Theses and Dissertations   Student Graduate Works

9-1-2013

Enhancing Electromagnetic Side-Channel Analysis in an Operational Environment

David P. Montminy

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Other Computer Engineering Commons, and the Other Electrical and Computer Engineering Commons

ENHANCING ELECTROMAGNETIC
SIDE-CHANNEL ANALYSIS
IN AN OPERATIONAL ENVIRONMENT

DISSERTATION

David P. Montminy, Major, USAF

AFIT–ENG–DS–13–S–01

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–ENG–DS–13–S–01

ENHANCING ELECTROMAGNETIC

SIDE-CHANNEL ANALYSIS

IN AN OPERATIONAL ENVIRONMENT

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

David P. Montminy, B.S.E.E., M.S.C.E.
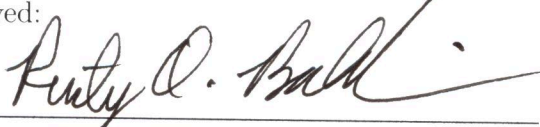
Major, USAF

September 2013

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ENHANCING ELECTROMAGNETIC
SIDE-CHANNEL ANALYSIS
IN AN OPERATIONAL ENVIRONMENT

DISSERTATION

David P. Montminy, B.S.E.E., M.S.C.E.
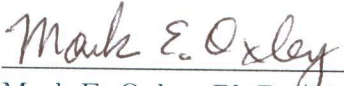
Major, USAF

Approved:

_____   28 Jun 13
Rusty O. Baldwin, Ph.D. (Chairman)   Date

_____   28 June 2013
Mark E. Oxley, Ph.D. (Member)   Date

_____   28 Jun 13
Michael A. Temple, Ph.D. (Member)   Date

Accepted:

_____   31 July 2013
HEIDI R. RIES, Ph.D.   Date
Interim Dean, Graduate School of Engineering
    and Management

To my parents and grandparents,

The values you instilled in me as a child continue to served me well.

To my wife and children,

The love and support you have given me over the last three years has kept me strong and focused. This dissertation would not have been possible without your support.

Acknowledgements

Table of Contents

viii

List of Figures

List of Tables

AFIT–ENG–DS–13–S–01

Abstract

Side-channel analysis has been used to determine the secret key from crypto-graphic devices in a controlled laboratory environment. In many cases, it is assumed that a powerful attacker is able to place a near-field probe within close proximity of a device, modify the device to gain precise timing information, and have access to a training device having side-channel emissions identical to those produced by the target device. Attacks in a laboratory setting utilize expensive digital storage oscilloscopes. To make side-channel attacks more effective in an operational environment, this research identifies ways to 1) reduce the control an attacker must have on a cryptographic device, and 2) reduce the cost of required attack equipment.

A new unknown-plaintext attack is developed to exploit redundancy in the AES key schedule and successfully extract keys from "poor" quality collections. Algebraic cryptanalysis is used to determine the correct key schedule even when maximum likelihood-based template attacks do not identify correct intermediate values by attacking more intermediate values and exploiting the redundancy of the key schedule, the new attack is superior to known plaintext attacks when only a small number of traces for a target device are available. The quality of collected traces is intentionally degraded to show the attack robustness, and a novel thresholding technique is developed to identify possible values for each targeted key schedule byte. Even with poor quality traces, the new attack is successful in 97.5% of trials where a standard template attack that does not employ algebraic cryptanalysis fails 100% of the time.

Profiling attacks assume an adversary has access to a training device identical to the target device being attacked. Although it was previously assumed the side-channel emissions from similar devices were identical, or at least similar, this assumption is challenged here by performing template attacks using traces collected

from 40 16-bit microcontrollers. When the standard template attack methodology fails to produced adequate results, each step is evaluated to identify device-dependent variations. A simple pre-processing technique, i.e., normalizing the trace means and variances from the training and test devices, is evaluated for various test data set sizes. Normalization improves the key-byte extraction success rate from 65.1% to 100% for same part number cross-device template attacks and from 39% to 82.8% for attacks using similar devices for training. Additionally, a procedure is developed to create a single set of templates using training data from multiple devices that can be used to attack all 40 devices at a 99.95% byte extraction success rate.

The new mean and variance normalization technique is also shown to compensate for differences in probe placement, increasing the number of locations at which successful attacks can be performed by 226% on a 32-bit microcontroller. When combined with a new technique that identifies and filters signals in collected traces that are unrelated to the encryption operation, the number of traces required to perform successful attacks is reduced by 85.8% on average. These simple techniques can be performed on the same traces collected for a standard template attack—improving the results through post-collection processing only.

Finally, the use of Software Defined Radios (SDRs) to collect side-channel emissions is introduced and eliminates the need for an attacker to modify the target device. Side-channel emissions are collected passively, and encryption operations are identified in the collected emissions. A correlation-based frequency-dependent leakage mapping technique is introduced to evaluate a 32-bit microprocessor and shows how individual key bytes leak at different frequencies. Key-byte dependent leakage is observed in both SDR collected and triggered oscilloscope-based collections used to validate the SDR methodology; this research is the first to demonstrate effective differential side-channel attacks using SDRs. Successful attacks are demonstrated using two different SDRs, including a commercial $20 USD digital television receiver with modified drivers.

ENHANCING ELECTROMAGNETIC SIDE-CHANNEL ANALYSIS IN AN OPERATIONAL ENVIRONMENT

1. Introduction

Modern cryptographic algorithms provide confidentiality and authenticity but their security relies on computational intractability [132]. The algorithms themselves are public knowledge but secret keys are used to encrypt and decrypt the information. Cryptographic systems based on reusable keys can be broken through a brute force attack, with the amount of time required to do so being an exponential function of key length. Security is achieved by making the amount of work needed to attack the cipher greater than the ability of an adversary to muster [114]. A cryptographic system is considered to be computationally secure if the number of calculations needed to decode the message or determine the key is impossible through practical means.

Modern ciphers are typically implemented on electronic devices that produce both intentional and unintentional emissions. The intentional emissions are the ciphertext resulting from an encryption operation or the plaintext resulting from a decryption operation. Using only the input and intentional emissions, i.e., the plaintext and ciphertext, the key used during the encryption cannot be determined because the computational complexity of the cipher is very high. The unintentional emissions are called side-channels. The side-channels that can be used to extract information from a device depend on the implementation, but may include power consumption [67], acoustic, electromagnetic (EM) [2], optical [116], and photonic [109] emissions, as well as variations in computation time [66].

Side-channel analysis (SCA) effectively bypasses the computational complexity of a cipher by attacking the implementation instead of the cipher itself [67]. Using the side-channel emissions from a device, properties of the intermediate values calculated by the cipher can be determined. When side-channel analysis is used to attack a device, the attack is referred to as a *side-channel attack* and cryptographic devices are a common target. The goal of a side-channel attack against a cryptographic device is to determine the secret key being used for encryption and decryption. If the operations being performed by the device are key dependent it may be possible to determine the key from a single observation of the side-channel. If only the data being processed changes, as is the case with the Advanced Encryption Standard (AES), differential statistics must be used to determine the secret key [67].

1.1 Motivation

The field of SCA has continued to grow since timing and power consumption based attacks were first demonstrated by Kocher et al. in the 1990s [66,67]. Although the effectiveness of these attacks has been demonstrated in laboratory environments, many rely on the assumption that a powerful adversary has complete control over the cryptographic device being attacked [73]. It is frequently assumed the attacker 1) knows the plaintext or ciphertext being processed, 2) can place the EM probe within close proximity of the encryption device, and 3) can modify the cryptographic device to add a trigger; the trigger identifies when the encryption operation is being performed, providing precise timing information for collections made with a digital storage oscilloscope. For attacks based on profiling a similar device, it is assumed that similar devices produce EM emissions identical to the target device [24]. While these assumptions are practical in a academic setting, they may not be rational in operational scenarios where modifying the device is not an option or there is no access to a digital storage oscilloscope.

The objective of this research was to identify ways to reduce the number of assumptions needed for EM-based SCA attacks to make these attacks more practical

in an operational scenario. Ideally, these new techniques would not reduce attack effectiveness. However, if a given technique reduces attack effectiveness but makes the attack possible by a less powerful attacker, it would still be considered useful.

1.2 Research Contributions

First, algebraic cryptanalysis is used to enhance an attack on the key schedule of the AES. By attacking the key schedule, the attack can be performed without knowledge of the plaintexts or ciphertexts associated with each collected side-channel emission [82]. Since the key is fixed and the key schedule is recalculated for each encryption operation on the target device, the side-channel emissions from multiple encryption operations can improve the key extraction rate. Uncertainty in the key extraction phase of the attack is reconciled using a satisfiability solver and an algebraic description of the cipher. A novel technique is developed to identify possible values of portions of the key schedule. The robustness of the attack is demonstrated by intentionally degrading the quality of the results by gradually moving the EM probe away from the encryption device [82].

Template attacks are a form of two-stage profiling attack, with the initial stage obtaining 'a priori' knowledge of the side-channel leakage for a specific device [24]. The profiling stage estimates the multivariate probability densities of the observable side-channels for the targeted intermediate value of the internal calculations performed within a cryptographic device. It is assumed that a powerful attacker would be able to procure a training device identical to the device being attacked. Although template attacks were originally proposed using power consumption data, they were extended to EM emissions [122]. For the attack to be successful, the EM emissions from the training device must be sufficiently similar to the EM emissions from the test device. While previous research assumed implicitly this to be true by collecting training data from the same device being attacked [3, 9, 24, 73, 92], this

assumption is challenged by performing template attacks with 40 PIC and 2 ARM microcontrollers [83, 85].

When attacks performed with different training and test devices show degraded performance, each step of the template attack methodology is analyzed and evaluated. A number of simple, yet powerful, techniques are developed to improve cross-device template attacks [85]. A *cross-device* template attack is defined as a template attack that specifically uses traces from two different physical devices for the training and classification phases. These techniques identify and remove device dependent EM signals and compensate for differences in the distribution of collected EM emissions from different devices. In addition to differences between devices, these techniques effectively compensate for differences in collection parameters including probe type and placement [83]. Additionally, a process for creating a master template to attack any device within a family of devices is developed and shown to be effective.

Finally, the requirement for an attacker to have control of the target device is eliminated by collecting EM emissions using a Software Defined Radio (SDR) [84]. SDRs down-convert EM emissions from a device allowing data collection at reduced sampling rates, allowing side-channel data to be collected in real-time. Although individual traces must be identified in post-processing, the trigger signal is no longer necessary. The use of a SDR also dramatically reduces the cost of performing side-channel analysis [84].

1.3 Organization

This dissertation is organized as follows. Chapter 2 contains background information and a summary of recent publications that pertain to this research. Chapter 3 describes common methodology used in two or more of the focus areas. The next three chapters each contain the unique methodology and results from the four focus areas described above. Chapter 4 develops an algebraic cryptanalysis-based key

schedule redundancy attack. Chapter 5 improves the effectiveness of cross-device template attacks for PIC microcontrollers. In Chapter 6, cross-device attacks are expanded to more complex 32-bit microcontrollers and a new method is developed to identify and remove interfering signals. Chapter 7 introduces the used of SDRs to collect the EM emissions from a microprocessor. Finally, Chapter 8 concludes the dissertation and recommends areas for further study.

2. Background

2.1 Introduction

Modern cryptographic algorithms provide confidentiality and authenticity services based on computational hardness assumptions [132]. Algorithms to secure information are public knowledge but secret keys are used to encrypt and decrypt the information. While cryptographic systems based on reusable keys can be broken through a brute force attack, the amount of time needed is an exponential function of the length of the key. Effective security is achieved by making the amount of work needed to attack the cipher more than the ability of an adversary to muster [114]. A cryptographic system is considered to be computationally secure if the number of calculations needed to decode the message or determine the key is impossible by any practical means.

In his article on the communication theory of secrecy systems, Claude Shannon said, breaking a good cipher should require "as much work as solving a system of simultaneous equations in a large number of unknowns [114]". In theory, a technique known as algebraic cryptanalysis can break ciphers by describing the cipher as a system of polynomial equations and solving this system to obtain the secret key. In practice, the multivariate systems of polynomial equations derived from modern ciphers are large and complex and it is not possible to solve the system in a reasonable amount of time [26]. To break the cipher, the complexity of such systems of equations must be reduced.

The complexity of solving the system can be reduced by finding a weakness in the algorithm, or by determining intermediate values calculated by an implementation of a cipher. A category of relatively low cost, non-intrusive attacks are called side-channel attacks. Side-channel analysis (SCA) can determine the properties of intermediate values calculated by a physical implementation of a cipher by

6

collecting and processing the side-channel emissions from the device performing the cryptographic operation [73].

The target of the side-channel attacks, the Advanced Encryption Standard (AES), and cryptanalysis techniques used against block ciphers are introduced in Section 2.2. The type of side-channels produced by modern electronic devices is discussed in Section 2.3. Next, how SCA is used to identify intermediate values is discussed in Section 2.4, followed by a brief discussion on countermeasures, and collecting and pre-processing of side-channel emissions. Finally, the process and benefit of combining algebraic cryptanalysis with SCA attacks is explored in Section 2.8.

2.2 Cryptography Preliminaries

Information prior to encryption is referred to as *plaintext*. Encrypted information is called *ciphertext*. *Encryption* is the process of converting plaintext to ciphertext. Similarly, *decryption* is the process of converting ciphertext to plaintext. The *cipher* is a pair of algorithms used to encrypt and decrypt information.

In symmetric key cryptography, the two communicating parties share a piece of secret information, the key, and a public encryption system. *Breaking* a cipher consists of "finding a weakness in the cipher that allows the cipher to be exploited with a complexity less than brute-force [110]".

2.2.1 Block Ciphers.

Block ciphers are symmetric key ciphers that operate on groups of bits called blocks. The block cipher is keyed to a family of permutations which operate on n-bits at a time. A permutation is selected from the family using a key; the same key for both encryption and decryption [125]. Two commonly used ciphers are the Data Encryption Standard (DES) and AES.

The rounds of a block cipher are usually based on substitution boxes (S-boxes), bit permutations, arithmetic operations, and exclusive-ORs (XOR). S-boxes are non-linear substitution tables that map input bits to output bits. They are typically the

7

only part of a block cipher that is non-linear [16]. An iterated block cipher applies the round functions sequentially, taking the result of one round as the input to the next round.

Modern block ciphers can be traced back to Claude Shannon [114]. Shannon discussed the block ciphers based on the concepts of diffusion and confusion [28]. *Diffusion* spreads the influence of all parts of the block cipher inputs to all parts of the output, the ciphertext. For a block cipher, the inputs include the plaintext and the key. *Confusion* attempts to make the relationship between the ciphertext, the plaintext and the key complicated. In modern ciphers diffusion is typically achieved using permutations or linear transformations. Simple operations are repeated multiple times to achieve the desired level of security. Encryption/decryption operations are key dependent because key material is introduced in each round.

2.2.2 Advanced Encryption Standard. The Advanced Encryption Standard (AES) was developed to replace the Data Encryption Standard (DES) and triple-DES. The U.S. National Institute of Standards and Technology (NIST) conducted an open competition to develop AES which was to be as secure as triple-DES but much more efficient [39]. A block cipher called Rijndael was selected as the AES. Since Rijndael was announced as the AES in October 2000, AES has been used throughout the U.S. government and been adopted by banks, industry and governments around the world.

AES is a symmetric block cipher that processes blocks of 128 bits using cipher keys with lengths 128, 192 and 256 bits [88]. The basic processing unit for AES is a byte. The AES algorithm operations are performed on a two-dimensional array of bytes called the *State*. The State is a four row by four column matrix, with a byte in each cell. One round of AES is composed of the following four different byte-oriented transformations. [88]:

1. AddRoundKey: The state matrix is XOR-ed with the round key.

8

2. SubBytes: Each byte of the state matrix is substituted for another byte value based on a one-to-one non-linear invertible mapped called S-box,

3. ShiftRows: The last three rows of the state matrix are cyclically shifted column-wise using different offsets,

4. MixColumns: The state matrix is mixed column by column using a linear operation.

One round key, based on the original cipher key, is produced by the key expansion routine for each round. The final round does not include the MixColumns operation and another AddRoundKey operation is performed to produce the ciphertext.

SubBytes is the only non-linear step in each round. To resist linear and differential cryptanalysis (Ref. Section 2.2.3.1), the S-box was specifically chosen to be non-linear and have a high algebraic complexity. Confusion is achieved using carefully chosen S-boxes in AES. The S-box is based on the inversion over a field of order 2^8 [39]. The S-box is defined in [88]. Since inversion and matrix multiplication are computationally expensive, the S-box is frequently precomputed and stored in a table.

A key expansion routine generates the key schedule containing each of the round keys. The number of rounds is determined by the key size. AES performs 10, 12, and 14 rounds for key sizes of 128, 192, and 256 bits of key length respectively. Eleven round keys are produced by the key expansion routine for AES-128. The first round key is simply the cipher key and the subsequent round keys are calculated using the following transformations.

1. SubWord: The SubBytes transformation is performed on each byte of the four-byte input word to produce a four byte output word,

2. RotWord: Performs a cyclic permutation on the four-byte input word, and

3. AddRcon: Bit-wise XOR with the round constant.
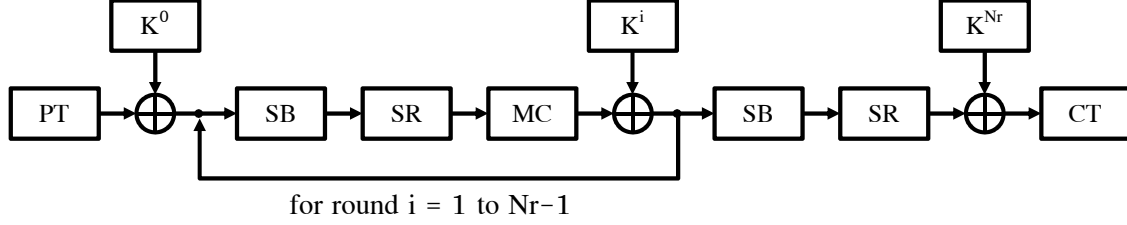
Figure 2.1   AES Cipher Structure [88].

The round constant is designed to eliminate symmetries [39]. For a given implementation, the round keys are either calculated "on the fly", or pre-calculated and stored in memory. On devices with a low amount of memory the round keys are generated as needed, writing over the previous round key in the process. More detailed information on the structure of the key schedule for AES-128 is presented in Chapter 4. The $N_r = 10$ rounds of AES-128 is shown graphically in Figure 2.1. For conciseness the round transformations are abbreviated AddRoundKey (ARK), SubBytes (SB), ShiftRows (SR) and MixColumns (MC) in Figure 2.1.

In a 32-bit microprocessor, the operations performed in the round transformation can be combined into a single look-up-table, called a T-Box to create a faster implementation [38]. The 8×32-bit tables defined

$$
T_0[a] = \begin{bmatrix} SB[a] \bullet 02 \\ SB[a] \\ SB[a] \\ SB[a] \bullet 03 \end{bmatrix} \quad T_1[a] = \begin{bmatrix} SB[a] \bullet 03 \\ SB[a] \bullet 02 \\ SB[a] \\ SB[a] \end{bmatrix}
$$

$$
\tag{2.1}
$$

$$
T_2[a] = \begin{bmatrix} SB[a] \\ SB[a] \bullet 03 \\ SB[a] \bullet 02 \\ SB[a] \end{bmatrix} \quad T_3[a] = \begin{bmatrix} SB[a] \\ SB[a] \\ SB[a] \bullet 03 \\ SB[a] \bullet 02 \end{bmatrix}
$$

combine the SubBytes, ShiftRows and MixColumns operations.

10

The complete round transformation for a 32-bit block is calculated

$$e_c = T_0\left[a_{0,c}\right] \oplus T_1\left[a_{1,c-1}\right] \oplus T_2\left[a_{2,c-2}\right] \oplus T_3\left[a_{3,c-3}\right] \oplus k_j, \qquad (2.2)$$

where c denotes the column of the output e, $a_{r,c}$ denotes the row r and column c of one byte of the input state a, and k_j is corresponding 32-bit portion of the round key for round j. Column indices are taken modulo $N_b = 4$ for AES-128. For AES-128, row $r = n_b \mod 4$ for byte number $n_b = 1, ..., 16$.

Hence, each of the four 32-bit portions of the round output are implemented with four table lookups and four XORs. After performing the initial AddRoundKey, the T-box implementation is used to calculate the first 9 rounds of AES-128. In the 10th round, since the MixColumns operation is not performed, the SubBytes and ShiftRows operations are performed separately.

2.2.2.1 Modes of Operation.   Since AES only encrypts data one block at a time, but the amount of data that must be encrypted is typically greater than one block, modes of operation have been developed. A *mode of operation* is a scheme that allows a block cipher to perform encryption and decryption on groups of plaintexts. NIST special publication 800-38A lists the modes of operations recommended for use with AES: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and (CTR) [87].

In ECB mode the message is divided into blocks and each block is encrypted separately. The drawback of this approach is blocks with identical plaintexts are encrypted into identical ciphertexts. Patterns in the ciphertext may reveal information about the message being sent. However, since each block is encrypted separately, encryption of multiple blocks can be performed in parallel.

CBC, CFB, and OFB incorporate the output of previous encryption operations and utilize initialization vectors. In CTR mode the input blocks, called counters,

are encrypted using AES and the output of the encryption operation is XORed with the plaintext to produce the ciphertext. Since other modes incorporate additional information and restrict the order in which traces can be processed for side-channel analysis, only ECB mode is considered in this dissertation. For side-channel analysis using ECB mode allows for each trace to be analyzed independently.

2.2.3 Cryptanalysis of Block Ciphers. The goal of cryptanalysis is to break ciphers. The goal of most attacks is to recover the encryption or decryption key. Attacker's capabilities may vary. In this regard, the amount of information an adversary has access to changes the types of attacks that are possible. However it is assumed the attacker has full knowledge of the encryption algorithm and the key is always secret. Below is a taxonomy of cryptographic attacks adapted from [28] listed from most practical to most hypothetical.

1. *Ciphertext-only:* The adversary only has access to encrypted messages and some information about the distribution of the plaintext messages. Most modern ciphers are not susceptible to this type of attack.

2. *Known plaintext:* In addition to the ciphertext, the attacker has full or partial knowledge of corresponding plaintext messages. Since messages contain common words or patterns, such as headers, this type of attack is realistic.

3. *Chosen plaintext or ciphertext:* In a chosen plaintext attack the adversary has the ability to choose the plaintext messages to be encrypted. In a chosen ciphertext attack, the adversary can choose the ciphertext to be decrypted and has access to the corresponding plaintext. Although less common, this scenario is still realistic.

4. *Adaptive chosen plain text or ciphertext:* The adversary adapts his choices of the text to be encrypted and decrypted based on information learned during the attack.

5. *Related Key:* The adversary exploits a known relationship between keys (e.g., they only change by a certain number of bits). This attack is conducted in conjunction with one or more of the scenarios above.

This attack model applies to both algebraic cryptography attacks and side-channel analysis. Proposed attacks cover the entire range of attack types, but known plaintext attacks are most common [28].

2.2.3.1   Linear and Differential Cryptanalysis.   Linear cryptanalysis and differential cryptanalysis are the most established methods of attacking block ciphers. Statistical in nature, the attacker constructs probabilistic patterns through as many rounds of the cipher as possible. The goal is to distinguish the cipher from a random permutation and recover the key.

Linear cryptanalysis looks for the *effective* linear expression for a cipher [76]. A *linear approximate* is constructed by building a statistical linear path between input and output bits of each S-box. The linear approximate is the probability that S-box inputs coincide with an S-box output bit. Since this method is based on statistics developed for a specific key, it requires a large number of known plain-texts. Once each S-box is described as a linear approximate the entire algorithm is represented without any intermediate values.

Differential cryptanalysis analyzes the effect of particular differences in plain-text pairs on the differences in corresponding ciphertexts. Using these differences, probabilities can be assigned to possible keys to identify the most probable key [16]. Typically this type of attack is done with chosen plaintext, but can be done with known plaintext if a sufficient number are available [76].

AES was developed to be resistant to both linear and differential cryptanalysis. The wide tail strategy employed in AES maximizes the level of mixing within each round to provide fast diffusion ensuring security against differential and linear cryptanalysis [39].

Due to their statistical basis these attacks require enormous amounts of known or chosen plaintext/ciphertext and are therefore impractical for complex ciphers [5]. Conversely, a relatively new form of attack, algebraic cryptanalysis, requires very few, if any, known plaintexts.

2.2.3.2 Algebraic Cryptanalysis. Algebraic cryptanalysis breaks ciphers by solving polynomial systems of equations and exploiting the intrinsic algebraic structure of the cipher. Typically an attacker converts the encryption transformation into a large system of low degree multivariate polynomial equations and solves the system to reveal information about the key. A number of methods have been proposed for solving these systems of equations. If a cipher is well constructed, however, the system of equations will not be directly solvable. Since the systems are typically very sparse, over-defined, and structured, it is believed they can be solved faster than generic non-linear equation systems [5]. The used of algebraic cryptanalysis against AES is discussed in Section 2.8.

2.3 Side-Channel Leakage

A *side-channel* is an *unintended* observable phenomenon that is correlated with the internal state, operations or data being processed within a device. These correlations can be exploited to recover the leaked information. Electronic devices can leak information via a number of side-channels including power consumption [67], acoustic, electromagnetic (EM) [2], and optical [116] emissions, as well as variations in computation time [66]. Although initial research focused on timing, power consumption and EM emissions, optical analysis is becoming more practical [109].

2.3.1 Power Consumption. Digital circuits consume power during operation. The received energy is dissipated as heat and EM emissions. Many modern integrated circuits are based on Complementary Metal Oxide Semiconductors (CMOS) transistor technology that are the basis of various types of devices includ-
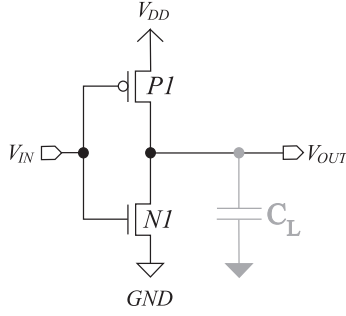
Figure 2.2   Lumped capacitor model of a CMOS inverter [73].

ing general purpose microprocessors and Field Programmable Gate Arrays (FPGAs). The power consumption of CMOS devices consists of static power consumption and dynamic power consumption. CMOS cells are based on complementary pull-up and pull-down networks. For constant input signals, the pull-up and pull-down networks never conduct at the same time. For the inverter shown in Figure 2.2, $P1$ is conducting and $N1$ is insulating when the input is set to GND. When the input is set to V_{DD}, $P1$ is insulating and $N1$ is conducting. For constant input signals there is only a small leakage current which contributes to the static power consumption, $P_{stat} = I_{leak} \cdot V_{DD}$ [73].

Dynamic power consumption occurs when internal transistors change state. However, the power consumed by internal state changes is much lower than the power consumed by changing the CMOS cell output signal, therefore it can be ignored. When the value of the cell output does not change, only static power is consumed, but transitioning from $0 \to 1$ or $1 \to 0$ requires both static and dynamic power. One component of dynamic power consumption is due to the CMOS cell drawing a charging current from the power supply to change the output capacitance C_L during a transition. C_L is the intrinsic capacitance of the CMOS cell and the extrinsic capacitance of wires connected to subsequent CMOS cells. The second component of dynamic power consumption is due to the temporary short circuit that occurs when a CMOS cell switches and both pMOS and nMOS transistors conduct simultaneously. Dynamic power consumption is much higher than static power

15

consumption, in fact it is the primary source of power consumption. Furthermore, dynamic power consumption is always data dependent [73].

There are many factors that affect the power consumption of a microprocessor, including the instruction being executed and the memory address the instruction was retrieved from. Additionally, the data memory address, and the contents of the data being manipulated, and location of the data registers being accessed affect power consumption [95].

2.3.2 Electromagnetic Emissions. Electromagnetic (EM) emissions are caused by three types of coupling: conductive, inductive and radiative. This coupling is caused by time-varying current flows due to transistors turning on and off. Conductive coupling occurs when there is a physical conductive path between a source and a receptor allowing the signal to be transmitted through the system. Conductive emissions can be observed in the power supply, ground line, and cables attached to the device [2, 20].

An EM field is created when current flows through a wire. When two conductors are separated by less than a wavelength, mutual-inductive coupling or magnetic coupling can occur. Through EM induction, current flowing in one wire can induce a voltage across the ends of another wire. Low frequency signals are typically transmitted by inductive coupling. High frequency signals are more easily transmitted by capacitive coupling, whereby energy is transferred between to device nodes due to the capacitance between the two nodes. Inductive and capacitive coupling occur when the conductors are typically less than a wavelength apart. Radiative coupling occurs when the source and the receptor are separated by more than a wavelength; part of the source circuit acts as an antenna and transmits undesired EM waves [93].

EM emissions from digital electronics can be either differential-mode or common-mode radiation. Differential-mode radiation is generated by a flow of current around loops formed by conductors in the circuit during the circuit's normal operation.
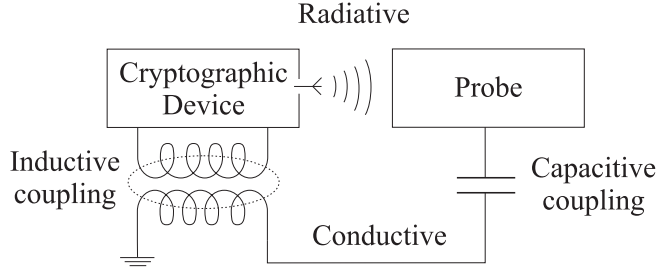
Figure 2.3   EMI coupling modes [40].

The loops act as small antennas that primarily radiate magnetic fields. Differential-mode radiation emission is proportional to the loop area, frequency squared, and the differential-mode current in the loop. Common-mode radiation is caused by parasitics in the circuit and unintentional voltage drops in the conductors. Differential-mode currents flowing through the ground impedance produce a voltage drop in the device ground system, causing some grounded circuits to rise above the real ground potential. Bond wires and pins connected to the affected ground act like antennas radiating components of the common-mode potential as electric fields. Common-mode radiation is proportional to frequency, cable length and the common-mode current in the circuit [93].

2.3.2.1   Direct and Unintentional Emissions.   The EM emissions from a device can be separated into two broad categories: direct and unintentional [2]. *Direct* emissions result from of intentional current flows, which consist of short bursts of current with sharp rising edges. These short bursts result in emissions observable over a wide frequency band. Components of the emissions at higher frequencies may be more useful if less interference or noise is present at higher frequencies. Isolating direct emissions can be very difficult in complex circuits due to interference by other signals [1]. To capture direct emissions with minimal interference, tiny near-field probes should be placed as close as possible to the signal source.

Modern CMOS devices have electronic and EM field coupling between components in close proximity, producing compromising *unintentional* emissions [2]. Mod-

17

ulations of a carrier signal, such as the harmonic rich "square-wave" clock may be produced within the device. As a result, odd harmonics of the clock can be strong carriers of modulated signals. Non-linear coupling between a carrier signal and a data signal can result in an amplitude modulated signal emanating from the device. The coupling between circuits can result in angle (or frequency) modulated signals. The modulated signals can propagate further than the direct emissions enabling attacks from further distances. Once collected, the data signals can be recovered using amplitude and angle demodulation techniques [2].

2.3.2.2 Exploiting Electromagnetic Emissions. Each current carrying component of a device produces EM emissions based on its physical and electrical characteristics as well as the data being processed [2]. An attacker that can analyze emissions and determine how the data being processed corresponds to the emissions would be able to compromise the system. As a result, methods originally developed for power analysis have also been applied to EM Analysis (EMA) [47].

Since power measurements are collected by placing a resistor in series with the power or ground of the cryptographic device physical access to the device is required. EM measurements are less invasive because physical contact with the device is not necessary. Even so, to reduce noise and increase the signal strength EM measurements are typically performed as close as possible to the chip using a near-field probe. Ideally, the probe is placed near the part of the device with the most intense data-dependent signal. This is typically near the CPU, data lines or power supply lines [47]. Although no physical contact to the device is required, it is assumed the attacker has the ability to place the probe in the near-field.

The phenomenon of compromising power and EM emissions has been known and exploited for decades. Declassified TEMPEST documents reveal vulnerabilities of United States cryptographic systems to EM analysis in 1962 and Soviet guidelines for radio frequency interference indicated they recognized the threat before the

United States [20]. Academic research on the vulnerability of cryptographic devices to EM analysis has flourished in the last decade when attacks performed using power analysis were extended to collected EM emissions [47, 94]. Since power and EM attacks are related and many of the same techniques apply, attacks are not grouped by side-channel and are presented in Section 2.4

2.3.3 Other Side-Channels. Although the power and EM side-channels are the most commonly attacked, other side-channels have been used to attack cryptographic systems. By carefully measuring and analyzing the amount of time required to perform a cryptographic operation, a secret key can be determined when the length of the operation depends on the secret key [66]. The optical side-channel has been used to enhance side-channel attacks. By observing photon emissions from switching of transistors, the active area of an integrated circuit can be identified to allow targeted EM and power attacks [116]. Attacks based on acoustic emissions have been demonstrated on desktop computer CPUs [111], keyboards [10, 138] and dot matrix printers [51].

Differential fault injection extends existing side-channel analysis methods by actively injecting faults into a system in the hope the internal state of the system will be revealed. Faults can be induced in a variety of ways including over-clocking, powering at unsupported voltages, or even targeting the device with radiation [17].

2.3.4 Leakage Models. At the transistor level the static and dynamic power consumption can be modeled with approximations that describe the power consumption well. For more complex circuits, power models and simulations can estimate the power consumption, efficiency and security. Highly accurate models require a high level of memory, time to simulate, and intricate knowledge of the device. Analog simulations use transistor netlists and circuit parasitics to calculate power consumption. Precise circuit parasitics will result in a precise simulation. Logic level simulations requiring fewer resources are less accurate, but still require

19

a netlist containing all logic cells in the circuit and the connections between them. More accurate logic level simulations will contain signal delays, rise and fall times and accurate power models for each cell. Analog and logic level models required detailed knowledge that is typically only available to the device designers [73].

Models based on Hamming Weight and Hamming Distance are presented in the next two sections. Although these models are not as accurate, they are useful because they do not required detailed information about the layout and device being used.

2.3.4.1 Hamming Weight Model. A model based on the Hamming Weight (HW) assumes the power consumption is proportional to the number of bits equal to 1 in the processed value and does not require any information about the values processed before or after [73]. Although CMOS power consumption depends on whether a transition occurs and not on the values being processed, HW models can still be useful for some applications. The utility of each model depends on the implementation.

In the best case, the preceding or succeeding values are known, for example, a precharged bus on a microprocessor. If all of the bits of a data bus are set to 0 before the value of interest is placed on the bus, the HW is determined only by the values being placed on the bus [78]. Figure 2.4 shows an example of how the power consumption changes based on the HW of the data being processed.

2.3.4.2 Hamming Distance Model. Hamming Distance (HD) is the number of bit-level transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) that occur during a certain interval. As discussed above, power consumption is primarily caused by the output of logic cells transitioning from one state to another. The HD model is a simplified power model based on a count of the transitions over a period of time. For simplicity it is assumed that the power required for transitions from $0 \rightarrow 1$ and $1 \rightarrow 0$ are equal. The parasitic capacitances of wire and cells and the static power consumption

20
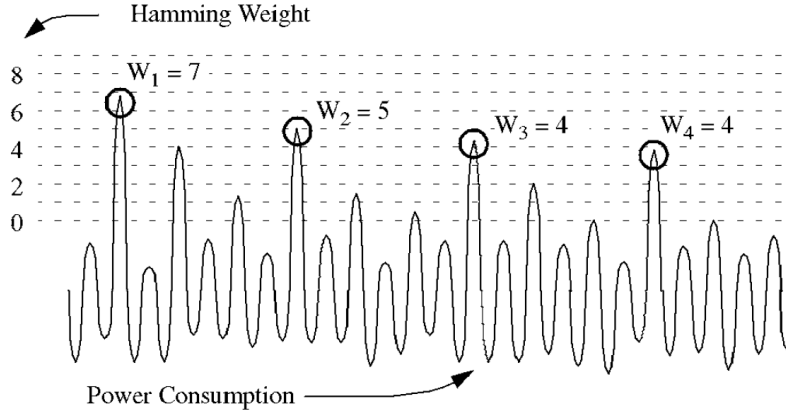
Figure 2.4   HW information revealed by the a power consumption side-channel [78].

are also ignored for simplicity. The HD of two values v_0 and v_1 is equal to the HW of $v_0 \oplus v_1$. Since HW is equal to the number of bits that are set to one, $HD(v_0, v_1) = HW(v_0 \oplus v_1)$ [73].

To calculate HD, consecutive data values processed in part of a circuit must be known or guessed. HD can be effective for modeling the power consumption of registers and buses when the values placed in the register or on a bus are determined by the algorithm being attacked, known plaintext or ciphertext value, and key guesses. Advanced HD models assign the power consumption for the transitions between $0 \to 1$ and $1 \to 0$ differently. Figure 2.5 shows how the number of transitions effects the power leakage of an 8-bit smartcard microcontroller.

The worst case is when the preceding and succeeding values are random and uniformly distributed. If this occurs the HW and HD models will not be highly correlated with the power consumption. However, since the power consumed for the transitions between $0 \to 1$ is not truly equal to the power consumed by $1 \to 0$ transition, the HW model will still be weakly related in some way to the actual power consumption [73].

21
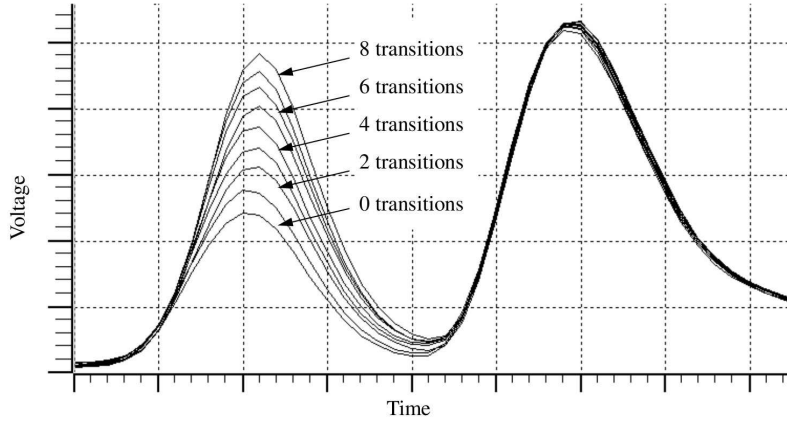
Figure 2.5   HD power leakage from an 8-bit smart-card micro-
controller performing a load operations [78].

