiftRows);  % ShiftRows
21         IV_S(4+4*(round-1),:,:) = S;
22         a.S = S;
23         a.MixColumns;   % MixColumns
24 %         MCIVs(round,:,:,:,1) = a.MC_IV_1;
25 %         MCIVs(round,:,:,:,2) = a.MC_IV_2;
```

```
        S = a.S;
        IV_S(5+4*(round-1),:,:) = S;
        S = bitxor(S, w((1+4*round):(4+4*round),:)');  % AddRoundKey
        IV_S(6+4*(round-1),:,:) = S;
    end

    S = a.SBox(S + 1);  % SubBytes
    IV_S(3+4*(Nr-1),:,:) = S;
    S = S(a.ShiftRows); % ShiftRows
    IV_S(4+4*(Nr-1),:,:) = S;
    S = bitxor(S, w(Nb*Nr+1:Nb*Nr+4, :)'); % AddRoundKey
    IV_S(5+4*(Nr-1),:,:) = S;

    a.S = S;
    out = a.HexOutput;
    a.IV.S = IV_S;
    a.IV.MC = MCIVs;

end
```

Listing E.3 Decrypt.m

```
function out = Decrypt(a)

    % Don't recalc these each time...!
    Nr = a.Nr;
    Nb = a.Nb;
    w = a.w;

    IV_S = zeros(5+4*(Nr-1),4,4);

    % Set initial state to input
    S = reshape(a.IntInput, 4, 4);
    IV_S(1,:,:) = S;

    S = bitxor(S, w(Nb*Nr+1:Nb*Nr+4, :)'); % AddRoundKey
    IV_S(2,:,:) = S;


    for round = (Nr - 1):-1:1
        S = S(a.InvShiftRows);                          % InvShiftRows
        IV_S(3+4*(Nr - round - 1),:,:) = S;
        S = a.InvSBox(S + 1);                           % InvSubBytes
        IV_S(4+4*(Nr - round - 1),:,:) = S;
```

```
23          S = bitxor(S, w((1+4*round):(4+4*round),:)');     % AddRoundKey
24          IV_S(5+4*(Nr - round - 1),:,:) = S;
25          a.S = S;
26          a.InvMixColumns;                                  % InvMixColumns
27          S = a.S;
28          IV_S(6+4*(Nr - round - 1),:,:) = S;
29      end
30
31      S = S(a.InvShiftRows);                                % InvShiftRows
32      IV_S(3+4*(Nr-1),:,:) = S;
33      S = a.InvSBox(S + 1);                                 % InvSubBytes
34      IV_S(4+4*(Nr-1),:,:) = S;
35      S = bitxor(S, w(1:4,:)');                             % AddRoundKey
36      IV_S(5+4*(Nr-1),:,:) = S;
37
38      a.S = S;
39      out = a.HexOutput;
40      a.IV.S = IV_S;
41
42  end
```

Listing E.4 MixColumns.m

```
1   %
2
3   function MixColumns(a)
4
5   %  Commented out for efficiency.  This version is SLOW!  Using table lookup
6   %  of polymul instead.
7   %
8   %     function out = polymul(x,y)
9   %
10  %         [~, r] = deconv(conv(x, y), a.P);
11  %         tmp = mod(r, 2);
12  %
13  %         % Binary to decimal conversion...much faster than bi2de!
14  %         out = [128 64 32 16 8 4 2 1] * tmp(end-7:end)';
15  %
16  %     end
17
18      S = a.S;
19
20      for col = 1:4
21
```

```
22          s0 = S(1,col);
23          s1 = S(2,col);
24          s2 = S(3,col);
25          s3 = S(4,col);
26
27          S(1, col) = bitxor(bitxor(a.PM(2+1, s0+1), a.PM(3+1, s1+1)), bitxor(s2, s3↩
                ));
28          S(2, col) = bitxor(bitxor(s0, a.PM(2+1, s1+1)), bitxor(a.PM(3+1, s2+1), s3↩
                ));
29          S(3, col) = bitxor(bitxor(s0, s1), bitxor(a.PM(2+1, s2+1), a.PM(3+1, s3+1)↩
                ));
30          S(4, col) = bitxor(bitxor(a.PM(3+1, s0+1), s1), bitxor(s2, a.PM(2+1, s3+1)↩
                ));
31
32  %  This version is SLOW!
33  %          a.S(1, col) = bitxor(bitxor(polymul([1 0], s0b), polymul([1 1], s1b)), ↩
        bitxor(s2, s3));
34  %          a.S(2, col) = bitxor(bitxor(s0, polymul([1 0], s1b)), bitxor(polymul([1 ↩
        1], s2b), s3));
35  %          a.S(3, col) = bitxor(bitxor(s0, s1), bitxor(polymul([1 0], s2b), polymul↩
        ([1 1], s3b)));
36  %          a.S(4, col) = bitxor(bitxor(polymul([1 1], s0b), s1), bitxor(s2, polymul↩
        ([1 0], s3b)));
37
38      end
39
40      a.S = S;
41
42  end
```

Listing E.5 InvMixColumns.m

```
1   function InvMixColumns(a)
2
3   % Commented out for efficiency.  This version is SLOW!  Using table lookup
4   % of polymul instead
5   %
6   %     function out = polymul(x,y)
7   %
8   %         gfx = de2bi(x, 'left-msb');
9   %         gfy = de2bi(y, 'left-msb');
10  %
11  %         [~, r] = deconv(conv(gfx, gfy), a.P);
12  %         tmp = mod(r, 2);
```