2.3.4.3 Applying Power Models to Electromagnetic Emissions.   Com-
bining Ohm's law ($I = V/R$) with with Joule's law ($P = IV$), the power in a resistive
circuit current is directly proportional to the current squared ($P = I^2 R$). It follows
that models that describe power consumption also correspond with EM emissions.
If the power consumption of a device is data dependent, the EM emission will also
be data dependent.

Before introducing how these models can be used to extract information from
a device, it is helpful to understand the cipher being targeted by the attacks. Block
ciphers are presented in the following section. Side-channel attacks that used HW
and HD models are introduced in Section 2.4.

2.4   Side-Channel Attacks

Modern cryptographic ciphers, including AES, were developed assuming that
the hardware used to implement them was secure. Given that assumption, the focus
was on proving the underlying mathematical structure of the cipher is computation-
ally secure. When developing a cryptographic cipher it is generally assumed that the
cryptographic systems behave like a black box in which plaintext is securely turned
into ciphertext.

22

Rather than attack the cryptographic algorithm itself in the hopes of finding a mathematical vulnerability, side-channel analysis targets the devices used to implement the cryptographic algorithms. The goal of Side-Channel Analysis (SCA) is to learn information about the internal state, data or operations being performed. Side-channel attacks can also be used to bypassing or compromise the system's security.

2.4.1 Types of Implementation Attacks. Attacks on cryptographic devices can be *active* or *passive* [73]. During a passive attack the device performs its normal operations with little or no interference by the attacker. In an active attack, the device, its environment or inputs are manipulated to make the device behave abnormally. The abnormal behavior is analyzed to compromise the secret key.

Attacks can be *invasive* [8], *semi-invasive* [117] and *non-invasive*. In an invasive attack there is no limitation to what can be done to the device. The devices are typically depackaged to access specific components of the device using a probing station. If the probing station only observes the component, the attack is *passive*. If signals in the device are changed to alter the function of the device, the attack is *active*. Invasive attacks typically require expensive specialized equipment. In a semi-invasive attack the device is depackaged but, no direct electrical contact is made with the chip surface. Active semi-invasive attacks may induce faults using X-rays, EM fields, light or lasers. In a non-invasive attack, the device is attacked without altering the device leaving no evidence of an attack. Active non-invasive attacks attempt to cause faults without depackaging the device. The faults can be introduced by clock glitches, power glitches or by changing the operating environment. In general, side-channel attacks are non-invasive or semi-invasive.

2.4.2 Adversary Models. Similar to the cryptanalysis model in Section 2.2.3, the abilities and knowledge attackers possess varies. The power of an adversary is determined by the amount of knowledge and control he theoretically has over the

cryptographic system during a particular attack. A *weak* attacker will have very restricted access, while a *powerful* attacker will have complete control of the device.

Powerful attackers are able to choose the number and contents of device inputs, and are able to observe the encryption/decryption operation in an ideal environment. Measures may be taken to optimize the quality of the observed side-channel, such as decapsulation and adding a hardware trigger to precisely determine when the cryptographic operation begins [73]. In extreme cases, the attacker may have the ability to load new keys into the device or a similar training device. The attacker may also collect multiple traces for each plaintext and average the traces together to reduced environmental noise in the trace.

A *weak* adversary has less control over the device. The adversary has the ability to observe the device being attacked in some way, but no special measures are taken to improve the quality of the collected traces. As a result, the traces may be noisy and timing may be poor. Typically it is assumed that the adversary is able to collect either the plaintext or the ciphertext from the device. In extreme cases, only the side-channel can be observed.

2.4.3 Power and EM Analysis.   The introduction of power analysis in 1999 by Kocher et al. gave rise to a new field of side-channel attacks and countermeasures [67]. By observing and analyzing the power consumption of a device performing a cryptographic operation, information about the device's operation and data the device is processing can be determined. A side-channel attack targets a vulnerability in the implementation rather than attacking the cryptographic algorithm. Depending on the implementation and resolution of measurement devices an attack may be performed with a single trace. A *trace* is a set of side-channel emission measurements over the length of the cryptographic operation of interest. Any observable behavior that can be correlated to the internal operation of a device can reveal information about the device.
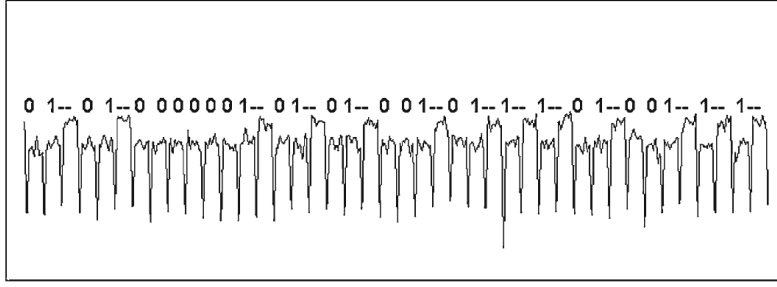
Figure 2.6   SPA analysis from an RSA implementation [68].

A large number of attack types have been developed, but the three most important distinguishing characteristics are the analysis approach, the number of traces used, and the number of phases involved in an attack.

2.4.4   Simple Side-Channel Analysis.   Some cryptographic implementations are susceptible to *Simple Side-Channel Analysis* (SSCA) in which information about the device's operation and key material is determined from direct interpretation of the power or EM emission traces. In [67] Kocher et al. introduced *simple power analysis* (SPA) noting that weaknesses in the implementation of an algorithm such as conditional jumps based on key bit's value and computational intermediates, reveal information about the key. Processing time may vary for various reasons including conditional branches, cache misses, pipeline stalls, interfacing with memory and external devices [66]. As a result, SPA can determine the sequence of operations for a cryptographic implementation. If the order or length of operations are dependent on key bit values, the value of the key bits may be determined from the power trace. The techniques used for SPA were extended to EM emissions in [94] and called *simple EM power analysis* (SEMA).

Figure 2.6 shows power traces from a device implementing public-key cryptography algorithm RSA[1] [68]. In this implementation, a square operation is performed in every iteration of the exponentiation loop but a multiplication is only performed

_____

[1]The RSA algorithm is named for Ron Rivest, Adi Shamir and Leonard Adleman [33].

when a bit of the exponent is 1. Each 1 bit in the secret key appears as a shorter bump followed by a taller one, each 0 bit appears only as a shorter bump. The key can be read directly from the measured power consumption.

2.4.5 *Differential Side-Channel Analysis.* Even when SSCA is not possible, differential side-channel analysis (DSCA) can be used whenever a physical, measurable property of the device depends on the data it processes. Differential Power Analysis (DPA), introduced in [67], takes small variations in power consumption between multiple traces to find correlation between the intermediate values in the cryptographic computation and the measured power consumption. Although the variations are small, by collecting a large number of traces, the implementation can be broken using statistical functions tailored to the target algorithm and device [67]. A DPA attack uses the power traces from multiple observed encryption operations, typically with different plaintext/ciphertext. Using the recorded plaintext or ciphertext and traces from the operations, the attacker calculates differential statistics based on a key block guess. DPA is capable of extracting information even when the variations in side-channel are too subtle to be identified using SPA. The techniques used in DPA were extended to EM emissions and called differential EM analysis (DEMA) [94]. The step-by-step process for conducting model-based DPA is explained in Section 3.4. Differential attacks typically assume a powerful adversary that can arbitrarily change the plaintext to perform desired encryption operations. At a minimum the attacker must have knowledge of the plaintext or ciphertexts associated with each trace.

Differential attacks can be used even if detailed knowledge of the implementation is not known. While SSCA requires the attacker recognize when certain operations occur in side-channel leakage, DSCA techniques identify the points in time when side-channel leakage is correlated with a hypothetical intermediate values in the cryptographic operation. As a result, the attacker only needs to know the underlying algorithm so that hypothetical intermediate values can be calculated, to

carry out an attack. Even when the algorithm is not known, an attacker can perform SSCA and DSCA to learn details about the implementation sufficient to perform a successful attack [73].

The number of observations required to successfully perform DSCA depends on the implementation, statistical technique, environmental factors and countermeasures protecting the device. The number of traces required can vary from a few dozen to millions [73].

2.4.5.1   Statistical Methods.   DSCA uses statistical methods to make inferences about the data processed on the device. Using multiple traces, the statistical method reduces the noise from measurement error and non-data dependent emissions while amplifying the data-dependent contributions. DSCA can detect very small correlations provided a sufficient number of traces are collected and analyzed. A number of statistical methods have been applied to DSCA to determine how the data being processed is correlated with side-channel emissions.

DSCA techniques attempt to identify an affine relationship between the predicted leakage and one or more columns of the observed matrix data. In the observed data matrix each column corresponds to a particular time sample in relation to the start of a cryptographic operations and each row corresponds to a possible key value. When the key hypothesis is correct, there is a linear relationship between the hypothesized leakage value based on the leakage model and the observed leakage.

The original method, proposed in [67] and formalized by [77, 78], known as difference of means (DoM), takes traces collected for known plaintext and divides the traces into two groups according to the intermediate value predicted by a key guess and the trace's corresponding plaintext. If the average power trace from each group differs from each other in a significant way, it is likely the key guess is correct. Key guesses can be made at the bit level, byte level or for multiple bytes at a time.
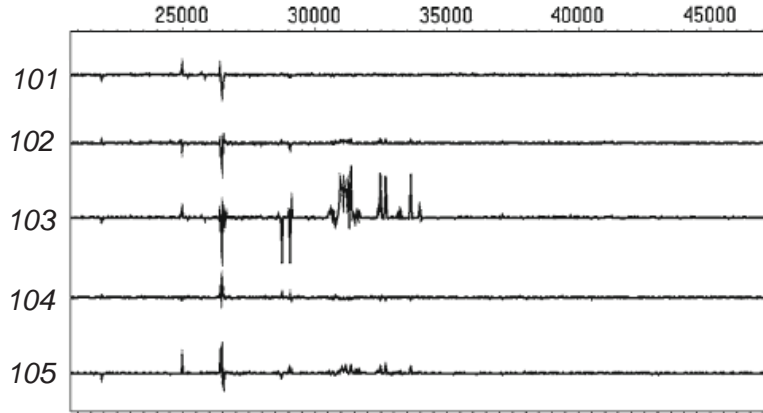
Figure 2.7   Five differential traces for a DPA test predicting
the least significant bit of an S-box output. Traces
for key guesses 101 to 105 are shown. Key guess
103 is the correct value [68].

The architecture of the device used to implement the cipher and the way in which
it was implemented will effect which type of attack will be most effective.

The use of Pearson's correlation coefficient was first proposed in [22]. Using
Pearson's correlation coefficient, the highest correlation coefficient indicates the cor-
responding sub-key guess most likely to have produced the observed results. The
location of the peak indicates the time at which the targeted intermediate value
is manipulated. Figure 2.7 shows a plot of the correlation coefficient for five sub-
key hypotheses. Since the plot for sub-key 103 contains the highest peak, it is the
sub-key most likely to have produced the observed trace. Similar keys can produce
high correlations, leaving some ambiguity. If it is not clear which peak is correct,
additional traces can be collected or multiple possible sub-keys can be identified.

A comparison of four DSCA distinguishers was conducted in [40]. In addition
to the DoM and Pearson's correlation coefficient, the Spearman's coefficient [15],
variance test [121], and Student's T-test [34, 53] distinguishers were evaluated using
power traces collected from a hardware implementation of DES. Although the results
are likely implementation dependent, Pearson's correlation coefficient had the highest
first-order success rate. That is, the sub-key identified by Pearson's correlation

28

coefficient was most often correct. However, in cases where Pearson's correlation coefficient did not identify the correct sub-key as the most likely key, the correct sub-key was ranked low. DoM yielded the highest guessing entropy. When the first-order DoM guess was wrong, the correct sub-key was still highly ranked. Guessing entropy is an important consideration when a SAT solver is available to evaluate the validity of sub-key guesses.

A more precise definition of Pearson's correlation coefficient is presented in Section 3.4.1.

2.4.6 *Profiling Attacks.* Profiling attacks differ from standard SPA and DPA attacks because they require two stages rather than one. The first stage is a profiling stage which is used to obtain *a priori* knowledge on the side-channel leakage for a specific device. Unlike standard DPA attacks, it is possible to conduct a profiling attack without power consumption models that accurately predict the side-channel leakage [53].

The key assumption for a profiling attack is that a powerful attacker has access to a training device, identical to the target device, over which he has full control. The training device is used to create a precise multivariate distribution of the device's side-channel leakage for each sub-key dependency [53]. It is assumed the side-channel leakage of the device being attack is sufficiently similar to the leakage of the training device. The training phase is sometimes referred to as the *offline* phase.

During the attack, or *online*, phase the distributions calculated during the profiling phase are used to classify side-channel observations from a target device. The attacker does not need to have control of the device, but must be able to observe the side-channel leakage. The trace collected from the target device is classified to determine the most likely sub-keys. Profiling attacks are considered to be very powerful because they utilize all information in each side-channel sample [24].

A number of different profiling attacks have been proposed including template attacks [24] and the stochastic model attack [108]. Chari et al. observed that using multivariate statistics allows for stronger attacks [24]. They believe the new attack, which they called a *template attack*, is the strongest side-channel attack possible from an information theoretic sense. Template attacks use all information present in each portion of a side-channel trace for classification, making them a strong attack even when only a few traces from the device being attacked are available. Rather than try to eliminate or reduce noise, the noise present in the side-channel emission is assumed to be key dependent and precisely modeled. The profiling stage creates mean and covariance matrices for each of the possible sub-keys. The profiling stage allows attacks to be conducted using as little as one trace. Agrawl et al. expanded template attacks to differential power analysis, allowing multiple traces, or even side-channels, to be incorporated into the template [4].

Building templates is the optimum way to describe the side-channel characteristics of a device. When the side-channel leakage fits a multivariate-Gaussian model a template attack is the optimal DPA attack because it minimizes the probability of error when determining the key. This optimality is only achieved when the ideal data dependent points of interest in the encryption operation are located and used to build the template [73].

The efficiency and effectiveness of stochastic model attacks [108] and template attacks [24] are evaluated in [53]. Unlike template attacks, the stochastic method developed by Schindler et al. presumes that side-channel noise is independent of the sub-key and it is not incorporated into the model. The stochastic method is more efficient, but is not more effective than template attacks [108]. The stochastic model is useful when there is a bound on the number of traces that can be collected during profiling; template attacks are more effective when there are enough traces to construct a full set of templates.

Agrawal et al. introduced the single-bit template attack and template-enhanced DPA attack [3]. Rather than building templates for each of the possible 256 key-byte values, templates were created to attack a single bit in an intermediate value. The single-bit templates are built from peaks observed in DPA attacks and predict the value of single-bit with high probability using only one side-channel observation during the attack phase. An attacker can build a large number of single-bit templates, each of which can be used to identify the values of individual bits during the attack phase. With enough precomputed templates, the entropy of the key is reduced significantly so a brute force attack is practical. The single-bit template attacks can be incorporated into a template-enhanced DPA attack which was able to defeat standard random masking techniques on smart cards.

Renauld et al. explore the increasing variability in device leakage for cryptographic devices with features sizes of 65-nanometers and smaller [103]. Using a prototype S-box implemented in a 65-nanometer low-power CMOS technology, they demonstrate that with reduced feature size cryptographic devices may not follow common leakage models. Additionally, they show the increased variability leads to degraded leakage models and template attack performance when using one device to attack another. To account for inter-chip variability, they propose training across multiple devices, but note that incorporating the inter-chip variability into the template makes the models less accurate when attacking any individual chip.

A powerful unknown-plaintext, unknown-ciphertext template attack based on HW templates, which incorporates an algebraic description of AES, is discussed in Section 2.8.4.1. The steps required to perform a template attack are discussed in Section 3.6.

2.4.6.1 Identifying Important Components of the Trace. Templates consist of a vector of means and a matrix of covariances for each class. Templates are constructed for specific points in the encryption operation related to the tar-

get operation. Since the size of the covariance matrix grows quadratically with the number of points included in the template, and calculating the observation probability involves a matrix inversion, the number of points included in each template dramatically effects processing time [99].

It is computationally infeasible to include all of the points in each trace to construct each template. Ideally, only points that distinguish between the different classes considered by the template attack will be included. DSCA methods can be used to identify these points. One option is to sum the absolute differences of mean traces and select the highest points [24], or use the cumulative difference between the mean traces [99] ensuring only one point per clock cycle is chosen. While heuristic approaches for selecting these points have been effective, a number of more systematic approaches have been developed.

Assuming the majority of information content of a leakage trace is contained at the time instances of maximum inter-class variability, Principal Component Analysis (PCA) can be used[2] [9]. PCA is an orthogonal linear transformation that maximizes the inter-class distance when projecting the data into a lower-dimensional space. As a result, the dimensionality of the data set is reduced while retaining the majority of the information. An alternative linear transformation is Fisher's Linear Discriminant Analysis (LDA), which maximizes the ratio between inter- and intra-class variance. While for some implementations LDA has been shown to be more effective than PCA, it is substantially more computationally expensive then PCA [122].

Archambeau et al. used PCA to perform template attacks in the principal subspace of the mean traces for each class [9]. They applied PCA to collected data from an implementation of RC4 running on a PIC 8-bit micro controller and an FPGA implementation of AES. Traces with 300,000 time samples were collected from the PIC. Using heuristic methods, 42 test samples were selected for building the templates

---

[2]Depending on the field of application, PCA is also known as the Karhunen-Loève transform, the Hotelling transform or proper orthogonal decomposition.

and the average classification success rate was 91.8%. Using PCA, 7 components were identified and proved to be sufficient to ensure a correct classification of 93.3%. For the FPGA, traces with 500,000 time samples were collected and PCA identified 20 components. Using 128 encrypted messages the average classification success was 86.7%. Archambeau et al. did not test the performance of the template attack using heuristic methods to choose the test samples.

Preprocessing leakage traces using PCA provides a systematic way to consolidate/identify the most important features of a class, and may allow for superior classification results using a smaller number of test points. The computational effort required to build the templates is thereby dramatically reduced. PCA identifies which components account for the majority of the variance between classes. How PCA is used in this research is explained in Section 3.6.5.1.

2.5 Countermeasures

Numerous side-channel analysis countermeasures have been proposed. In practice, protecting implementations against side-channel analysis is difficult and expensive. All countermeasures attempt to make the power consumption of a cryptographic device independent of the data being processed. Since most countermeasures only increase the cost of attacking a device without fully protecting it, the cost of implementing the countermeasures must be compared with the additional security it provides. Countermeasures can be broken into two categories, hiding and masking.

2.5.1 *Masking.*   *Masking* attempts to randomize the intermediate values being processed by the cryptographic device changing the power consumption characteristics of the device [73]. Masking techniques are based on various secret sharing schemes [23, 54]. Masking does not reduce the side-channel emissions, but rather attempts to make the leaked intermediate values independent of the key. A mask is used to conceal the value of intermediate values. Since the mask is generated within

the cryptographic device and varies from execution to execution, it is unknown to the attacker. A masked intermediate value v_m is an intermediate value v concealed using a random value m such at $v_m = v * m$, where $*$ is the masking operation. Logical XOR (boolean masking) and modular addition and multiplication (arithmetic masking) are typical masking operations. To prevent leakage of actual intermediate values, the masked values are processed by the algorithm, and the final result is unmasked [73].

2.5.2 *Hiding.* *Hiding* attempts to make the power consumption of cryptographic devices independent of the intermediate values and independent of the operations performed. The signal-to-noise ratio (SNR) can be decreased by making changes to the implementation of the cryptographic algorithm or hardware via hiding techniques. *Hiding* includes algorithmic countermeasures to randomize the intermediate results processed during a cryptographic operation. Common hiding techniques include inserting dummy instructions, random process interrupts, clock skipping, randomly changing the clock frequency and including multiple clock domains in the device [73].

Hiding techniques, in general, attempt to make the timing of the implementation non-deterministic. Correctly implemented, these techniques are very effective against first-order DPA, but can easily be defeated using higher-order DPA attacks [72]. The operation of the circuit can be modeled as a finite state machine [63] and the randomization can be analyzed. Resynchronization techniques can be used to bypass the randomizations. Furthermore, randomization techniques require additional resources and clock cycles to implement, making them expensive.

The SNR can also be decreased by adding noise. Adding noise does not provide any fundamental protection against side-channel analysis, but may make an attack more difficult. The data dependent signal is still being generated by the device and can still be recovered [123]. Noise is typically generated by adding additional logic

to the device to perform unrelated, ideally random, operations. As a result, noise generation may be expensive and increase the power consumption of the device. The frequency of the generated noise must be matched to the frequency of the information containing signal, otherwise, the noise can be detected and filtered from the collected trace [68].

FPGA implementations are more difficult to exploit for two reasons. First, performing parallel computations in hardware significantly reduces the SNR. Second, FPGAs operate at higher frequencies, making side-channel data collection more difficult. Combining pipelined and unrolled implementation, or unrelated operations, on the same device is an effective way of efficiently increasing noise [123].

Some countermeasures are algorithm independent. These countermeasures implemented in hardware include special leakage resistant logic styles, with the goal of reducing SNR. To gain resistance, leakage resistant logic styles try to equalize power consumption for all operations. As a result, implemented ciphers required twice as much space and power consumption is doubled, making this approach less practical [98]. For dynamic and differential logic, the output capacitance is independent of the input transitions. Power consumption differences are due to parasitic capacitances in the designs and can only be predicted with *transistor-level* knowledge of the circuit. Without this knowledge an attacker is not able to create a precise power consumption model, however, template attacks can still be used [123].

Although FPGAs are not constructed using dynamic differential logic, gate level designs with the same properties can be implemented on an FPGA [131]. A customized design flow is used to implement AES using dynamic differential logic on an FPGA requiring a 50% time delay and a 90% increase in slice utilization.

2.6 Collecting Electromagnetic Emissions

To reduce the noise present in the traces, EM side-channel collections are usually performed in the near-field [73]. Although a number of far-field attacks have

been demonstrated, they often use simplified versions of cryptographic implementations [64], specialized collection processes, or are limited to SEMA [2].

For example, with the use of a directional antenna and 30dB pre-amplifier Kim et al. were able to successfully attack an implementation of a single S-box on a FPGA using a hardwired trigger for timing [64] from 1 meter away. Agrawal et al. performed SEMA attacks on an Intel-based server containing a commercial PCI bus based SSL accelerator from 40 ft away using biconical and log-periodic wide-band antennas as well as hand-crafted, high-gain Yagi antennas [2].

2.6.1 Electronic Noise. Power and EM traces are subject to noise and as a result repeated traces for constant inputs will be different. Noise can be categorized as electronic noise, and switching noise [73]. Although steps can be taken to reduce electronic noise, every trace will contain some noise from the power supply, clock generator, conducted emissions from other components connected to the device under attack, and radiated emissions from other electronic devices near the device. Additionally, since the side-channel data is digitized for analysis, quantization noise is also present. In addition to the power consumption and EM emissions from the circuit of interest during the attack, many other operations may be conducted simultaneously on the device. The variation in the power and EM traces caused by cells not involved in the operation of interest, and therefore not relevant to the attack, is called switching noise. DPA and DEMA techniques, which perform signal processing and statistical estimation on a collection of traces, can mitigate measurement noise. Hundreds or thousands of traces are often required to perform this type of attack [73].

2.6.2 Improving Collections. A powerful adversary will be able to take measures to improve the quality of collected traces by various means. Traces are typically collected with a high speed digital capturing oscilloscope, due to their ability to sample and store the side-channel at a high sampling rate. Software-

defined radios (SDRs) have been used to perform SEMA attacks on unprotected implementations of RSA [61], but no differential attacks have been demonstrated using SDRs. The use of SDR for differential side-channel analysis is introduced in Chapter 7.

2.6.2.1 Hardware Trigger. Adding a hardware trigger to the cryptographic device being attacked can dramatically improve the timing of collected traces. The cryptographic device is modified by the attacker or designer to signal immediately before the encryption operation begins, allowing the capturing digital oscilloscope to trigger at the same time relative to the start of each encryption operation. As a result, corresponding samples in each trace correspond to the same portion of the encryption operation.

2.6.2.2 Clock Signal. The harmonic content of a square wave is determined by its rise time and not its fundamental frequency. EM compatibility guides recommend using a dithered clock which intentionally varies the clock frequency by a small amount to spread the emission out in the frequency spectrum. This will reduce the strength of the emission at any one frequency [93]. If a powerful attacker has the ability to use a clock with a quicker rise time than the FPGAs internal clock, the concentration of the leakage at clock harmonics can be increased [73].

2.6.2.3 Cartography. Compared to power analysis, EM analysis has the advantage of being able to target the leakage of specific areas of a chip. Using small probes placed in the near-field allows the EM emissions from specific parts of a device to be isolated. Every element on the FPGA will contribute in some way to the captured EM field. Efforts to isolate the leakage from specific portions of a device have been shown to reduce the number of traces needed to perform a successful attack. Using near-field probes, EM cartography was demonstrated to enable more efficient attacks on FPGAs in [107].

To find the location on the device with the highest data-correlated emissions, a scan of the surface can be performed. Creating a coarse resolution EM leakage map allows physical locations on the device with greater leakage to be identified and targeted. It has been shown that EM emissions correlated to the data being processed is not restricted to the area on the device where the data is being processed. For example, leakage was observed to originate from both the encryption processor implementation on the FPGA and from a surface mount ceramic capacitor located outside of the FPGA in [107]. Power, ground networks, clock paths, and buffer trees all leak information. By targeting the leakage with a directional near-field probe, the number of traces required to attack the device was significantly reduced compared to collections made with larger probes.

2.6.2.4 Benefit of Isolation. To reduce interfering signals from other devices, Gandolfi et al. placed the cryptographic device into a Faraday cage [47]. They determined that isolation of the device had little effect on the noise present in the readings. Furthermore, even if the device and probe can be isolated, the remaining trace collection equipment is still subject to ambient noise and is prone to cross-talk. This result conflicts with the benefit of isolation reported by Mangard ([70] as cited in [40]). All isolation studies were conducted using near-field measurements.

2.7 Pre-Processing Processing Techniques

In cases where special efforts to improve the quality of collected traces cannot be made, pre-processing can improve the effectiveness of side-channel analysis. The techniques in the following sections identify the frequencies which leak information, improve alignment of traces and reduce the complexity of analysis through data reduction.

2.7.1 Detecting Compromising Frequency Components. One technique to detect potentially compromising emissions is to use a wide-band receiver tuned to a specific frequency. High-end TEMPEST receivers can scan across a range of frequencies to identify potential compromising emissions, and demodulate the signal using AM and FM demodulation. The demodulation can also be performed using software.

In [136] a software radio was constructed using the Universal Software Radio Peripheral (USRP) and GNU Radio project. The software radio was able to scan from DC to 2.9 GHz using various daughterboards. The GNU radio project which includes libraries for processing AM and FM modulation and performing signal processing techniques such as filtering and Fast Fourier Transform (FFT) processed the collected signals. Combining the USRP with the GNU Radio project, a wide-band receiver and a spectral analyzer with software-based FFT computation was constructed [135, 136].

If a signal is composed of irregular peaks and erratic carrier frequencies, methods such as spectral analyzers and scanning with wide-band receivers may fail to identify some direct and indirect EM emissions [136]. Signal analyzers require constant or long duration carrier frequencies and since the scanning process is not instantaneous, emissions may be missed by a wide-band receiver. Typically when a frequency is identified using one of these methods, the frequency is isolated using narrow band antenna and filters. This method is not ideal because the signal is captured at base band and with limited bandwidth. Important information at other frequencies may be lost, reducing the entropy of the signal.

Meynard et al. showed a SEMA attack against an RSA processor implemented on a side-channel Attack Standard Evaluation Board (SASEBO) FPGA board can be enhanced using a hardware demodulation receiver [79]. Starting with a SEMA resistant implementation of RSA, the raw recorded EM traces do not allow discrimination between the square and multiply operations. To determine the appropriate

demodulation frequency, the spectral signature of each operation was found by isolating each operation and performing a FFT. Using mutual information techniques [52], a metric for the amount of information contained at different frequencies was calculated and multiple frequencies with high information content were identified. Using the frequencies identified with mutual information analysis in the frequency domain as the demodulation frequency, the start of square and multiple operations are easily identified in the demodulated signal. In addition to identifying harmonics of the clock frequency, other frequencies believed to be caused by direct emissions were found to have high information content [79].

The use of signal processing to enhance the effectiveness of side-channel attacks is introduced in [13]. A leakage model identified which frequencies contain useful information by examining the Discrete Fourier Transform (DFT) of the collected traces. Since power consumption in CMOS devices is mostly due to signal transitions and power consumption is proportional to the voltage swing and operating frequency of the device components, the magnitudes of the clock harmonics in the DFT are significantly greater than other frequencies. The information leakage is amplitude modulated on harmonics of the clock. To improve the SNR by removing components of the signal not correlated to the information, a filter was designed to remove signal components not related to the clock or clock harmonics.

Using the DFT of the power traces, the harmonics of the clock frequency were identified. After locating the main clock and each significant harmonic using the DTF, 500kHz wide passbands were centered around each. The Chebyshev windowing technique was chosen to implement the bandpass filters due to its rapid side lobe roll-off and uniform side lobe attenuation. The filtering method was validated using a 32-bit Cortex-M3 processor running a software implementation of the AES-128 without DPA countermeasures. Performing the identical DPA attack before and after filtering, the number of traces required for a successful attack was reduced from 6000 to 450. To reduce the amount of environmental noise, 16 measurements are

40

taken with each plaintext and averaged. The technique also enabled better alignment of traces because artifacts in the original signal that do not carry information are removed [13].

The above filtering technique is extended to efficiently search the frequency spectrum and identify which frequency bands contain important information [14]. The algorithm focuses on harmonic components in the measured signal to avoid sweeping the entire frequency range. To characterized the leakage in certain frequency bands, the spectrum is split into equally sized shares and filtered as in [13]. A series of DPA attacks are conducted on the filtered output for each frequency band to determine the number of traces required for a successful attack. The searching function is recursively called for frequency regions that yield successful attacks.
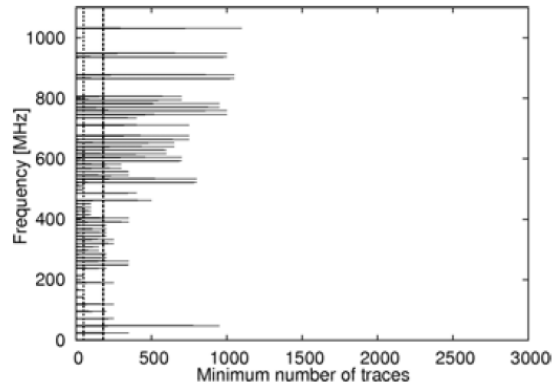
Figure 2.8   The minimum number of traces required for a successful attack for each slice of the frequency spectrum. The leftmost dashed line indicates the minimum number of traces for a successful attack. The rightmost line indicates the number of unfiltered traces required [14].

When finished, the algorithm provides a list of both the shortest frequency intervals that yield a successful attack and the largest frequency intervals that fail using the entire trace set. The number of traces required to successfully recover the key using the traces filtered for a specific frequency band is used as an estimator of the information leakage over that frequency band. Figure 2.8 shows the results for

a DEMA attack using up to 3000 traces. Unsuccessful attacks are represented as requiring zero traces. Since the information leakage tends to be clustered across the frequency domain, the search algorithm is more efficient than conducting a brute force search of the entire frequency spectrum [14].

2.7.2   Trace Alignment.   Most side-channel analysis techniques compare the recorded side-channel emission values at specific points in the encryption operation. The traces must be aligned so the cryptographic operation being attacked occurs at the same time in each trace. When attackers have complete control of the system they typically build a trigger signal into the hardware to ensure that each collected trace starts at the same point in the encryption operation. When the attacker is not able to modify the device, real-time processing of the signal to identify a pattern in the signal can be performed. Once the traces have been collected, they can be analyzed to identify the portions corresponding to the encryption operation in each trace [73].

Even when a trigger signal is used, it may contain jitter-related deviations from the timing of the cryptographic computations. Displacement errors can cause significant data loss of secret information when analysis techniques average the waveforms together. Traces can also be misaligned for a number of reasons, including countermeasures like random dummy operations and shuffling. Various methods have been proposed and demonstrated for aligning traces.

2.7.2.1   Pattern Matching.   Pattern matching in the time domain is the most common alignment technique. A portion of the first trace is selected as the pattern. The attacker tries to find the pattern in all other traces to identify the offset between the traces. There are a number of important considerations when selecting a pattern [73].

First, the pattern should be unique. The more distinct the portion of the trace is, the better the alignment process will work. Targeting a unique operation, such

42

as loading the initial registers of an AES-128 implementation will work better than building a pattern for the output of the S-box since a similar operation occurs in each of the 10 rounds. Second, the pattern should be from a portion of the operation that is *not* data dependent. For example, when attacking a device with a fixed key where the key schedule is generated on the fly, portions of the trace that correspond with round keys being calculated will not be data dependent.

Next, the length of the pattern is important. Longer is only better if the operations in the pattern do not depend on intermediate results. Including data dependent portions of the trace in the pattern can degrade the matching results. Finally, if countermeasures such as inserting random dummy operations are used, the pattern should be as close as possible to the portion of the trace that is dependent on the intermediate value being attacked.

Least squares and correlation coefficient methods are the most common approach used to identify the pattern in a trace. To improve accuracy and reduce processing time rather than trying to find the pattern in the entire trace, an attacker should focus on a smaller search interval based on the location of the pattern in the first trace [73].

2.7.2.2 *Phased-Based Alignment.* A number of methods to perform alignment using the frequency domain have been demonstrated. The phase-only correlation technique allows fine grain alignment of traces with high noise tolerance. The technique uses a cross-phase spectrum formed with the reference trace and the subject trace. If the two traces are similar, the inverse discrete Fourier transform of this spectrum forms a distinct peak at the location of the translational displacement between the reference and subject trace. Using an analytical model of the correlation peak, the displacement between the waveforms can be estimated with higher resolution than the sampling resolution. An interpolation technique can then be used to finely align the traces. The magnitude of the correlation peak can also be

used to identify inaccurately measured traces that would have adverse effects on the statistical analysis if included in the attack [59].

Gebotys and White demonstrated a phase-based technique for temporal alignment EM traces [49]. The method is robust for complex systems, even those with random delays and random operations. The technique, called phase substitution, is based on the fact that a shift of a signal in the time domain corresponds to a change in the phase of the signal in the frequency domain. To perform phase substitution, the FFT of each trace is calculated. Next, one trace is randomly chosen as the reference from the collection of traces and the phase of all other traces is replaced by the phase of the reference trace. Finally, the inverse FFT is performed to transform the trace back into the time domain. Once the phase substitution alignment is completed, an appropriate side-channel analysis attack can be performed. Phase substitution adds noise in the time domain, but it is largely averaged out by differential analysis techniques.

2.7.3 Frequency-Based Analysis. In addition to using the frequency domain to perform alignment, side-channel attacks can be performed using representations of the traces in the frequency domain. By ignoring phase information in the frequency domain, alignment problems can be mitigated. Gebotys et al. proposed frequency-based DEMA based on the spectrogram of the collected traces [48]. Gebotys and White the demonstrate a frequency-based DEMA attack using the Power Spectral Density (PSD) in [50]. After the portion of each trace with the operation of interest is identified, the regions are extracted an the PSD of these regions are used for the attack. Unfortunately, the amount of preprocessing required makes this type of attack impractical. Hodgers et al. use an overlapping window method to reduce the amount of preprocessing and eliminate the problem of sampling boundaries [58]. This ensures the entire region of attack is contained within the portion of the trace that will be used for at least one PSD provided the window is large enough. The technique follows typical correlation-based attack methodology (cf. Section 3.4.1),

but uses the PSD data set in place of the collected time-domain EM traces. The method is shown to be effective for both aligned and misaligned traces.

Pre-processing traces by taking their FFT has been shown to be a viable method to enhance template attacks [99]. The FFT of each trace is used in place of the collected EM trace in the template attack methodology. Basing the template attack on the FFT was shown to allow for a successful attack even when the ambient noise in the time domain did not allow for successful classification.

2.8 Algebraic Cryptanalysis

As block ciphers have become more important, a number of powerful cryptanalysis methods have been developed; these include differential and linear attacks as discussed in Section 2.2.3. Most of these methods submit particular statistical patterns through rounds of the cipher to determine if non-random behavior can be observed in the output [18]. Newer ciphers, including AES, were developed to be resistant to these techniques and are thus not vulnerable to these types of attacks.

As introduced in Section 2.2.3.2, an alternate approach for the cryptanalysis of block ciphers such as AES and DES is to exploit the algebraic structure of the cipher by constructing algebraic systems of equations which completely describe the cipher.

2.8.1 Describing a Cipher. In algebraic attacks, equations describe the output bits of a cipher in terms of its input bits and key. Since modern block ciphers are implemented in hardware or software, their operations are typically defined over $\mathbb{GF}(2)$. As a result, the Boolean equations are written as polynomial systems over $\mathbb{GF}(2)$. The Galios Field arithmetic required for AES is reviewed in [88].

Theoretically most modern block ciphers can be fully described by a system of multivariate polynomial equations over a finite field. In practice, the majority of

these systems are too complicated for any practical purpose. However, due to its algebraic structure AES may be vulnerable to algebraic cryptanalysis [26].

As in most block ciphers, the only non-linear element of the AES is the S-Box. Since the S-Box is based on an inverse function, a small set of quadratic multivariate equations in terms of the input and output bits completely define the S-box. For an S-box of any practical size, a basis of linear independent multivariate polynomials can be generated which span the space of all possible equations between the input and output bits [18]. By limiting the set of equations to the basis equations, the size of the system is reduced.

2.8.1.1   Strategies for Describing Ciphers as Equations.   Writing a set of equations for the linear components of a block cipher, including linear diffusion layers and key additions, is straightforward. These are combined with the equations for the non-linear components to completely define the cipher. Although a cipher can be described in the terms of a multivariate system of equations over $\mathbb{GF}(2)$, that does not guarantee it can be broken. Solving a system of multivariate quadratic equations (known as a MQ problem) is NP-hard. Such systems of equations have a number of properties that describe their computational complexity.

Shamir et al. showed that the complexity of a MQ problem drops substantially when a system is over-defined [112]. An *overdefined* system has more equations than unknowns. For a block cipher, using additional plaintext/cipher text pairs is a straightforward way to create an over-defined system of equations.

To reduce the complexity of the system of equations, equations with common terms can be combined. Rather than write separate equations for bit permutations, for example, variables are renamed to prevent redundant variables. Once equations have been derived for each component, they are combined into a system of equations for the system. Both [36] and [86] construct simple algebraic equations to describe AES.

There are numerous ways to represent the same system of equations. Since most modern cryptographic systems are implemented on inexpensive hardware they have moderately low gate counts, resulting in a sparse system of equations [36]. The *sparsity* of a system is the ratio of coefficients that are non-zero to the total number of possible coefficients. Some algebraic cryptanalysis techniques work better on densely defined systems, while others are more efficient for sparse systems. The technique used to solve the system of equations should be kept in mind when equations are generated.

The number of variables in each equation affects how difficult the system is to solve. A system of any degree can be written as a degree 2 system using the following step repeatedly

$$\{l = wxyz\} \Rightarrow \{a = wx; b = yz; l = ab\}. \tag{2.3}$$

Likewise, any equation can be written as a system of smaller equations. The maximum degree of the smaller equations is known as the cutting number [11]. A methodology for describing ciphers as systems of multivariate polynomials is described in Appendix A.

2.8.1.2 Systems of Equations for AES. Biryukov and De Canniere show in [18] that each of the 160 8-bit S-boxes in AES can be completely defined by a system of 23 quadratic equations in 80 terms. The 11 linear layers in AES can be written as a system of 128 linear equations and the complete implementation of AES has been defined as a system of 4000 multivariate quadratic equations with 1600 variables [36]. Unfortunately, the systems of equations have not been made available from either of these research efforts.

However, two systems of polynomial generators have been published. The small scale variants of the AES (SR) Polynomial System Generator [7] based on [25], and SYMAES, a fully symbolic polynomial generator for AES-128 [134]. These tools both run in Sage Mathematics Software [124] and can generated systems of equations

for AES-128. The SR Generator is used in Chapter 3 to create the AES-128 SAT solver tool used in this dissertation.

Once the cipher has been fully defined as a system of equations, the system can be solved to determine the cryptographic key.

2.8.2 Solving a System of Equations. The most naïve approach to solve a system of equations is to guess all of the variables (brute force). If the system has n unknowns, 2^{n-1} guesses must be tried to have a 50% chance of finding the correct solution. Algebraic cryptography attempts to exploit the algebraic properties of the cipher to solve the system of equations in less time. Various methods have been developed for solving non-linear multivariate systems of equations. The method employed by this research is a satisfiability (SAT) solver. Alternative methods not used in this research are included in Appendix A.

Using a SAT solver for cryptanalysis was first proposed by Massacci and Marraro [75]. They demonstrated that DES could be written as a system of equations and used three different SAT solvers to solve DES reduced round implementations. Courtrois, Bard and Jefferson discovered that SAT solvers and Gröbner bases algorithms such as F4, can solve very sparse or over-defined systems of quadratic equations efficiently even in cases where the performance of algebraic elimination methods is greatly degraded [12]. Furthermore, Courtois and Bard showed it is possible to solve very large systems of multivariate equations with more than 1000 unknowns derived from a contemporary block cipher such as DES [37].

2.8.3 Using SAT Solvers. Solving systems of multivariate quadratic polynomials is known to be NP-complete [35]. Therefore, rather than solving the system directly, the system is translated into a SAT problem.

SAT solvers determine an assignment of a set of variables over a domain such that a set of equations or constraints holds true for those variables or, alternatively,

determine that no such assignment exists [127]. The term SAT refers more specifically to the problem of assigning values to variables in a given Boolean formula to find a variable assignment which makes the Boolean statement true, or satisfied. Although SAT problems are also known to be NP-complete, they are a well-studied class of problem and the development and enhancement of SAT solvers is ongoing.

SAT solvers determine if a particular set of constraints have a solution. These constraints are often written in conjunctive normal form (CNF). Each element in the constraint (a or $\bar{a}$), is called a *literal*. A *clause* is a disjunction (or statement) of literals. Constraints presented to a SAT solver in CNF are written as a conjunction of clauses. See Appendix A for more detail.

Conflict-driven SAT solvers attempt to find a satisfying variable assignment. For example, MiniSat uses a backtracking-based, depth-first search algorithm [120]. The algorithm branches on a variable by guessing true or false and determining if other variables depend on the guess. Variables affected by this guess are assigned values and the algorithm continues to branch until no more assignments can be made. This period is called *propagation*. If a clause is found that cannot be satisfied a *conflict* is identified and a *learned clause* is generated that records the incorrect guesses that led to the conflict. Based on the learned clause, the top most guess allowed is reversed and propagation continues. The collection of learned clauses trims the search tree and guides the algorithm in choosing the next guess. The algorithm eventually identifies a satisfying variable assignment or the search tree is exhausted meaning that no solution exists. Figure 2.9 shows an example search path taken by a SAT solver.

2.8.3.1   Optimization for Cryptography.   SAT solvers typically require the cipher to be described as a system of equations written in CNF. This process can be cumbersome and adds additional complexity to the problem. To tailor SAT solvers for use in cryptography, Soos extended CryptoMiniSat's input language to
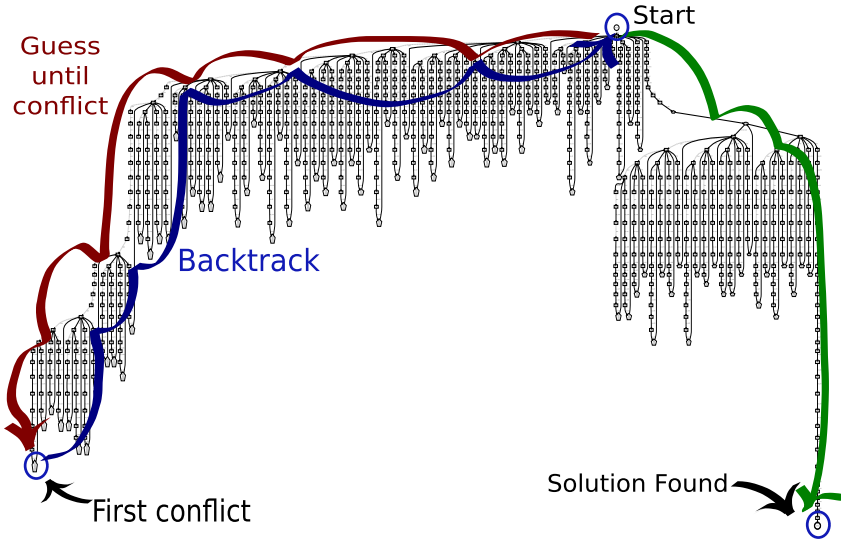
Figure 2.9   Visualization of a SAT solver's search for a solu-
tion. The first conflict clause and path to the sat-
isfying assignment are highlighted [118].

support the XOR operation and created functions to reconstruct XOR operations
from CNF clauses [120]. Since many ciphers are described using XOR functions,
this provides a more natural and compact representation. Using a number of stream
ciphers for testing, Soos showed that describing the ciphers using XOR operations
allows a SAT solver to solve systems more quickly. The exponential expansion of
XOR clauses into CNF is described in Appendix A.3.

2.8.4   Algebraic Side-Channel Analysis.   To break a cipher using algebraic
cryptanalysis, in addition to describing the cipher as a system of multivariate polyno-
mial equations, the system must be solvable. Computational complexity prevents the
system of equations from being solved using only one plaintext-ciphertext pair [36].
Information about intermediate values determined using side-channel analysis can
reduce the complexity of the system.

Combining algebraic cryptanalysis and side-channel analysis has a synergistic
effect, making the system easier to solve while simultaneously reducing the number

of side-channel measurements required to perform the attack [101]. On vulnerable implementations, with enough measurements, the entire key can be recovered.

2.8.4.1 Attacks with SAT Solver Stage. Renauld and Standaert combined side-channel and cryptanalysis techniques into a two-stage attack on an implementation of the PRESENT block cipher [101]. The first stage uses a template attack to recover as many intermediate values from a single power consumption trace as possible. Since each intermediate value provides partial information, an adversary should determine as many intermediate values as possible. Ideally, intermediate values are the output of a surjective function, such as an S-Box, so they reveal information about previous values.

In the second phase, the adversary uses the intermediate values recovered using side-channel analysis to write the block cipher as a system of quadratic (or cubic) equations, including the previously defined surjective functions with outputs recovered using side-channel analysis. The block cipher can be represented as a boolean satisfiability problem and the intermediate values can be fed into an SAT solver to recover the key [101]. In an unknown-plaintext/ciphertext scenario, they recovered the PRESENT block cipher key after observing only one encryption.

Renauld et al. combine algebraic cryptanalysis with a HW-based template attack to exploit an implementation of AES on a 8-bit PIC microcontroller in [102]. They demonstrate that an AES key can be recovered using only HW information and show that not knowing the plaintext or ciphertext does not significantly reduce the probability of determining the correct key. The system of equations includes multiple intermediate values for the AddRoundKey and SubBytes as well as all of the intermediate calculations to perform the MixColumn operation. This provided a total of 788 possible HWs over 10 rounds. Renauld et al. demonstrated that knowing all of the HWs for three consecutive AES-128 rounds allows for key recovery in 95% of trials, using simulated data. When the same number of HWs are known for random

intermediate values the success rate is dramatically reduced. The sensitivity of the SAT solver to incorrect information is highlighted as one of the weaknesses of using a SAT solver. Although they state that up to 200 HWs can be extracted correctly from a single trace, no algebraic side-channel attacks are performed with real data [102].

2.8.4.2 Pseudo-Boolean Optimizers.   To compensate for the noise in side-channel analysis measurements, DPA calculates statistics based on multiple traces. Measurement noise has multiple sources including electronic noise, quantization noise, and switching noise. To get highly reliable side-channel information, a large number of traces must be collected and analyzed. Allowing for errors in the side-channel data can reduce the number of traces that need to be collected. The process described for algebraic side-channel analysis proposed by Renault et al. in [102] is extremely sensitive to noise [90].

Renault et al. proposed using algebraic methods during the key recovery phase to convert the key recovery problem into a Boolean SAT problem and using a SAT solver to recover the key. However, testing showed the SAT solver could only find a solution when the error rate was very low (well under 1%) [102]. Oren et al. propose a new method called Tolerant Algebraic Side-Channel Analysis (TASCA) in which the side-channel analysis problem is transformed into a pseudo-Boolean optimization problem (PBOPT) [90], with the main benefit being higher tolerance for errors.

The SAT representation does not offer an efficient method for handling errors in the side-channel measurements or analysis. If the SAT representation is given enough errorless side-channel information the SAT solver will be able to recover the key successfully. However, even a single error in the side-channel measurement can result in unsatisfiability, or a wrong key.

By writing the system of equations in the more flexible language of non-linear pseudo-Boolean optimization, additional variables can represent errors. Unlike a SAT solver which attempts to find a single solution that satisfies a system of equa-

tions or determine that the system is not satisfiable, PBOPT algorithms find the solution which minimizes an objective function. Oren et al. demonstrated successful single-trace attacks against the Keeloq block cipher with intermediate value error rates of 10-20% [90] which was significantly more tolerant to errors than using a SAT solver directly [102].

Recently, the TASCA technique was enhanced by specifying a goal function to indicate which key byte guesses are more probable than others using the posterior probabilities of each key-byte guess and tested against an 8-bit implementation of AES-128. This technique is called probabilistic TASCA [89]. Probabilistic TASCA has a higher correct key identification rate than standard TASCA, and reduced solve times. While technique is provided the same fixed number of byte or HW value guesses per targeted intermediate value, only probabilistic TASCA incorporates data from the posterior probability calculated in the attack phase of a template attack. It is important to note that experiments in [89, 102] were performed on simulated data.

Cold boot attacks are a related research field in which cryptographic keys are reconstructed from partial information. The similarities and differences of cold boot attacks and side-channel analysis are explored in the following section.

2.8.5   *Related Key Recovery Techniques.*   It is generally believed that dynamic random access memory (DRAM) loses its contents immediately when it loses power, but it has been found that the loss of contents is in fact gradual. Although typically DRAMs will lose their contents gradually over a period of seconds at room temperature, if the chips are kept at low temperatures the data will persist for minutes or even hours [55].

Cold boot attacks attempt to extract a cryptographic key stored in a computers memory [55]. Since the memory decays gradually, some of the bits will have already decayed to their *ground states*. Memory bits not in their ground state have a high

probability of being correct and some researchers assume they are. Using a direct approach, an AES key can be extracted from memory and candidate keys can be generated in order of HD from the recovered key. If few bits have flipped, the true key can be recovered quickly, but the search time grows exponentially as the number of bit flips increases.

To increase efficiency, encryption software may pre-calculate and store the AES key schedule in memory. Knowing how the key schedule is constructed from the key allows an entire decayed key schedule to be used to reconstruct the key [55]. From the decayed key schedule, small sets of key bytes can be recovered and key candidates identified. Rather than being an algebraic attack, this attack is based on probability. The key candidates can be checked against the decayed key schedule to determine which candidate most likely produced the recovered key schedule.

The use of a SAT solver for cold boot key recovery is proposed by Kamal and Youssef in [62]. Since bits in memory are expected to decay to their ground state, this approach assumes that any bit in the recovered key and key schedule not in its ground state is correct and the remaining bits are discarded. Due to the amount of redundant information in the AES key schedule, the encryption key can still be recovered by writing and solving a set of SAT clauses.

Albrecht and Cid extend cold boot key recovery attacks to additional ciphers with more complex key schedules and use integer programming techniques to solve sets of non-linear equations with noise to determine the most likely key [6]. This effort included describing the AES key schedule as a system of polynomial equations. Key recovery is written as a Polynomial System Solving (PoSSo) problem. The goal is to find a solution to the system of polynomials over some field. In the presence of errors, Max-PoSSo can find a solution that maximizes the number of polynomials equal to 0. The Max-PoSSo problem is analogous to the Max-SAT problem which, rather than find a solution that satisfies all clauses, tries to find the maximum number of clauses that can be satisfied.

Cold boot attacks are similar to side-channel cryptanalysis because both reconstruct keys from unreliable data using the algebraic structure of the cipher to relate recovered values to the key. Cold boot attacks assume errors are asymmetric, giving the attacker a simple way to identify intermediate values with a high probability of being correct.

2.9 Summary

Algebraic cryptanalysis and side-channel analysis are two techniques for determining the relationship between the plaintext, key and ciphertext in a cryptographic operation. Algebraic cryptanalysis breaks ciphers by solving systems of multivariate polynomials created from the intrinsic algebraic structure of the cipher [11].

While describing the cipher as a system of polynomials does not reduce the complexity of solving for the key given only the plaintext and ciphertext, it allows for intermediate values to be introduced [27]. SAT solvers are an effective way to solve the system of polynomials because they allow for any known intermediate value to be incorporated and can quickly recover the key if enough intermediate values are known. Combining SAT solvers with side-channel analysis enables powerful attacks that often require less traces than side-channel attacks that do not incorporate algebraic cryptanalysis.

The side-channels from cryptographic devices leak information about the operations performed and the data being manipulated by the device [68]. The emissions can be used to determine the intermediate values of the cipher and ultimately the key used for an encryption or decryption operations. The most powerful side-channel analysis techniques require a very powerful attacker. Many attacks require the attacker to have complete control over the device, allowing the attacker to add a trigger signal, place a probe as close as possible to the target device, and be able to performed encryption or decryption operations at will [73]. For template attacks, it is

assumed the leakage from two identical devices is identical and the noise present in the side-channel emission is key dependent and precisely modeled [24].

The goal of this research is to identify ways to reduce assumptions that must be made by an attacker, making side-channel attacks more viable in an operational setting. After common methodology is introduced in Chapter 3, the following four chapters introduce novel ways to eliminate some of these assumptions to making side-channel attacks more effective for less powerful attackers.

3. Methodology

This chapter describes the methodology used to collect and process the electromagnetic (EM) emissions from target encryption devices. The three contribution areas of this dissertation, an algebraic side-channel Key Schedule Redundancy Attack (KSRA), cross-device template attacks, and using software-defined radios (SDRs) to perform differential Side-Channel Analysis (SCA) are all based on the standard Correlation-based EM Analysis (CEMA) and template attack methodology. All data collection and the common methodology shared by these techniques are explained in this chapter. The unique and novel enhancements that allow these techniques to be used by a less powerful attacker are presented in Chapters 4, 5, 6 and 7.

The process used for collecting the EM emissions from each of the encryption devices used in the attacks are described in Section 3.1. The devices used for side-channel attacks are introduced in Section 3.2, followed by signal processing techniques in Section 3.3. The steps required to perform CEMA for unknown-key and known-key analysis are found in Section 3.4. Section 3.6 outlines the steps required to perform a template attack. Finally, how the system of equations and SAT solver incorporate side-channel data to perform algebraic cryptanalysis is discussed in Section 3.7.

3.1 Data Collection

The EM side-channel of the microprocessors are collected using the hardware from AFIT's commercial Riscure Inspector side-channel collection and analysis system. However, custom software is used to gain more control of the process. A Riscure low-sensitivity probe with a 1 GHz bandwidth is used for all collections. For non-SDR collections the probe is connected to a LeCroy WaveMaster 804Zi oscilloscope through an anti-aliasing filter. The oscilloscope has a 4 GHz bandwidth, a maximum sampling rate of 40GSa/sec on 4 channels and memory to store 128 Mpts

Figure 3.1   The Riscure Inspector Side-Channel Test Tool [105].

per channel. One of the novel approaches in this research is to use SDRs to collect EM emissions. This methodology is found in Section 7.5.

A low-sensitivity Riscure probe is mounted on a computer-controlled motorized XYZ table that, with proper calibration, allows repeatable probe placement. Traces are collected and analyzed to determine the best location to place the probe. Spectral intensity and the results of correlation analysis are used in this research. For the highest quality collections, unless otherwise stated, the probe is placed as close as possible without touching the package of the device. To prevent aliasing, an analog low-pass filter with a cutoff frequency of approximately 36% of the sampling frequency is placed in-line with the probe. For collections for sampling frequency $f_s = 2.5$ GSa/sec a Mini-Circuits BLP-1000+ (-3 dB at 900 MHz) low-pass filter is used. For collections for sampling frequency $f_s = 250$ MSa/sec a Mini-Circuits BLP-90+ (-3 dB at 90 MHz) low-pass filter is used.

For all collections made with the oscilloscope the encryption device is programmed to produce a signal at the start of each encryption operation on a general purpose I/O pin. This signal is used to trigger the oscilloscope, resulting in collected traces that are well aligned. Where indicated, correlation-based alignment is used to correct any difference in the time at which the start of the encryption operation occurs in each trace.
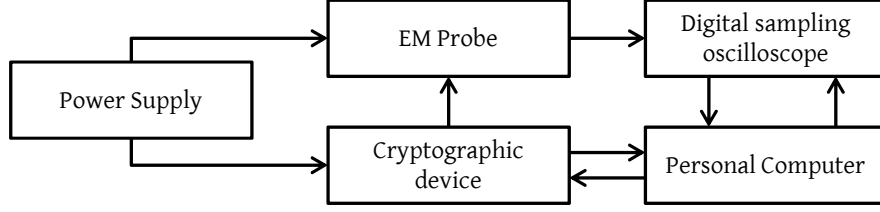
Figure 3.2   Block diagram of a typical measurement setup for collecting side-channel emissions [73].

The collection of EM traces is automated to a great extent by controlling the process with a PC. A block diagram of the process of collecting side-channel emissions with an oscilloscope is shown in Figure 3.2. The XY position of the probe above the device is controlled by the computer. For safety, the height above the device is set manually. The plaintexts and keys used by the cryptographic device are send to the device by the computer using an RS-232 serial interface. The encryption operation is initiated by the PC and the output of the encryption operation is returned to the computer to verify correct encryption operation. The oscilloscope, configured and controlled through a PC interface, collects the EM emissions for the duration and sampling rate set by the PC each time the trigger is asserted. The plaintext, key, ciphers and optionally the intermediate values calculated during each encryption operations are stored with the saved trace.

Traces collected from a device being attacked, with a fixed key are referred to as *test* traces. The key is randomly generated and stored only for the purpose of determining if the correct key was recovered using SCA. Plaintexts are generated randomly and encrypted until the desired number of traces are collected. If traces are collected at multiple probe locations for the purpose of evaluating the effect of probe placement, the random seed is reset for each location causing the same set of plaintexts being generated. For the *training* traces collected for template attacks, a both they key and ciphertext are random for each encryption operation.

Table 3.1   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

3.2   Targeted Devices

This research attacks two types of microcontrollers. The 16-bit PIC[1] micro-controllers are representative of low cost microcontroller used in various embedded applications. The 32-bit ARM[2] Cortex-M4F high performance, lower power device intended for applications such as industrial automation, stepped motor and motion control [128].

3.2.1   PIC Microcontrollers.   The PIC24 is a 16-bit general purpose microcontroller. The collection of PIC microcontrollers tested come from 4 different part numbers. There are 10 unique devices from each part number, for a total of $N_D = 40$ devices. The full part numbers and nomenclature used to refer to each individual device is shown in Table 3.1. These part numbers were selected because they have similar device architectures. The 10 chips from each part number were all manufactured in the same lot.

Although all the PIC devices have the same basic architecture, Part A devices have several on-board peripherals that are not included in the other three. Parts B, C and D devices all have identical architectures with the exception of the amount of on-board flash Random Access Memory (RAM) which is 32, 48, and 64 KB of RAM respectively. Part A devices have 64 KB of RAM. Individual devices are reference

---

[1]The original PIC microcontroller was designed as a Peripheral Interface Control. The name was retained despite the future devices being used for other applications.

[2]The term ARM refers to a family of RISC-based microprocessor architecture design licensed by British company ARM holdings.

by an alphanumerical device number that includes part type and chip number, i.e., $A1, A2, ..., D9, D10$, as shown in Table 3.1.

The chips were fabricated using an unspecified 180 nm process. Since all 10 chips for each part number were produced in the same lot, they contain identical architectural features. Uncontrolled manufacturing variations in the die fabrication and packaging process are believed to be the only physical differences between devices with the same part number.

AES-128 is implemented using separate SubBytes, ShiftRows, and MixColumn functions as specified in the AES standard [88]. The C++ used to program the UART interface and assembly code used to program the AES operation are identical for each device. The compiled versions may vary slightly for each part number due to part specific header files.

3.2.1.1 Data Collection. The collection process was designed to make measurements as repeatable as possible. A single evaluation board is used to collect side-channel emissions from all $N_D = 40$ devices. Each device was programmed to respond to commands over a RS-232 serial interface. The evaluation board was modified with a Zero Insertion Force socket (ZIF) to allow the devices to be easily swapped out. To improve trace alignment, a trigger signal was programmed to go high immediately before the encryption operation started and to go low immediately after completion.

To find the best probe position, an XY scan is performed with the near-field probe as close to a reference device as possible without touching the packaging of the microcontroller. The point above the device yielding highest spectral intensity is chosen for collections. Lateral movement of the circuit board is minimized between signal collections using a custom made jig that fixed the microprocessor position relative to the probe. A DC power supply (Agilent E3631A) minimizes variation in the supply voltage.

Training data for each of the $N_D = 40$ devices is generated by performing 5,000 AES-128 encryption operations using randomly chosen plaintexts and keys. Similarly, test traces for each device are collected while performing 500 encryption operations using a fixed key and random plaintexts.

Traces are time aligned by shifting them based on the location of highest cross-correlation of a trace segment with a segment from the reference trace [104]. Traces are collected at a sampling rate of 2.5 GSa/sec with a 1 GHz low-pass anti-aliasing filter inserted between the probe and the oscilloscope. Since the target device operates at $f_{sys} = 29.48$ MHz, the traces were down-sampled to make the trace sets easier to process. Different down sampling techniques were used for the KSRA and cross-device template attacks.

For the KSRA in Chapter 4, the collected traces were down-sampled by averaging groups of adjacent time samples to an effective sampling rate of 200 MSa/sec. This method was performed by the Riscure Inspector software [104]. Although this technique is computationally simple, it is not a common down-sampling technique used in other types of signal processing. A more common technique was used the cross-device template attacks in Chapter 5 where the collected data is down-sampled to 250 MSa/sec using decimation. Decimation is discussed in Section 3.3.2.

3.2.2 ARM Cortex-M4F. The second target device is a 32-bit Stellaris LM4F232 ARM Cortex-M4F-based microcontroller. The Stellaris LM4F232 USB +CAN evaluation kit features a Stellaris microcontroller in a 144-LQFP package, a color organic light emitting diode display, Universal Serial Bus (USB) 2.0, multiple Universal Asynchronous Receiver/Transmitters (UARTs), as well as other network and interface standards. The Stellaris LM4F232H5QD microcontroller contains a number of analog features including two 12-bit analog-to-digital converters (ADCs), three analog comparators, two temperature sensors and a three-axis accelerometer.

The microcontroller is fabricated using a 65 nm process and operates at up to 80 MHz.

The microprocessor is programmed to communicate with the PC using a RS-232 serial interface, allowing the PC to set the key, encrypt plaintext and retrieve the ciphertext using the device on demand. The key schedule is generated when the key is set and not generated as part of the encryption operation. For testing collections with and without a trigger signals, whether a trigger is asserted during an encryption operation is determined by which version of the encryption command is sent to the microprocessor by the PC. For compatibility with an existing UART interface, the system clock is set to $f_{sys} = 50$ MHz.

The ARM Cortex-M4F implements AES-128 in Electronic Codebook (ECB) mode using the T-box method as described in Section 2.2.2 and [38]. Internal read-only memory stores forward S-box, reverse S-box, forward polynomial and reverse polynomial tables, however no countermeasures are implemented on the device. The four T-boxes are generated and stored in random access memory on initialization. Two devices with the same part number are used for testing. Both ARM devices are used to test cross-device attacks in Chapter 6. The devices are referred to as *ARM1* and *ARM2*. ARM1 is used for the SDR testing in Chapter 7.

3.2.2.1 Data Collection.   A custom jig was fabricated to allow for repeatable placement of each ARM development board on the XYZ stage. Custom software was written to control the XYZ stage, allowing for calibration and repeatable probe placement. Since the LM4F232H5QD microcontroller is placed at a 45 degree angle on the development board, the board sits in the jig at a 45 degree angle to allow for more efficient XY scans. Note that in the board orientation seen in Figure 3.3, the device package is upside-down. To compensate for manufacturing variations in the boards, the coordinates of the upper left-hand corner and lower right-hand, which specify the bounds of the XY scan, are adjusted for each board.
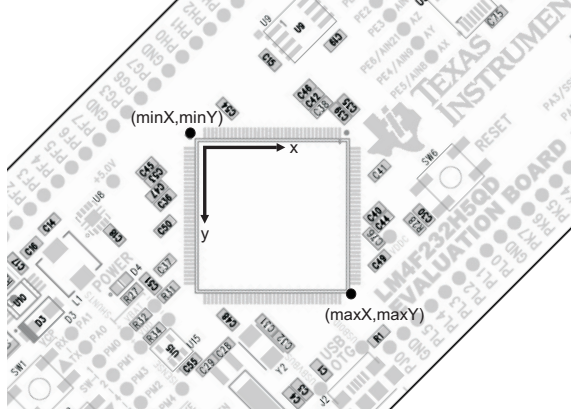
Figure 3.3   Orientation of the ARM development board in the jig [129].

For all ARM collections, locations are based on a $25 \times 25 = 625$ location scanning grid. Locations are numbered left-to-right, then top-to-bottom as the microcontroller is oriented in Figure 3.3. For example, location 1 is the upper left-hand corner, location 25 is in the upper right-hand corner and location 625 is in the bottom right-hand corner of the device.

Depending on the application, up to $n_t = 2500$ test traces are collected at each location. A single trace is used to find the spectral intensity of the EM emission collect at each location above the device package. The spectral intensity is calculated by finding the power spectral density (PSD) of the trace and finding the maximum power over a range of frequencies. Since the goal of the scan is the find the location with this highest spectral intensity, the PSD values are normalized across all locations.

Collecting $n_t = 2500$ test traces allows CEMA attacks to be performed, and for the attack phase of a template attack to be conducted. Locations selected based on maximum power spectral density and CEMA attack performance are used for collecting template attack training data. The results of the maximum PSD plots and CEMA attacks for the ARM devices are discussed in Sections 6.3.1 and 7.4.1.

Test traces are collected at 2.5 GSa/sec for ARM1 for the baseline test in Chapter 7. However, due to the large amount of training and test traces that must be

collected for the cross-device template attacks, the sampling rate used in Chapter 5 is reduced to 250 MSa/sec.

3.3   Signal Processing Techniques

This research uses a number of pre-processing techniques to improve the effectiveness of side-channel attacks by making it easier to extract information from the collected side-channel. Alignment, decimation and filtering are the primary techniques used in this research. Software demodulation was evaluated, but for the devices studied it did not improved the effectiveness of the attack. Since these techniques may be applied to both CEMA-based attacks and template attacks, they are outlined here before these techniques are introduced in Sections 3.4 and 3.6 respectively.

3.3.1   *Filtering.*   Although template attacks model the noise generated by the target device due to non-data-dependent operations [24], the classification results may be better if some of this noise is reduced through filtering, especially if this noise is not present in all traces. For more complex devices, such as the ARM Cortex-M4F, where components not used to perform the encryption operation are housed within the same package, signals unrelated to the encryption operation may be present. Although filtering has been shown to dramatically reduce the number of traces required for CEMA [14], no research has been found indicating that intelligently filtering traces used in a template attack will reduced the number of traces required. Ideally, signals unrelated to the encryption operation will be filtered without attenuating frequencies containing useful information.

Digital filtering is used two ways in this research: to isolate and eliminate frequency components of the collected EM emissions. Bandpass filters are used to isolate the frequency components to determine if the frequencies retained contain information that can be used to successfully attack a device. To ensure low attenuation
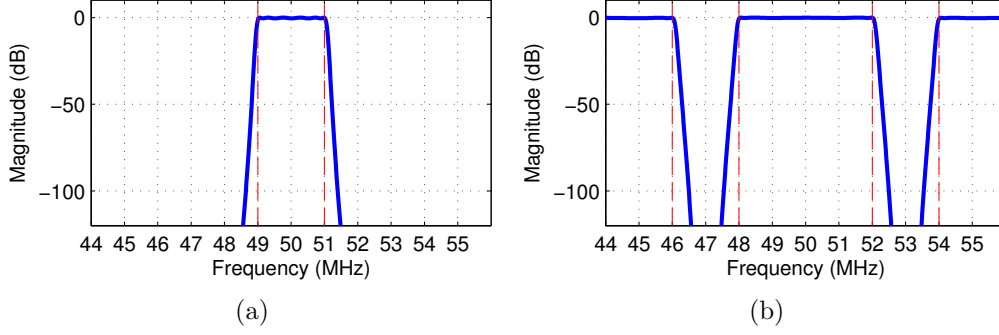
Figure 3.4   (a) Magnitude of the impulse response for a bandpass filter centered at $f_c = 50$ MHz with a bandwidth $W_{BW} = 2$ MHz in the frequency domain. (b) Magnitude of the of impulse response for two notch filters in series. The specified cutoff frequencies are shown as dashed lines.

at the cutoff frequencies of the bandpass filters, sixth-order Chebyshev Type I filters are implemented with a passband ripple of $r = 0.1$ dB. The term *frequency interval* refers to the passband of a bandpass filter. The magnitude of the impulse response in the frequency domain for a bandpass filter with $f_c = 50$ MHz and $W_{BW} = 2$ MHz is shown in Figure 3.4(a).

Notch filters are used to eliminate frequencies that are believe to interfere with the side-channel attack. If multiple frequencies are identified, the traces are filtered with a series of notch filters. Each filter is a twelfth-order Chebyshev Type I bandstop filter with a stopband between the specified frequencies. Since the filters are intended to be used in series a passband ripple of $r = 0.1$ dB is used as a design parameter to ensure low attenuation outside of the stopband. The magnitude of the impulse response in the frequency domain for two stopband filters in series is shown in Figure 3.4(b). The stopband for filter 1 is 46 MHz $< f <$ 48 MHz. The stopband for filter 2 is 52 MHz $< f <$ 54 MHz. At the cutoff frequencies, the magnitude of the filter's response is -0.1 dB due to the desired passband ripple. Although it may be more straightforward to use a filter design that allows for the desired attenuation to be reached at the cutoff frequency, having low attenuation and ripple for frequencies outside the stopband is highly desirable for notch filters used in series.

The process for identifying the cutoff frequencies of filters are discussed when these filtering techniques are applied.

3.3.2   Decimation.   Decimation is used to down-sample traces collected at a higher sampling frequency to a lower effective sampling frequency. Traces collected at $f_s = 2.5$ GSa/sec are downsampled to $f_s^D = 250$ MSa/sec using the following method. First the traces are filtered with an eighth-order low-pass Chebyshev Type I filter having a cut-off frequency of 100 MHz ($0.8 \times f_s/2$), and then the filtered traces are properly decimated by 10 (every 10th sample retained and all others discarded). Since the 8-bit traces are converted to double precision before filtering and not converted back to 8-bit precision after decimating, the decimated traces are higher quality[3] than traces collected at 250 MSa/sec directly using the 8-bit oscilloscope. Decimated traces are denoted f_s^D to indicate they were not directly sampled at that rate.

3.3.3   Alignment.   Traces collected with the oscilloscope are aligned using the Riscure Inspector software *static alignment* module. Static alignment shifts all samples in a trace by the same offset to align the trace to a reference trace [104]. The module allows the user to quickly select a portion of the trace that is visually distinctive and it expected to be present in every trace. The shift value is determine by correlating the selected part of the reference traces with the trace being aligned. The relative position of the best correlation is used as the shift value. Traces can also be discarded if the maximum correlation between the trace being aligned and the selected portion of the reference traces does not meet the desired minimum correlation level. Traces collected using a SDR are also aligned using correlation. Additional details on how SDR traces are identified and aligned is found in Section 7.2.1.

---

[3]In testing, using decimated traces resulted in more effective SCA attacks.
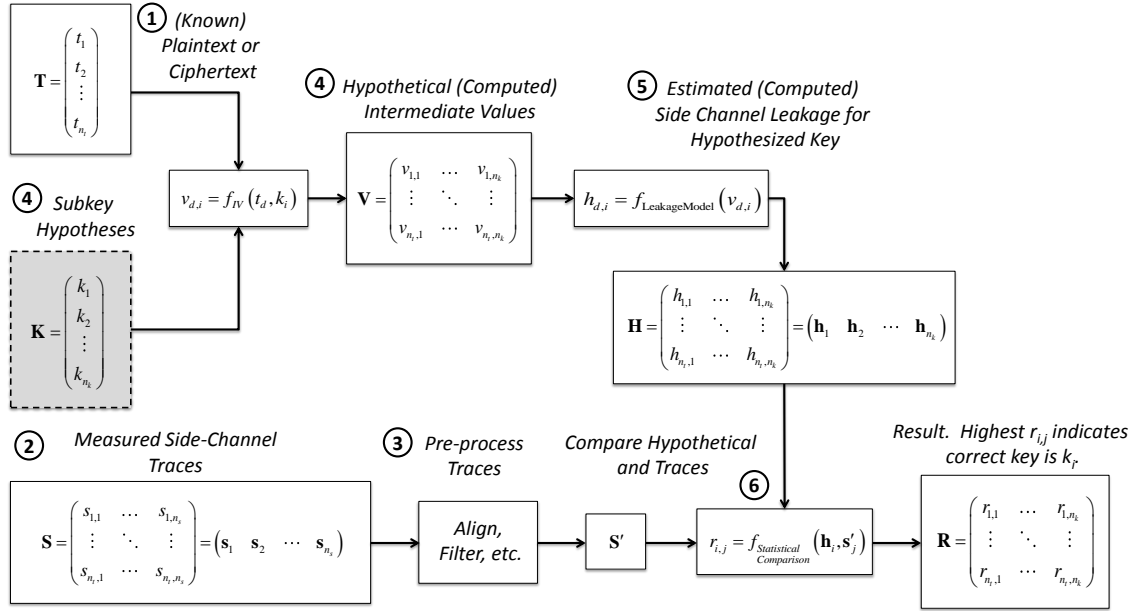
Figure 3.5   Differential Side-Channel Analysis Process [30, 73].

3.4   Correlation-Based Electromagnetic Analysis

The methodology for correlation-based EM analysis (CEMA) and template attacks are presented in Sections 3.4.1 and 3.6 respectively. A CEMA-based filtering processing to identify which frequencies leak information is explained in Section 3.5, but is important to understand CEMA first.

3.4.1   CEMA Attack Methodology.   The same general strategy is used in all differential SCA attacks. The step-by-step process is presented below and shown in Figure 3.5. Most of the following steps are outlined in [73] but an optional pre-processing step has been added.

Step 1: Choose an Intermediate Value to Attack. The first step in a differential attack is to choose an intermediate value calculated by the cryptographic device to attack. The intermediate value must be a function, $f(t, k)$, where k is

a small portion of the key and t is a non-constant data value. When attacking cryptographic devices, t is typically part of the plaintext or ciphertext.

Step 2: Measure the Side-Channel Emissions. The next step is to measure the side-channel emissions from the device while it encrypts or decrypts n_t different data blocks. For each operation, the attacker must control or at least observe the value of t. The values of known values can be written as a vector $\mathbf{t} = (t_1, ..., t_{n_t})'$. Power or EM traces corresponding to each known value are collected. The trace for data block t_d can be written as $\mathbf{s}'_d = (t_{d,1}...t_{d,n_s})$, where n_s denotes the length of the trace. The $n_t \times n_s$ matrix $\mathbf{S}$ contains n_t traces of length n_s samples.

Step 3: Pre-process Traces (Optional). Proper trace alignment is critical for differential attacks. If the traces are properly aligned, this step is optional. Alignment methods are discussed in Section 2.7.2. Data reduction, demodulation, and filtering techniques can also be applied. These methods are discussed in detail in Section 3.3.

Step 4: Calculate Hypothetical Intermediate Values. Since the intermediate values are a function of t and k, and values of t are known, hypothetical intermediate values can be calculated for each possible choice of k, the key hypothesis. A list of the n_k possible key hypotheses is written as $\mathbf{k} = (k_1, ..., k_{n_k})$. Calculating the hypothetical intermediate values for each of the n_t values of t and n_k values of k results in the matrix $\mathbf{V}$ of size $n_t \times n_k$. The individual elements of $\mathbf{V}$ can be calculated $v_{d,i} = f(t_d, k_i)$ where $d = 1, ..., n_t$ and $i = 1, ..., n_k$. The column i of $\mathbf{V}$ contains the intermediate results based on the key guess k_i.

Step 5: Calculate Hypothetical Leakage. Using an appropriate power consumption or EM emission model, the hypothetical intermediate values in $\mathbf{V}$ are used to calculate the hypothetical emission values in matrix $\mathbf{H}$. The hypothetical intermediate value $v_{d,i}$ is used to calculate the values of $h_{d,i}$. The most commonly used models are discussed in Section 2.3.4.

Step 6: Compare Hypothetical Leakage to Collected Traces. The final step compares the hypothetical emission values for each key guess in $\mathbf{H}$ to the collected traces in $\mathbf{S}$ using statistical methods. Each column $\mathbf{h}_i$ from the matrix $\mathbf{H}$ is compared with each column $\mathbf{s}_j$ from the matrix $\mathbf{S}$. It is assumed there is a statistical correlation between the hypothetical values for the correct key guess h_{k_i} and the collected traces s_j where j is the sample index corresponding to some unknown time t. Finding the interdependence reveals both the correct key value,k_i, and the time t at which the intermediate value is computed.