```
13 %
14 %         out = bi2de(tmp, 'left-msb');
15 %
16 %     end
17
18     S = a.S;
19
20     for col = 1:4
21
22         s0 = S(1,col);
23         s1 = S(2,col);
24         s2 = S(3,col);
25         s3 = S(4,col);
26
27         S(1, col) = bitxor(bitxor(a.PM(14+1,s0+1), a.PM(11+1,s1+1)), ...
28                            bitxor(a.PM(13+1,s2+1), a.PM(9+1,s3+1)));
29         S(2, col) = bitxor(bitxor(a.PM(9+1,s0+1), a.PM(14+1,s1+1)),  ...
30                            bitxor(a.PM(11+1,s2+1), a.PM(13+1,s3+1)));
31         S(3, col) = bitxor(bitxor(a.PM(13+1,s0+1), a.PM(9+1,s1+1)),  ...
32                            bitxor(a.PM(14+1,s2+1), a.PM(11+1,s3+1)));
33         S(4, col) = bitxor(bitxor(a.PM(11+1,s0+1), a.PM(13+1,s1+1)), ...
34                            bitxor(a.PM(9+1,s2+1), a.PM(14+1,s3+1)));
35
36 %  This version is SLOW!
37 %
38 %         a.S(1, col) = bitxor(bitxor(polymul(14,s0), polymul(11,s1)), ...
39 %                              bitxor(polymul(13,s2), polymul(9,s3)));
40 %         a.S(2, col) = bitxor(bitxor(polymul(9,s0), polymul(14,s1)),  ...
41 %                              bitxor(polymul(11,s2), polymul(13,s3)));
42 %         a.S(3, col) = bitxor(bitxor(polymul(13,s0), polymul(9,s1)),  ...
43 %                              bitxor(polymul(14,s2), polymul(11,s3)));
44 %         a.S(4, col) = bitxor(bitxor(polymul(11,s0), polymul(13,s1)), ...
45 %                              bitxor(polymul(9,s2), polymul(14,s3)));
46
47     end
48
49     a.S = S;
50
51 end
```

Appendix F. MemMapTRS Source Code

This appendix contains the source code for to wrap the Riscure Inspector '`.trs`' format with a Matlab memory-mapped interface. This allows the very large data files to be manipulated from disk in a manner similar to native Matlab matrices. This code is used extensively by the RF DNA Fingerprinting and Leakage Mapping procedures. See the Matlab help files for general information on memory-mapping.

Listing F.1 MemMapTRS.m

```
%% Class to access to Riscure's .trs format via Matlab memmapfile
%
%  Syntax:
%
%       memmap = MemMapTRS(TracePath, TRSFileName)
%       new_memmap = MemMapTRS(TracePath, TRSFileName, TraceDataMat, ...
%                              PTCTDataMat, XDelta)
%
%       First two arguments (directory & filename) are mandatory.  Others
%       are only needed if creating a new .trs file from an existing matrix
%       and data.  To create a new .trs, use the second syntax along with
%       an (N_t X N_s) matrix containing the trace data.  N_t is the number
%       of traces (rows); N_s is the number of samples per trace (columns).
%       XDelta is the time per sample or 1/SampleRate, used for scaling
%       X-Axis.
%
%  This class allows direct access to the Riscure '.trs' trace set file
%  format using Matlab's memmapfile capability.  Allows efficient
%  manipulation of -very- large data sets.
%
%  Current weakness is the inability to fully treat it like a matrix.
%  It's not currently possible to index across specific columns and
%  multiple rows of the traceset.
%
%  ! *TODO* !  -- Look into using Riscure's Java API to improve this aspect
%
%  Author:  Maj Will Cobb
%  Created:  28 Feb 2010
%  Last Modified:  28 Jul 2010 - added writeable flag to allow file
%                  31 Aug 2010 - added capability to create a .trs file from
%                                a matrix and associated PT/CT/KY data
```

```
%                  9 Sep 2010  - updated comments to better document
%                                functionality.

classdef MemMapTRS < handle

    properties (Constant)  % Constants used in .trs header structure
        NT = hex2dec('41');  % Number of traces
        NS = hex2dec('42');  % Number of samples per trace
        SC = hex2dec('43');  % Sample coding:  '000ABBBB' where A indicates ↩
            integer(0) or floating point(1); B is length in bytes (must be in ↩
            {1,2,4})
        DS = hex2dec('44');  % Length of cryptographic / supplementary data
        TS = hex2dec('45');  % Title space reserved for each trace
        GT = hex2dec('46');  % Global trace title
        DC = hex2dec('47');  % Description
        XO = hex2dec('48');  % Offset in X-axis for trace representation
        XL = hex2dec('49');  % Label of X-axis
        YL = hex2dec('4A');  % Label of Y-axis
        XS = hex2dec('4B');  % Scale value for X-axis
        YS = hex2dec('4C');  % Scale value for Y-axis
        TO = hex2dec('4D');  % Trace offset for displaying trace numbers
        LS = hex2dec('4E');  % Logarithmic scale
        TB = hex2dec('5F');  % Trace block marker; '5f 00' marks end of header
    end; % properties (Constant)

    properties (Access = public)
        NumTraces = 0;
        NumSamples = 0;
        SampleCoding = 0;
        INTEGER_CODING = 1;     % Default is int8 coding
        FLOAT_CODING = 0;
        SampleLength = 1;       % Default is int8 coding
        DataLength = 0;
        TitleLength = 0;
        GlobalTitle = '';
        Description = '';
        XOffset = 0;
        XLabel = '';
        YLabel = '';
        XDelta = 0;
        YDelta = 0;
        TraceOffset = 0;
        LogScaleFlag = 0;
```

```
73          coding_type = 0;
74          data_format = {};
75
76          memmap;
77
78      end; % public properties
79
80      properties (Access = private)
81          idx = 0;
82      end; % private properties
83
84      methods
85
86          %↩
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%↩
87          %%%   Constructor -- pass in directory (TracePath) and name of
88          %%%   TRS file to be memory mapped
89          %↩
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%↩
90          function m = MemMapTRS(TracePath, TRSName, TraceDataMat, PTCTDataMat, ↩
                XDelta)
91
92              filepath = fullfile(TracePath, TRSName);
93
94              if ~exist(filepath, 'file')
95                  if nargin < 3
96                      error('Specified file:\n\n %s\n\n does not exist!', filepath);
97                  end
98                  m = CreateTRSFile(m, filepath, TraceDataMat, PTCTDataMat, XDelta);
99              end;
100
101             m = ParseHeader(m, filepath);
102             m = MakeFormat(m);
103
104             % Call memmap constructor
105             m.memmap = memmapfile(filepath,     ...
106                'offset', m.idx,                 ...                % This offset ↩
                       is 0 based !!
107                'format', m.data_format,         ...
108                'repeat', m.NumTraces,           ...
109                'Writable', true);
```