Various discriminators have been proposed for use in Step 6. The most commonly used is Pearson's correlation coefficient [22]. The elements $r_{i,j}$ of the result matrix ($\mathbf{R}$) are

$$r_{i,j} = \frac{\sum_{d=1}^{n_t}(h_{d,i} - \bar{\mathbf{h}}_i)(s_{d,j} - \bar{\mathbf{s}}_j)}{\sqrt{\sum_{d=1}^{n_t}(h_{d,i} - \bar{\mathbf{h}}_i)^2 \cdot \sum_{d=1}^{n_t}(s_{d,j} - \bar{\mathbf{s}}_j)^2}} \in \mathbb{R}, \tag{3.1}$$

where $i = 1, ..., n_k$ and $j = 1, ..., n_s$ and $\bar{\mathbf{h}}_i$ and $\bar{\mathbf{s}}_j$ denotes the means of the columns $\mathbf{h}_i$ and $\mathbf{s}_j$ [22].

When using Pearson's correlation coefficient as the discriminator, the correlation coefficient between each of the columns of hypothetical power consumption matrix $\mathbf{H}$ and each column of the recorded side-channel information matrix $\mathbf{S}$ is calculated and stored in the matrix $\mathbf{R}$. The correlation coefficient is an indication of the linear relationship between the observed side-channel and hypothetical leakage model [22]. The correlation matrix $\mathbf{R}$ can be visualized a number of different ways. Each row of $\mathbf{R}$ corresponds to one key guess. Plotting each of the rows yields a plot of the correlation coefficient vs. time for each key hypothesis. The most likely key hypothesis produces the highest correlation coefficient. The time at which the peak occurs indicates when an operation correlated to the model takes place in the

cryptographic operation. Figure 3.6(a) is a plot of the correlation coefficients for all key guesses vs. time. The correct key hypothesis is shown in black while other key hypotheses are shown in gray. A plot of how the correlation coefficient changes as the number of traces increases is shown in Figure 3.6(b). This graph is created by plotting the columns of $\mathbf{R}$ at a specific time correlated with an intermediate value in the encryption operation. Plots similar to Figure 3.6(b) are often used to determine the number of traces required before the most likely key hypothesis can be identified.

3.4.2   Example CEMA Attack.   The output of the AddRoundKey and SubBytes operations in the first round of AES-128 are common intermediate values to target in a CEMA attack [73]. Since it provided the best results against the PIC microprocessors, the output of the SubBytes operation in the first round of AES is the targeted intermediate value for the baseline template attacks in Chapter 4 and all attacks in Chapter 5.

Since these are byte-wise computations, each byte can be considered separately. Let t_d^n denote the value of byte n of the dth input plaintext (which corresponds to the dth trace) and let k^n denote byte n of the fixed secret key. When attacking the input to the SubBytes operation in the first round, the target intermediate values is calculated $l_{d,i}^n = t_d \oplus k_i$. When attacking the output of the SubBytes operations in the first round, the target intermediate values is calculated $v_{d,i}^n = \text{SubBytes}(t_d \oplus k_i)$. In both cases, $k_i \in \{0, ..., 255\}$ represents possible values for k^n. Assuming t_d^n is known and k^n is unknown, the CEMA attack identifies the most likely candidate k_i^n based on the collection side-channel observations $\mathbf{S}$ that correspond with the intermediate value $l_{d,i}^n$ or $v_{d,i}^n$ calculated for each $d \in \{0, ..., n_t\}$ trace in being processed on the target device.

The plots in Figure 3.6 are generated by performing an attack on byte 1 of the output of SubBytes from traces collected from PIC A01.
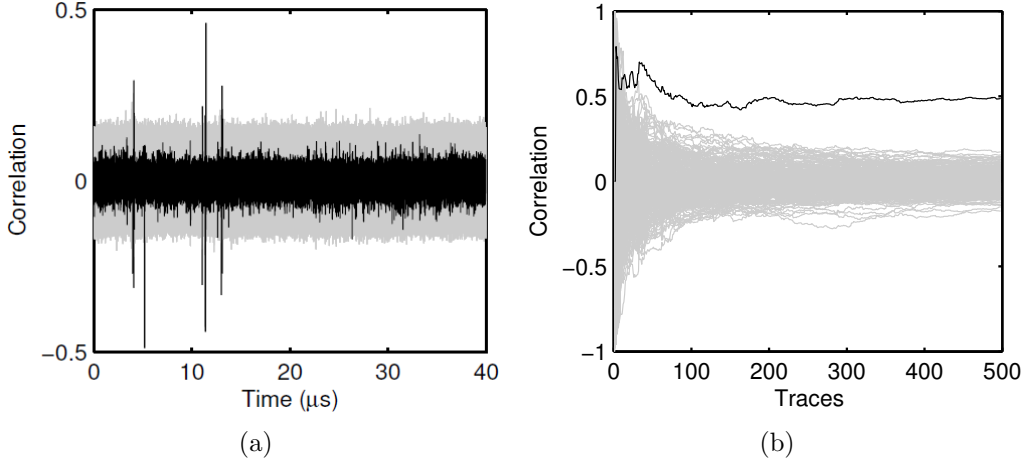
Figure 3.6 (a) All rows of $\mathbf{R}$. Correct key hypothesis is plotted in black, others in gray. (b) Columns of $\mathbf{R}$ at time t corresponding to an intermediate specific operation for different numbers of traces. Correct key hypothesis is plotted in black.

3.4.3   Known-Key Correlation Analysis.   Since the correct key-byte is expected to produce the highest correlation with the observed side-channel, if the correct key is known (3.1) simplifies to

$$r_j = \frac{\displaystyle\sum_{d=1}^{n_t}(h_d - \bar{\mathbf{h}})(s_{d,j} - \bar{\mathbf{s}}_j)}{\sqrt{\displaystyle\sum_{d=1}^{n_t}(h_d - \bar{\mathbf{h}})^2 \cdot \sum_{d=1}^{n_t}(s_{d,j} - \bar{\mathbf{s}}_j)^2}} \in \mathbb{R}, \tag{3.2}$$

where $j = 1, ..., n_s$ and $\bar{\mathbf{s}}_j$ denotes the mean of the column $\mathbf{s}_j$. Since the key is known, $\bar{\mathbf{h}}$ is the mean of hypothetical leakage $\mathbf{h}$ for the correct intermediate values. Since the targeted key byte is known for each trace, the correct hypothetical values can be calculated even if the key changes for each trace (as with training data).

The correlation vector $\mathbf{r}$ is used to identify time samples highly correlated with the leakage model as in the leakage mapping tecnique developed by Cobb et al. [30]. This is one heuristic method for identifying the points of interested used in template attacks.

72

3.4.4   Comparing Effectiveness of CEMA Attacks.   A number of methods have been developed to determine the effectiveness of CEMA attacks. The correlation coefficient $r_{i,j}$ defined in (3.1) is an indication of the linear relationship between the observed side-channel and hypothetical leakage model. For byte-wise attacks, each column of $\mathbf{R}$ corresponds to one key byte guess. Each row of $\mathbf{R}$ corresponds to a time sample. Although additional insight may be gained by examining the columns of $\mathbf{R}$ graphically, for this research the most likely key k_{max} is chosen according to

$$k_{max} = \arg\max_{i \in \{0,...,n_k-1\}} \left(\max_{j \in \{1,...,n_s\}} |r_{i,j}| \right). \tag{3.3}$$

To evaluate the effectiveness of a CEMA attack the maximum correlation coefficient, r_{max}, is compared with the next highest correlation coefficient, r_{next}. Let $r_{\max}$ be the maximum correlation coefficient where

$$r_{max} = \max_{i \in \{0,...,n_k-1\}} \left(\max_{j \in \{1,...,n_s\}} |r_{i,j}| \right). \tag{3.4}$$

The next highest correlation coefficient r_{next} is

$$r_{next} = \max_{\substack{i \neq k_{max} \\ i \in \{0,...,n_k-1\}}} \left(\max_{j \in \{1,...,n_s\}} |r_{i,j}| \right). \tag{3.5}$$

Comparing r_{max} to r_{next} is used in this research to identify which frequencies are the most important to the success of the CEMA attack.

The confidence intervals for each correlation coefficient can be calculated and compared as in [14]. To determine the confidence interval for a correlation coefficient the Fisher's transformation [46]

$$Z(r) = \frac{1}{2} \ln \frac{1+r}{1-r} = \operatorname{arctanh}(r), \tag{3.6}$$

is used.

73

To calculate the confidence interval for a sample correlation r, the upper and lower bounds are calculated $\xi_l = z_r - \frac{z_{1-\alpha/2}}{\sqrt{N_t-3}}$, $\xi_u = z_r + \frac{z_{1+\alpha/2}}{\sqrt{N_t-3}}$ where $z_r = \text{arctanh}\,(\text{r})$, and $z_{1\pm\alpha/2}$ is the standard normal cumulative distribution function evaluated at $1 - \alpha/2$ [14]. The lower and upper confidence interval bounds for a correlation coefficient r are

$$\varepsilon_l\,(r) = \tanh\left(\text{arctanh}\,(\text{r}) - \frac{z_{1-\alpha/2}}{\sqrt{N_t - 3}}\right), \text{and} \tag{3.7}$$

$$\varepsilon_u\,(r) = \tanh\left(\text{arctanh}\,(\text{r}) + \frac{z_{1-\alpha/2}}{\sqrt{N_t - 3}}\right). \tag{3.8}$$

3.5 *Identifying Information Leaking Frequencies*

Barenghi et al. propose a systematic way of determining the frequencies at which information is leaked from cryptographic devices [14]. They show the effectiveness of differential power analysis can be improved by isolating the frequencies that leak information using software filtering. The goal of Barenghi's attack was to determine the minimum number of filter traces required to perform a successful correlation attack with a given confidence level. The number of traces is found by repeatedly performing the CEMA attack while increasing the number of traces until $\varepsilon_l\,(r_{max}) \geq \varepsilon_u\,(r_{next})$ as calculated in (3.7) and (3.8). The algorithm tries to identify the smallest frequency interval that contains information by repeatedly dividing frequency intervals that yield successful attacks with the desired confidence in less than the maximum number of traces. The algorithm does not split up frequency intervals that were not successful with the maximum number of traces. Although this approach dramatically reduces the number of correlation attacks that must be performed, testing on the PIC microcontroller showed that it may fail to identify frequency intervals that carry information.
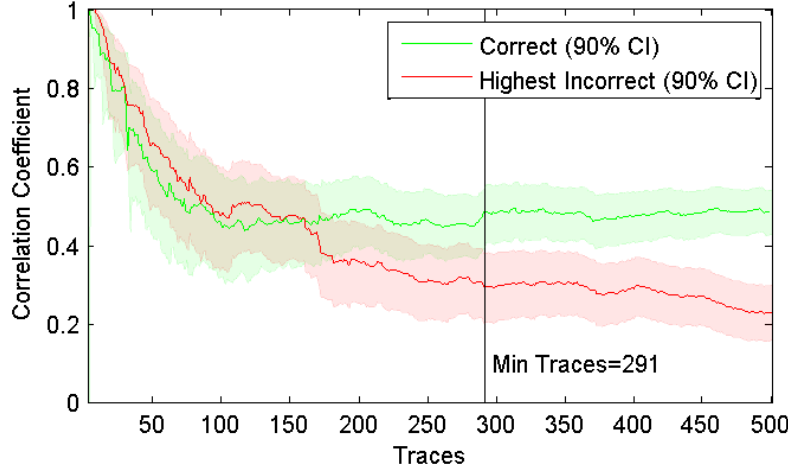
Figure 3.7   Identifying the number of traces needed for the de-
sired confidence in a CEMA attack

Figure 3.7 illustrates the result from this approach graphically. Using traces from PIC A01, decimated to $f_s^D = 250$ MSa/sec a CEMA attack is performed using up to 500 test traces in the order they were collected. The confidence intervals are shown as shaded regions. Although the correct value for key byte 1 is identified for $n_t \geq 160$ traces, the 90% confidence intervals overlap until $n_t \geq 291$ traces are used. The minimum number of traces required for the desired confidence is only one metric that can be used to compare attacks.

Two alternative approaches were developed as part of this research. The *frequency interval break down approach* and *overlapping frequency interval approach* are developed in Sections 3.5.1 and 3.5.2. Both correlation-based frequency-dependent leakage analysis techniques complement each other, one providing a variable frequency interval width and the other providing overlapping frequency intervals and easier comparison of multiple key bytes.

3.5.1 Frequency Interval Break Down Approach.   The frequency break down approach splits the frequency interval $[f_{\min}, f_{\max}]$ by a branching factor γ, λ times (levels). Initially $f_{\min} = 0$ and $f_{\max} = f_s/2$, where f_s is the sampling frequency, but these bounds can range between 0 and $f_s/2$ to focus on a frequency interval of
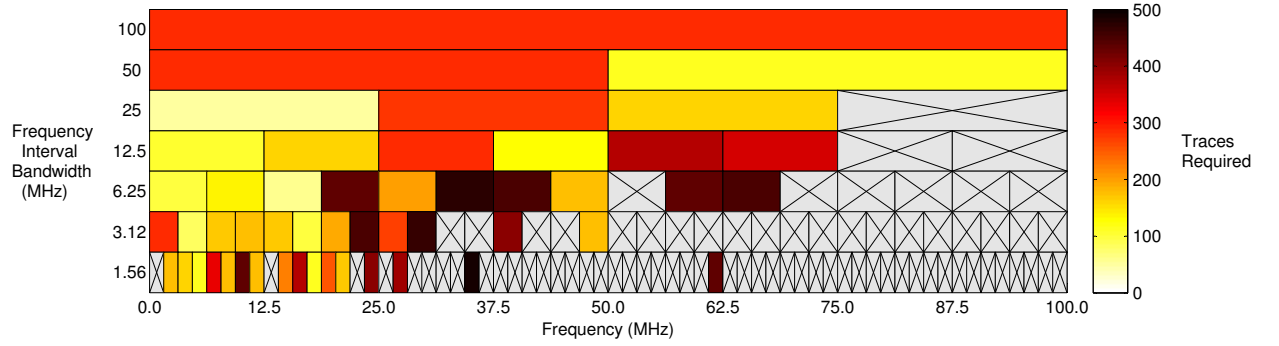
Figure 3.8   The number of traces needed for 90% confidence in a byte 1 CEMA attack using traces from PIC A01 filtered with a bandpass filter. Attacks that are not successful with the desired confidence are shown with an × through the rectangle representing the bandwidth.

interest. CEMA attacks are performed on all intervals regardless if the attack on the level above it was successful. Like, [14] the minimum number of traces required to achieve the desired confidence level is used as the metric to compare attacks.

Using traces collected from PIC A01 decimated to $f_s = 250$ MSa/sec, this method was applied and the results are shown in Figure 3.8. In this case, $f_{min} = 0$, $f_{max} = 100$ MHz, $\gamma = 2$ and the first $\lambda = 7$ levels are shown. Compared with the 291 traces needed in Figure 3.7, it is clear from Figure 3.8 that filtering can dramatically reduce the number of traces required to extract byte 1 with the desired level of confidence. Filtering reduced the number of traces required to achieve the designed confidence level to as few as 53 traces for the 0 MHz to 25 MHz frequency interval.

   3.5.2   *Overlapping Frequency Interval Approach.*   There are a number of problems with the frequency interval break down approach. Since multiple levels are represented in Figure 3.8, in order to compare all key-bytes 16 figures must be compared. While it is possible to extract a single level and compare the results for each byte on a single figure, the first algorithm also does not allow for overlap between adjacent frequency intervals. As a result, the frequencies near the filter cutoff are

76

attenuated. To address these problems, the overlapping frequency interval approach uses a single level for each bytes, allowing the filter bandwidth, f_{BW}, frequency interval, $[f_{\min}, f_{\max}]$, and percent overlap between adjacent filters to be specified. As many bandpass filters with bandwidth f_{BW} that fit in the frequency interval with desired the overlap of 50% are created and used to filter the collected traces.

Finding the minimum number of traces required for a CEMA attack to have a desired confidence is computationally expensive. To produce Figure 3.7, the CEMA attack is repeated 499 times[4] for each key byte. A binary search method was developed which reduces the number of CEMA attacks needed to determine the minimum number of traces, but this approach relies on the assumption that once $r_{max} \geq r_{next}$ with the desired confidence, the confidence will not go down when additional traces are added.

An alternative metric to compare the effectiveness of attacks is the confidence $r_{max} \geq r_{next}$ for a fixed number of traces. A hypothesis test is performed to determine if r_{max} is statistically different than r_{next} using Fisher's transformation and a Z-test [46]. The Z values for r_{max} and r_{next}, z_{max} and z_{next} respectively, are calculated using (3.6).

The Z-score is calculated by finding the difference between the z_{max} and z_{next} and dividing by the pooled standard error, $SE = \sqrt{2/(n_t - 3)}$, or

$$Z_{test} = \frac{z_{max} - z_{next}}{SE} = (z_{max} - z_{next})\sqrt{\frac{n_t - 3}{2}}. \tag{3.9}$$

---

[4]The attack cannot be performed with less than two traces

Rather than compare this test statistic with a cutoff, for example $Z_{crit} = 1.95$ for 95% confidence ($\alpha = 0.05$), the confidence p with which the null hypothesis can be rejected is found using the standard normal cumulative density function

$$p = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{Z_{test}} e^{\frac{-t^2}{2}} dt. \tag{3.10}$$

This approach is used to compare the frequencies at which key byte information leaks for the A01 PIC microcontroller in Figure 3.9. A total of 99 overlapping filters are created for $f_{min} = 0$, $f_{max} = 100$ MHz and $f_{BW} = 2$ MHz with 50% overlap. Each row represents a different key byte. To make the differences between key bytes easier to see, only $n_t = 250$ traces are used. Since this is a relatively small number of traces, it is possible that differences between key bytes are due to the distribution of the plaintext bytes processed, but the change in the confidence between frequency intervals is real since only f_c changes for each filter. The traces are filtered once for each frequency interval and used to attack each key byte. For this plot, since intervals overlap by 50% the width of the rectangle in Figure 3.9 representing each frequency interval does not reflect the true frequency interval bandwidth. The box is centered on the correct f_c.

Since it is possible for r_{max} to correspond with an incorrect key guess, an $\times$ is drawn through the box presenting a unsuccessful attack. As expected, attacks with lower calculated confidence are more likely to be incorrect. Since by definition $z_{max} \geq z_{next}$ according to (3.9), $Z_{test} \geq 0$ and $0.5 \leq p \leq 1$. Figure 3.9 shows that some key bytes values are leaked more for certain frequency intervals than others. Additionally, the value of key byte 4 does not appear to leak as well as the other key bytes for this device.
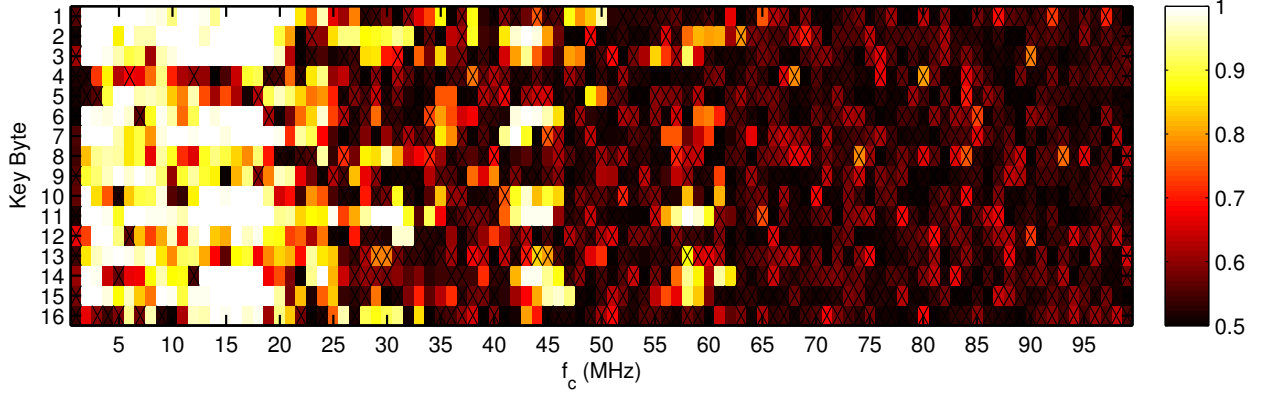
Figure 3.9   Colored boxes represent the confidence $r_{max} \geq r_{next}$ for each CEMA attack on PIC A01. Attacks are performed on each key byte using $n_t = 250$ filtered traces. The boxes represent bandpass filters with center frequency f_c, and bandwidth of $W_{BW} = 2$ MHz. An $\times$ indicates the CEMA attack produced an incorrect result.

3.6   Template Attacks

A profiling stage can be used to build multivariate statistical models of the device's side-channel leakage [24]. Incorporating a profiling stage allows template attacks to use all information present in a side-channel trace for classification, making them a strong attack even when only a single or few traces from the attacked device are available. Rather than try to eliminate or reduce noise, the noise present in the side-channel emission is assumed to be key dependent and precisely modeled. Templates are created during the training or profiling stage using training traces with known key and plaintext values. The templates are used during the attack or classification phase to determine the most likely class a collection of test traces belong to.

Since the introduction of template attacks in [24], a number of variations and improvements have been proposed as discussed in Section 2.4.6. However, all template attacks fundamentally contain the following steps.

Step 1: Data Collection. The training device and test device must both be observed performing encryption operations. It is assumed the attacker has complete control of the training device, can change the key and plaintext at will and can associate a collected trace with the plaintext and key used to produce it. While the key on the target device is always unknown, some attack scenarios assume a powerful attacker is able to match observed test traces with corresponding plaintext or ciphertext.

Step 2: Identify Classes. The goal of a template attack is to correctly determine to which category or *class* an observed trace (or set of traces) from a target device belongs. The number and definition of the classes is determined by the attack scenario and the type of information leaked from the target device. When attacking a microprocessor running AES, classes are commonly based on byte value (256 classes) [24,99], byte Hamming Weight (HW) (9 classes) or bit value (2 classes) [3].

Step 3: Feature Generation. Preprocessing techniques may be applied to the traces or the extracted samples before they are used for training or classification. Examples of preprocessing techniques include Principal Component Analysis, down-sampling or filtering.

Step 4: Feature Extraction. The samples in the collected or preprocessed traces that distinguish between classes are identified and extracted from each trace.

Step 5: Classifier Training. Using the known plaintexts and keys from the training phase, the attacker can estimate the class from which the observed training trace belongs. One template is created for each class using the extracted distinguishing features from the training traces belonging to that class (Ref. to Sec. 3.6.2).

Step 6: Classification. Using distinguishing features generated from one or more test traces, the classifier estimates the class to which the test traces most likely

belong. If the plaintexts or ciphertexts are known, hypothetical intermediate
values may be used in this process.

The remainder of this section provides additional information for the more
complicated template attack steps. Although distinguishing features are generated
and selected before constructing templates in an actual attack, it is more insightful to
discuss the rationale for feature selection after explaining the mechanics of template
attacks. Let vector $\mathbf{x}$ be the list of γ distinguishing features. Methods for selecting
distinguishing features are discussed in Section 3.6.5.

3.6.1 Class Identification. To evaluate the effectiveness of template attacks
on the target devices, a CEMA-based template attack is performed on the PIC and
ARM microcontrollers. Since the plaintext and keys are known during the training
phase, the actual values of the target intermediate value is known for each of the
training traces. For both microprocessors, attacks using $K = 256$ classes, one for each
byte possible byte value, were the most effective and yeilded the most information
to the attack. Different templates are constructed for each intermediate value byte
being attacked, but the number of classes is used in all attacks.

3.6.2 Classifier Training. The classifier is trained by constructing tem-
plates for each class. A fundamental assumption is that side-channel leakage for
a particular operation follows a multivariate Gaussian distribution. This assump-
tion has been shown to provide adequate performance in previous template attack
research [3, 9, 24, 73, 92]. The probability density function of a γ-dimensional multi-
variate normal distribution is

$$p\left(\mathbf{x}\right) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \hat{\mu}_{k_i})^T \hat{\mathbf{\Sigma}}_{k_i}^{-1}\left(\mathbf{x} - \hat{\mu}_{k_i}\right)\right)}{(2\pi)^{\gamma/2}\left|\hat{\mathbf{\Sigma}}_{k_i}\right|^{1/2}}, \tag{3.11}$$

where empirical mean vector $\hat{\boldsymbol{\mu}}_{k_i}$ and empirical noise covariance matrix $\hat{\boldsymbol{\Sigma}}_{k_i}$, form the template of class k_i. One template is constructed for each of the $K = 256$ possible byte values, $k_i \in \{0, ..., 255\}$. The estimates are constructed using n_{k_i} distinguishing feature vectors that belong to class k_i. Each distinguishing feature vector is represented as $\mathbf{x}_\delta$ where $\delta \in \{1, ..., n_{k_i}\}$.

The empirical mean vector, $\hat{\boldsymbol{\mu}}_{k_i}$, and the $\gamma \times \gamma$ empirical noise covariance matrix, $\hat{\boldsymbol{\Sigma}}_{k_i}$, are

$$\hat{\boldsymbol{\mu}}_{k_i} = \frac{1}{n_{k_i}} \sum_{\delta=1}^{n_{k_i}} \mathbf{x}_{k_i,\delta} \tag{3.12}$$

and,

$$\hat{\boldsymbol{\Sigma}}_{k_i} = \frac{1}{n_{k_i} - 1} \sum_{\delta=1}^{n_{k_i}} \left(\mathbf{x}_{k_i,\delta} - \hat{\boldsymbol{\mu}}_{k_i} \right) \left(\mathbf{x}_{k_i,\delta} - \hat{\boldsymbol{\mu}}_{k_i} \right)^T, \tag{3.13}$$

respectively.

3.6.3 Classifying Observed Traces.   Since each targeted intermediate values are dependent one byte of the key, there are $K = 256$ key-byte values, $k_i \in \{0, ..., 255\}$. For matrix $\mathbf{X}$, which contains the distinguishing features from one trace in each row, the probability that a key-byte guess is correct is [73]

$$p(k_i | \mathbf{X}) = \frac{\prod_{d=1}^{n_t} p\left(\mathbf{x}_d^T \middle| k_i \right) \cdot p(k_i)}{\sum_{l=0}^{K-1} \left(\prod_{d=1}^{n_t} p\left(\mathbf{x}_d^T \middle| k_l \right) \right) \cdot p(k_l)}, \tag{3.14}$$

where i is the index of the n_t test traces.

Since AES key-bytes are uniformly distributed, it is initially assumed that $p(k_l) = 1/256$ for all $l \in \{0, ..., 255\}$. The Bayesian classification process produces the probabilities $p(k_i | \mathbf{X})$ for all $i \in \{0, ..., 255\}$.

3.6.4 Class Selection. Classification is based on a maximum-likelihood (ML) decision rule. After $p(k_i | \mathbf{X})$ is calculated for each possible round key value, the most likely key-byte value is

$$\hat{k}_i = \arg\max_{k_i} p(k_i | \mathbf{X}).$$ (3.15)

3.6.5 Distinguishing Feature Selection. Device EM traces are typically collected at a very high sampling rate resulting in a large number of samples ($n_s > 10^4$). Building templates based on every sample is not feasible due to storage requirements of the covariance matrix and complexity of matrix inversion required to calculate the observation probability [99].

The processing time and complexity of constructing the templates can be reduced by identifying n out of n_s points that provide the most information to the template attack. Since these samples must allow classes to be distinguished from each other, they are referred to as *distinguishing features* herein. They are also referred to as *points of interest* in related literature.

Previous research has focused on improving how distinguishing features are generated and selected. A number of heuristic approaches have been proposed, including selecting samples with the largest difference between mean traces [24], or the point at which the largest variance between the mean traces (for each class) occurs. Benefits of pre-processing using a Fast Fourier Transform before selecting the samples, with the highest cumulative difference between pairs of mean traces, was evaluated in [99]. Requiring a minimum number of samples between successive selected time samples has also been proposed as a way to reduce the number of distinguishing features by reducing redundant information [99].

The known-key CEMA described in Section 3.4.3 was determined to be the most effective method for selecting distinguishing features for both target devices. Points with high correlation coefficients are dependent on the key and plaintext being

processed. The n points with the highest correlation can be selected, or all points with an minimum correlation coefficient could be used. To prevent points of interest from being chosen that are not significantly greater than the average correlation coefficient of the trace, only points with correlation coefficients greater than 5 times the average correlation coefficient are used as points of interest for the template attacks.

3.6.5.1 Principal Component Analysis (PCA). While heuristic methods for selecting distinguishing features have been effective, more systematic approaches have been developed. PCA can reduce the dimensionality of trace data using a linear transform that maximizes the inter-class variance between empirical mean traces $\{\hat{\boldsymbol{\mu}}_s\}_{s=1}^{K}$ for each class in the subspace [9]. To find this transform, PCA identifies the principal directions $\{\mathbf{w}_i\}_{i=1}^{n_p}$ such that $n_p \leq n_s$, which forms an orthonormal basis capturing the maximal variance of $\{\hat{\boldsymbol{\mu}}_s\}_{s=1}^{K}$ in an n_p-dimensional subspace. The principal directions are the eigenvectors $\mathbf{U}$ of the empirical covariance matrix

$$\bar{\mathbf{S}} = \frac{1}{\mathbf{K}} \sum\nolimits_{\mathbf{s=1}}^{\mathbf{K}} \left(\hat{\boldsymbol{\mu}}_{\mathbf{s}} - \bar{\boldsymbol{\mu}}\right)\left(\hat{\boldsymbol{\mu}}_{\mathbf{s}} - \bar{\boldsymbol{\mu}}\right)^{\mathrm{T}},\tag{3.16}$$

where $\bar{\boldsymbol{S}} = \mathbf{U}\boldsymbol{\Delta}\mathbf{U}^{\mathrm{T}}$, and $\bar{\boldsymbol{\mu}} = \frac{1}{K}\sum_{s=1}^{K}\hat{\boldsymbol{\mu}}_s$ is the average of the mean traces. The principal directions $\{\mathbf{w}_i\}_{i=1}^{n_p}$ are the columns of $\mathbf{U}$ that correspond to the n_p largest eigenvalues of $\boldsymbol{\Delta}$. The n_p-largest eigenvalues are denoted by the diagonal matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{n_p \times n_p}$ and the corresponding matrix of principal directions is denoted $\mathbf{W} \in \mathbb{R}^{\mathbf{n_s} \times \mathbf{n_P}}$. To perform an attack in the principal subspace, a Gaussian model after projection is assumed. The projected means $\{\boldsymbol{\nu}_s\}_{s=1}^{K}$ and projected covariance matrices $\{\boldsymbol{\Lambda}_s\}_{s=1}^{K}$ are given by

$$\boldsymbol{\nu}_k = \mathbf{W}^{\mathrm{T}}\hat{\mu}_k \tag{3.17}$$

and,

$$\boldsymbol{\Lambda}_k = \mathbf{W}^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}\mathbf{W}. \tag{3.18}$$

A collection of traces from the test device, $\mathbf{X}$, is classified by

$$\hat{k}_i = \arg\max_{k_i} p\left(k_i | \mathbf{W}^{\mathrm{T}}\mathbf{X}\right) \tag{3.19}$$

3.6.6   Comparing Effectiveness of Template Attacks.   In a template attack the most likely key-byte is selected using (3.15) or (3.19) if PCA is used. For this research, the template attack is only considered successful if the most likely key-byte value is the correct key-byte value. Since key-byte selection is dependent on the set of test traces $\mathbf{X}$ used, the attack can be repeated multiple times with different sets of test traces. Although mathematically when using a Bayesian classifier, the order in which the processed traces are added does not matter, in practice due to machine precision limitations the result can be different depending on the order the traces are added to the classifier. This may occur if the distributions of the training and test data differ significantly. To repeat a template attack multiple times, permutations of the collected traces are generated to specify multiple trace orders for a given test trace set. To determine the success rate for an attack, the classification phase of the template attack is performed using the first n_t traces from each permutation, and the percentage of attacks that yield the correct key byte is calculated for each byte individually or for all 16 key-bytes (global success rate).

Alternatively, a single set/order of traces can be used to compare the effectiveness of two template attacks. By plotting the posterior probabilities for call key-byte guesses found using (3.14) as in Figure 3.10, the number of traces at which an attack is successful for a given trace set can be compared [73]. The posterior probability for the correct key byte value is drawn with a thick green line, and the other key bytes are drawn using thin lines. Using training and test traces collected from ARM1, a template attack is performed using the 40 samples most highly correlated with the HW of output of SubBytes for Byte 1 as distinguishing features. Next, the attack is repeated using the 80 most highly correlated samples. The effectiveness of these

(a) 40 features	(b) 80 features
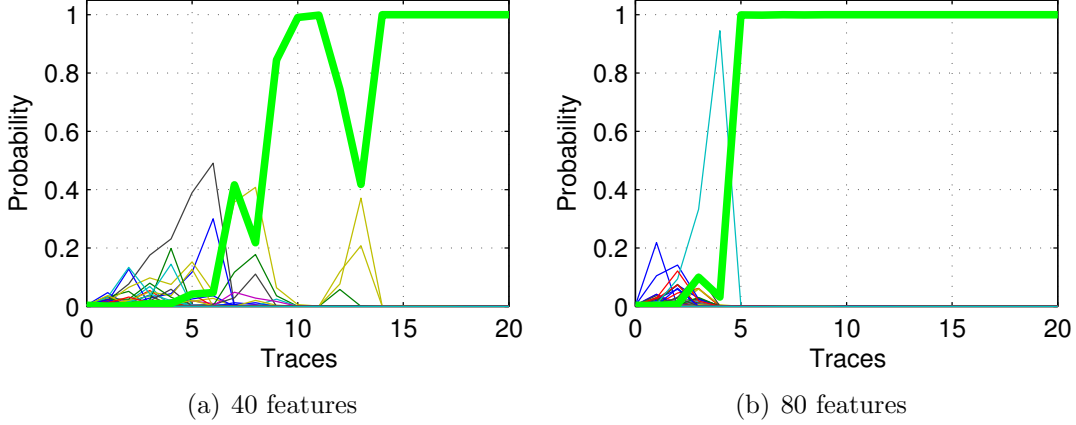
Figure 3.10   Plot of the posterior probabilities for all possible key-byte values using the indicated number of distinguishing features selected using known-key CEMA.

template attacks can be compared in Figure 3.10. Using the same trace set, the template attack using 80 distinguishing features identifies the correct key in fewer traces than the template attack using only 40 distinguishing features.

3.7 Algebraic Cryptanalysis

Algebraic cryptanalysis is used in this dissertation to combine the results from multiple template attacks. The AES SAT solver tool developed in this section uses a multivariate system of polynomials to describe the relationship between the plaintext, key, ciphertext and intermediate values of AES-128. The AES SAT solver tool allows any information known about the plaintext, ciphertext, and any intermediate value or a pair of intermediate values to be added as a constraint for the system of equations. For direct attacks on the key schedule, an AES Key Schedule SAT Solver tool is developed which only includes the equations for the key schedule.

3.7.1 Generating a System of Equations for AES-128.   The small scale variants of the AES are designed to incorporate the design features of AES and provide a framework for comparing cryptographic methods [25]. In addition to implement-

ing the small scale variants described in [25] and [27], a full scale implementation of AES-128 can be constructed using the SR polynomial generator [7] based in Sage mathematical software, a free software tool created with the goal of being an "open source alternative to Magma, Maple, Mathematica, and MATLAB" [124].

For a full AES-128 encryption operation this system includes 7288 polynomials in Algebraic Normal Form (ANF) and 4544 variables. The variables are specified at the *bit level* for the start of each AES-128 round, the output of each SubBytes inversion and each bit of each round key. Variables are added to the system to represent the plaintext and ciphertext. The polynomials define the relationship between each round, the inversion in SubBytes, and the key schedule. Thus the system of equations fully defines the relationship between each of the variables defined by AES-128. If enough of the variables are known, the key can be determined by finding a solution to the system of equations. The value of the variables may be known from plaintext, ciphertext, and intermediate values found using side-channel analysis. Methods for writing constraints for known values are explained in Section 3.7.2.1.

For the attack in Chapter 4, the key schedule is attacked directly by identifying possible values for multiple bytes of the key schedule. Since the entire key schedule depends only on the cipher key and not the plaintext, only the equations for the key schedule are included in the *AES Key Schedule SAT Solver tool*.

3.7.2 Converting to a SAT Problem. The polynomials produced by the SR polynomial generator are in ANF. Since the system of polynomials contains thousands of polynomials and variables, it is not practical to convert to CNF by hand. Mate Soos updated a converter originally written by Martin Albrecht which converts ANF to Conjunctive Normal Form (CNF) (`anf2cnf`) and can produce a DIMACS file[5] from equations written in ANF [119]. DIMACS is a standardized format for writing CNF equations for input to a SAT solver. The output of the SR

---

[5]DIMACS format is named for the Center for Discrete Mathematics & Theoretical Computer Science (DIMACS) at Rutgers University.

polynomial generator can be used with the `anf2cnf` converter to produce a DIMACS file which describes the relationship between the intermediate values, the plaintext, ciphertext and key for AES-128.

3.7.2.1 Introducing Side-Channel Information. SCA can identify properties of intermediate values or pairs of intermediate values. For example, if the exact value of a S-box input is determined using a template attack, the value can be added as constraints on bit values in the system of polynomials. The attack could also identify multiple possible byte-values.

Given enough traces, template attacks can usually identify the correct byte value. However, in many cases a template attack can identify multiple possible key-byte candidates using less traces than required to determine the exact key value. Rather than using the exact byte value as a constraint for the SAT solver tool, a list of possible bytes can be used. If the template attack determines the HW of an intermediate value, all byte values with that HW would be added as possible values. For example, if the HW of an intermediate value byte is equal to 1, the actual decimal representation of the 8-bit intermediate value byte may be 1, 2, 4, 8, 16, 32, 64 or 128.

A list of possible values for each byte can also be identified from the posterior probability for each key byte value calculated using 3.14. When using the maximum likelihood decision rule in (3.15), only the key byte value with the highest posterior probability is selected. If this fails to produce the correct key value, more than one possible key value can be allowed. A fixed number of possible values could be added to the system of equations for each attacked intermediate value or all key byte values a meet a minimum posterior probability threshold could be added. Methods for setting this threshold are explored in Chapter 4. If more than one byte value is allowed, additional information is needed to identify which byte is correct. This could

be information from more than one round of AES, or a known plaintext/ciphertext pair.

As the number of possible byte values increases, the probability a byte value combination will satisfy the system of equation increases and the SAT solver may identify the incorrect key. Increasing the number of byte candidates increases solve time, but because of the data redundancy in AES, the SAT solver tool may still identify the correct key. The data redundancy in AES allows the correct key to be recovered even if each byte value cannot be uniquely identified.

3.7.3 Solving the System of Equations. Once constraints on the byte values are identified, they must be written in terms of the bit variables used to construct the system of equations. Rather than add the constraints directly to the system of equations in ANF, the constraints are written separately. The benefit of this approach is that the system of equations describing AES can be generated once, and Sage Mathematics is not required for each attack. Constraint statements describing the relationship between the bits of the intermediate value are written using Boolean logic. A tool called Limboole [60], takes constraints written in Boolean logic and converts them to DIMACS CNF form. Once the variable assignments in the DIMACS file from `anf2cnf` and Limboole have been deconflicted, the DIMACS files can be combined and used as the input to the SAT solver. The consolidated DIMACS file is processed using CryptoMiniSat2 v2.9.1 [118] and the result is compared with the correct key schedule. If an incorrect key is found, it can be added as a constraint to the system of equations, to ensure the SAT solver will not find the same incorrect solution again. More information on this process is found in Appendix B.

3.7.4 Unique Contributions of this SAT Solver Tool. Although this tool is based on a freely available polynomial system generator, extensive work was performed to introduce the properties of the intermediate values recovered from SCA

to constrain the SAT solver and translate the SAT solver solution back into the intermediate value variables.

The AES-128 SAT solver tool in this research is the first to use multiple models to simultaneously constrain the solution of the SAT solver. Since the properties that can be extracted using SCA are different for each round- or key-bit, properties can be specified in terms of bytes or bits. Models currently incorporated into the tool are exact bit/byte value, exact byte value plus random byte values, byte HW, and possible byte values based on posterior probability. Using a thresholding technique based on the posterior probabilities for all key guesses, is also unique and explained further in Chapter 4.

3.8   Summary

This chapter introduced the common methodology used in this dissertation including, data collection, pre-processing techniques, CEMA-based attacks and template attacks. The techniques presented in the following chapters enhance the effectiveness of these attacks and remove or challenge one or more of the assumptions required to preform these attacks. Additional methodology presented in the following chapters build off of the common methodology presented here.

4. Key Schedule Redundancy Attack

This chapter is based on methodology and results submitted to the *International Journal of Applied Cryptography* in a paper titled "An algebraic side-channel attack on the AES key schedule". The article was coauthored by Dr. Rusty Baldwin and Dr. Michael Temple.

4.1 Introduction

Side-Channel Analysis (SCA) exploits a physical implementation rather than the mathematical cryptographic strength of a cipher. An implementation can leak information about the data being processed on the device, which can lead to the extraction of the key used to perform the cryptographic operation. Various side-channels have been used to attack cryptographic devices including timing [66], power consumption [67], and electromagnetic (EM) emanations [47,94]. Template attacks [24] are a powerful type of two-phased attack in which an adversary builds probabilistic models known as templates during a training phase, and compares key-dependent predictions with observed emissions using those templates during an attack phase. The single key or key portion guess with the highest probability of being correct is typically chosen based on a Maximum Likelihood (ML) decision rule [24,73]. The ML decision rule produces a single guess for each portion of the key. If the collected emissions used for classification are of poor quality, due to poor probe placement, noisy device operation, noise introduced from the collection process or in the collection environment, the ML decision rule may produce an incorrect key guess.

Incorporating algebraic cryptanalysis into the template attack methodology allows the most likely guesses to be considered rather than a single guess. As a result, even if the correct byte is not identified using the ML decision rule, the algebraic structure of the cipher can be used to identify which guess is correct. To demonstrate this, we introduce a new unknown plaintext attack, called the Key Schedule Redun-

dancy Attack (KSRA), that combines template attacks with algebraic cryptanalysis such that the AES key schedule can be recovered even from poor quality collections. The attack has three phases: 1) template construction, 2) trace classification, and 3) key schedule reconciliation. The template construction phase identifies distinguishing features of interest using correlation analysis [22] and builds templates for each targeted intermediate value. The list of potential values for each targeted intermediate byte is used to calculate possible round key-byte values in the attack phase. These possible round key-bytes are then reconciled into a working key schedule using algebraic cryptanalysis in the final phase. While, demonstrated here using the AES key schedule, the proposed method is generally applicable to other ciphers.

Various aspects of this attack are different than previously proposed attacks. This attack directly targets the key schedule, identifying possible values for portions of multiple round keys based on the posterior probability for each byte guess calculated from a template attack. A novel thresholding technique is used to gradually include additional key-byte guesses based only on the posterior probability for each key-byte guess. Since the target microcontroller calculates the key schedule as part of each encryption operation, traces from multiple encryption operations can be used without requiring individual plaintexts or ciphertext to be matched with their corresponding side-channel emissions, eliminating one of the biggest assumptions made in side-channel analysis. Since each round key contains all of the information required to reconstruct the entire key schedule, this attack takes advantage of the redundancy in the key schedule to resolve uncertainty in template classification. Since the goal of KSRA is to combine template attacks and algebraic cryptanalysis to improve performance when using poor quality traces, rather than a hardened design, the attack is performed on an unprotected implementation.

This chapter is organized as follows. Section 4.2 provides a brief overview of the AES key schedule, template attacks and algebraic cryptanalysis. Related work

is outlined in Section 4.3. The new attack is explained in Section 4.4, and results are presented in Section 4.5.

4.2   Background

This attack targets an Electronic Codebook (ECB) implementation of AES on a 16-bit PIC microprocessor. AES is summarized in Section 2.2.2 and fully described in the Federal Information Processing Standards Publication 197 [88]. Since the KSRA focuses on the key schedule, it is explained in detail here.

The KSRA uses known-key Correlation-based Electromagnetic Analysis (CEMA) to identify points of interest for template attacks. Background on known-key CEMA and template attacks can be found in Chapter 3.

4.2.1   Key Schedule Background.   The template attacks in the KSRA target the SubWord operation output in the AES key schedule[1]. How this output is related to portions of the round keys is explained below and shown in Fig. 4.1. The full key expansion routine is explained in AES standard [88].

For AES-128, the variant of AES with a 128-bit key, there are $N_r = 10$ rounds. The key expansion routine takes the original cipher key, K, and generates a total of $N_w = 44$ 4-byte words, $[w_i], 0 \leq i \leq 43$. The first 4 words of the key schedule are from the original 128-bit round key $K = K^0 = [w_0, w_1, w_2, w_3]$ and an additional 4 words are generated for each of the 10 rounds. Each round key is denoted by K^r where r is the round index $0 \leq r \leq 10$ so, $K^r = [w_{4r}, w_{4r+1}, w_{4r+2}, w_{4r+3}]$. Furthermore, since key schedule transformations are performed at the byte level, each round key can be written in terms of its key-bytes as $K^r = [K_0^r, K_1^r, ...K_{14}^r, K_{15}^r]$.

There are multiple intermediate values calculated during the key schedule routine that are not used as the round keys. The last four bytes of each round

---

[1] The motivation for attacking the output of SubWord, rather than the key-bytes directly [24], is described in Section 4.4.2.
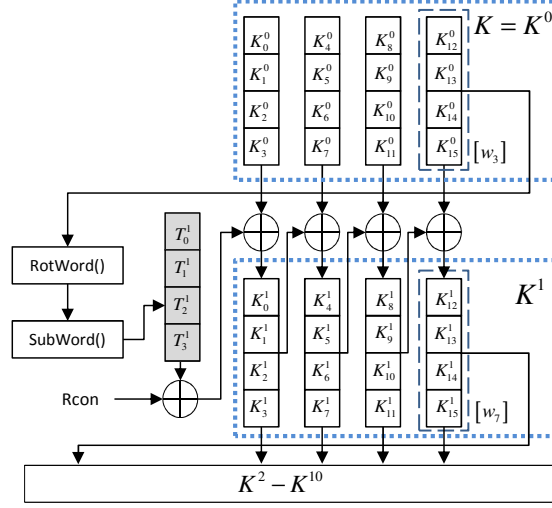
Figure 4.1   Process for transforming the original cipher key into the first round key performed in the key schedule algorithm (figure derived from [65]). This process is repeated for K^2 through K^{10}. The 4-byte outputs of the SubWord operations performed to calculate K^1 through K^{10} are the targets of this attack.

key, $[w_{4r+3}] = [K_{12}^r, K_{13}^r, K_{14}^r, K_{15}^r]$, are transformed using the cyclic permutation $\texttt{RotWord}\left([K_{12}^r, K_{13}^r, K_{14}^r, K_{15}^r]\right) = [K_{13}^r, K_{14}^r, K_{15}^r, K_{12}^r]$.

Next, the SubWord operation takes the 4-byte output of RotWord and applies the AES S-box to each byte to produce a 4-byte output word. This result is XOR-ed with the round word constant, Rcon[i] and finally XOR-ed with $[w_{4r}]$ to calculate $\left[w_{4(r+1)}\right]$. For this attack, since the SubWord operation occurs in the round after the targeted round, r, its output is $\left[T_0^{r+1}, T_1^{r+1}, T_2^{r+1}, T_3^{r+1}\right]$.

4.3   Related Work

A side-channel attack on the AES-128 key schedule is proposed in [71]. The goal of the attack is to substantially reduce the number of keys such that a brute force search is feasible. The attack assumes all 16 8-bit Hamming Weights (HWs) (i.e., the number of bits equal to 1 in the byte) can be extracted for a targeted round key. The round key is divided into four 5-byte overlapping parts. Additional

94

intermediate key schedule values solely determined by each 5-byte set are identified and it is assume the HWs of these values are known. Knowing the HWs for each round key-byte and dependent intermediate values allows lists of possible round keys to be created for each 5-byte part. The four lists are combined to define the new key search space. Not all HW values must be known to determine the key, but the attacker must be able to determine which HWs can be successfully determined during a side-channel attack [71]. With fewer HWs, the key search space becomes larger. The KSRA allows for uncertainty in extracted values, using the algebraic representation of the key schedule and the SAT solver to identify the correct value for each byte.

Renauld et al. combined algebraic cryptanalysis with HW-based template attacks to exploit an implementation of AES on a 8-bit PIC microcontroller [100]. The attack targeted an 8-bit PIC microcontroller which used multiple lookup tables and XOR operations to perform the MixColumn transformation resulting in more intermediate values than needed to perform AES on another microcontroller architecture. Renault et al. note the leakage of the MixColumn operation is "most critical when solving the system". If the HWs for all of the intermediate values for 3 consecutive rounds can be extracted with an error rate less than or equal to 1%, the AES key can be recovered in 95% of cases. Since this approach identifies a single HW for each intermediate value, an incorrect HW will result in the SAT solver determining the system as unsatisfiable or the SAT solver will identify an incorrect key. A SAT solver can use less precise leakages and still find the correct key if given a pair of HWs that includes the correct one along with sufficient rounds of HW information. All template attack results were simulated, and the authors do not state if enough HWs can actually be recovered from the 8-bit PIC microcontroller, to make the attack possible.

Building on the results of [100], Mohamed et al. improved the algebraic representation of AES for an 8-bit implementation to reduce the amount of data required

for both known and unknown plaintext attacks [81]. Additionally, their approach defined and tolerated practical levels of erroneous information recovered from simulated template attacks by increasing the set of possible HW values to reach a predetermined certainty threshold based on maximum-likelihood estimation. Oren et al. proposed dealing with errors by converting the problem of solving the system of equations into an integer programming optimization problem [90]. Successful attacks were demonstrated with 10-20% error rates for the Keeloq system [90] and extended to AES [89] allowing recovery of the secret key in 60-70% of trials with a single trace even when 20% of the trace is corrupted by noise. These attacks [81, 89, 100], were again demonstrated with simulated template attack results, and rely on the ability to extract the HWs from additional calculations used to implement AES on the targeted 8-bit microcontrollers.

Simultaneously, yet independent of this research, posterior probability values were incorporated into the template attack methodology to attack AES [89]. Using known plaintext and ciphertext pairs in conjunction with the algebraic description of AES-128 on an 8-bit microcontroller from [100], the posterior probability values identify the k most likely values for each intermediate value. A fixed number of possible values for each intermediate value produced poor results when used with a SAT solver. More favorable results were achieved when the posterior probability values were used to determine a goal term in an integer programming optimization representation. Rather than use a fixed number of guesses, our KSRA attack demonstrates a thresholding technique to identify the possible values for each intermediate value.

Albrecht and Cid demonstrated it is possible to recover a decayed key schedule in the presence of noise using polynomial system solving techniques to identify the most likely key schedule [6]. In their cold boot application, noise is modeled as the probability of an individual bit in the key schedule stored in Dynamic Random Access Memory (DRAM) flipping from its initial state. Since bit decay in DRAM is

usually asymmetric [55], the probability of a bit flipping to the memory's "ground state" is much higher than the probability of flipping in the opposite direction. This research uses the same system of equations, but possible byte values are determined by the posterior probability of the template attacks.

4.4 The Attack

The KSRA combines correlation-based EM analysis techniques, template attacks and algebraic cryptanalysis. The attack uses collected side-channel emissions from a PIC microcontroller. To test the robustness of the attack, the quality of the data is intentionally degraded by moving the near-field probe away from the device. Training and test data are collected at each probe height above the device. Known-key correlation analysis is performed on the training data to identify the points of interest for each output of the SubWord operation in the key schedule. Templates are built for the points of interest for each byte using the training data. The template classification phase is performed using test data, and the posterior probability for each key bytes guess is used to identify possible values for the last 4 bytes in each round key. The bytes values that meet a threshold posterior probability are included as possible values for the SAT solver. The SAT solver uses a description of the AES-128 key schedule and possible byte values to return a working key schedule if possible. Each of these steps are now explained in greater detail.

4.4.1 Data Collection. Training and test data is collected from PIC A01 as described in Section 3.1. The microprocessor performs MixColumns using the xtimes operation as described in FIPS 197 [88], not using substitution tables as in [81, 89, 100]. The SubBytes substitution table, used in both the rounds and key schedule, is the only substitution table used.

To test the robustness of the attack, the quality of the data is intentionally degraded by gradually increasing the distance between the device and probe from

97

$h = 0$ to $h = 5$ mm in 1 mm increments. At each height, $n_t = 10,000$ training traces and $n_t = 5,000$ test traces are collected. After down-sampling, the traces used in the attack have an effective sampling frequency of $f_s^D = 200$ MSa/sec.

4.4.2   Targeted Intermediate Values.   To identify the portions of the round key for round r which can be recovered from the targeted intermediate values, the output of SubWord in the following round is designated $\left[T_0^{r+1}, T_1^{r+1}, T_2^{r+1}, T_3^{r+1}\right]$ and shown in Fig. 4.1.

The temporary values produced by SubWord, $\left[T_0^{r+1}, T_1^{r+1}, T_2^{r+1}, T_3^{r+1}\right]$, are the target of the KSRA. Letting SB denote the AES S-Box, with SB^{-1} denoting its inverse, the relationship between the last 4 bytes of round keys K^0 through K^9 and the subsequent SubWord operation which calculates the next round key can be written,

$$\left[\text{SB}\left(K_{13}^r\right), \text{SB}\left(K_{14}^r\right), \text{SB}\left(K_{15}^r\right), \text{SB}\left(K_{12}^r\right)\right] = \left[T_0^{r+1}, T_1^{r+1}, T_2^{r+1}, T_3^{r+1}\right]. \tag{4.1}$$

Therefore,

$$
\begin{aligned}
K_{12}^r &= \text{SB}^{-1}\left(T_3^{r+1}\right), \\
K_{13}^r &= \text{SB}^{-1}\left(T_0^{r+1}\right), \\
K_{14}^r &= \text{SB}^{-1}\left(T_1^{r+1}\right) \text{ and,} \\
K_{15}^r &= \text{SB}^{-1}\left(T_2^{r+1}\right).
\end{aligned}
\tag{4.2}
$$

Possible values for $\left[T_0^{r+1}, T_1^{r+1}, T_2^{r+1}, T_3^{r+1}\right]$ are determined using template attacks. Since the original key can be determine using the last 4 bytes of any 4 consecutive round keys, the 40 bytes of round key data extracted using this method are redundant and can be used to determine the correct key schedule in poor quality trace sets. Given a set of guesses for the last 4 bytes of each round key, a SAT solver attempts to reconcile the guesses into a proper key schedule.

A more direct attack would be to build templates for each byte of the AES-128 key [24]. However, following the same methodology to identify distinguishing features to build and use the template attack described in this section, only the last 4 of the 16 key-bytes can be extracted successfully for the target device. Other methods of identifying distinguishing features including Principal Component Analysis [9] were tested, but did not improve the result.

While any intermediate value in the key schedule can be targeted, only comparison of the input and output SubWord operations provides additional information. All other intermediate values are linearly related to the output of the previous Sub-Word operation. The SubWord output was chosen as the target intermediate value for the template attack because it yielded the highest posterior probabilities for correct key-byte values out of all intermediate values tested.

In testing, including additional linearly related intermediate values degraded performance. Performance is degraded because including additional intermediate values introduces additional (incorrect) key-byte guesses for the same key-byte, increasing the likelihood of finding an incorrect key schedule.

4.4.2.1 Identifying Distinguishing Features. The goal of the template attack is to identify the most likely byte values of each output of the SubWord operation. To identify the correct byte, each bit must be correctly identified. Using the known cipher keys from the training data, the output of the SubWord operation is calculated and bit-level correlation analysis [22] is performed to determine which samples are highly correlated with the leakage for each bit of the SubWord output. This process is described in Section 3.4.3.

The samples with the highest correlation coefficients for each bit are added to the list of distinguishing features. If a sample has already been added because it was highly correlated with another bit, the sample with the next highest correlation coefficient for that bit is added. Only samples which have correlation coefficients

significantly ($5\times$) greater than the average correlation coefficient are added to the list of distinguishing features. Up to 10 points of interest for each bit are included in the list of distinguishing features for each byte. Including more than 10 points for each bit did not dramatically improve classification performance for $h = 0$. To allow for comparison, the maximum number of points was fixed for all heights.

The known-key correlation analysis is repeated using the data sets collected at each distance. As a result, the interesting points identified for the trace set collected with the near-field probe is at $h = 0$ mm may not be the same points as identified using the trace set collected when the probe is 5 mm above the device.

4.4.3 Template Attack. A template attack is performed on 40 intermediate values of the key schedule to determine possible values for T_b^{r+1} where $b \in \{1, ..., 4\}$ and $r \in \{0, ..., 9\}$. Although distinguishing features are selected using bit-wise known-key correlation analysis, the templates are constructed for $K = 256$ possible key-byte values (classes), $k_i \in \{0, ..., 255\}$, as described in Section 3.6.2.

For the matrix $\mathbf{X}$ containing distinguishing features from one test trace on each row, the probability that a byte guess is correct is [73],

$$p\left(k_i = T_b^{r+1} \middle| \mathbf{X}\right) = \frac{\left(\prod_{d=1}^{N_t} p\left(\mathbf{x^T}_d \middle| k_i = T_b^{r+1}\right) \cdot p\left(k_i = T_b^{r+1}\right)\right)}{\sum_{l=0}^{K-1}\left(\left(\prod_{d=1}^{N_t} p\left(\mathbf{x^T}_d \middle| k_l = T_b^{r+1}\right)\right) \cdot p\left(k_l = T_b^{r+1}\right)\right)}, \tag{4.3}$$

where i is the index of the N_t test traces.

Since AES key-bytes are uniformly distributed, it is initially assumed that $p\left(k_l = T_b^{r+1}\right) = 1/256$. Let p_{thr} be the posterior probability for which a key-byte guess is retained as a round key-byte candidate. Since the key schedule portion of each trace is the same, an alternative approach is to average multiple traces before performing classification. In testing, the Bayesian classification approach in (4.3) yielded better results.

If $p\left(k_i = T_b^{r+1} \middle| \mathbf{X}\right) \geq p_{thr}$ it is included in the list of possible values for T_b^{r+1}. The list of possible values for T_b^{r+1} are used to calculate possible values for K_{12}^r, K_{13}^r, K_{14}^r or K_{15}^r using (4.2).

Selecting the proper threshold at which to keep or reject a byte guess for each T_b^{r+1} is critical to the success of this attack. For the SAT solver to find the correct key schedule, the value of T_b^{r+1} that corresponds to the actual key-byte must be included in the list of possible bytes for each targeted byte. If all correct round key-byte values are not included in the list of possible values, the SAT solver will either produce an incorrect key schedule or return an UNSAT result. If too many round key-byte guesses are included, the SAT solver will either find an incorrect key schedule or not identify a working key schedule in a timely manner.

To calculate the ideal threshold, p_{thr}, the maximum threshold that allows all of the correct round key-bytes to be included in the list of possible round key-byte values, the minimum posterior probability of the correct values for the 40 targeted round key-byte values is found. The ideal threshold is calculated for each attack. It is shown in Section 4.5.3.2 that for an actual attack, the posterior probability matrices for each targeted round key-byte can be used to determine possible values for p_{thr} by evaluating the number of round key-byte guesses that would be included for a given p_{thr}.

4.4.4 Reconciling Round Key-Byte Guesses. The AES Key Schedule SAT Solver tool described in Section 3.7 is used to reconcile the round key-byte guesses. The guesses for the byte value for 40 key schedule bytes are included as constraints to the system of equations describing the AES-128 key schedule. The SAT solver attempts to identify a working key schedule. Since the key schedule rounds contain redundant information, it may identify the correct key schedule even if the correct byte value would not have been selected using a ML decision rule. If a working key schedule cannot be found the SAT solver will identify the system as unsatisfiable

(UNSAT). If a working key schedule is returned, the correct key schedule is compared to the result to determine if the attack is successful.

4.5 Results and Comparison

4.5.1 Evaluating Performance. After collecting training and test traces as described at $h = 0$ mm, a template attack is performed using the ML decision rule on each of the 40 targeted key schedule intermediate values. The attack is repeated with $n_t = 500$ test traces, and overall the correct round key-byte value is identified in 98.5% of the template attacks using only a single trace. To intentionally degrade the quality of the collected traces, the distance between the probe and device is increased from $h = 0$ to $h = 5$ mm in 1 mm increments.

Increasing the distance between the probe and the device reduces the signal-to-noise ratio (SNR). The SNR is estimated for one bit changing for each set of training traces collected at different distances using the methodology proposed in [73]. In addition to the variation caused by the data, each trace contains electronic and switching noise as well as noise from the collection equipment and environment. Digitizing the signal introduced quantization noise. The SNR for one bit changing at a specific sample of a collected signal S is estimated to be $SNR \equiv Var\left(S_{data}\right)/Var\left(S_{noise}\right)$.

To estimate $Var\left(S_{data}\right)$, the mean signal observed value when the bit of interest is zero, μ_0, and the mean when the bit is one, μ_1, are calculated to average out the contributions of noise. Next, the value observed for a sample in each of the $n_t = 10,000$ training traces is replaced with either μ_0 or μ_1 based on the actual bit value for that trace. To estimate $Var\left(S_{noise}\right)$, the variance of all traces when the bit value is equal to 0 is calculated. This process estimates the SNR for the LSB of K_{12}^1 for the training data collected at each distance. The SNRs are calculated for each time sample in the collected traces. Since the highest SNRs may occur at different samples for each trace collection, the SNRs are compared by calculating the average of the five highest SNRs at each distance and normalizing by the average of
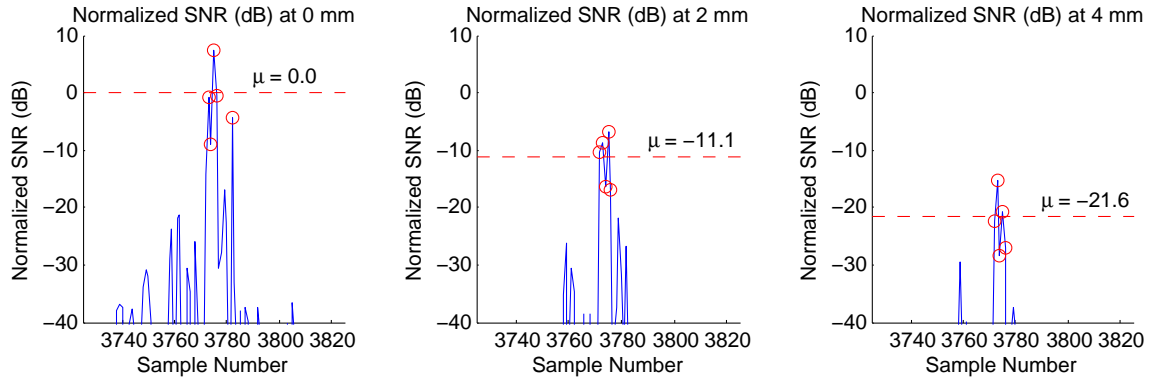
Figure 4.2   Estimated normalized SNR is calculated by averaging the 5 highest SNRs for each distance and normalizing using the average when the probe is at $h = 0$ mm (left). The change in SNR in dB when the near-field probe is placed $h = 2$ mm (middle) and $h = 4$ mm (right) above the device are shown.

Table 4.1   Calculated change in SNR (dB) from increasing the distance between the microcontroller packaging and the bottom of the probe.

	Probe Height h (mm)					
	0	1	2	3	4	5
Normalized SNR (dB)	0	-3.7	-11.1	-14.9	-21.6	-26.3

5 SNRs calculated when the probe is at $h = 0$ mm. Plots of the SNRs calculated for the distances of $h = 0, 2$ and 4 mm for samples 3725 to 3825 are shown in Fig. 4.2. The 5 highest SNR values, circled on in Fig. 4.2, correspond with the 5 highest correlation coefficients found during the single-bit correlation analysis used to identify distinguishing features for the template. The normalized SNR for each probe height is shown in Table 4.1.

SNR was estimated at the bit level because the distinguishing points are selected at the bit level. Performing these calculations on other bits produced consistent results, so only one example is shown.

4.5.2   Comparison of Distinguishing Features.   The samples used to build templates for $h = 0$ mm and $h = 5$ mm are compared in Figs. 4.3 and 4.4. Using the
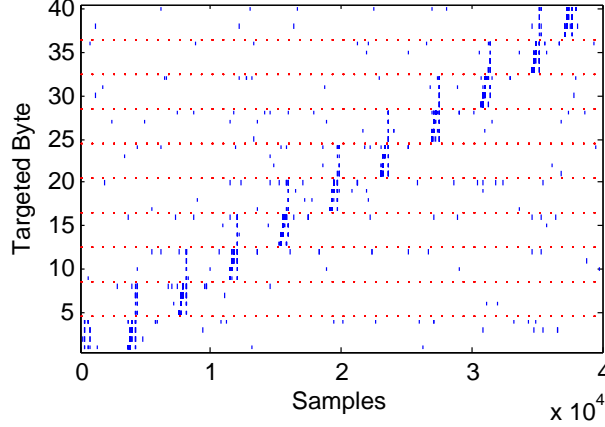
Figure 4.3   Plot of distinguishing features chosen for $h = 0$ mm. The round key for the next round is calculated before the round starts. The points which are highly correlated with a HW model are identified as distinguishing features. There are 4 targeted bytes per round. To help distinguish rounds, a dotted line is shown between each round.

methodology described in Sec. 4.4.2.1, between 59 and 80 points are identified for training data collected at $h = 0$ mm. When the distance is increased to $h = 5$ mm the number of points selected to build templates for each byte is reduced to between 30 and 66 because fewer samples have correlation coefficients significantly greater than ($\geq \times 5$) the average correlation coefficient. The reduced number of points, and the increased number of points scattered across the entire trace, indicate the traces collected at the greater distance do not follow the HW model as well. The reduced number of points and selection of points with lower correlation, make the template attack at $h = 5$ mm less effective than for $h = 0$ mm.

4.5.3   Experimental Results.   Since this attack directly targets the intermediate values of the key schedule, it has the benefit of being able to use traces from multiple encryption operations without requiring the plaintext or ciphertext to be associated with each collected side-channel emission. In fact, only a single plaintext/ciphertext pair is needed to verify the correct key has been found. The
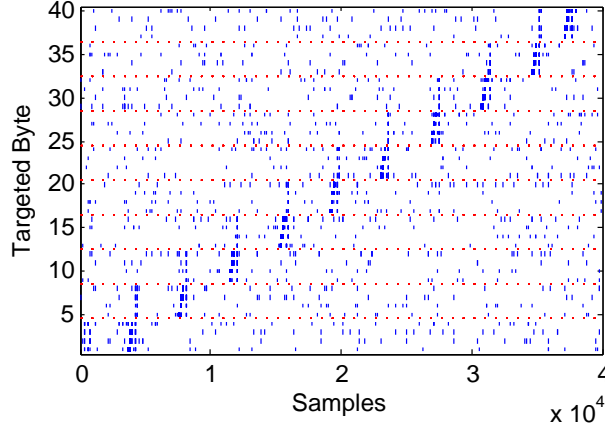
Figure 4.4   Plot of distinguishing features for $h = 5$ mm. At $h = 5$ mm additional noise causes various points across the entire trace to be identified as distinguishing features. Overall, less distinguishing features are chosen for $h = 5$ mm than for $h = 0$ mm.

performance of this attack is evaluated by repeatedly performing the attack 500 times using randomly chosen sets of 1, 5 or 50 traces from the $n_t = 1,000$ test traces.

4.5.3.1   Ideal threshold.   To reduce the time required to evaluate performance of the KSRA, the ideal threshold p_{thr} (discussed in Section 4.4.3) is used to identify possible values for each round key-byte for the results in Table 4.2. For these trials the SAT solver is only allowed to return one solution. If any portion of the key schedule is incorrect, the entire key schedule is incorrect. The average number of round key-bytes (of the 40 targeted values) correctly identified using the ML decision rule are shown in parentheses for comparison. Since they do not indicate the percentage of key schedules recovered using the ML method, they are shown as a ratio to avoid confusion.

With a probe height of $h = 2$ mm using only 1 test trace, the KSRA was able to recover the correct key schedule in 90.6% of the trials despite correctly identifying only 21.7 of the 40 extracted round key-bytes correctly using a ML decision rule. With a probe height of $h = 5$ mm, if 50 traces are used during the template matching

105

Table 4.2 Percentage of 500 trials resulting in the correct key schedule being identified at each probe height using the unknown plaintext attack proposed. The posterior probability threshold p_{thr} used to select possible `SubWord()` output is calculated for each trial to allow all correct bytes to be included. The average number of the 40 targeted key schedule values correctly identified using a ML decision rule are shown in parentheses.

# of Traces	Probe Height h (mm)					
	0	1	2	3	4	5
1	100.0% (39.4/40)	99.8% (33.9/40)	90.6% (21.7/40)	40.6% (13.9/40)	0.0% (4.7/40)	0.0% (1.9/40)
5	100.0% (40/40)	100.0% (39.1/40)	99.6% (35.2/40)	99.6% (29.40/40)	58.3% (15.8/40)	0.0% (6.8/40)
50	100.0% (40/40)	100.0% (39.9/40)	100.0% (38.1/40)	100.0% (37.0/40)	99.8% (28.0/40)	93.6% (17.8/40)

phase, the key schedule is successfully recovered in 93.6% of trials without knowledge of the plaintext.

4.5.3.2 Experimentally Determined Threshold. Clearly, in an actual attack the ideal threshold is not known. However, the posterior probability matrices for each round key-byte can be used to identify possible threshold values. The threshold can gradually be decreased based on the number of key-byte guesses included at a given threshold. The initial threshold is set to include at least one guess for all targeted round key-bytes. If this initial guess fails to identify the correct key due to the SAT solver returning an incorrect key or UNSAT, the threshold is lowered for subsequent attempts. Although there are potentially 40x256 thresholds to evaluate, good results were obtained when the threshold was adjusted to increase the *maximum* number of guesses per round key-byte each time the SAT solver returned an Unsatisfiable (UNSAT) or incorrect key schedule. These thresholds can easily be found by ranking the key schedule byte guesses for each intermediate values by posterior probability, and finding the maximum posterior probability for the desired number of guesses. Although this method still identifies up to 256 threshold values, these are quickly evaluated by the SAT solver.

To test the effectiveness of gradually lowering the threshold, KSRA is performed repeatedly, reducing the threshold for each subsequent attack. Since all correct key schedules in Table 2 were found in less than 60 seconds, the SAT solver timeout is set to 120 seconds. Additionally, since it is unlikely the SAT solver will find the correct answer when the average number of guesses for each round key-byte is greater than 50, or the minimum number of key guesses for any byte is greater than 10, these statistics were used to identify when the threshold should no longer be increased. These constraints allow multiple correct key-byte guesses to have low posterior probabilities. However since only one threshold is used to determine the possible values for all key schedule bytes, if the correct key-byte value for one or more bytes has a very low posterior probability due to a particularly noisy or poorly collected trace, the number of possible values for other key schedule bytes will be increased.

The effectiveness of using this method to find the threshold is compared to calculating the ideal threshold using sets of 50 traces collected at $h = 5$ mm. By gradually increasing the maximum number of round key-byte guesses, the correct key was identified in 488 of 500 (97.6%) of the trials. This is a marked improvement over the 93.6% success rate for the "ideal threshold". When the ideal threshold is known, the SAT solver was only run once. However, when the threshold is unknown and an incorrect answer is returned it is assumed the threshold was not low enough to include the correct round key-bytes and the threshold is decreased further. Although the SAT solver has a larger set of possible key-byte values in this case, it also has another chance to find the correct key.

4.5.3.3 Multiple Guesses. When the threshold is gradually lowered, the correct key is found for 12 of the 16 trials which produced incorrect key guesses using the "ideal" threshold. This discrepancy is resolved by allowing the SAT solver to return another guess if it returns an incorrect key schedule. Adding incorrect solutions to the SAT solver constraints prevents previous results from being returned

by the SAT solver in subsequent attempts. Allowing for up to 10 incorrect answers to be returned, 488 of 500 keys (97.6%) were recovered successfully using the ideal threshold, matching the performance when the threshold was unknown.

$4.5.3.4$ *Error Tolerance.* If the template attack classification process results in the correct byte value having a very low posterior probability due to noise, p_{thr} must be decreased until the correct value is included in the set of possible byte values before it will be possible for the SAT solver to find the correct key schedule. Unfortunately, reducing p_{thr} also increases the number of possible byte values for other round key-bytes. When a large number of byte values are possible for each of the targeted bytes, the SAT solver may find another working key schedule before finding the correct key. If the correct key is not found after a set number of attempts, the trial is considered a failure. For all trials shown in Table 4.2, the correct value for each byte is included in the list of possible bytes. All failures are due to an incorrect key schedule being returned from the SAT solver after the first attempt.

$4.5.4$ *Comparison.* Effectiveness of KSRA is compared to a template attack on the SubBytes output in the first round of AES-128, as described in Section 3.4.2. This type of side-channel attack is compared with KSRA because it also incorporates multiple traces using a Bayesian classification rule. To perform classification for a SubBytes attack using multiple traces, the plaintext must be known. The SubBytes template attack is implemented using the same distinguishing feature selection criteria and the ML decision rule is used to select the 16 key-byte candidates. The percentage of trials that successfully identified all 16 key-bytes for each probe height are shown in Table 4.3.

Even without knowledge of the plaintexts, performing a proposed SubWord attack with a SAT solver provides a higher success rate when only a small number of traces or plaintexts are available. If only one captured trace is available, the SubWord attack has a higher success rate than the SubBytes attack for $h = 0$ to 3

Table 4.3　Percentage of 500 trials resulting in the correct key being identified at each probe height using a known plaintext template attack targeting the output of the SubBytes operation in the first round of an AES-128 encryption operation. The average number of the 16 key values correctly identified using a ML decision rule are shown in parentheses.

# of Traces	Probe Height h (mm)					
	0	1	2	3	4	5
1	16.8% (14.2/16)	0.0% (4.8/16)	0.0% (4.8/16)	0.0% (3.4/16)	0.0% (1.5/16)	0.0% (0.7/16)
5	95.8% (15.9/16)	85.2% (15.6/16)	66.8% (15.6/16)	42.2% (15.1/16)	4.2% (13.1/16)	0.0% (8.67/16)
50	100.0% (16/16)	100.0% (16/16)	100.0% (16/16)	100.0% (16/16)	100.0% (16/16)	98.2% (15.9/16)

mm. If the associated plaintext is known for a large number of test traces, the SubBytes attack provides superior performance.

While the SPA key schedule attack in [71] and SKRA both exploit key schedule redundancy, there are key differences between the attacks which make them hard to compare directly. The SPA attack uses 81, 76, or 40 HWs extracted with perfect accuracy to reduce the size of the key schedule search space. A brute force iteration is required to determine which key in the reduced search space is correct.

The KSRA is based on actual side-channel attacks against the PIC microcontrollers. The 40 key schedule bytes are chosen because their values can be revealed through side-channel analysis. The SPA key schedule attack in [71] identifies which key schedule bytes could be used to determine the key schedule, but it does not take into account which byte values (or HWs) can actually be determined on a physical implementation.

The KSRA uses the posterior probabilities for the 256 possible byte values for 40 key schedule bytes. With perfect byte-value extraction accuracy this attack could be performed with only 16 bytes from the key schedule. Targeting 40 bytes allows key-byte values to be consider by the SAT solver that would not be chosen using the ML decision rule. Since the SPA attack assumes perfectly extracted HW

data, despite increased brute force search times for lower quantities of HWs, it always recovers the correct key. Unlike the SPA attack, the KSRA is able to use information from multiple observations and is able to tolerate imprecise classification. Since guesses for any key-byte can be specified in the SAT solver constraints, KSRA can be easily adapted to implementations of AES-128 that leak different key schedule bytes. As such, direct comparison between the SPA attack and the KSRA is not appropriate.

4.6   Conclusion

This chapter demonstrates the benefit of using algebraic cryptanalysis to reconcile uncertainty in the classification stage of template attacks. The KSRA is a new unknown-plaintext attack that exploits the redundancy in the key schedule to compensate for measurement errors. Unlike previous attacks, which merely simulated the results of template attacks [81, 89, 90, 100], the robustness of the KRSA is demonstrated using collected traces that were intentionally degraded by moving the probe away from the device.

The KSRA identifies possible round-key-byte candidates using the posterior probability from the classification phase of a template attack rather than selecting the byte guess with the highest posterior probability. The resulting list of possible round-key-byte values are reconciled into a working key schedule using a SAT solver. In addition to not requiring the plaintext, this method was shown to be more effective than targeting the output of the SubBytes operation in the first AES round based on the ML decision rule when a small number of plaintexts are available.

Unlike previous SAT solver based unknown plaintext attacks [81, 89, 90, 100] which targeted intermediate values from the AES round transformation, traces from multiple encryption operations can be used if the key schedule is calculated during each encryption operation. The use of multiple traces dramatically improves performance for poor quality trace sets. For data collected at $h = 5$ mm with only one

110

trace, the key schedule was not successfully recovered in any of the 500 attempts. However, when 50 traces and multiple SAT solver attempts are used the success rate improves to 97.6%.