```
110
111         end; % MemMapTRS method
112
113         function m = ParseHeader (m, filepath)
114
115             %% Open the file and parse the header
116             try
117                 fid = fopen(filepath, 'r');
118
119                 data_loc_found = 0;
120                 m.idx = 0;
121
122                 while (data_loc_found == 0)
123
124                     tmp = fread(fid, 1, 'int8');
125
126                     if (tmp == m.NT)
127                         trash = fread(fid, 1, 'int8');
128                         m.NumTraces = fread(fid, 1, 'int32', 'l');
129                         m.idx = m.idx + 6;
130                     elseif (tmp == m.NS)
131                         trash = fread(fid, 1, 'int8');
132                         m.NumSamples = fread(fid, 1, 'int32', 'l');
133                         m.idx = m.idx + 6;
134                     elseif (tmp == m.SC)
135                         trash = fread(fid, 1, 'int8');
136                         m.SampleCoding = fread(fid, 1, 'int8');
137                         if (bitget(m.SampleCoding, 5) == 0)
138                             m.INTEGER_CODING = 1;
139                             m.FLOAT_CODING = 0;
140                         else
141                             m.INTEGER_CODING = 0;
142                             m.FLOAT_CODING = 1;
143                         end;
144                         m.SampleLength = bitand(m.SampleCoding, 7);
145 %                         switch bitand(SampleCoding, 7)
146 %                             case 1
147 %                                 SampleLength = 1;
148 %                             case 2
149 %                                 SampleLength = 2;
150 %                             case 4
151 %                                 SampleLength = 4;
152 %                             otherwise
```

```
153 %                                 display('ERROR:  INVALID SAMPLE CODING DETECTED↩
        ');
154 %                                 break;
155 %                             end;
156                             %m.idx = m.idx + 2;
157                             m.idx = m.idx + 3;  % I think 2 was WRONG...?? should be 1↩
                                     byte for code, 1 byte for length, 1 byte for sample ↩
                                    coding
158                         elseif (tmp == m.DS)
159                             trash = fread(fid, 1, 'int8');
160                             m.DataLength = fread(fid, 1, 'int16');
161                             m.idx = m.idx + 4;
162                         elseif (tmp == m.TS)
163                             % IMPORTANT:  THIS WON'T CORRECTLY HANDLE TITLE
164                             % LONGER THAN 127 CHARACTERS (YET)!!!!!
165                             trash = fread(fid, 1, 'int8');
166                             m.TitleLength = fread(fid, 1, 'int8');
167                             m.idx = m.idx + 3;
168                         elseif (tmp == m.GT)
169                             length = fread(fid, 1, 'int8');
170                             m.GlobalTitle = char(fread(fid, length, 'int8'));
171                             m.idx = m.idx + length + 2;
172                         elseif (tmp == m.DC)
173                             length = fread(fid, 1, 'uint8');
174                             m.Description = char(fread(fid, length, 'int8'));
175                             m.idx = m.idx + length + 2;
176                         elseif (tmp == m.XO)
177                             trash = fread(fid, 1, 'int8');
178                             m.XOffset = fread(fid, 1, 'int32');
179                             m.idx = m.idx + 6;
180                         elseif (tmp == m.XL)
181                             length = fread(fid, 1, 'int8');
182                             m.XLabel = char(fread(fid, length, 'int8'));
183                             m.idx = m.idx + length + 2;
184                         elseif (tmp == m.YL)
185                             length = fread(fid, 1, 'int8');
186                             m.YLabel = char(fread(fid, length, 'int8'));
187                             m.idx = m.idx + length + 2;
188                         elseif (tmp == m.XS)
189                             trash = fread(fid, 1, 'int8');
190                             m.XDelta = fread(fid, 1, 'float32', 'l');
191                             m.idx = m.idx + 6;
192                         elseif (tmp == m.YS)
```

```
193                         trash = fread(fid, 1, 'int8');
194                         m.YDelta = fread(fid, 1, 'float32', 'l');
195                         m.idx = m.idx + 6;
196                     elseif (tmp == m.TO)
197                         trash = fread(fid, 1, 'int8');
198                         m.TraceOffset = fread(fid, 1, 'int32');
199                         m.idx = m.idx + 6;
200                     elseif (tmp == m.LS)
201                         trash = fread(fid, 1, 'int8');
202                         m.LogScaleFlag = fread(fid, 1, 'int8');
203                         m.idx = m.idx + 3;
204                     elseif (tmp == m.TB)
205                         data_loc_found = 1;
206                         m.idx = m.idx + 2;  %  *** Not sure why this is 7, but ↩
                                trial & error found this works...
207                     else
208                         m.idx = m.idx + 1;
209                     end;
210 
211                 end;
212 
213             catch SomeException
214 
215                 SomeException % Display any error information
216                 error('Error parsing .trs file header for file:  %s', filepath);
217 
218             end;
219 
220             % Done parsing header...close the file and prepare to map to memory
221             fclose(fid);
222 
223         end; % ParseHeader method
224 
225         function m = CreateTRSFile(m, filepath, trace_data, PTCT_data, XDelta)
226 
227             m.NumTraces = size(trace_data, 1);
228             m.NumSamples = size(trace_data, 2);
229             m.SampleCoding = bin2dec('00010100'); % Floating point (bit 5 = '1'), ↩
                    4 bytes/sample (bits 3..0 = '4')
230             m.DataLength = 48;
231             m.TitleLength = 0;
232             m.GlobalTitle = 'Global Title Goes Here';
233             m.Description = 'Description Goes Here';
```

```
234             m.XOffset = 0;
235             m.XLabel = 'Seconds';
236             m.XDelta = XDelta;
237             m.YLabel = 'Volts';
238             m.YDelta = 1;
239             m.TraceOffset = 0; % What is this?
240             m.LogScaleFlag = 0;
241             m.FLOAT_CODING = 1;
242             m.SampleLength = 4;
243
244             try
245
246                 %% Open the file and write the header
247                 fid = fopen(filepath, 'w');
248
249                 fwrite(fid, m.NT, 'int8');
250                 fwrite(fid, 4, 'int8');
251                 fwrite(fid, m.NumTraces, 'int32', 'l');
252
253                 fwrite(fid, m.NS, 'int8');
254                 fwrite(fid, 4, 'int8');
255                 fwrite(fid, m.NumSamples, 'int32', 'l');
256
257                 fwrite(fid, m.SC, 'int8');
258                 fwrite(fid, 1, 'int8');
259                 fwrite(fid, m.SampleCoding, 'uint8');
260
261                 fwrite(fid, m.DS, 'int8');
262                 fwrite(fid, 2, 'int8');
263                 fwrite(fid, m.DataLength, 'int16');
264
265                 fwrite(fid, m.TS, 'int8');
266                 fwrite(fid, 1, 'int8');
267                 fwrite(fid, m.TitleLength, 'int8');
268
269                 fwrite(fid, m.GT, 'int8');
270                 fwrite(fid, length(m.GlobalTitle), 'int8');
271                 fwrite(fid, uint8(m.GlobalTitle), 'uint8');
272
273                 fwrite(fid, m.DC, 'int8');
274                 fwrite(fid, length(m.Description), 'int8');
275                 fwrite(fid, uint8(m.Description), 'uint8');
276
```