Pre-calculating and storing the key schedule is an obvious countermeasure for poor quality collections requiring more than one trace to determine the key schedule. In this case, an attacker would not be able to observe the same key schedule being calculated during multiple encryption operations and multiple traces could not be used to identify the most probable byte guesses. However, unlike the SubBytes attack, KSRA was successful using but a single trace for all trials when the probe was placed as close to the microprocessor as possible.

Although determining the secret key for an unprotected implementation of AES-128 on a microprocessor using side-channel analysis is trivial using high quality collections, the KSRA has shown the value of incorporating algebraic cryptanalysis and a SAT solver into template attacks. Even with poor quality traces, the cipher redundancy can be exploited to determine the correct key value justifying the extra effort required to create and incorporate the algebraic description of the cipher into the template attack methodology.

5. *Improving Cross-Device Template Attacks*

This chapter contains text of an article submitted and accepted to the Journal of Cryptographic Engineering [85] titled "Improving cross-device attacks using zero-mean unit-variance normalization" based on the cross-device attack on PIC micro-controllers. This article was co-authored by Dr. Rusty Baldwin, Dr. Michael Temple and Mr. Eric Laspe. It was published online 29 September 2012, and in Volume 3, Issue 2, pp 99-110 of the print edition. The background section was reduced to avoid redundancy and notation has been homogenized between chapters of this dissertation. Improvements on the cross-device methodology, to create a single template for a family of devices, and improve cross-device attacks for more complicated devices are discussed in Chapter 6.

5.1 *Introduction*

Template attacks [24] are a form of two-stage profiling attack, with the initial stage obtaining 'a priori' knowledge of the side-channel leakage for a specific device. The profiling, or training stage estimates the multivariate probability densities of observable side-channels for the targeted key-dependent internal state of the cryptographic implementation. The estimated probability densities are used during the attack phase to determine the device's internal state. The key assumption for a profiling attack is that a powerful attacker has access to a training device, identical to the target device, over which he has full control. The training device is used to create a precise multivariate model of the device's side-channel leakage for each key dependency. Implicit in using a training device is that both devices produce similar side-channel emissions. This assumption was originally introduced in [24], and has since been repeatedly accepted without challenge [3, 9, 53, 92, 102, 122].

It has recently been shown that in addition to operation and data dependent components of electromagnetic (EM) emissions, the emissions exhibit signif-

icant device-dependent characteristics [31]. This is likely due to random process variations introduced during fabrication and packaging [69]. Although the structural variations introduced in the manufacturing process are relatively small, and the devices produced meet the desired specifications, no two chips are exactly alike. Therefore, the emissions produced by similar devices are indeed similar to some degree but not identical. These variations are significant enough to allow a specific device to be uniquely identified based only on the devices EM emissions [29]. The work here examines the differences in cross-device emissions to determine if such differences are sufficient enough to prevent template attacks from being effective if *similar* devices are used for training and testing, versus using the *same* device for training and testing.

Template attack research has expanded the capabilities of template attacks to use multiple test traces [92], multiple side-channels [4], reduced the number of features required to build templates using heuristics [99] and systematic methods [9, 122], and employed templates to defeat countermeasures [3, 92]. Template attacks have been adopted as an attack methodology without evaluating the underlying assumption the power consumption and EM emissions from two separate devices are sufficiently similar to make the attacks practical. In each of the papers cited above, the same smartcard or microprocessor was used to create both the training and test data.

Research here focuses on the EM side-channel of a device performing the Advanced Encryption Standard (AES) encryption operations. Unlike power consumption methods, the EM side-channel can be collected without physically modifying the device. This makes repeating the collection process more difficult. Instead of monitoring the voltage change across a single shunted resistor, careful consideration must be given to placing the EM probe in exactly the same position and configuration between collections on a device. Even template attacks performed using the same-device can fail if the probe is moved between the collection of training and test

traces. Recently, differences in collection equipment, methodology, synchronization and target device age were shown to reduce the effectiveness of template attacks even when attacking using templates created with the same device [41]. This paper explores the differences between devices.

The remainder of this chapter is organized as follows. Background on the target cipher, the AES [88] and template attacks were presented in Section 2.2.2 and Section 3.6 respectively. Known-key Correlation-based EM Analysis (CEMA), used to identify distinguishing features, was introduced in Section 3.4.3. Differences in side-channel emissions between devices and development of the mean and variance normalization technique are explored in Section 5.2, followed by the experimental methodology in Section 5.3. Results are presented in Section 5.4 followed by the conclusion in Section 5.5.

5.2   Cross-Device EM leakage

The EM side-channel can be divided into various components as was done for the power consumption side-channel in [73]. Each sample in the EM side-channel trace is made up of various components: a operation-dependent component S_{op}, a data-dependent component S_{data}, electronic noise $S_{el.\,noise}$, and a constant component S_{const}. A sample in the total EM side-channel trace is a sum of these components, or

$$S_{total} = S_{op} + S_{data} + S_{el.\,noise} + S_{const}. \tag{5.1}$$

The distribution of $S_{el.\,noise}$ in a microprocessor has been shown to be $S_{el.\,noise} \sim \mathcal{N}(0, \sigma^2_{el.noise})$. The contribution of S_{data} is proportional to the Hamming Weight (HW) of the data being processed and its distribution can be approximated with the normal distribution when the data is uniformly distributed, or $S_{data} \sim \mathcal{N}(0, \sigma_{data})$. Note that $\sigma_{el.noise}$ and σ_{data} are device specific.

114

For a differential side-channel attack where only the data is changing between traces, it can be assumed that $Var\left(S_{const}\right) = Var\left(S_{op}\right) = 0$, and $E\left(S_{op}\right) = E\left(S_{data}\right) = E\left(S_{el.noise}\right) = 0$. Therefore, $E\left(S_{total}\right) = E\left(S_{const}\right) = \mu_{const}$, where μ_{const} is a constant contribution for the operation being performed at a specific time on a specific device.

Assuming S_{data} and $S_{el.noise}$ are statistically independent, $S_{data} + S_{el.noise} \sim \mathcal{N}\left(0, \sigma^2_{data} + \sigma^2_{el.\,noise}\right)$ and $S_{total} \sim \mathcal{N}\left(\mu_{const}, \sigma^2_{data} + \sigma^2_{el.\,noise}\right)$. For a cross-device template attack to be successful, μ_{const} and $\sigma^2_{data} + \sigma^2_{el.\,noise}$ must be consistent across devices at samples identified as distinguishing features.
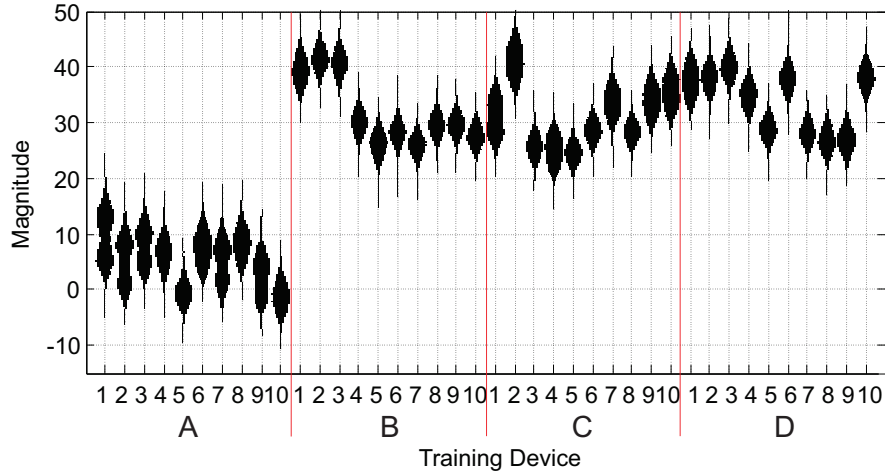
Figure 5.1   Violin plots showing the distribution of 5,000 observations of sample #972 on an AES encryption operations with random plaintext and keys for $N_D = 40$ similar microcontroller devices.

Figure 5.1 uses a violin plot [57] to show the distributions of 5,000 observations of sample (#972) for $N_D = 40$ different training devices. These 40 devices are from the same family of PIC microcontrollers, with devices within groups of 10 (denoted A, B, C and D) having identical part numbers. More information on the devices can be found in Section 3.2.1. It is shown in Section 5.4.2 that if the differences in means and variance between device groups are not compensated for some template attacks will fail.

5.2.1 Compensating for Device Differences. To compensate for the device-to-device differences in μ_{const} and $\sigma^2{}_{data} + \sigma^2{}_{el.\,noise}$, a transformation of variables is performed. The transformation ensures the test data have approximately the same distribution as the training data.

Using collected training and test data at a specific time sample collected across multiple traces, the mean and variance of that sample can be estimated for each data set. Let variable X_{train} represent the value of EM traces at a specific sample in the training data, and let variable X_{test} represent the value of the test data at the same sample.

Let $\hat{\mu}_{train}$ and $\hat{\sigma}^2_{train}$ be the estimated mean and variance of X_{train}, and let $\hat{\mu}_{test}$ and $\hat{\sigma}^2_{test}$ represent the estimated distribution of the test data. X_{test} is used to calculate transformed X'_{test} having the same mean and variance as the training data using

$$X'_{test} = \frac{(X_{test} - \hat{\mu}_{test})}{\hat{\sigma}_{test}} \hat{\sigma}_{train} + \hat{\mu}_{train}. \tag{5.2}$$

Test data transformation is performed for each sample selected as a distinguishing feature for the template attack.

Alternatively, both X_{test} and X_{train} can be transformed to the standard normal via

$$X'_{test} = \frac{(X_{test} - \hat{\mu}_{test})}{\hat{\sigma}_{test}}, \text{and} \tag{5.3}$$

$$X'_{train} = \frac{(X_{train} - \hat{\mu}_{train})}{\hat{\sigma}_{train}}. \tag{5.4}$$

To reduce attack time and eliminate the need to retain the training data, the classifier can be trained using (3.12) and (3.13) before collecting the test traces. Transforming both X_{test} and X_{train} eliminates the need to retain the training data or store $\hat{\mu}_{train}$ and $\hat{\sigma}_{train}$ for each distinguishing feature.

The remaining steps of the template attack are performed as usual using transformed test data. This process is referred to as the zero-Mean, unit-Variance Normalization (MVN) technique herein. Although this technique was developed independently for this research, it was first published in [41] as a way to compensate for differences in the collected trace sets from the same device before and after device modifications and aging.

5.3   Experimental Methodology

5.3.1   Targeted Devices.   To test the effectiveness of the MVN technique, template attacks are performed to attack 40 unprotected 16-bit PIC24F microcontrollers. The PIC device naming conventions can be found in Table 3.1. The process used to collect $n_t = 500$ test traces and $n_t = 5,000$ training traces with an effective sampling frequency of $f_s^D = 250$ MSa/sec can be found in Section 3.1. The attacks are performed with and without the MVN technique using two methods for generating distinguishing features.

5.3.2   Template Attack Methodology.   Template attacks performed with training and test data from the same-device are referred to as *same-device* attacks. For cross-device attacks, each device is used as a training device and used to attack all 40 devices. In all attacks, feature selection is performed using a single training device. If an attacker only has access to one training device, it is assumed that he would follow a similar process.

The initial step develops two highly effective same-device template attacks using both a heuristic method and Principal Component Analysis (PCA) to select distinguishing features. Classifier training and trace classification are performed as described in Sections 3.6.2 and 3.6.3 respectively. Since they are specific to this attack, the rationale for class selection and distinguishing feature selection are discussed here.

5.3.2.1   Class Selection.   In the classification stage of the template attack, the classifier attempts to determine which class an observed side-channel emission most likely belongs to. The training traces are separated into $K = 256$ classes and templates are created for each class.

5.3.2.2   Correlation-based Feature Selection.   Distinguishing features are identified using known-key CEMA as described in Section 3.4.3. The intermediate value attacked for all devices is the input to SubBytes. This intermediate value was chosen because it yielded higher success rates for same-device attacks than attacking the output of SubBytes. Since both the plaintext, $\mathbf{t} = (t_1, t_2, ...t_{n_t})^T$, and sub-keys, $\mathbf{k} = (k_1, k_2, ...k_{n_t})^T$, are known for each of the n_t collected traces in the training data, the correct intermediate value $v_{d,i} = f(t_d, k_i) = t_i \oplus k_i$ can be calculated. The leakage model $h_{d,i}$ based on the HW of $v_{d,i}$ is also easily calculated.

Selecting distinguishing features based on known-key CEMA with a byte HW model for the targeted intermediate value byte produces adequate results. However, classification is improved by performing correlation analysis separately for each bit of the intermediate value byte. This approach produces a vector of correlation coefficients for each of the 8 bits in the targeted byte. Samples with the highest correlation coefficient for each bit are added to the list of distinguishing features for the byte. If a sample has already been added because it was highly correlated with another bit, the sample with the next highest correlation coefficient for that bit is added. Only samples which have correlation coefficients significantly greater ($\geq 5\times$) than the average correlation coefficient are added to the list of distinguishing features. It was determined empirically through experimentation that including up to 10 points of interest for each bit produced very good results.

5.3.2.3   PCA-based Feature Selection.   PCA is also used to generate and select distinguishing features in the principal subspace. Based on the samples selected using correlation analysis, only the first $n_{PCA} = 3500$ samples are used for

each trace. For byte-wise analysis with $K = 256$ classes same-device PCA-based template attacks are not always successful unless approximately 80 components in the principal subspace are used as distinguishing features. This may be due to the relatively low number of traces (~ 19 on average) from each class used to construct the mean traces. Like the correlation-based feature selection process, the probability of correctly identifying the key-byte improves by performing bit-wise analysis.

PCA is performed for each bit of the target byte, with $K = 2$ classes for each bit. A PCA transformation matrix $\mathbf{W}_b \in R^{n_{PCA} \times 1}$ is constructed for each bit $b \in \{1, ..., 8\}$ retaining a single principal component. Rather than perform 8 bit-wise template attacks, these 8 transformation matrices are combined column-wise,

$$\mathbf{W} = \left[\begin{array}{ccc} \mathbf{W}_1 & \cdots & \mathbf{W}_8 \end{array} \right] \tag{5.5}$$

where $\mathbf{W} \in R^{n_{PCA} \times 8}$. The new transformation matrix is used to perform byte-wise template attacks using (3.17), (3.18), and (3.19).

5.3.3 Distinguishing Feature Data Normalization. Motivation for the MVN technique can be seen in Figure 5.1, which provides violin plots [57] for the distributions of the 5,000 observations of sample #972 for each device. Sample #972 is one of the 16 samples chosen as a distinguishing feature for all 40 devices. The violin plots show that observation mean and variance changes from device to device.

For each sample selected as a distinguishing feature, the distributions of training and test data at that sample are normalized using (5.3) and (5.4) respectively. Normalization is performed independently on training and test data because the mean and variance at corresponding samples may not be the same for different devices. For simplicity when utilizing PCA the distribution for training and test data are normalized for each sample across all traces before PCA transformation into the primary component subspace. The test data set must contain enough traces to es-

timate the distribution of each sample accurately. The amount of traces required to estimate the distribution is explored in Section 5.4.5.

5.4 Results

5.4.1 Selected Features. The correlation-based feature selection methodology in Section 5.3.2 is repeated for each of the $N_D = 40$ devices. Each byte of the input to the SubBytes operation in round 1 of AES-128 is targeted separately with $K = 256$ templates constructed for each byte. The process is repeated to identify distinguishing features to build templates for each training device. Figure 5.2 is a graphical representation of the samples selected for each device for byte 1. Since only these samples are used as distinguishing features, and there are large gaps between them, the time axis is segmented multiple times to compress the plot.

Recall that part A devices have different peripherals than devices in groups B-D. There is some intra-device type variation in the samples for parts B-D devices but 19 of the samples are the same across the three part types. It is important to note that a number of samples which are consistently selected for devices in group A are not identified as distinguishing features for any of the devices in group B, C, and D. Likewise, some samples commonly selected in groups B, C and D are not selected for group A.

When performing a PCA-based template attack, the transform matrix $\mathbf{W}$ that maps the trace samples into the principal subspace is generated separately for each set of training data. Plotting the eigenvectors is one way of visualizing contributions of the original samples in the principal subspace [122]. Since the magnitude of the eigenvector elements determine the weight of a sample's contribution in a component, a plot of samples which contribute most to one or more of the retained components can be generated by calculating statistics for the magnitude of the eigenvector elements for the retained components. Figure 5.3 shows the maximum value of the eigenvector element magnitude for each of the 8 retained components found

120
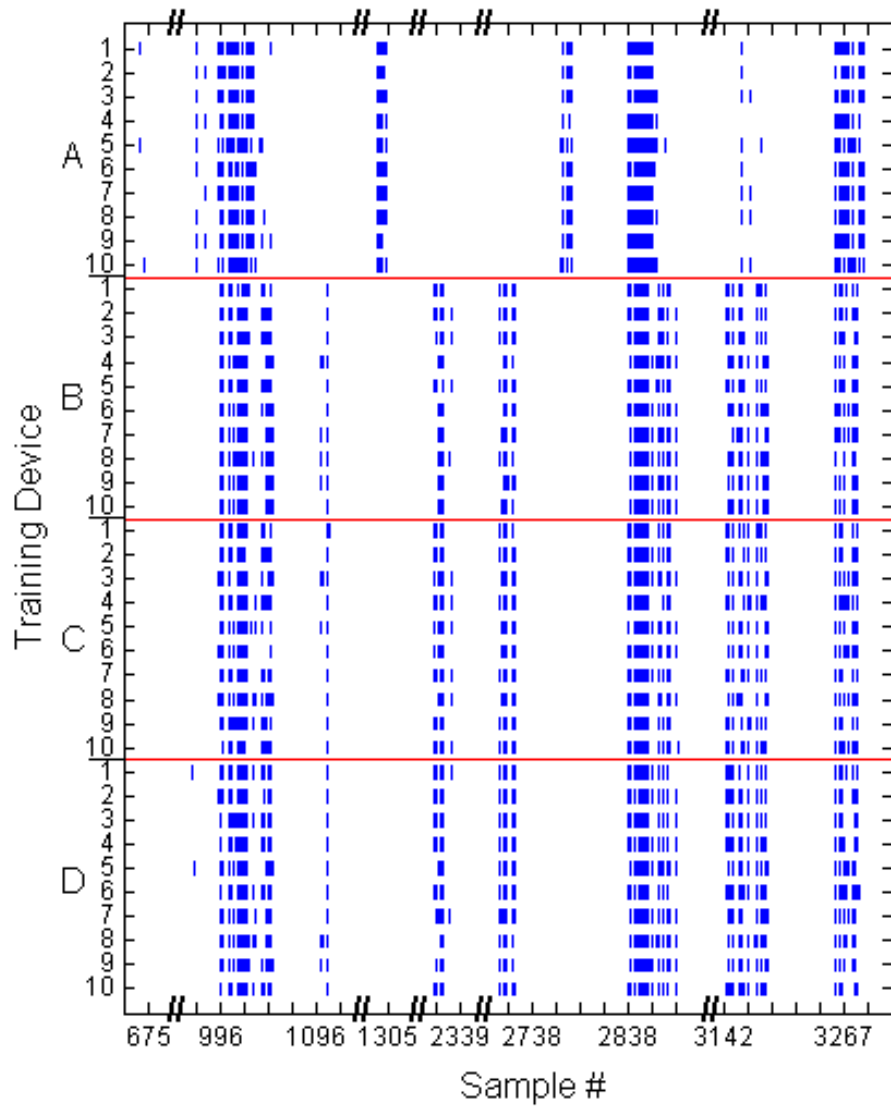
Figure 5.2   Samples selected as distinguishing features for each of the 40 devices using the correlation-based feature selection process when attacking byte 1 of the input to the SubBytes operation. All B, C and D devices share 19 common features while only 6 of those features are identified for all of the type A parts.

121

Table 5.1   Standard template attack cross-device key-byte extraction success rate using correlation-based selection of distinguishing features (without the MVN technique).

Test	Training Device			
Device	A	B	C	D
A	59.7%	4.3%	5.2%	5.1%
B	13.1%	63.0%	68.3%	70.2%
C	11.6%	67.9%	63.7%	69.5%
D	14.7%	69.8%	69.2%	70.3%

using bit-wise PCA. Since a majority of the samples contribute to one or more of the retained primary components the plot is not segmented as in Figure 5.2. Note that different samples are weighted more heavily based on the training device. As is the case for the correlation-based feature selection, devices in groups B, C and D are more similar to each other than they are to group A.

5.4.2   Baseline Standard Template Attack.   A standard template attack assumes a multivariate Gaussian distribution for each sample and assumes the distributions for corresponding training data samples and test data samples are identical [24]. Figure 5.4 shows the percent of key-bytes correctly extracted for 1600 template attacks, one for each device used as a training device and as a test device. Each attack is repeated using 100 randomly chosen sets of 30 test traces from the $n_t = 500$ available test traces for each test device. The same 100 trace sets for each test device are used in the attack performed in Section 5.4.3 and Section 5.4.4. When the same-device is used to generate both the training and test data, the template attacks identify each of the 16 key-bytes correctly in all trials. Same-device attacks are found on the diagonal of the chart. The overall percent of successful key-byte extractions using one device from groups A-D to attack another device from groups A-D is shown in Table 5.1. The cross-device success rates do not include same-device attacks.
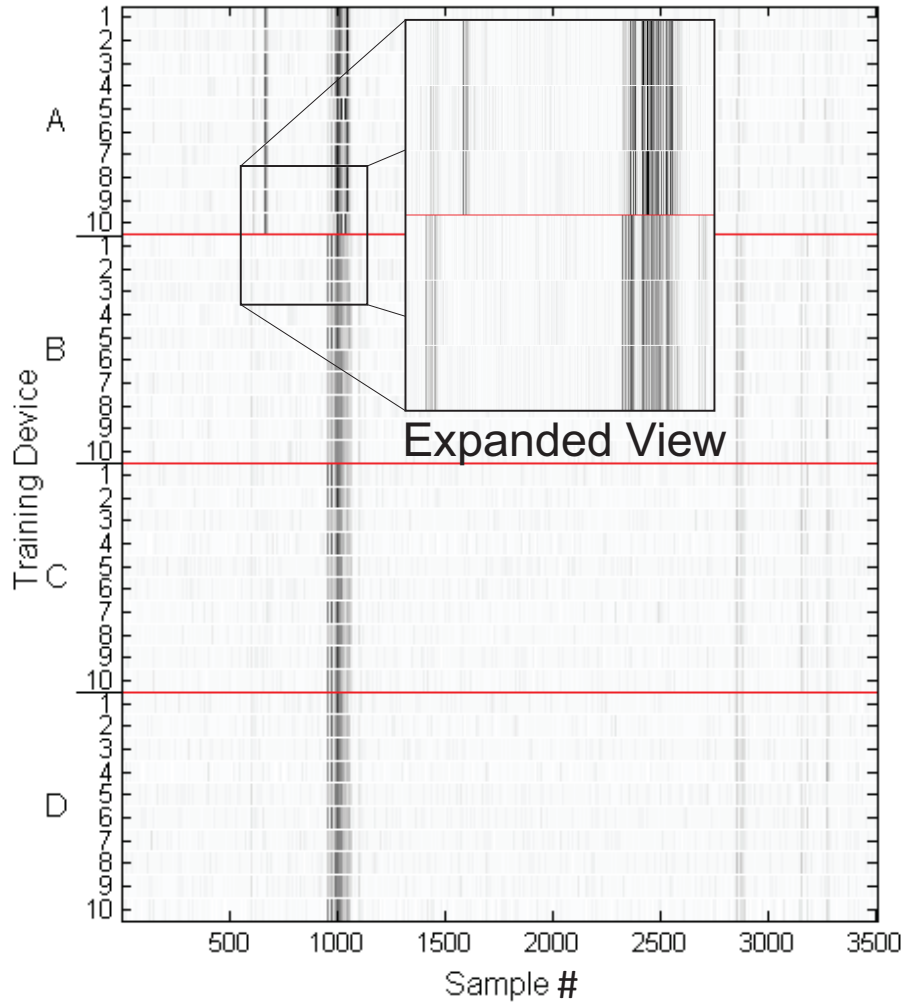
Figure 5.3   Plot of the maximum magnitude of the eigenvector elements for the eight retained components found using bit-wise PCA. Darker points have higher maximum eigenvector element magnitude, indicating they contribute more to one or more of the retained components in the principal subspace.
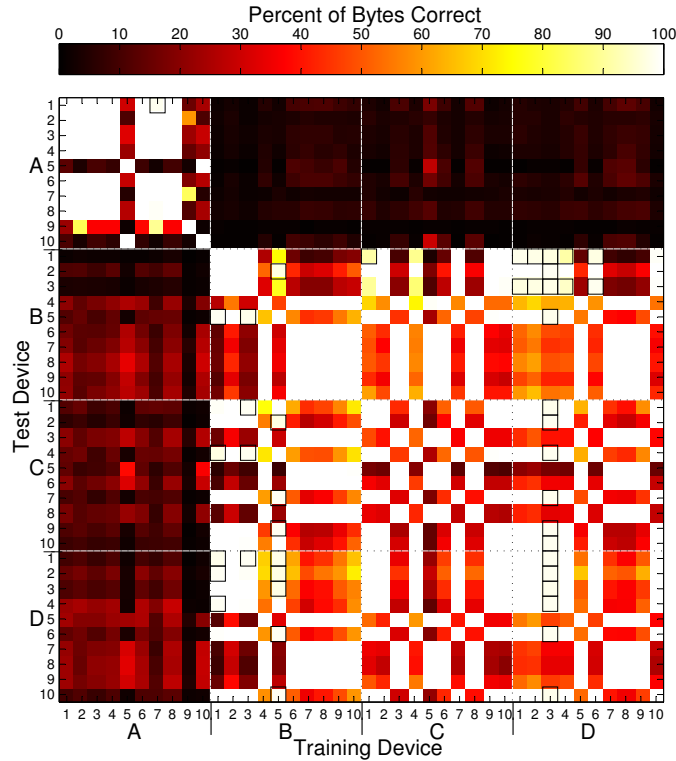
Figure 5.4   Standard attack results using same- and cross-device templates without the MVN technique. The percentage of correctly extracted key-bytes in 100 trials is indicated by the color of the block. Percentages $\geq 90\%$ and $< 100\%$ are highlighted with a box.

The reduced number of correct key-bytes when training using devices from group A to attack devices from groups B, C, or D can be explained in part by the difference in the distinguishing features used to build templates. More surprising is the poor results for intra-group attacks which construct templates using many of the same samples as distinguishing features. The poor results are due to location and spread differences in the distributions for collected side-channel emissions at the points used as distinguishing features for training and test data. For example, Figure 5.1 shows the distribution of sample #972 for 5,000 observations in each set of training traces. Since the test data collected for each device is consistent with the distribution of the training data from that device, it is not shown separately.

Table 5.2   MVN-enhanced cross-device key-byte extraction success rate for matched distribution correlation-based template attack.

Test	Training Device			
Device	A	B	C	D
A	99.9%	70.0%	70.3%	67.8%
B	58.8%	100.0%	100.0%	99.9%
C	64.8%	100.0%	100.0%	99.9%
D	61.5%	100.0%	100.0%	99.9%

5.4.3   MVN Technique Results.   The template attack results can be improved by transforming the test data to match the distribution of the training data or by mapping both the training and test data to the standard normal. This transformation is performed separately for the data from each distinguishing feature before templates are built. These template attacks are performed with $n_t = 5,000$ training traces and $n_t = 30$ test traces. Unlike the attacks in [41] which normalized the data using 50,000 measurements, only the 30 test traces in each trial are used to estimate the distribution of the test data. Since the main benefit of template attacks is the low number of test traces required, limiting the number of traces used for normalization is more realistic. Using this simple pre-processing step, the successful byte extraction rate is improved for cross-device attacks for both the same part number and similar devices.

The correlation-based template attack is repeated after pre-processing the training data and each of the 100 test trace sets for each device. The results are shown in Figure 5.5 and Table 5.2. With the MVN technique, any device from groups B, C or D can be used to successfully attack any test device in groups B, C, or D. Any device in group A can be used to attack any device within that group. The worst same-part-number performance was using A9 to attack A5 where the correct key-byte was returned in only 94.4% of the attacks. Device D3 has the 'poorest' results as a training device when attacking similar devices with only 98.7% of bytes-correct for devices in groups B-D, however in practice such an attack would likely be

125
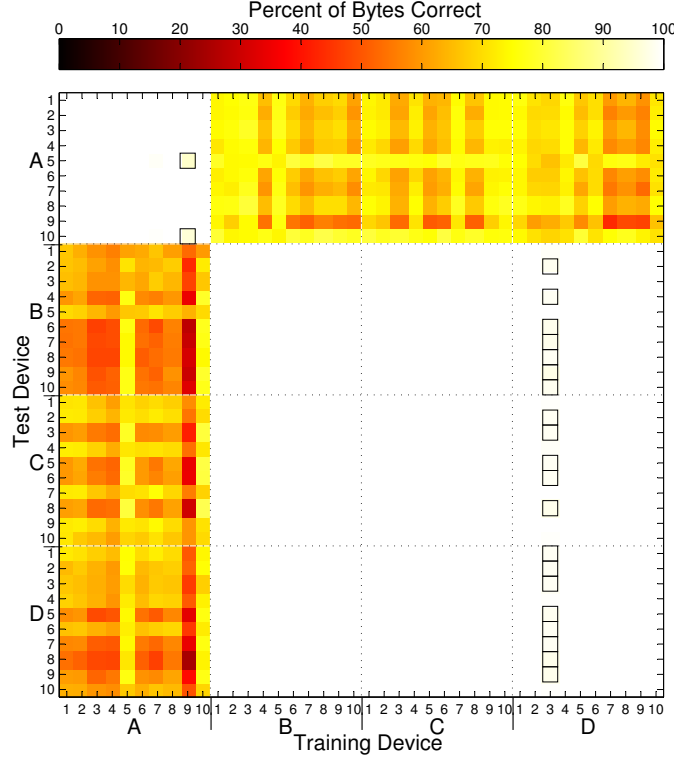
Figure 5.5   Results from same and cross-device template attacks using MVN technique with correlation-based distinguishing features. The percentage of correctly extracted key-bytes is indicated by the color of the block. Percentages $\geq 90\%$ and $< 100\%$ are highlighted with a box.

successful. The MVN technique also dramatically improves the percentage of bytes correctly identified when attacking between groups B-D and group A.

A same-device attack can be performed successfully for all 40 devices using 1-4 training traces. For same-device attacks, normalizing the training and test data reduces the posterior probability of the correct key-byte guess found during the classification step for an equivalent number of traces, but the correct key-byte is still chosen based on the Maximum Likelihood (ML) decision rule. The additional traces required for cross-device attacks allow the distribution of the test data to be estimated accurately. It is important to note that plaintexts only need be known for traces used in the classification process and not for all traces used to estimate the

Table 5.3   Cross-device key-byte extraction success rate for MVN PCA-based template attack. These numbers do not include the same-device attacks.

Test	Training Device			
Device	A	B	C	D
A	99.6%	30.4%	29.9%	30.0%
B	16.7%	100.0%	100.0%	99.7%
C	17.3%	100.0%	100.0%	99.7%
D	19.5%	100.0%	100.0%	99.7%

distribution. In this case, 30 traces were used for both distribution estimation and classification.

5.4.4 PCA-based Attack. The PCA-based attack incorporates the MVN processing step. Normalizing the mean and variance is a common PCA pre-processing step when data is collected using various scales for different dimensions. It is not clear if this step is performed in [9, 122], as testing showed it is not necessary for same-device template attacks.

The PCA-based attack uses 8 distinguishing features generated by transforming the training data and test data using $\mathbf{W}$ found using (5.5) and performing a byte level template attack. The PCA-based template attack is repeated 100 times for each training and test pair and the success rate is shown in Figure 5.6. Cross-device byte extraction success rate is shown in Table 5.3. Although all same-device attacks are successful, a small number of the attacks within a group only correctly recover 15 of 16 bytes. The number of bytes correctly extracted when training using devices in group A to attack devices in group B-D is significantly lower than for attacks using the same test traces with the correlation-based distinguishing feature selection process. The PCA-based attack can be improved by increasing the number of principal components retained for each byte (or bit), or by performing PCA only on the points identified using the correlation-based selection process.
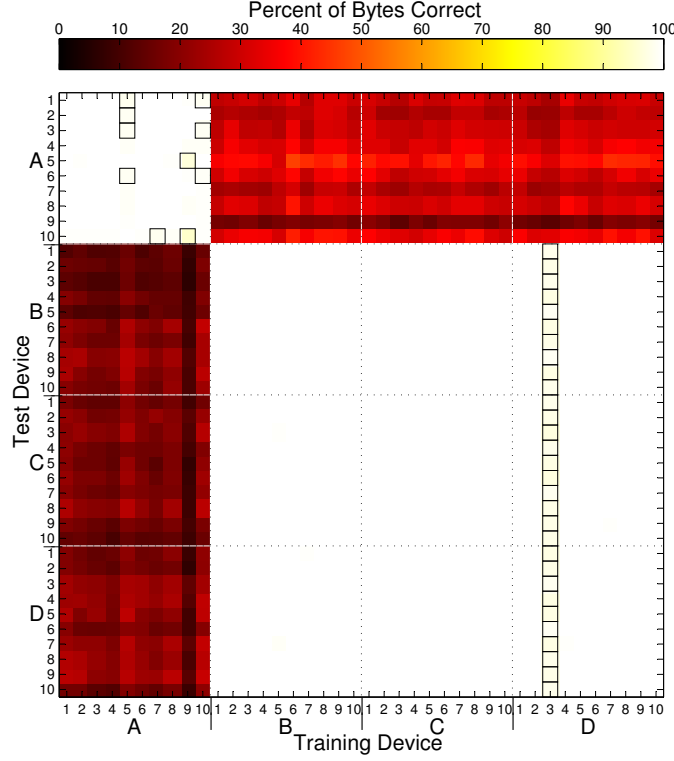
Figure 5.6   Results from same and cross-device template attacks using MVN tech-
nique with bit-wise PCA performed on the distinguishing points vectors
found using correlation analysis. The percentage of correctly extracted
key-bytes is indicated by the color of the block. Percentages $\geq 90\%$
and $< 100\%$ are highlighted with a box.

5.4.5 Comparison of Attacks.   This section examines how the MVN tech-
nique affects the probability of successful key-byte extraction for a single training
and test device pair. The correlation-based and PCA-based template attacks are
performed using A1 as the training device and A5 as the test device. This pair is
chosen because the MVN technique improves the results for both the correlation and
PCA-based attacks for 30 traces. The attacks in Figs. 5.4–5.6 are performed using
sets of 30 traces from the $n_t = 500$ collected test traces for each device. The number
of traces required to perform each type of attack is evaluated below.

When using a Bayesian classifier, the order in which traces are processed the-
oretically does not matter. In practice, traces may be ignored if they cause the
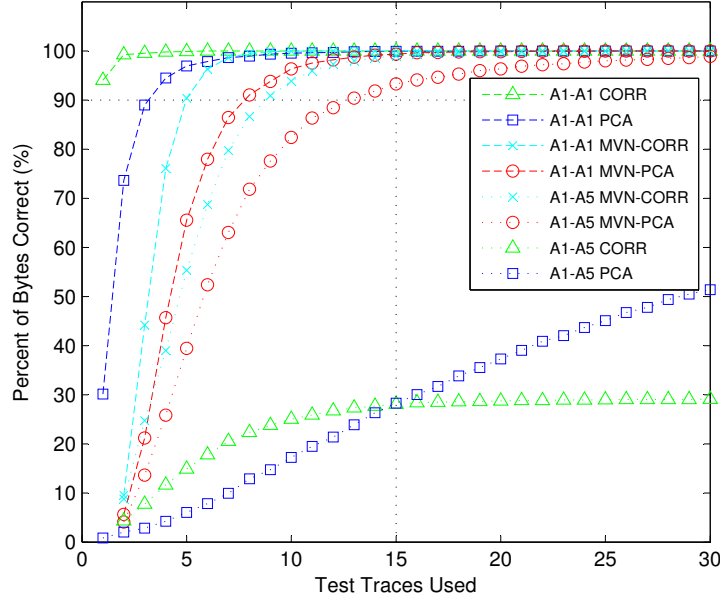
Figure 5.7   Comparison of same-device (A1-A1) and cross-device (A1-A5) template attacks using the baseline standard template attack and attacks using the MVN technique. CORR and PCA indicates whether correlation analysis or PCA was used to identify/generate distinguishing features.

denominator of (3.14) to be zero. This occurs frequently for cross-device attacks without the MVN technique when the test data samples have different distributions than the training data.

To randomize the order test traces are added to the template attacks, 500 permutations of the $n_t = 500$ test trace indices are generated to specify trace order. The percent of bytes correct in Figures 5.7 and 5.8 are the percentage of bytes correct across all 16 key-bytes using the template attack for the 500 randomly generated trace orders. The same 500 trace orders are used for each template attack methodology.

Same-device template attacks are very effective using only a small number of traces. All results in Figure 5.7 are performed using only the indicated number of test traces for both normalization and classification. As expected, for a low number of traces, i.e., less than 15, where the distribution of each sample cannot be accurately estimated, using MVN for same-device attacks dramatically reduces the
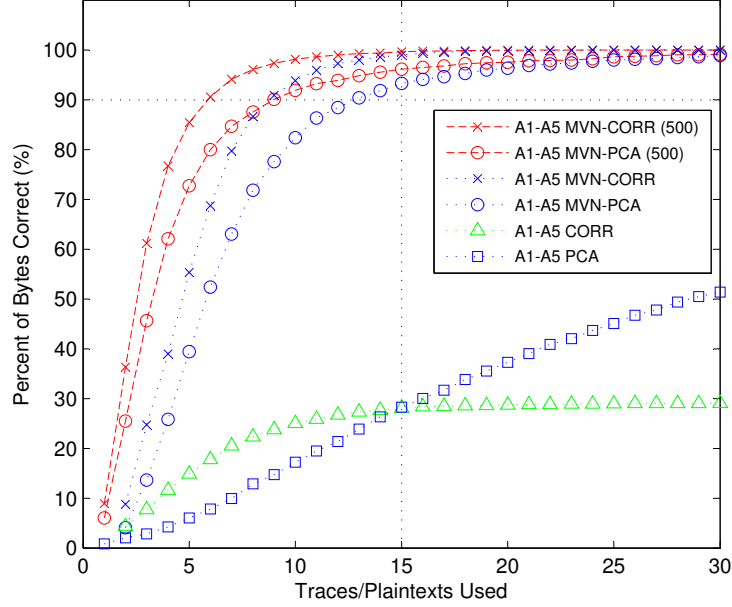
Figure 5.8   Using additional traces to estimate the distributions improves the results for MVN technique-based attacks. When $n_t = 500$ traces are used for the MVN process (denoted 500), with the indicated number of plaintexts, the results for both the correlation-based (CORR) and PCA-based MVN attacks improve.

effectiveness of the attack. However, for a same-device attack the standard template attack methodology can be used.

For A1-A5 cross-device correlation-based attacks, a relatively poor successful byte extraction rate of approximately 28% is achieved after 15 traces without the MVN technique. The PCA-based template attack gradually improves to 51.3% for 30 traces. The A1-A5 cross-device attacks, however, are greatly enhanced by the MVN technique with 99.6% successful byte extraction by 15 traces for the correlation-based attack and 99.1% successful byte extraction by 30 traces for the PCA-based attack.

Figure 5.8 shows the benefit of using more traces to estimate the distributions before normalization. By performing the MVN technique on all $n_t = 500$ traces before using the traces for classification, fewer traces are needed to achieve the same percentage of correct bytes for both correlation- and PCA-based cross-device template attacks. Using $n_t = 500$ traces for the MVN processing technique, the

correlation-based attack reaches 90% successful extraction in 6 traces. It takes 9 traces to reach the same byte extraction rate when only traces with plaintexts are used to estimate the distribution. Using $n_t = 500$ traces for MVN, the PCA-based attack reaches 90% at 9 traces. When only using the traces with plaintexts, 13 traces are required before the 90% extraction rate is reached.

5.5 Conclusion

This chapter explored whether *similar* devices can be used as effective training devices for a template attack. It was shown that while template attacks based on mean and covariance matrices work well for attacking the same device on which the training traces are collected, the slight differences in emissions from similar devices may be sufficient to cause a template attack to fail. However, if the zero mean, unit variance normalization (MVN) technique is used to pre-process both the test and training data before building templates, the effectiveness of cross-device template attacks is significantly improved. These results are consistent with the benefit of the MVN technique utilized in [41] for differences in measurement conditions and device age for training and test data.

Additionally, the distinguishing features selected may be different from device to device, even for devices with the same part number produced in the same lot. If enough distinguishing features are different between two devices, the template attack will fail. Even small changes such as internal peripherals are sufficient to degrade the byte extraction success rate. While the goal for a same-device attack is to reduce the number of distinguishing features to make the templates easier to create, increasing the number of distinguishing features improves the cross-device attack success rate.

Ultimately, the original assumption that training and target devices have sufficiently similar side-channel emissions in [24] is validated with an added caveat that device-dependent differences in sample means and variances must be compensated for before performing the template attack.

One limitation of the MVN technique is that sufficient traces from the target device must be available to estimate sample distributions accurately. This is a drawback of the MVN technique but nevertheless it allows for successful attacks that would fail otherwise. Furthermore, even if a relatively large number of traces are required to estimate the distribution, only a small number of plaintext or ciphertext must be known for the classification process.

5.6 Constructing a Master Template

This section was not included in the original "Improving cross-device attacks using zero-mean unit-variance normalization" paper [85], but introduces an efficient way to create single master template for a family of devices. This is the first known template attack based on traces from more than one device.

To construct a single template for multiple devices, the distinguishing features for one device from each part number (A1, B1, C1, and D1) are identified separately using known-key CEMA as described in Section 5.4.1. The four lists of up to 80 distinguishing features for each byte are then combined and duplicate sample indices are removed to create a single list of distinguishing features for each byte.

Next the training data from each of the four devices are pre-processed with the MVN technique before being combined into a consolidated training trace set. Using the combined training set, templates are constructed for each byte. These templates are used to attack all 40 devices using test traces processed using the MVN technique. The attack is repeated 100 times using sets of $n_t = 30$ test traces for both estimating the distribution and in the classification phase. The success rate is shown for each byte and test device in Figure 5.9. On average 99.95% of key bytes where correctly identified. Since performing this process without the MVN technique, produces results worse than using a single device to build templates without the MVN technique, results are not shown for this method.
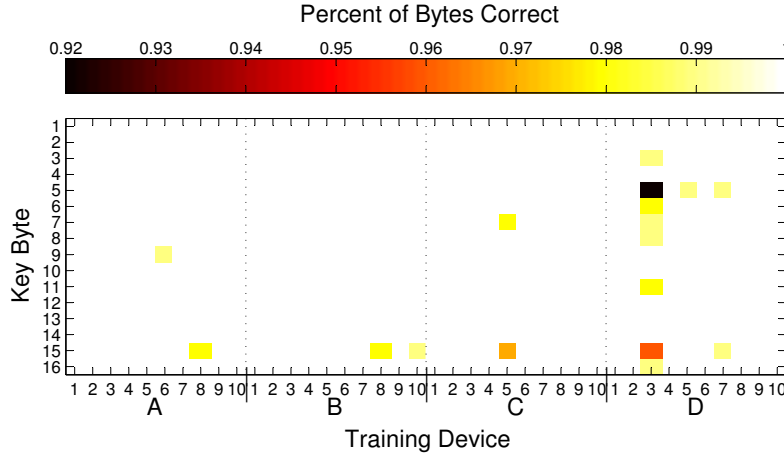
Figure 5.9   Attack results per key-byte using a set of templates built from train-
ing trace sets combined after performing the MVN technique on each
device training set separately. The percentage of correctly extracted
key-bytes in 100 trials is indicated by the color of the block. Note all
byte extraction rates are greater than 90%.

An alternative approach is to process the training data using the MVN tech-
nique and combined the trace sets before identifying distinguishing features. Using
a fixed number of distinguishing features (≥ 80), this approach reduces the effec-
tiveness of the attack. In this case, since devices B1, C1, and D1 are very similar
(compared to A1) the distinguishing features they have in common have higher cor-
relation with the data being processed than the features important for device A1. As
a result, attacks using test traces from device A1-A10 have poor results. Although
more distinguishing features could be allowed when using this approach, identifying
distinguishing features separately for each device type is simple and effective.

Since known-key CEMA attacks produced better results than PCA, the mas-
ter template was constructed using this technique. When performing PCA for a
template constructed from multiple training devices, including additional principal
components in (5.5) from the PCA transformation matrix constructed for each bit
may improve results.

6. Cross-Device Attacks on Complex Microprocessors

This chapter contains text of an article submitted to the Journal of Cryptographic Engineering [83] titled "Cross-device attacks on complex microprocessors" based on the cross-device attack on ARM microcontrollers. This article was co-authored by Dr. Rusty Baldwin and Dr. Michael Temple. The background section was reduced to avoid redundancy and notation has been homogenized between chapters of this dissertation.

6.1   Introduction

The PIC microcontrollers used in Chapter 5 made an ideal target for testing cross-device template attacks. They are inexpensive and all four part numbers could be placed on the same development board. By mounting a Zero Insertion Force (ZIF) socket to the development board, the microcontrollers were quickly and easily swapped out while maintaining the lateral position of the probe above each device. Since the same development board was used for all collections, the contribution due to the microprocessor was isolated.

Even if the development board could not have been modified to accept the ZIF socket, the small size of the PIC microcontroller package makes it easy to reposition the probe on each device. Side-channel analysis becomes more challenging as device complexity increases. More complex devices may have faster operating frequencies, and more noise from other parts. Since the PICs are relatively simple compared with the more complex microcontrollers used in devices, such as cell phones and tablets, this chapter evaluates the effectiveness of cross-device attacks on the ARM Cortex-M4F microcontroller.

While cross-device template attacks are shown to be effective even without applying the zero-Mean and unit-Variance Normalization (MVN) technique developed in Chapter 5, the MVN technique is shown to improve the attack success rate. A

cartography scan is performed to show the MVN technique increases the area above the device where test traces can be collected and used for a successful template attack.

When the MVN technique is combined with a new technique to identify and reduce the differences between training and test data, the number of test traces required for a template attack is dramatically reduced. The power spectral density of each trace is used to identify the frequencies that have different amounts of power from trace to trace. For the ARM Cortex-M4F devices tested, these frequencies can be identified using only 10 traces. Combining the two techniques reduced the average number of traces required to attack a key-byte by 85.8%.

Background on the T-Box implementation of the target cipher, the Advanced Encryption Standard (AES) [88] and template attacks were presented in Section 2.2.2 and Section 3.6 respectively. Known-key Correlation-based Electromagnetic Analysis (CEMA), used to identify distinguishing features, was introduced in Section 3.4.3. The remainder of this chapter is organized as follows. Research most directly related to the techniques developed in this chapter are reviewed in Section 6.2. Techniques to improve cross-device attacks for the ARM Cortex-M4F microcontrollers, are developed in Section 6.3, and results are presented in Section 6.4. Finally, Section 6.5 concludes this chapter.

6.2 Related Work

Mapping the training and test data to the standard normal, is used by Elaabid and Guilley to compensate for "carrier-induced degradation" of the training device and changes collection setup [41]. Since a differential voltage probe was used to measure the voltage across a resistor, the collection location was fixed. This chapter will explore the effectiveness of the MVN technique to compensate for changes in probe placement on the test device.

Although filtering was used by Barenghi et al. to enhance the effectiveness of CEMA attacks [13], no research has been found suggesting filtering can improve the success rate for template attacks. This may be due to the belief that template attacks build the noise into the templates in the training phase [24]. While this may be effective if the noise is present in both the training data and test data, as it is for a same-device template attack, it may not be true for cross-device attacks.

Kocher et al. state that "digital filtering" can help "reduce noise" and "focus on parts of the spectrum where the leakage signal is present" without specifically stating how the filtering is performed [68]. Trace compression (adding successive measurements), subtracting unwanted effects, and average traces from identical operations are given as examples. Unfortunately, averaging test traces in a template attack, where the attacker does not have complete control over the target device, is not practical.

6.3 Methodology

Two identical ARM Cortex-M4F development boards are used to test the MVN technique and the new techniques developed in this chapter. The training device is designated ARM1 and attacks are performed using test data from both ARM1 and ARM2. Attacks that used ARM1 for both training and test are called *same-device* attacks. Attacks that use traces from ARM1 for training, and traces from ARM2 for testing are called *cross-device* attacks.

6.3.1 *Device Leakage Cartography.* To identify the best location to collect training data for a template attack, traces are collected at each of the 625 locations specified by a 25×25 grid, as described in Section 3.2.2.1. A total of $n_t = 2,500$ traces, with random plaintexts and fixed key, are collected with $f_s = 250$ MSa/sec at each location on both ARM1 and ARM2. The trace sets are not aligned because there is no distinguishable structure in the collected signal for many of the locations

and alignment for traces with structure only improves the CEMA attack results marginally. To be consistent across all locations, no alignment is used. The trace sets from ARM1 are used for CEMA attacks, and traces from both devices are used during the classification phase of template attacks.

CEMA attacks are performed using $n_t = 1,500$ traces to determine which locations result in effective CEMA attacks. The same set of plaintexts are used at each location. Based on the results for ARM1 shown in Figure 6.1, location 303 (indicated by a black circle) was chosen because it is in the middle of a cluster of locations from which the majority of the key-bytes can be extracted from the collected traces. Since an attacker targeting ARM2 may only be able to collect a small number of test traces, making a XY scan of the target device impractical, placing the probe in an area where a small change in probe position can be tolerated is more desirable than placing the probe in a location with better CEMA attack performance if the attack success depends on exact probe placement. Even if the probe can be placed at the same location above the device package, manufacturing differences may cause the attack to fail in a cross-device attack.

Although the locations with good CEMA attack performance are similar for ARM1 and ARM2, the attacks on traces collected from ARM1 consistently yield a higher number of bytes correct than attacks on traces from ARM2. This comparison is shown in Figure 6.1. Since an attacker does not have full access to the test device (ARM2), the attacker would not be able to produce Figure 6.1(b) to compare the two devices.

6.3.2 Identifying Unrelated Signals. The power spectral density (PSD) of a signal describes how the power in the signal is distributed in the frequency domain. Since the same instructions are being executed for each encryption operation, the power for each frequency should be approximately the same for each trace. Changes in the PSD between traces may be due to changes in the data being processed or due

(a) ARM1　　　　　　　　　　　　　　　　　　(b) ARM2
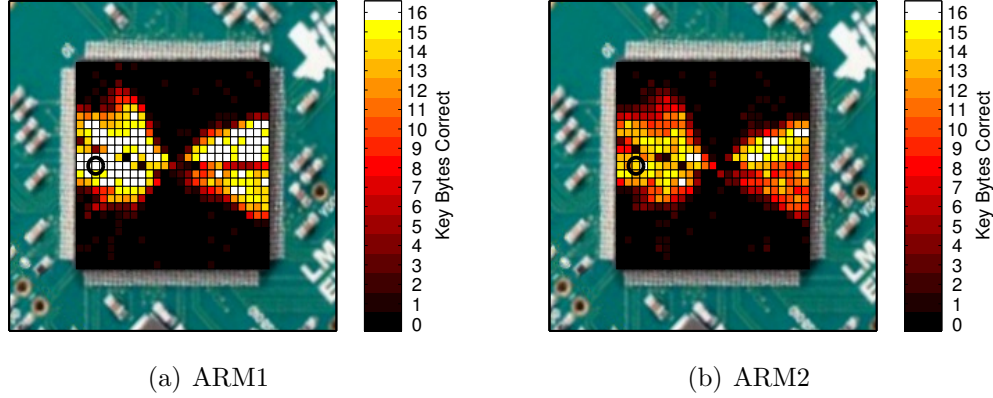
Figure 6.1　Comparison of the number of bytes correctly identified using $n_t = 1,500$ test traces to perform CEMA attacks on (a) ARM1 and (b) ARM2. CEMA attacks are more effective for ARM1 with more locations that recover 15 or 16 bytes. The box location represents the location of the center of the probe. The black circles in (a) and (b) denote collection location 303 used to collect training traces for template attacks.

to operations unrelated to the encryption operation being performed on the device. To determine if the differences are due to the data being processed, multiple traces can be collected with a fixed key and fixed plaintext[1].

The PSD of a non-periodic signal $x_T(t)$ observed only in the interval $(-T/2, T/2)$, with a proper Fourier transform $X_T(f)$ is defined in the limit as [115]

$$G_x(f) = \lim_{T\to\infty} \frac{1}{T} |X_T(f)|^2.$$

To calculate an approximate value for the PSD, the Fourier transform $X_T(f)$ is estimated using the Fast Fourier transform (FFT). The magnitude of the FFT is squared and divided by the number of samples in a trace multiplied by the sampling frequency. Finally, since the magnitude of the PSD is symmetric around zero, it can be represented as a single-sided PSD by doubling the magnitude for frequencies

---

[1]Identifying differences in frequency content between traces was inspired by viewing the signals near $f = 31.9$ MHz and $f = 63.7$ MHz (on ARM1) change frequency in a software defined radio waterfall display. Additional details are found in Chapter 7.
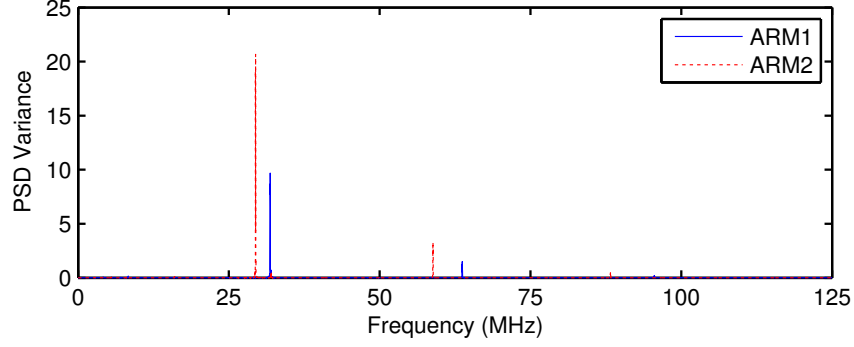
Figure 6.2   Variance of Power Spectral Density for traces collected from ARM1 and ARM2. Only $n_t = 10$ traces are used to identify the frequency of signals that change in power level between traces.

between 0 and $f_s/2$. Next, variance of the trace PSDs for test traces and training traces are calculated separately.

Finding the variance of PSDs for traces from a device, is a simple and effective way of identifying the frequency of signals unrelated to the encryption operation on that device. A set of of $n_t = 20,000$ training traces are collected from ARM1. Sets of $n_t = 60,0000$ test traces are collected at location 303 on both ARM1 and ARM2. All traces are collected at $f_s = 250$ MSa/sec. Since template attacks try to minimize the number of traces used in the attack phase, it is more realistic to use a small number of traces to find the variance of the PSDs. The variance of the PSDs for the first $n_t = 10$ traces from each device is shown in Figure 6.2.

The frequencies that have high power variance between traces are $f_1^{ARM1} \approx$ 31.9 MHz and $f_2^{ARM1} \approx 63.7$ MHz for ARM1 and $f_1^{ARM2} \approx 29.4$ MHz and $f_2^{ARM2} \approx$ 58.8 MHz for ARM2. Although the source of these signals is unknown, the signals are not related to the data being processed by the device. Notch filters can be used to reduce the contributions of signals at these frequencies in the collected traces. Notch filters are implemented in series to eliminate each of these frequencies from both the training data and test data.
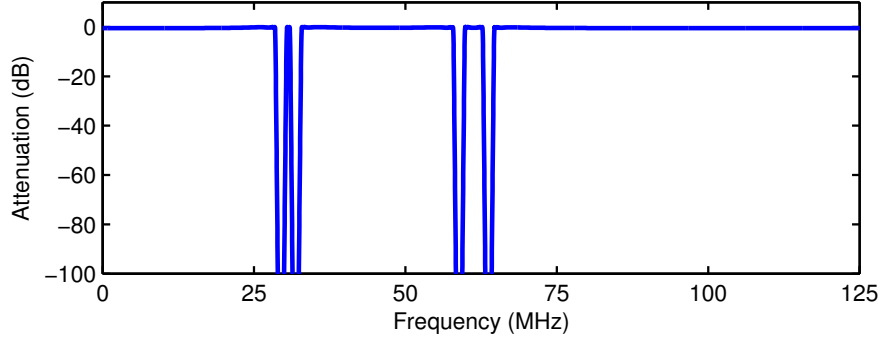
Figure 6.3   Plot of the magnitude of the impulse response for four notch filters in series.

As described in Section 3.3.1, for Chebyshev Type I stopband filters the cutoff frequencies define the passband. The cut-off frequencies are set to ± 1 MHz for each frequency identified in Figure 6.2. Figure 6.3 shows the magnitude of the filter series impulse response in the frequency domain. For the remainder of this chapter, *filtering* refers to filtering both the training data and test data with a series of notch filters. For cross-device attacks, the filter attenuates the contributions of signals around f_1^{ARM1}, f_2^{ARM1}, f_1^{ARM2}, and f_2^{ARM2}. For same-device attacks, the filter attenuate signals around f_1^{ARM1} and f_2^{ARM1}.

6.3.3   Combining Techniques.   After filtering both the training and test trace sets, template attacks are performed. The methodology for template attacks is described in Section 3.6. Known-key CEMA, described in Section 3.4.3, is performed on the $n_t = 20,000$ training traces to identify the 80 time samples that are the most highly correlated with a Hamming Weight model for each 8-bit output of SubBytes. Although the ARM uses the T-Box approach, template attacks that used the output of SubBytes as the target intermediate value produce results superior to attacks targeting the 32-bit output of the T-Box. The template training and classification phases are unchanged.

When used with filtering, the MVN technique is applied after filtering. The known-key CEMA step can be performed before or after the MVN technique. Tem-

plate attacks using training and test traces from location 303, are compared for each combination of these two techniques in the following section.

6.4   Results

6.4.1   Effectiveness of Cross-Device Methods.

To test the effectiveness of the filtering and the MVN technique on the ARM Cortex-M4F processors, $n_t = 20,000$ training traces are collected for ARM1 and $n_t = 60,000$ test traces are collected from both ARM1 and ARM2. All traces are collected using a Riscure low-sensitivity probe at location 303, as described in Section 3.1. Templates are build using all $n_t = 20,000$ training traces. To repeat the template attacks 1,000 times, the order in which test traces are used in the classification phase is randomly assigned. The same trace order is used for each of the pre-processing techniques, making the attacks identical except for the preprocessing technique used. Up to 1,000 traces are used in the classification phase. The following preprocessing techniques are evaluated: none[2], notch-filtering, MVN and notch-filtering with MVN. Since the filtering may change the structure of the traces (including sample mean and variance), the known-key CEMA-based distinguishing feature identification and MVN are performed after filtering. All test traces are used to estimate the mean and variance when performing the MVN technique.

The maximum-likelihood (ML) decision rule in (3.15) is used to select the key-byte in the classification phase of the template attack. If the correct key-byte has the highest posterior probability for all key-byte guesses for a given number of traces, the template attack is successful for that number of traces. Since the success of a template attack depends on the quality and order of the training and test data, not every trial will necessarily be successful for a given number of traces. To compare the preprocessing techniques, the number of trials needed to achieve a 90% attack success rate for each byte is shown in Figure 6.4.

---

[2]A standard template attack has no pre-processing.
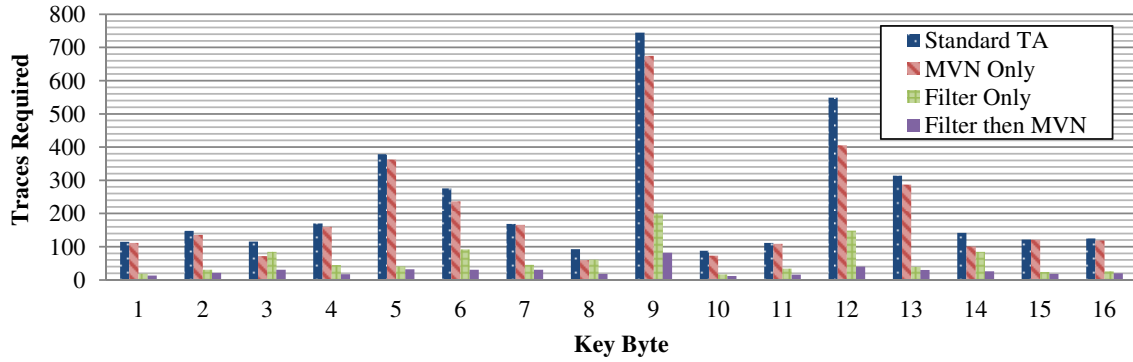
Figure 6.4   Comparison of the number of test traces need to achieve a 90% success rate for template attacks for each key-byte using test traces from ARM2 with ARM1 training traces using the indicated pre-processing technique.

It is clear from Figure 6.4 the correct value of some key-bytes are easier to extract than other key-bytes. For the standard template attack, 745 test traces are required to achieve the desired 90% success rate for key-byte 9. Applying the MVN technique reduces the number of test traces to 675, but filtering has the biggest impact. With filtering only 200 traces are required. When filtering is followed by the MVN technique, only 82 test traces are needed to achieve the 90% success rate. The number of test traces required is reduced by 88.9%. Across all key-bytes, the number of test traces required is reduced between 73.3% and 92.7%, with an average reduction of 85.8%.

To examine the benefit pre-processing methods have on the effectiveness of the cross-device template attack, the percent of attacks successful for key-bytes 9 and 10 are compared in Figure 6.5. These key-bytes were chosen because they have the worst and best performance of the standard template attack. Although the MVN technique results in a slight improvement, filtering results in a dramatic improvement in the percent of attacks successful. Applying the MVN technique after filtering continues to improve the attack success rate for key-byte 9, but has little effect on key-byte 10. For key-byte 9, with 82 traces the standard template attack is only successful for

(a) Key-Byte 9	(b) Key-Byte 10
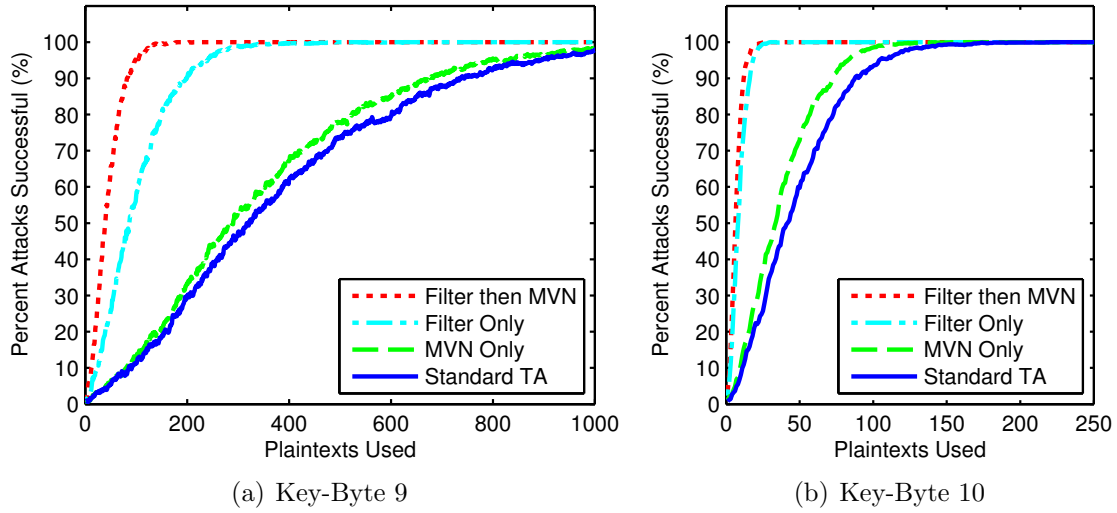
Figure 6.5   Comparison of the percent of 1000 trials correct for each type of cross-device template attack for (a) key-byte 9 and (b) key-byte 10 using the indicated number of test traces. Training traces are from ARM1 and test traces are from ARM2 (location 303).

8.2% of the attacks, with filtering and the MVN technique the attack us successful in 90% of the 1000 attacks. For key-byte 10, with 12 traces the standard template attack is only success for 10.9% of trials but with filtering and the MVN technique, the success rate improves to 90%.

The overall key-byte extraction success rate for same-device and cross-device attacks are shown in Figure 6.6. Filtering the traces results in a slight improvement in the percentage of bytes correct when ARM1 training traces are used to attack ARM1. Since training and test data are from the same device, MVN has no effect and the results are omitted from Figure 6.6(a). For the cross-device attack, using training traces from ARM1 and test traces from ARM2, the overall key-byte extraction success rate for template attacks with different pre-processing techniques are shown in Figure 6.6(b). For a 90% byte extraction rate, 264 traces are needed for a standard template attack, but only 30 traces are needed if filtering and the MVN technique are used. All trials are correct when 625 traces are used with filtering. Only 212 traces are needed when filtering and MVN are combined. With the MVN

(a) Same-Device   (b) Cross-Device
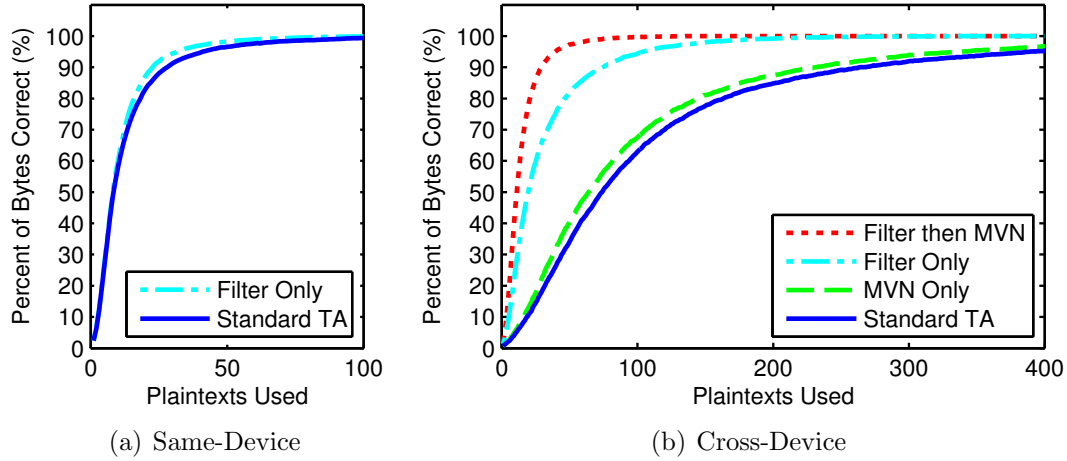
Figure 6.6   Comparison of the percent of all bytes correct in 1,000 trials for each
pre-processing technique for same-device and cross-device attacks using
ARM1 training traces. In (a) ARM1 is attacked and in (b) ARM2 is
attacked. Since the MVN technique does not change the success rate for
the same-device attacks, processing techniques that use it are omitted
from (a).

technique, 2079 traces are required to achieve a 100% success rate for 1000 trials.
Standard template attacks required 2293 traces.

6.4.2 Probe Position Tolerance.   The MVN technique was shown to be
effective at compensating for differences between devices in Chapter 5, but it can also
be used to tolerate changes in the positioning of the probe above a device. Ideally,
both the training and test traces will be collected from the exact same location above
the device. To test the effectiveness of the MVN technique to mitigate the effects of
poor probe placement repeatability, template attacks are conducted using the test
data collected at location 303 above ARM1 and the training data collected at each
of the 625 locations above both ARM1 and ARM2.

Probe placement repeatability is important for cross-device template attacks.
For same-device attacks, test and training data can be collected without moving the

144

probe. For cross-device attacks, the probe (or target device) must be moved and it is possible the probe may not be placed in the same location above each device.

Template attacks are performed using all $n_t = 20,000$ training traces from ARM1 location 303 and $n_t = 2,500$ test traces from each location on ARM1 and ARM2. The number of traces required before the correct byte is permanently identified using the ML rule is determined from the posterior probability for each key-byte guess. For the purposes here, *permanently identified* means that no other key-byte guess has a higher posterior probability than the correct key-bytes even as additional traces are added to the Bayesian classifier. The use of posterior probabilities to identify the correct key-byte value was discussed in Section 3.6.6.

Same-device and cross-device attacks are conducted using each pre-processing technique and a summary of the results are shown in Figure 6.7. The color of the box indicates the mean number of traces needed before the template attack identified the correct key-byte for each of the 16 bytes of the AES-128 key. The location of the box indicates where the test traces were collected. Only attacks which yielded all 16 of the key-bytes correctly are shown. To improve the contrast for attacks that required less than 500 traces, all attacks that required 500 or more traces are listed as *500+*.

Same-device attacks with and without the MVN technique are shown in Figures 6.7(a) and (b). Using the MVN technique both increases the number of locations where all 16 bytes can be extracted from 91 to 140, and increases the number of locations where less than 100 traces are needed on average from 53 to 76.

Cross-device attacks for each combination of filtering and the MVN technique, are shown in Figures 6.7(c)-(f). As with the same-device attack, the MVN technique increases the number of locations that template attacks yield all 16 key-bytes correctly from 84 to 135 (compare Figures 6.7(c) and (d)). The number of traces required is higher for the cross-device attack for both the standard template attack and attacks with the MVN technique than for same-device attacks. For the standard

(a) Same-Device Standard TA

(b) Same-Device MVN

(c) Cross-Device Standard TA

(d) Cross-Device MVN

(e) Cross-Device Filtering

(f) Cross-Device Filtering then MVN
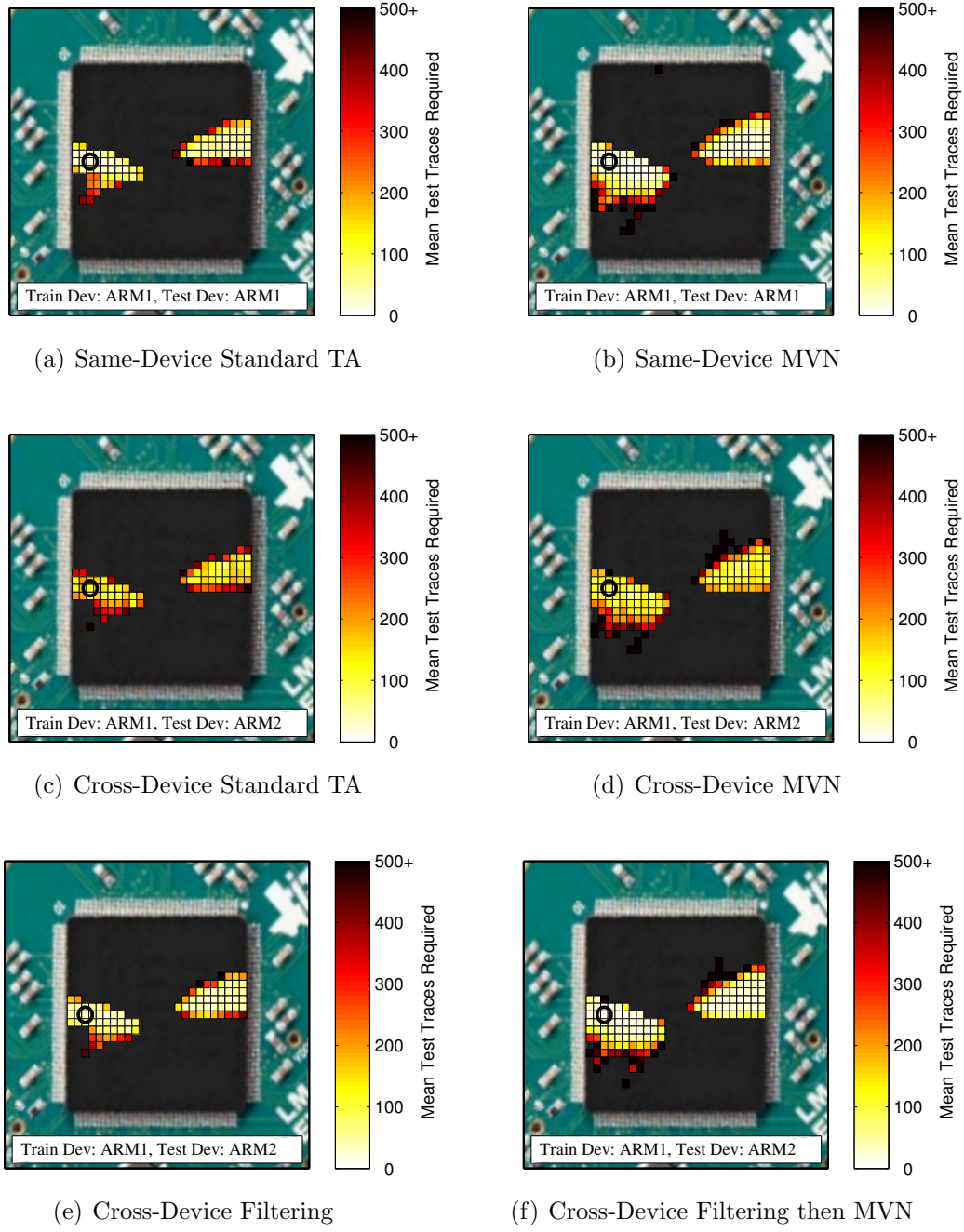
Figure 6.7   The mean number of traces required per byte for successful template attacks. All attacks are performed with training data collected at location 303 from ARM1 (denoted by black circles). Plots (a) and (b) are same device attacks. Plots (c)-(f) are cross-device attacks. The location of each square represents the location of the probe when collecting test traces. Only locations where all 16 bytes are identified with less than $n_t = 2,500$ test traces are shown.

146

cross-device template attack, only 4 locations required a mean number of traces less than 100. The number of locations is increased to 18 when the MVN technique is used.

In the cross-device attack shown in Figure 6.7(e) using filtering only, the average number of test traces required per key-byte is significantly reduced. The number of locations that yield all 16 bytes is increased to 89, and for 55 locations the mean number of traces required is less than 100. In Figure 6.7(f), both filtering and the MVN technique are used to improve the cross-device attack. With both techniques, the number of locations with all 16 key-bytes correct increases to 139 and the number of locations with a mean number of traces less than 100, increases to 79. These results are comparable to the success rate of the same-device attack with the MVN technique.

6.4.3 Comparison of Successful CEMA and Template Attacks Locations.
Using the MVN technique on a same-device attack increases the number of locations that can be used for a successful template attack. Since template attacks typically require less test traces than CEMA attacks, the number of locations with successful results should increase for the template attack. To compare the attack using an equal number of traces, the CEMA attack is repeated using all $n_t = 2,500$ test traces from ARM1. The number of bytes correctly identified for the CEMA attack and the same-device template attack are shown in Figure 6.8. While there are regions of the device where the template attack is more successful than the CEMA attack, there are also large portions of the device where the CEMA attack is successful, but the template attack is not.

There are at least two reasons why the template attack could fail. The distinguishing features could be different, or the distribution of the samples for the distinguishing features could be different. Although the MVN technique compensates for differences in the mean and variance of collected traces, it does not compensate for

147

(a) CEMA Attack (b) MVN Template Attack
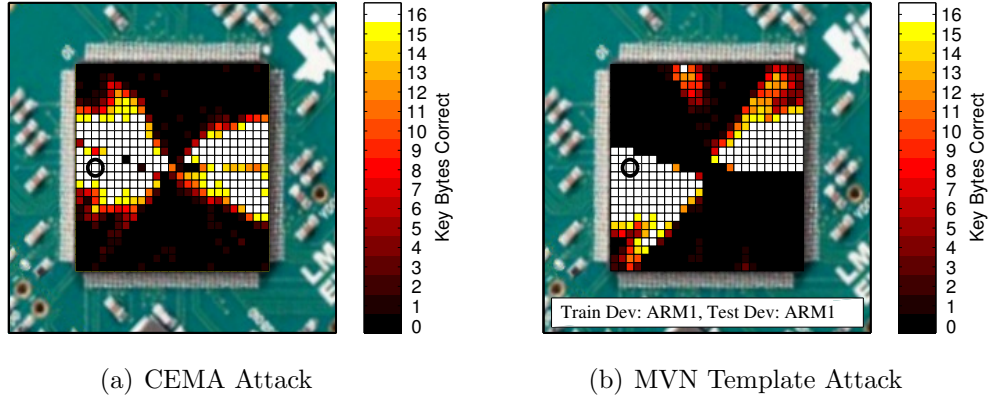
Figure 6.8   Comparison of (a) the number of bytes correct for CEMA attacks and (b) the number of bytes correct for template attacks with the MVN technique. All attacks use $n_t = 2,500$ test traces from ARM1. Training data is from ARM1 location 303 (denoted by black circles).

other differences in the leakage distribution. To evaluate the difference in leakage across the device, training data is collected at location 202. Known key CEMA is performed, and the top 80 highest correlated samples are compared with the top 80 highest correlated samples from location 303. Out of the 80 samples chosen for each location, 65 of the samples are the same. With the large number of test traces, the attacks should still be successful with this many distinguishing features in common.

Since the CEMA attack works in locations the template attack does not, the leakage in other regions of the device must still be correlated with the HW of the data being processed. To make the CEMA attack be effective on various microprocessor architectures, the absolute value of the correlation coefficient is taken before identifying the samples with the highest correlation magnitude. As a result, leakage that is both negatively or positively correlated with the HW of the targeted intermediate values can be used to identify the correct key-byte using (3.3).