```
277                 fwrite(fid, m.X0, 'int8');
278                 fwrite(fid, 4, 'int8');
279                 fwrite(fid, 0, 'int32', 'l');
280
281                 fwrite(fid, m.XL, 'int8');
282                 fwrite(fid, length(m.XLabel), 'int8');
283                 fwrite(fid, uint8(m.XLabel), 'uint8');
284
285                 fwrite(fid, m.YL, 'int8');
286                 fwrite(fid, length(m.YLabel), 'int8');
287                 fwrite(fid, uint8(m.YLabel), 'uint8');
288
289                 fwrite(fid, m.XS, 'int8');
290                 fwrite(fid, 4, 'int8');
291                 fwrite(fid, m.XDelta, 'float32');
292
293                 fwrite(fid, m.YS, 'int8');
294                 fwrite(fid, 4, 'int8');
295                 fwrite(fid, m.YDelta, 'float32');
296
297                 fwrite(fid, m.T0, 'int8');
298                 fwrite(fid, 4, 'int8');
299                 fwrite(fid, m.TraceOffset, 'int32', 'l');
300
301                 fwrite(fid, m.LS, 'int8');
302                 fwrite(fid, 1, 'int8');
303                 fwrite(fid, m.LogScaleFlag, 'uint8');
304
305                 fwrite(fid, m.TB, 'int8');
306                 fwrite(fid, 0, 'int8');
307
308                 % Write each row with PTCTData....
309                 for trace_idx = 1:size(trace_data, 1);
310                     fwrite(fid, PTCT_data(trace_idx,:), 'uint8');
311                     fwrite(fid, trace_data(trace_idx,:), 'single');
312                 end
313
314                 % Done writing file...close
315                 fclose(fid);
316
317             catch SomeException
318
319                 SomeException % Display any error information
```

```
320                 error('Error creating file header for specified file:  %s', ↩
                        filepath);
321
322             end;
323
324         end; % CreateHeader method
325
326
327         % After reading the appropriate parameters from the file header,
328         % construct the format that will be used to map the file to memory
329         function m = MakeFormat (m)
330
331             if (m.FLOAT_CODING == 1)
332                 m.coding_type = 'single';
333             elseif m.SampleLength == 1
334                 m.coding_type = 'int8';
335             elseif m.SampleLength == 2
336                 m.coding_type = 'int16';
337             elseif m.SampleLength == 4
338                 m.coding_type = 'int32';
339             else
340                 error('ERROR:  INVALID CODING TYPE DETECTED');
341             end;
342
343             % Four possible combinations for actual trace data block...must have ↩
                    actual
344             % tracedata, but Title and Data are optional.
345             if ((m.DataLength > 0) && (m.TitleLength > 0))
346
347                 m.data_format = { 'uint8'                                       ...
348                                           [1 m.TitleLength]   'Title';      ...
349                                   'uint8'                                       ...
350                                           [1 m.DataLength]    'PTCTData';   ...
351                                    sprintf(m.coding_type)                       ...
352                                           [1 m.NumSamples]    'tracedata'};
353
354             elseif (m.DataLength > 0)
355
356                 m.data_format = { 'uint8'                                       ...
357                                           [1 m.DataLength]    'PTCTData';   ...
358                                   sprintf(m.coding_type)                       ...
359                                           [1 m.NumSamples]    'tracedata'};
360
```

```
361             elseif (m.TitleLength > 0)
362
363                 m.data_format = { 'uint8'                                    ...
364                                         [1 m.TitleLength]  'Title';      ...
365                                   sprintf(m.coding_type)                 ...
366                                         [1 m.NumSamples]   'tracedata'};
367
368             else  % No title or data areas allocated...
369
370                 m.data_format = { sprintf(m.coding_type)                 ...
371                                         [1 m.NumSamples]   'tracedata'};
372
373             end;
374
375         end; % MakeFormat method
376
377     end; % Methods
378 end
```

Bibliography

AA04. Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy*, pages 3–11. IEEE Computer Society, 2004.

AARR02. Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002. Available from: `http://dx.doi.org/10.1007/3-540-36400-5`.

AARR07. Dakshi Agrawal, Bruce Archambeault, Josyula Rao, and Pankaj Rohatgi. The EM side-channel(s): Attacks and assessment methodologies (revised paper). Online, 2007. Retrieved 9 August, 2011 from `http://www.research.ibm.com/intsec/emf-paper.ps`.

ABDM00. Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. Power analysis, what is now possible... In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502. Springer, 2000. Available from: `http://link.springer.de/link/service/series/0558/bibs/1976/19760489.htm`.

ABK+07. Dakshi Agrawal, Selçuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. Trojan detection using IC fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 296–310. IEEE Computer Society, 2007. Available from: `http://dx.doi.org/10.1109/SP.2007.36`.

And01. Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, January 2001.

APSQ06. Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006. Available from: `http://dx.doi.org/10.1007/11894063_1`.

ARR03. Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel attacks. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2003. Available from: `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=2779&spage=2`.

ARRS05. Dakshi Agrawal, Josyula R. Rao, Pankaj Rohatgi, and Kai Schramm. Templates as master keys. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2005. Available from: `http://dx.doi.org/10.1007/11545262_2`.

Bak07. R. Jacob Baker. *CMOS: Circuit design, layout, and simulation*. Wiley-IEEE Press, 2007.

BB05. David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005. Available from: `http://dx.doi.org/10.1016/j.comnet.2005.01.010`.

BCO04. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. Available from: `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3156&spage=16`.

BEA08. Andrew Burnside, Ahmet T. Erdogan, and Tughrul Arslan. The re-emission side channel. In Adrian Stoica, Tughrul Arslan, Daniel Howard, Tetsuya Higuchi, and Ahmed O. El-Rayis, editors, *BLISS*, pages 154–159. IEEE Computer Society, 2008. Available from: `http://doi.ieeecomputersociety.org/10.1109/BLISS.2008.22`.

Ber05. Daniel J. Bernstein. Cache-timing attacks on AES. Online, 2005. Retrieved on August 9, 2011 from `http://http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.

BGLT06. Marco Bucci, Luca Giancane, Raimondo Luzzi, and Alessandro Trifiletti. Three-phase dual-rail pre-charge logic. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 232–241. Springer, 2006. Available from: `http://dx.doi.org/10.1007/11894063_19`.

BGP+11. Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2011.

BILT04. Jean-Claude Bajard, Laurent Imbert, Pierre-Yvan Liardet, and Yannick Teglia. Leak resistant arithmetic. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2004. Available from: `http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3156&spage=62`.

BMV05. Lejla Batina, Nele Mentens, and Ingrid Verbauwhede. Side-channel issues for designing secure hardware implementations. In *IOLTS*, pages 118–121. IEEE Computer Society, 2005.

Boa73. David G. Boak. A history of U.S. communications security: The David G. Boak lectures, July 1973. Retrieved 9 August, 2011 from `http://www.nsa.gov/public_info/_files/cryptologic_histories/history_comsec.pdf`.

BZ08. Karthik Baddam and Mark Zwolinski. Divided backend duplication methodology for balanced dual rail routing. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 396–410. Springer, 2008. Available from: `http://dx.doi.org/10.1007/978-3-540-85053-3_25`.

CBL11. William E. Cobb, Rusty O. Baldwin, and Eric D. Laspe. Leakage mapping: A systematic methodology for asessing the side channel information leakage of cryptographic implementations. *Information and System Security, ACM Transactions on*, 2011. Under review.

CCD00. Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000. Available from: `http://link.springer.de/link/service/series/0558/bibs/1965/19650252.htm`.

CGB+10. William E. Cobb, Eric W. Garcia, Rusty O. Baldwin, Michael A. Temple, and Yong C. Kim. Physical layer identification of embedded devices using RF-DNA fingerprinting. In *Military Communications Conference, 2010 – MILCOM 2010*, pages 2168 –2173, November 2010.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. Available from: `http://dx.doi.org/10.1007/3-540-48405-1_26`.

Cla04. Christophe Clavier. Side channel analysis for reverse engineering (SCARE) — an improved attack against a secret A3/A8 GSM algorithm. Cryptology ePrint Archive, Report 2004/049, 2004. Available from: `http://eprint.iacr.org/2004/049`.

Cla07. Christophe Clavier. An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. In Patrick Drew McDaniel and Shyam K. Gupta, editors, *ICISS*, volume 4812 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2007.

CLB+11. William E. Cobb, Eric D. Laspe, Rusty O. Baldwin, Michael A. Temple, and Yong C. Kim. Intrinsic physical layer authentication of integrated circuits. *Information Forensics and Security, IEEE Transactions on*, PP(99):1, 2011. To appear.

CNPQ03. M. Ciet, M. Neve, E. Peeters, and J.-J. Quisquater. Parallel FPGA implementation of RSA with residue number systems - can side-channel threats be avoided? In *Micro-NanoMechatronics and Human Science, 2003 IEEE International Symposium on*, volume 2, pages 806–810 Vol. 2, Dec. 2003.

CRR02. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. Available from: `http://link.springer.de/link/service/series/0558/bibs/2523/25230013.htm`.

DC09. Boris Danev and Srdjan Capkun. Transient-based identification of wireless sensor nodes. In *IPSN*, pages 25–36. ACM, 2009.

DHBČ09. B. Danev, T.S. Heydt-Benjamin, and S. Čapkun. Physical-layer identification of RFID devices. In *Proceedings of the 18th conference on USENIX security symposium*, pages 199–214. USENIX Association, 2009.

DK07. Gerald DeJean and Darko Kirovski. RF-DNA: Radio-frequency certificates of authenticity. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 346–363. Springer, 2007.

DLMV05. Rémy Daudigny, Hervé Ledig, Frédéric Muller, and Frédéric Valette. SCARE of the DES. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 393–406, 2005. Available from: `http://dx.doi.org/10.1007/11496137_27`.

DR01. Joan Daemen and Vincent Rijment. *The Design of Rijndael*. Springer-Verlag New york, Inc. Secaucus, NJ, USA, 2001.

DS06. Geir Olav Dyrkolbotn and Einar Snekkenes. A wireless covert channel on smart cards (short paper). In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 249–259. Springer, 2006.

Dyr07. Geir Olav Dyrkolbotn. Analysis of the wireless covert channel attack: Carrier frequency selection. Online, 2007. Available from: `http://www.nik.no/2007/17-Dyrkolbotn.pdf`.

FCC09. Federal Communications Commission FCC. Code of federal regulations, Title 47, 2009. Available from: http://wireless.fcc.gov/index.htm?job=rules_and_regulations.

fISR11. Research Center for Information Security RCIS. SASEBO GII. Online, 2011. Retrieved July 15, 2011, from http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-en.html.

FMP03. Pierre-Alain Fouque, Gwenaëlle Martinet, and Guillaume Poupard. Attacking unbalanced rsa-crt using spa. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 254–268. Springer, 2003.

Fro11. Frontier Economics. Estimating the global economic and social impacts of counterfeiting and piracy: A report comissioned by busines action to stop counterfeiting and piracy (BISCAP). Online, 2011. Retrieved August 9, 2011, from http://www.iccwbo.org/uploadedFiles/BASCAP/Pages/Global%20Impacts%20-%20Final.pdf.

GBPV10. Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting higher-order dpa attacks:. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2010.

GBTP08. Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

GHT05. Catherine H. Gebotys, Simon Ho, and C. C. Tiu. EM analysis of Rijndael and ECC on a wireless Java-based pda. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2005.

Gib09. W. Gibbs. How to steal secrets. *Scientific American Magazine*, 300(5):58–63, 2009.

GLRP06. Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006. Available from: http://dx.doi.org/10.1007/11894063_2.

GMO01. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 251–261, London, UK, 2001. Springer-Verlag. Available from: http://dx.doi.org/10.1007/3-540-44709-1_21.

HBK06. Jeyanthi Hall, Michel Barbeau, and Evangelos Kranakis. Detecting rogue devices in bluetooth networks using radio frequency fingerprinting. In João José Neto, editor, *Communications and Computer Networks*, pages 108–113. IASTED/ACTA Press, 2006.

HTM09. Neil Hanley, Michael Tunstall, and William P. Marnane. Unknown plaintext template attacks. In Heung Youl Youm and Moti Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2009. Available from: `http://dx.doi.org/10.1007/978-3-642-10838-9_12`.

Inc10. Microchip Technology Inc. *PIC24F Family Reference Manual*, 2010. Available from: `http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2575`.

Jaf07. Joshua Jaffe. A first-order dpa attack against AES in counter mode with unknown initial counter. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007. Available from: `http://dx.doi.org/10.1007/978-3-540-74735-2_1`.

JRP04. A.K. Jain, A. Ross, and S. Prabhakar. An introduction to biometric recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):4–20, 2004.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. Available from: `http://dx.doi.org/10.1007/3-540-48405-1_25`.

KKMP09. Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking textscKeeLoq in a flash: On extracting keys at lightning speed. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer, 2009. Available from: `http://dx.doi.org/10.1007/978-3-642-02384-2_25`.

Koc96. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. Available from: `http://dx.doi.org/10.1007/3-540-68697-5_9`.

KSWH98. John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *ESORICS*, volume 1485 of *Lecture Notes in Computer Science*, pages 97–

110. Springer, 1998. Available from: http://dx.doi.org/10.1007/BFb0055858.

KSWH00. John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8(2,3):141–158, 2000.

KTC+08. Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. Designing and implementing malicious hardware. In Fabian Monrose, editor, *LEET*. USENIX Association, 2008. Available from: http://www.usenix.org/events/leet08/tech/full_papers/king/king.pdf.

KTM09. Randall W. Klein, Michael A. Temple, and Michael J. Mendenhall. Application of wavelet-based RF fingerprinting to enhance wireless network security. *Journal of Communications and Networks*, 11(6, Sp. Iss. SI):544–555, DEC 2009.

Kuh04. Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *4th Workshop on Privacy Enhancing Technologies, 23–25 May 2004, Toronto, LNCS 3424*, pages 23–25. Springer, 2004.

Lab94. RSA Laboratories. Rsaref: A cryptographic toolkit. [Online], 1994. Available from: ftp://ftp.funet.fi/pub/crypt/cryptography/rpem/ripem/rsaref/doc/rsaref.txt.

Lam73. Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973. Available from: http://doi.acm.org/10.1145/362375.362389.

LKG+09. Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. Trojan side-channels: Lightweight hardware trojans through side-channel engineering. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 382–395. Springer, 2009. Available from: http://dx.doi.org/10.1007/978-3-642-04138-9_27.

LPdH10. Jiqiang Lu, Jing Pan, and Jerry den Hartog. Principles on the security of AES against first and second-order differential power analysis. In Jianying Zhou and Moti Yung, editors, *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 168–185, 2010.

LR09. Kerstin Lemke-Rust. Ecrypt side channel cryptanalysis lounge. Online, 2009. Retrieved on August 6, 2011 from http://www.crypto.rub.de/en_sclounge.html.

LU02. Joe Loughry and David A. Umphress. Information leakage from optical emanations. *ACM Transactions on Information Systems Security*,

5(3):262–289, 2002. Available from: http://doi.acm.org/10.1145/545186.545189.

Lyo04. R. G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall PTR Upper Saddle River, NJ, USA, second edition, 2004.

Mac03. David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

Man03. Stephen Mangard. Attacks on cryptographic ICs based on radiated emissions. In *Proceedings of Austrochip*, pages 13–16, 2003.

MBKP11. Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks extracting keys from Xilinx Virtex-II FPGAs. Cryptology ePrint Archive, Report 2011/390, 2011. Retrieved on 27 July, 2011 from http://eprint.iacr.org/.

MDS99. Thomas Messerges, Ezzy Dabbish, and Robert Sloan. *Power Analysis Attacks of Modular Exponentiation in Smartcards*, page 724. 1999. Available from: 10.1007/3-540-48059-5_14;http://dx.doi.org/10.1007/3-540-48059-5_14.

MGV08. Nele Mentens, Benedikt Gierlichs, and Ingrid Verbauwhede. Power and fault analysis resistance in hardware through dynamic reconfiguration. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 346–362. Springer, 2008. Available from: http://dx.doi.org/10.1007/978-3-540-85053-3_22.

MKP09. Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Techniques for design and implementation of secure reconfigurable PUFs. *ACM Transactions on Reconfigurable Technology and Systems*, 2:5:1–5:33, March 2009. Available from: http://doi.acm.org/10.1145/1502781.1502786, doi:http://doi.acm.org/10.1145/1502781.1502786.

MKP11. Amir Moradi, Markus Kasper, and Christof Paar. On the portability of side-channel attacks an analysis of the Xilinx Virtex 4 and Virtex 5 bitstream encryption mechanism. Cryptology ePrint Archive, Report 2011/391, 2011. Retrieved on 25 July, 2011 from http://eprint.iacr.org/.

MOP07. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

MOS09. Stefan Mangard, Elisabeth Oswald, and Francois-Xavier Standaert. One for all - all for one: Unifying standard DPA attacks. Cryptology ePrint Archive, Report 2009/449, 2009. http://eprint.iacr.org/.

MPG05. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005. Available from: http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=3376{&}spage=351.

Nat99. National Institute of Standards and Technology (NIST). FIPS PUB 46-3: Data encryption standard (DES), 1999. Retrieved 9 August, 2011, from http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

Nat01. National Institute of Standards and Technology (NIST). FIPS PUB 197: Announcing the advanced encryption standard (AES), November 2001. Retrieved on 25 July, 2011, from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Nov03. Roman Novak. Side-channel based reverse engineering of secret algorithms. In *Proceedings of the 12th International Electrotechnical and Computer Science Conference, ERK 2003*, pages 445–448, 2003. Available from: http://www-e6.ijs.si/~novak/papersERK2003.pdf.

Off05. Office of the Under Secretary of Defense for Acquisition, Technology and Logistics. Report of the defense science board task force on high performance microchip supply, 2005. Available from: http://www.acq.osd.mil/dsb/reports/ADA435563.pdf.

OM07. Elisabeth Oswald and Stefan Mangard. Template attacks on masking - resistance is futile. In Masayuki Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2007. Available from: http://dx.doi.org/10.1007/11967668_16.

OMPR05. Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the AES s-box. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005. Available from: http://dx.doi.org/10.1007/11502760_28.

OT11. Elisabeth Oswald and Michael Tunstall. Opensca: A matlab-based open source framework for side-channel attacks, 2011. Retrieved on 1 August, 2011, from http://opensca.sourceforge.net/.

Pag02. D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169, 2002. Available from: http://eprint.iacr.org/2002/169.

Par10. TELECOM Paristech. DPA contest v2. Online, 2010. Retrieved July 15, 2011, from http://www.dpacontest.org/v2/.

PEK+09. Christof Paar, Thomas Eisenbarth, Markus Kasper, Timo Kasper, and Amir Moradi. Keeloq and side-channel analysis-evolution of an attack. In Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, and Jean-Pierre Seifert, editors, *FDTC*, pages 65–69. IEEE Computer Society, 2009. Available from: `http://dx.doi.org/10.1109/FDTC.2009.44`.

PH98. David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware / Software Interface*. Morgan Kaufmann Publishers, Inc., San Francisco, California, second edition, 1998.

PHF09. Thomas Plos, Michael Hutter, and Martin Feldhofer. On comparing side-channel preprocessing techniques for attacking rfid devices. In Heung Youl Youm and Moti Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 163–177. Springer, 2009. Available from: `http://dx.doi.org/10.1007/978-3-642-10838-9_13`.

PM05. Thomas Popp and Stefan Mangard. *Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints*, pages 172–186. 2005. Available from: `10.1007/11545262_13;http://dx.doi.org/10.1007/11545262_13`.

PRB09. E. Prouff, M. Rivain, and R. Bevan. Statistical analysis of second order differential power analysis. *Computers, IEEE Transactions on*, 58(6):799–811, June 2009. `doi:10.1109/TC.2009.15`.

PRTG02. Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical One-Way Functions. *Science*, 297(5589):2026–2030, 2002. Available from: `http://www.sciencemag.org/content/297/5589/2026.abstract`, `arXiv:http://www.sciencemag.org/content/297/5589/2026.full.pdf`, `doi:10.1126/science.1074376`.

PSQ07. Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration*, 40(1):52–60, 2007.

QS00. Jean-Jacques Quisquater and David Samyde. A new tool for non-intrusive analysis of smart-cards based on electro-magnetic emissions, the SEMA and DEMA methods. Rump session of Eurocrypt, 2000.

QS01. Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *E-smart*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001. Available from: `http://link.springer.de/link/service/series/0558/bibs/2140/21400200.htm`.

QS02. Jean-Jacques Quisquater and David Samyde. Automatic code recognition for smart cards using a Kohonen neural network. In *CARDIS'02: Proceedings of the 5th conference on Smart Card Research and Advanced Application Conference*, pages 6–6, Berkeley, CA, USA, 2002. USENIX Association.

RDG+08. Denis Réal, Vivien Dubois, Anne-Marie Guilloux, Frédéric Valette, and M'hamed Drissi. SCARE of an unknown hardware Feistel implementation. In Gilles Grimaud and François-Xavier Standaert, editors, *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 218–227. Springer, 2008.

Ris09. Riscure. Inspector - the side channel test platform. Online, 2009. Retrieved on 25 July, 2011, from `http://www.riscure.com/inspector/product-description.html`.

RN88. J.L. Rodgers and W.A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

RO04a. C. Rechberger and E. Oswald. Stream ciphers and side-channel analysis. In *Proceedings of the ECRYPT Workshop, SASC-The State of the Art of Stream Ciphers*, pages 320–326. Citeseer, 2004. Available from: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.1451&rep=rep1&type=pdf`.

RO04b. Christian Rechberger and Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung, editors, *WISA*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2004. Available from: `http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=3325{&}spage=440`.

Roh06. Pankaj Rohatgi. *Handbook of Information Security*, volume 3, chapter Side-Channel Attacks, pages 241–257. Wiley, 2006.

RPT11. Donald R. Reising, U. K. Prentice, and Michael A. Temple. Real-time RF fingerprinting using field programmable gate arrays. In *Proc. of 2011 IEEE Int'l Conf. on Comm (ICC11)*, pages TBD–TBD, 2011.

RS09. Mathieu Renauld and Franois-Xavier Standaert. Combining algebraic and side-channel cryptanalysis against block ciphers. In *30-th Symposium on Information Theory in the Benelux*, May 2009.

RSVC09. Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009. Available from: `http://dx.doi.org/10.1007/978-3-642-04138-9_8`.

RTM10. Donald R. Reising, Michael A. Temple, and Michael J. Mendenhall. Improved wireless security for GMSK-based devices using RF fingerprinting. *International Journal of Electronic Security and Digital Forensics*, 3(1):41–59, 2010. Available from: `http://dx.doi.org/10.1504/IJESDF.2010.032330`.

RWB04. Girish B. Ratanpal, Ronald D. Williams, and Travis N. Blalock. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(3):179–189, 2004. Available from: `http://doi.ieeecomputersociety.org/10.1109/TDSC.2004.25`.

SA03. Sergei Skorobogatov and Ross Anderson. *Optical Fault Induction Attacks*, pages 31–48. 2003. Available from: `10.1007/3-540-36400-5_2; http://dx.doi.org/10.1007/3-540-36400-5_2`.

SA08. François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008. Available from: `http://dx.doi.org/10.1007/978-3-540-85053-3_26`.

Sch96. Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wileh & Sons, New York, NY, USA, 1996.

SGV08. François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected cmos devices. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2008.

Sha00. Adi Shamir. Protecting smart cards from passive power analysis with detached power supplies. In Çetin Kaya Koç and Christof Paar, editors, *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 71–77. Springer, 2000. Available from: `http://link.springer.de/link/service/series/0558/bibs/1965/19650071.htm`.

SITMM08. W.C. Suski II, M.A. Temple, M.J. Mendenhall, and R.F. Mills. Radio frequency fingerprinting commercial communication devices to enhance electronic security. *International Journal of Electronic Security and Digital Forensics*, 1(3):301–322, 2008.

Sko06. Sergei P. Skorobogatov. Optically enhanced position-locked power analysis. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249

of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2006. Available from: http://dx.doi.org/10.1007/11894063_6.

SLP05. Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005. Available from: http://dx.doi.org/10.1007/11545262_3.

SMPQ06. François-Xavier Standaert, François Macé, Eric Peeters, and Jean-Jacques Quisquater. Updates on the security of FPGAs against power analysis attacks. In Koen Bertels, João M. P. Cardoso, and Stamatis Vassiliadis, editors, *ARC*, volume 3985 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2006.

SMY09. François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

SÖP04. François-Xavier Standaert, Siddika Berna Örs, and Bart Preneel. Power analysis of an FPGA: Implementation of rijndael: Is pipelining a DPA countermeasure? In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2004.

SS06. Daisuke Suzuki and Minoru Saeki. *Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style*, pages 255–269. 2006. Available from: 10.1007/11894063_21;http://dx.doi.org/10.1007/11894063_21.

ST04. A. Shamir and E. Tromer. Acoustic cryptanalysis—on nosy people and noisy machines. Online, 2004. Retrieved on Aug 6, 2011, from http://tau.ac.il/ tromer/acoustic/.

ST07. Patrick Schaumont and Kris Tiri. Masking and dual-rail logic don't add up. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 2007. Available from: http://dx.doi.org/10.1007/978-3-540-74735-2_7.

TAV02. K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, pages 403–406, Sept. 2002.

TJ09. Randy Torrance and Dick James. The state-of-the-art in IC reverse engineering. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 363–381. Springer, 2009. Available from: `http://dx.doi.org/10.1007/978-3-642-04138-9_26`.

TK09. Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, fourth edition, 2009.

TSU04. Ö. Tekbaş, N. Serinken, and O. Üreten. An experimental performance evaluation of a novel radio-transmitter identification system under diverse environmental conditions. *Electrical and Computer Engineering, Canadian Journal of*, 29(3):203 –209, 2004.

TV04. Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004. Available from: `http://csdl.computer.org/comp/proceedings/date/2004/2085/01/208510246abs.htm`.

TV06. K. Tiri and I. Verbauwhede. A digital design flow for secure integrated circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(7):1197–1208, July 2006.

van90. Tom van Vleck. Timing channels. Online, 1990. Retrieved on 9 August, 2011, from `http://www.multicians.org/timing-chn.html`.

Ver10. I. Verbauwhede. *Secure Integrated Circuits and Systems*. Springer, 2010.

vL85. Wim van Eck and Neher Laborato. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers and Security*, 4:269–286, 1985. Available from: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.1695`.

VP09. Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proceedings of the 18th USENIX Security Symposium, August 10-14, 2009, Montreal, Canada*, 2009. Available from: `http://lasecwww.epfl.ch/keyboard/`.

vWWB11. Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2011.

WH05. Neil H.E. Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, third edition, 2005.

WMTM10. McKay D. Williams, S.A. Munns, Michael A. Temple, and Michael J. Mendenhall. RF-DNA fingerprinting for airport WiMax communica-

tions security. In *2010 Fourth International Conference on Network and System Security*, pages 32–39. IEEE, 2010.

Wra91. John C. Wray. An analysis of covert timing channels. *Security and Privacy, IEEE Symposium on*, 0:2, 1991. Available from: http://doi.ieeecomputersociety.org/10.1109/RISP.1991.130767.

WTR10. McKay D. Williams, Michael A. Temple, and Donald R. Reising. Augmenting bit-level network security using physical layer RF-DNA fingerprinting. In *GLOBECOM*, pages 1–6. IEEE, 2010.

WW04. Jason Waddle and David Wagner. Towards efficient second-order power analysis. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004. Available from: http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=3156&spage=1.

Xil11. Xilinx. Virtex 5 family. Online, 2011. Retrieved on July 15, 2011, from http://www.xilinx.com/products/virtex5/.

ZF05. YongBin Zhou and DengGuo Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing, 2005. howpublished: Cryptology ePrint Archive, Report 2005/388. Available from: http://eprint.iacr.org/.

ZZT05. Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 373–382. ACM, 2005. Available from: http://doi.acm.org/10.1145/1102120.1102169.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–12–2011	Doctoral Dissertation	Sep 2008-Dec 2011

4. TITLE AND SUBTITLE

Exploitation of Unintentional
Information Leakage from Integrated Circuits

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Cobb, William E., Major, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765 DSN: 785-3636

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/DEE/ENG/11-06

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Military Intelligence Program (MIP)
ISR Portfolio Management Office (S33P)
Data Scquisitions (S3)
National Security Agency / Central Security Service (NSA/CS)
9800 Savage Rd, Fort Meade, MD 20755
(240) 373-2548

10. SPONSOR/MONITOR'S ACRONYM(S)

NSA/S33P

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Unintentional electromagnetic emissions are used to recognize or verify the identity of a unique integrated circuit (IC) based on fabrication process-induced variations in a manner analogous to biometric human identification. The effectiveness of the technique is demonstrated through an extensive empirical study, with results presented indicating correct device identification success rates of greater than 99.5%, and average verification equal error rates (EERs) of less than 0.05% for 40 near-identical devices. The proposed approach is suitable for security applications involving commodity commercial ICs, with substantial cost and scalability advantages over existing approaches. A systematic *leakage mapping* methodology is also proposed to comprehensively assess the information leakage of arbitrary block cipher implementations, and to quantitatively bound an arbitrary implementation's resistance to the general class of differential side channel analysis techniques. The framework is demonstrated using the well-known Hamming Weight and Hamming Distance leakage models, and approach's effectiveness is demonstrated through the empirical assessment of two typical unprotected implementations of the Advanced Encryption Standard. The assessment results are empirically validated against correlation-based differential power and electromagnetic analysis attacks.

15. SUBJECT TERMS

Side Channel Analysis, DSCA, Differential Power Analysis, DPA, RF Fingerprinting, RF-DNA, Leakage Mapping, Information Leakage, Cryptography, Physical Security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Rusty O. Baldwin (ENG)
U	U	U	UU	220	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4445; email:rusty.baldwin@afit.edu