In a template attack, the difference between negatively and positively correlated leakage can be compensated for by applying the MVN technique and multiplying the test data by -1. This is referred to herein as the *negative MVN technique.*

(a) Negative MVN Template Attack

(b) Combined Results
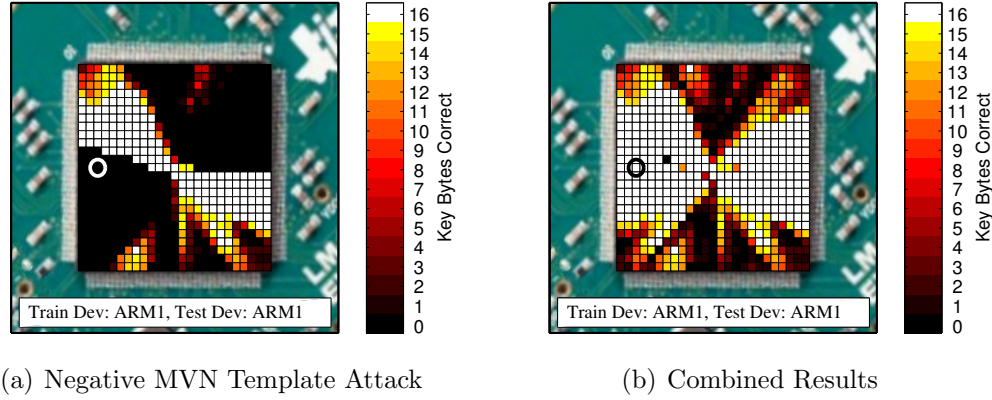
Figure 6.9   The number of bytes correctly identified using template attacks on ARM1 with $n_t = 2,500$ test traces at each location and training traces from location 303 (denoted by circle) when (a) the negative MNV technique is used on each test trace set, and (b) the negative MVN or MVN techniques are each used separately and the best result at each location is reported.

Using the MVN technique on training data from location 303 and the negative MVN technique on each set of test data, template attacks are repeated for all 625 locations. As expected, the locations that yield successful attacks, shown in Figure 6.9(a), are in areas that were not successful in Figure 6.8. The combined results for both techniques are shown in Figure 6.9(b). As expected, the combined attacks are successful in all the locations the CEMA attacks were successful (and more).

6.4.4   Notch-Filtering for CEMA Attacks.   Applying the notch-filtering technique to CEMA attacks significantly degrades attack performance. The bandpass filtering technique developed in Section 3.3.1 can improve the effectiveness of CEMA attacks. This approach is applied to ARM data in Chapter 7. Although bandpass filtering can dramatically improve CEMA attacks, bandpass filtering could not be shown to improve template attacks. All attempts to perform a template attack on bandpass filtered traces failed when the matrix inversion used to construct the covariance matrix could not be performed. This problem may be avoided by reducing the number of distinguishing features through principal component analysis or

multiple discriminant analysis to ensure the matrix of distinguishing features has full rank, or by constructing mean only templates. This is an area for future research.

6.5 Conclusion

The methods developed and tested in this chapter, to identify and filter out signals that are not consistently present in the collected traces, are simple and effective. Although template attacks incorporate the noise from a training device into the template [24], noise present in only the training or test trace sets can reduce the effectiveness of the attack. By calculating the variance in the PSDs of the collected traces, the frequencies of interfering components can be identified. Notch filters can effectively reduce the contributions of the signals at those frequencies in the collected traces, improving template attack performance.

Although little effect is seen for same-device attacks, notch-filtering dramatically reduces the number of traces required to achieve a 90% success rate for cross-device template attacks on the ARM Cortex-M4F. Filtering reduced the average number of traces required to attack each key-byte by an average of 69.3%. Filtering followed by the MVN technique reduced the number of traces required by 85.8%. For key-byte 9, the average number of traces required was reduced from 745, to just 82.

On the ARM Cortex-M4F, portions of each device have EM leakage negatively correlated with the HW of the data being processed, while other portions of the device have leakage positively correlated with the HW. Rather than collect data from two training locations, the negative MVN technique can be used. Use of the MVN and negative MVN technique makes template attacks more practical when probe placement can not be replicated or a new set of training data cannot be collected from a training device.

When the results from the MVN technique and the negative MVN technique are combined, the number of test locations where all 16 key-bytes are correctly

identified increased from 91 to 297, an increase of 226% vs. a standard template attack.

Although these techniques may not result in the same level of improvement for all cross-device attacks, since they use the same data collected for a standard template attack, they can be tried if the standard template attack fails to produce adequate results.

7. Differential Electromagnetic Attacks on a 32-bit Microprocessor Using Software Defined Radios

This chapter contains results submitted to the *IEEE Transactions on Information Forensics and Security* in a paper titled "Differential Electromagnetic Attacks on a 32-bit Microprocessor using Software Defined Radios". The article was coauthored by Dr. Rusty Baldwin and Dr. Michael Temple. For incorporation into this document, notation has been updated, and background and methodology that appeared previously in the dissertation have been removed from this chapter to avoid redundancy. The sections containing the removed information are referenced.

7.1 Introduction

Side-Channel Analysis (SCA) can extract sensitive information from power consumption [67] and Electromagnetic (EM) emissions [94] of cryptographic devices, including the cryptographic key used during encryption operations. Differential analysis determines the secret key used in multiple encryption operations by calculating statistics using observed side-channel traces from a cryptographic device and the plaintext or ciphertext associated with each trace [67]. To collect well aligned traces from multiple encryption operations, the cryptographic device is often modified to produce a trigger signal when the device is starting an encryption operation [73]. This trigger, of course, dramatically improves the alignment of the collected traces.

Agrawal et al. evaluated the leakage in EM signals, showing they contain a multiplicity of compromising signals, many of which can be used independently to break cryptographic implementations [2]. Observing that many compromising signals have very low energy, Agrawal et al. recommends separating those signals with useful information early in the acquisition process to negate precision limitations of signal capturing equipment. Receivers can be used for this purpose and although the receiver used for collections in [2] is not specified, the Dynamic Sciences R-1550

152

receiver and the Walkin-Johnson 8716 receivers are noted to be particularly effective receivers because they have a wide frequency range and bandwidth. Unfortunately, these receivers are also very expensive.

A less expensive approach is to sample the intermediate frequency output of a wide-band receiver using an oscilloscope and perform demodulation in software [2]. An oscilloscope could also directly sample the collected emissions at a rate greater than twice the targeted harmonic of the carrier frequency and the modulated signal can then be separated using software, but this approach does not provide the signal isolation of using a receiver before digitizing the signal.

In addition to a baseline attack performed using Correlation-based EM Analysis (CEMA) for traces collected using an oscilloscope, this paper evaluates the effectiveness of using SDRs to collect side-channel information. Software defined radios contain a Radio Frequency (RF) front end including a band-pass filter, RF amplifier and mixer to convert the signal to the intermediate frequency, followed by an analog-to-digital converter. Depending on the implementation, processing of the digitized signal is performed by a dedicated processor on the SDR or a general purpose processor, such as a personal computer.

The SDRs herein perform both the receiver and digitization functions, sending real-time observations of the side-channel to a PC, eliminating the need for an oscilloscope. The cost of performing side-channel analysis is significantly reduced by eliminating the oscilloscope, but the narrow bandwidth collected using an SDR and low number of samples per encryption operation increase the number of traces needed to perform successful side-channel attacks. However, since the sampling frequency is also reduced, trace information can be collected in real-time providing for continuous collection. Continuous collection allows attacks to be performed without modifying the device to add a trigger or training a real-time trigger generation device to create an external trigger. Despite sampling at rates well below the Nyquist rate, the encryption key can be successfully identified using SDRs.

Since this paper focuses on alternative collection techniques and determining the information contained at specific frequencies, countermeasures are not considered. An unprotected 32-bit ARM Cortex-M4F processor validates the SDR collection process. In both the oscilloscope-based and SDR-based analysis, differences in the frequencies at which key byte information is leaked from the device is observed.

This paper is organized as follows. Section 7.2 provides a brief overview of AES and correlation-based differential attacks. Related work is outlined in Section 7.3. The baseline attack is explained in Section 7.4, the SDR-based attack is explained in Section 7.5 and results are presented in Section 7.6.

7.2   Background

Devices running AES can be exploited using differential side-channel attacks. The goal of differential side-channel attacks against AES is to determine the secret key by measuring and analyzing the small statistical influence the computation of intermediate values has on the power or EM side-channel [68]. The CEMA attacks performed in this chapter are described in Section 3.4.

The targeted 32-bit ARM Cortex-M4F microprocessor performs AES-128 using the T-Box method described in Section 2.2.2 and [38]. The T-Box combined the SubBytes, ShiftRows and MixColumns operations into four 8×32 bit lookup tables. After performing the initial AddRoundKey, T-Boxes are used to calculate the first 9 rounds of AES-128. However, since the MixColumns operation is not performed in the last round of AES, the SubBytes implementation is used.

7.2.1   Triggering and Alignment.   When a powerful attacker has complete control of the cryptographic device, a hardware trigger is often added to improve collections [73]. The device is modified to produced a signal on an I/O pin when an encryption operation begins and/or ends. This signal is used to trigger the oscilloscope, resulting in collected traces that are closely aligned. In properly aligned

154

traces, corresponding parts of the encryption operations occur at the same sample in each trace.

In cases were a trigger is not added, but the attacker still has control of the cryptographic device, the oscilloscope can be triggered separately to capture a trace. Since the trigger is not created by the cryptographic device, the trace may not start at the same point relative to the start of each observed encryption operation and alignment techniques must be used [73].

If the attacker has no control over the start of an encryption operation, but is able to record at least one example encryption operation, a separate device capable of real-time pattern detection can be used to generate a trigger [106].

Even when a hardware trigger is used, post-collection alignment techniques can improve the alignment of the traces. Calculating the cross correlation between a reference trace and the trace being aligned, the point with the highest value indicates the offset between the two traces. Correlation-based alignment is used to align traces in both the baseline oscilloscope and SDR collected traces.

7.2.2 Software Defined Radios. Once restricted to military and academic applications, SDRs are now common in mobile communication networks [133], digital TV and FM reception [74]. Receivers typically use a variable frequency oscillator, mixer and filter to isolate and shift the target RF frequency to an Intermediate Frequency (IF) or baseband where it is amplified and sampled by a analog-to-digital converter (ADC) [133]. A Low Noise Amplifier (LNA) may amplify the RF signals before converting the RF signal to the IF or baseband. Alternatively, low frequency RF signals[1] may be sampled directly and down-converted digitally.

After the ADC, some or all of the signal processing is done in software. A number of free and/or open source software development toolkits are available to implement signal processing blocks in software including GNU Radio [19], and High

---

[1]Frequencies less than $\frac{1}{2}$ the sampling rate of the ADC

Definition Software Defined Radio (HDSDR) [126]. The interfaces used to collect the SDR traces used in the CEMA attacks are described in Section 7.5, but the HDSDR is used for spectrum analysis of the target microprocessor. The RF spectrum and waterfall displays are used to identify signals from the device that interfere with SDR-based collections.

7.3 Related Work

Cryptographic Research, Inc. demonstrated simple EM attacks on implementations of RSA and Elliptic Curve Cryptography (ECC) on smart phones [61]. The RSA attack was performed using a near field probe, and the ECC attack with a far-field antenna. A receiver demodulated the signal and an SDR was used as a digitizer. Additionally, it was shown that AES operations can be observed in the demodulated signal, but an attack is not performed on AES. No other examples of using SDRs for collecting SCA data have been found in literature.

Agrawal et al. explored how leakage changes across the EM spectrum by performing differential attacks on demodulated EM signals [2]. A Difference of Means attack (DoM) [67] was conducted on a single bit of a smartcard with a 3.68 MHz clock frequency performing the Data Encryption Standard (DES). The signal from a near-field probe was amplitude demodulated using a receiver for center frequencies of 188 MHz, 224.5 MHz, and 262 MHz with a bandwidth of 50 MHz. The demodulated signals were collected with a 12-bit, 100 MHz digital oscilloscope. Agrawal et al. found both the magnitude of the DoM results and time at which leakage occurs are affected by the carrier used.

Barenghi et al. identify the frequencies at which a device leaks information by performing a correlation-based differential attack on filtered power consumption data [13]. Focusing on a single key byte, they show that creating a filter with multiple passbands around harmonics of the clock frequency can reduce the number of traces required to determine the correct value for the byte. They expand this technique to

look at other frequencies in [14]. In both cases they average multiple traces for each plaintext.

The correlation-based frequency-dependent leakage analysis method developed in Section 3.5.2 in contrast does not average collected traces. To average traces, an attacker must be able to observe and align multiple encryption operations with the same plaintext. Since the goal of the SDR-based attack is to perform a passive attack with no device modification, the ability to average traces is not assumed. While Barenghi et al. focused on a single key byte, all key bytes will be attacked. Analysis shows that frequencies at which key bytes leak information can change from byte to byte and can even change if the device is reprogrammed. Since power consumption data cannot be collected without modifying the device, only EM emissions are collected.

7.4 Baseline Attack Performance

The target device is the LM4F232H5QD evaluation kit with ARM Cortex-M4F based microcontroller denoted ARM1 in Section 3.2.2. The baseline attack is performed using test traces collected with an oscilloscope sampling at $f_s = 2.5$ GSa/s as described in Section 3.1. The traces are downsampled to an effective sampling frequency of $f_s^D = 250$ MSa/s.

CEMA is used to identify the most likely value for each of the 16 bytes in the AES-128 key, as described in Section 3.4.1. Attacks targeting both the output of SubBytes and the output of the T-Box in the first round of AES can be successfully performed against the target microprocessor. However, since attacking the output of the T-Box yields the highest key extraction success rate, the T-Box outputs in the first round of AES are the target intermediate values herein.

The target intermediate values for these attacks are the 32-bit output of each T-Box in the first round of AES-128. The EM leakage from the ARM Cortex-M4F follows a 32-bit Hamming Weight (HW) model. Analysis is performed using

157

known plaintext bytes, $\mathbf{t} = (t_1, t_2, ...t_{n_t})^T$, and the collected side-channel emissions corresponding for each of the n_t plaintexts. Elements of the hypothetical leakage matrix $\mathbf{H}$ are calculated $h_{i,j} = HW(T_r(t_d \oplus k_i))$ where r is the row of the state matrix, $d = 1, ..., n_s$ samples per trace, and $i = 1, ..., n_k$. Where $n_k = 256$ possible key values and n_s is the number of samples in each trace. Since only the HW is used in the CEMA attack, the row of the input byte in the state matrix can be disregarded as the HWs of the output of all four T-Boxes are equal for any given input.

7.4.1   Electromagnetic Cartography Scan.   The LM4F232H5QD package has multiple power and ground connections and it is unclear in documentation [130] which pin supplies the power to the portion of logic the cryptographic computations will be performed on. To determine the best location above the device to position the probe for collections, a $25 \times 25 = 625$ location XY scan is performed capturing emissions from the device while it repeats one encryption operation at each location, as described in Section 3.2.2.1. Since the amplitude of the collected emissions varies greatly between locations, the magnitude of the collected trace is evaluated and the vertical sensitivity of the oscilloscope is adjusted to maximize dynamic range. After adjusting the vertical sensitivity the trace is recollected and stored with its corresponding plaintext and volts/div setting. When used to calculate the Power Spectral Density (PSD), each trace is scaled by the volts/div setting used to collect the trace.

Figure 7.1(a) is a plot of the normalized maximum PSD value across the 625 locations above the device. The LM4F232 has a maximum clock speed of 80 MHz, but the clock was set to $f_{sys} = 50$ MHz for compatibility with existing UART interface code. Results in Figure 7.1(a) are based on the PSD calculated for frequencies between 49 MHz and 51 MHz.
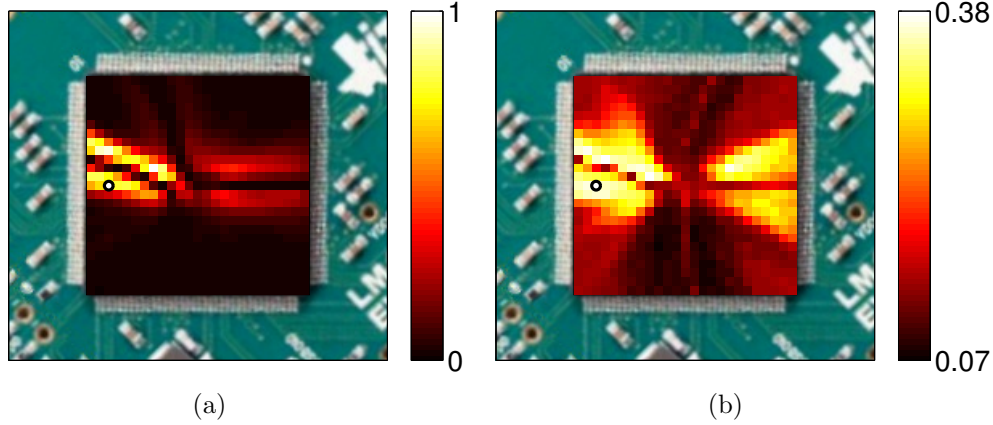
Figure 7.1   (a) Maximum Power Spectral Density (PSD) at $25 \times 25 = 625$ locations above the device package between 49 and 50 MHz. PSD values are normalized across all locations. (b) Mean magnitude of maximum correlation coefficient using $n_t = 1,000$ traces traces collected at $25 \times 25 = 625$ locations above the device package for each of the 16 AES key bytes.

To validate the effectiveness of using spectral intensity to determine the best location, the XY scan was repeated using a fixed set of plaintexts. A CEMA attack on the output of the T-Box is performed using the set of $n_t = 1,000$ traces from each location as described in Section 7.4. The average magnitude of the correlation coefficient for the correct key value for the 16 bytes in the AES-128 key are represented graphically in Figure 7.1(b). The locations identified using the PSD yield good results, but the correlation-based results find other locations above the device where collected traces have high correlation with the HW leakage model for each correct key byte. Plots of the minimum correlation and number of bytes correctly identified using (3.3) were also created, but are omitted here. Based on these plots, the position indicated by a small circle in Figures 7.1(a) and 7.1(b) was used for all oscilloscope and SDR collections herein. Using the grid system described in Section 3.2.2.1, this position is location 303.

7.4.2   Correlation-Based Frequency-Dependent Leakage Analysis.   To evaluate the information leakage of the ARM Cortex-M4F, a set of $n_t = 2,000$ traces with
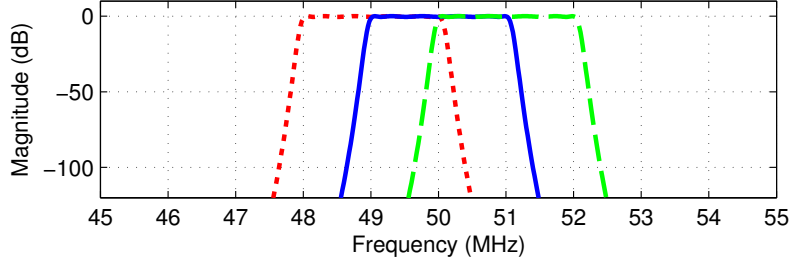
Figure 7.2   Magnitude of impulse frequency responses for 99 overlapping Cheby-
shev Type I filters bandpass filters with $W_{BW} = 2$ MHz used in the
correlation-based frequency dependent leakage analysis.

a fixed key and random plaintexts is collected and analyzed by filtering the traces
and performing CEMA. The traces were collected at $f_s = 2.5$ GSa/s. Since, ini-
tial filtering experiments showed frequencies below 100 MHz contained more leakage
when attacked using CEMA with a HW model than frequencies above 100 MHz, the
traces are downsampled to $f_s^D = 250$ MSa/s as described in Section 3.3.2 to make
them easier to process.

The goal of the baseline analysis is to determine which frequencies leak ex-
ploitable AES-128 key information from the ARM Cortex-M4F implementation to
aid in choosing a center frequency and sampling rate for the SDRs. Since the SDRs
will ultimately be used to collect at a single frequency, the filters are not combined
to create a multi-bandpass filter as in [14]. To ensure low attenuation at the cut-
off frequencies of the bandpass filters, twelfth-order Chebyshev Type I filters are
implemented with a passband ripple of 0.1 dB as described in Section 3.5.2. The
cutoff frequencies of the filters overlap by 50% to prevent gaps between the filters.
A plot of the magnitude of the impulse response in the frequency domain for three
overlapping filters with $W_{BW} = 2$ MHz is shown in Figure 7.2.

Since SDRs have much lower maximum sampling rates than oscilloscopes,
the filter bandwidth is chosen based on sampling rates that can be achieved us-
ing low-cost SDRs. Using a bandwidth of $W_{BW} = 2$ MHz, center frequencies
$f_c = \{1, 2, ..., 99\}$ MHz, 99 filters are constructed. If the passband includes 0 or

100 MHz, a sixth-order low-pass or high-pass filter is used, otherwise the filter is a twelfth-order bandpass filter. The term *frequency interval* refers to the passband of a filter. A CEMA attack is performed for each of the 16 key bytes on traces filtered using a zero-phase digital filter. The value of the key byte selected by the attack is found using (3.3).

The CEMA attack determines the correlation between the hypothetical leakage for each key byte value guess. To determine the confidence in the key value selected using a CEMA attack, the maximum correlation coefficient, r_{max}, is compared with the next highest correlation coefficient, r_{next}), as defined in (3.4) and (3.5) respectively. The confidence $r_{max} \geq r_{next}$ is calculated for each CEMA attack using the trace set filtered for each frequency interval as described in Section 3.5.2.

7.4.3 Baseline Results. Calculating the confidence at which r_{max} and r_{next} are statistically different allows CEMA attacks performed on different bytes and for traces filtered using different frequency intervals to be compared directly. Figure 7.3 shows the statistical confidence found using (3.10) for the 99 filters constructed with $W_{BW} = 2$ MHz and center frequencies $f_c = \{1, 2, ..., 99\}$ MHz. Higher statistical confidence is represented by lighter colors. Since the same trace set is filtered each time and the bandwidth is constant, the differences reflect confidence variation due to varying the center frequency of the filters. For some frequency intervals, even when the number of traces is increased, some key bytes cannot be extracted while other key bytes can be (see $n_t = 2000$ results in Sec. 7.6.3). Although high confidence does not guarantee the correct key byte value is chosen, in Figure 7.3 the highest p-value where an incorrect byte was chosen using (3.3) was 0.8252. If the CEMA attack using traces filtered over a frequency interval yielded the incorrect value, an $\times$ is drawn through the box representing that attack in Fig. 7.3.

Note that confidence changes for both frequency interval and key byte. The correct value for some key bytes can be easily extracted, while the values of other
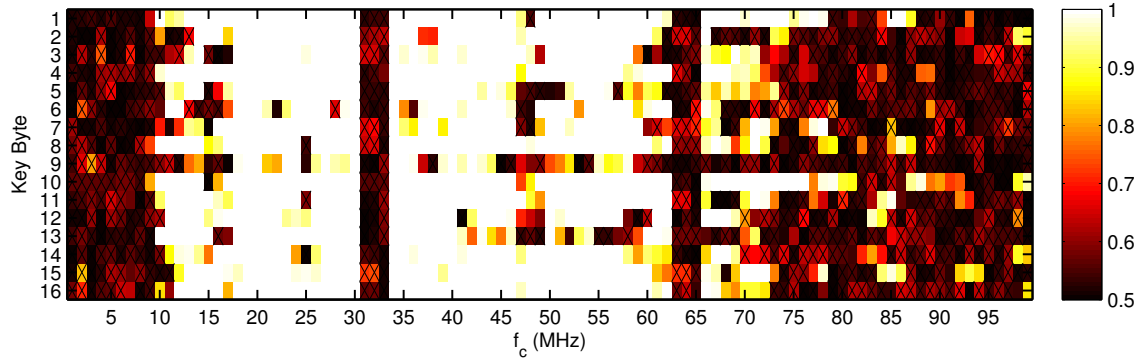
Figure 7.3   Colored boxes represent the confidence $r_{max} \geq r_{next}$ from each CEMA attack. Attacks are performed at each key byte using $n_t = 1000$ traces filtered using overlapping frequency intervals. Since $f_c \in \{1, 2, ..., 99\}$ MHz and $W_{BW} = 2$ MHz, the intervals overlap by 50%. Each box is centered at the correct f_c but their widths do not represent the actual W_{BW}. CEMA attacks that yielded incorrect key byte values are marked with an $\times$.

Table 7.1   Confidence $r_{max} > r_{next}$ using $n_t = 1000$ traces decimated to $f_s^D = 250$ MSa/s for key byte $k_i \in \{1, ..., 16\}$

k_i	Confidence	k_i	Confidence	k_i	Confidence	k_i	Confidence
1	1.00000	5	0.99998	9	0.98726	13	0.99996
2	1.00000	6	0.99990	10	1.00000	14	1.00000
3	1.00000	7	0.99970	11	1.00000	15	1.00000
4	1.00000	8	1.00000	12	0.99979	16	1.00000

key bytes may not be easily determined using the same set of filtered traces. For comparison, using the unfiltered traces the correct value is identified for all 16 key bytes. The confidence levels are listed in Table 7.1.

Some trends can be observed across all key bytes. For center frequencies near 31.9 MHz and 63.7 MHz the confidence $r_{max} > r_{next}$ is lower[2] Viewing the spectral waterfall display in HDSDR, there are unknown signals at these frequencies that

---

[2]The signal at 31.9 MHz and 63.7 MHz were also identified using the PSD variance technique in Section 6.3.2. Additional signals at 15.9 MHz and 47.7 MHz can be identified using the PSD variance method, but these signals have significantly less power than the signals at 31.9 MHz and 63.7 MHz and required a larger number of test traces identify them using this PSD variance method.

appear to vary in frequency; adding noise to the collected signal. Attacks performed using traces filtered with a passband filter which includes these unknown signals have lower r_{max} values and lower confidence. Testing isolated these signals to the development board, but their source is unknown.

Although the system clock of the ARM Cortex-M4F is set to $f_{sys} = 50$ MHz, frequency intervals with center frequencies above and below 50 MHz can be used to successfully attack the device. The target center frequencies for SDR collections are based on the baseline results in Fig. 7.3 and center frequency ranges of the two SDRs. The filtered traces for frequency intervals between 12 MHz and 60 MHz contain enough information to be able to extract a large majority of the 16 AES-128 key bytes with high confidence.

Since AES operations in the target device are performed with the T-Box implementation (as described in Section 2.2.2) the state matrix row determines which of the four T-Box tables are accessed in memory for each byte. There is no correlation between state row and the confidence with which a key byte can be extracted. The code is optimized to reduce execution time by the compiler. The effect of code optimization is evaluated in Section 7.6.3.

7.5   *Software Defined Radio Methodology*

Collecting differential side-channel traces using a SDR simplifies the collection process but requires additional post-collection processing. Since the SDR can collect data continuously, there is no need to modify the target cryptographic device to add a trigger. A near-field probe is placed just above the device and the SDR can immediately begin recording the emissions from encryption operations being performed. To allow comparison between the baseline oscilloscope-based collection and SDR-based collection results , the probe location is fixed at the location found in Sec. 7.4.1. However, an adequate location can easily be found by manually moving the probe over the device while monitoring the spectral intensity in SDR software.
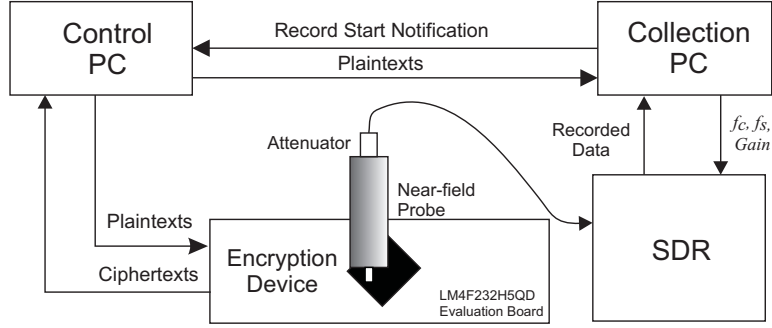
Figure 7.4   The collection setup uses two computers to simu-
late an attack scenario where the attacker is able
to place a probe on the encryption device to col-
lect the EM emissions corresponding to encryption
operations with known plaintext or ciphertexts.

Since the probe is amplified, and cannot be used without amplification, a 20 dB
attenuator is used to prevent damage to the SDRs.

When collecting using an oscilloscope, the PC used to collect and store the
traces from the oscilloscope also controls the target encryption device. To make
the SDR collection scenario more realistic, two separate PCs are used as shown in
Fig. 7.4. The *collection PC* continuously records the side-channel through the SDR
while the *control PC* requests the encryption device perform multiple encryption op-
erations. To verify the correct encryption operations are being performed, the control
PC receives the ciphertext from the ARM Cortex-M4F for the previous encryption
operation before sending the next plaintext for encryption.

To automate the collection of a large number of traces the collection PC and
control PC are connected via Ethernet. The collection PC notifies the control PC
when it is about to begin recording. The collection PC records for a given amount of
time and then retrieves the plaintexts from the control PC. Since a trigger is not used
to indicate the start or duration of individual encryption operations, each individual
encryption operation must be identified in the SDR recording via signal processing.

7.5.1 Sub-Nyquist Sampling. The baseline oscilloscope analysis indicated that signals with frequencies between 10 MHz and 70 MHz frequencies contain key information. The SDRs are used to target frequencies within this range. In order to perfectly reconstruct a bandwidth limited signal with spectral contents less than a maximum frequency f_{max}, the signal must be sampled at a rate of at least $2f_{max}$ [113]. This is known as the Nyquist rate. Since the SDRs sample at frequencies lower than the 20 MHz to 140 MHz Nyquist rates, all SDR collections will be *sub-Nyquist*. While many techniques have been developed to reconstruct signals sampled at sub-Nyquist rates using prior information on the signal structure [80], there is no need to reconstruct the signal to perform the CEMA attack. As an indicator of how far below the Nyquist rate the SDRs sample, the proportion of the Nyquist rate is calculated

$$N_q = \frac{f_s^D}{f_N},\tag{7.1}$$

where f_s^D and $f_N = 2f_{max}$ is the decimated sampling frequency output by the SDR. For simplicity, N_q is estimated with $f_N = 2f_c$.

We expect that lowering N_q will degrade the key byte extraction success rate, but successful attacks will still be possible. Additional traces may need to be collected to compensate for reduced N_q.

7.5.2 Software Defined Radios.

7.5.2.1 USRP. EM emissions are collected from the near-field probe using a Universal Software-Defined Radio Peripheral (USRP). The *USRP2* model uses dual 100 MSa/s 14-bit Analog to Digital Converters (ADCs) and interfaces with the collection PC via gigabit Ethernet. The USRP2 uses interchangeable daughter boards as the RF front end. Since the baseline test in Section 7.4 found compromising signals at frequencies less than 30 MHz, the LFRX daughterboard designed to receive

0-30 MHz is chosen. The LFRX can receive center frequencies up to 50 MHz, but filters the RF signal with a third-order low-pass filter with a cutoff of 30 MHz to prevent aliasing. The LFRX amplifies the RF signal using high-speed operational amplifiers [43]. Although the BasicRx daughterboard can sample in the DC - 50 MHz range, only the LFRX can be used without any external front end hardware.

A Digital Down-Converter (DDC) is implemented on the USRP Field Programmable Gate Array (FPGA) to down-convert the RF signal to baseband and decimate the signal. A Numerically-Controlled Oscillator (NCO) synthesizes the discrete-time, discrete-amplitude sine and cosine waveforms with frequency f_c within the FPGA. The sine and cosine functions are multiplied with the digitized samples from the ADC to produce In-phase Quadrature (I/Q) data and down-convert the center frequency of the collected signal to DC. Since only the magnitude of the baseband signal is use for SCA, one of the I/Q channels is filled with null-samples and only one 16-bit value is sent to the collection PC.

The USRP2 ADCs sample at $f_s = 100$ MSa/s. To achieve a decimated USRP2 output sampling rate of $f_s^D = 2$ MSa/s, the sampled RF signal is sent through a low-pass filter with a cut-off of $W_{LP} = f_{max}/n_d$ and decimated by $n_d = 50$. From the baseline test in Fig. 7.3, most key bytes appear to leak at frequencies between 18 MHz and 32 MHz. Using the USRP2, center frequencies between 18 and 40 MHz are targeted with sampling frequencies of both $f_s^D = 2$ MSa/s and $f_s^D = 4$ MSa/s.

7.5.2.2 *Low-Cost RLT-SDR.* Low-cost Digital Video Broadcasting–Terrestrial (DVB-T) Universal Serial Bus (USB) dongles can be used as SDRs. DVB-T dongles based on the Realtek RTL2832U[3] can transfer raw unsigned 8-bit I/Q samples to a host computer using alternative drivers [74]. The ezcap USB 2.0 DVB-T/DAB/FM dongle, henceforth referred to as the RTL-SDR, is used because it has the Elonics E4000, a highly integrated multi-band RF tuner integrated circuit im-

---

[3]The RTL2832U is a high-performance DVB-T Coded Orthogonal Frequency Division Multiplexing demodulator that supports a USB 2.0 interface [97]

plemented in CMOS [42]. The E4000 uses a direct conversion zero IF architecture, employing a single stage to mix the amplified and filtered RF signal to baseband. Before being digitized using a fast sampling ADC, the baseband signals DC offset is corrected, the signal is filtered using a low-pass filter and the signal is amplified. Although dongles with other tuners are available, the E4000 offers the widest frequency range [74].

According to specifications, the E4000 can accurately tune to frequencies between 64 and 1700 MHz [42], but can also be used out-of-spec from 50 MHz - 2.2 GHz [74]. In practice, the lowest center frequency the RTL-SDR dongle used for this research could be set to through the collection interface [32] was $f_c = 53.5$ MHz. Although the RTL-SDR's highest sampling rate is 3.2 MSa/s, rates less than 2.8 MHz are used to avoid sample loss when the I/Q data is transfered over USB 2.0 to the collection PC.

Since the targeted frequencies are down-converted to baseband before being digitized, the sampling rate is much lower than would be needed to sample the targeted frequencies directly. To minimize sample loss, the RTL-SDR was configured to sample at $f_s^D = 2.0$ MSa/s making the maximum frequency of the sampled baseband signals 1.0 MHz. From the baseline test in Fig. 7.3, there is at least one frequency interval above 50 MHz where each key byte has a high probability of $r_{max} \geq r_{next}$. Center frequencies between 53.5 and 73 MHz are used for collections with the RTL-SDR. The gain is set to 1.5 for all collections.

7.5.3 Identifying and Aligning Encryption Operations. Associating the correct plaintext to each recorded encryption operation is essential for an efficient differential attack. Since multiple encryption operations are being performed during each collection, the collection computer must be able to identify each operation. To simplify this problem, a set number of encryption operations are performed in each group. The control PC performs groups of $n_g = 250$ encryption operations using

randomly generated plaintexts and a fixed key. This number was chosen to make the processing of each SDR collection more manageable. The entire EM emission recorded by an SDR is referred to as a *collection*. To be consistent with SCA nomenclature, the portion of the collection produced by an individual encryption operation is referred to as a *trace*. Each collection should contain n_g traces. Since it is vital each plaintext is matched with the correct trace, if the collection PC cannot identify n_g traces in the collection, the collection and associated set of plaintexts are discarded.

The SDR collection not only contains the encryption operations of interest, but also all other operations being performed by the microprocessor. However, since the ARM Cortex-M4F is being used as a dedicated encryption device, when the device is not performing a key schedule operation or performing encryption or decryption operations it is waiting to receive or process a command.

When the collection device is triggered the portion of the collection where the encryption operation is performed is easily isolated and no other device activity need be collected. When recording continuously, device activity such as responding to interrupts, identifying commands, receiving the plaintext and transmitting the ciphertext are evident in the recorded trace. The magnitude of a USRP2 collection at $f_c = 22$ MHz and $f_s^D = 2$ MSa/s for the operations associated with one encryption operation is shown in Figure 7.5. Since the microcontroller waits for an interrupt between commands, distinct periods of activity can be seen before and during each encryption operation. The last period of activity in Figure 7.5 is where the encryption operation is performed.

Various methods can be used to identify the n_g encryption operations in the SDR collection. Using a manually identified reference trace, cross correlation can identify the start time of each encryption operation. However, in addition to requiring manual intervention for each center frequency and sampling frequency, this process is computationally intensive due to the length of the SDR collections. Since
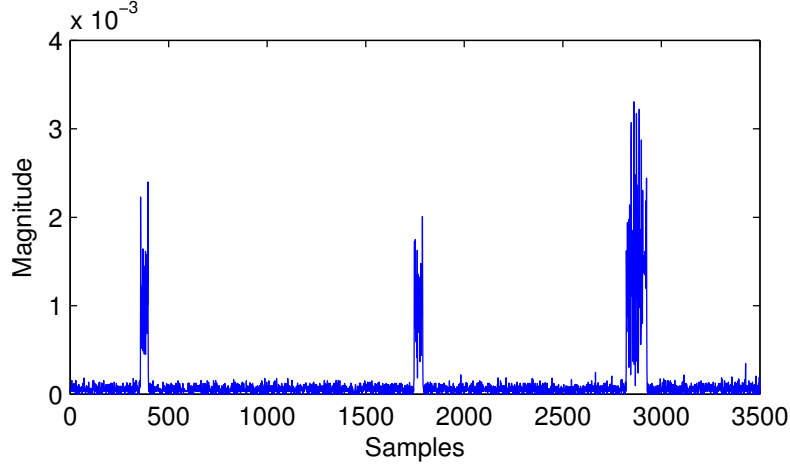
Figure 7.5   Magnitude of the data recorded for interrupt han-
dling, command identification, receipt of plaintext,
encryption of plaintext and returning ciphertext
for single encryption operation using the USRP2
with $f_c = 22$ MHz and $f_s^D = 2$ MSa/s.

the microprocessor is idle when not performing operations associated with an en-
cryption operation, faster detection is performed by counting and identifying the
location of n_d peaks with a minimum distance greater than the length of the opera-
tions shown in Figure 7.5 and a height greater than h_{min}. The initial value of h_{min} is
the overall maximum value in the collected trace. While $n_d < n_g$, h_{min} is gradually
lowered. If $n_d = n_g$, the collection is separated into n_g traces by retaining a fixed
number of samples before and after each identified peak. If $n_d > n_g$, the collection
is discarded.

The n_g traces should each contain three distinct periods of activity. Since only
the last period of activity contains the encryption operation, traces are truncated to
remove the first two peaks. Using the first trace as a reference, the remaining traces
are aligned by finding the offset that produces the highest cross correlation between
the trace being aligned and reference. The traces are circularly shifted to align
them. To show the similarity between aligned traces, 250 traces are superimposed
and shown with the mean of the 250 traces in Figure 7.6. Groups of n_g traces are

169
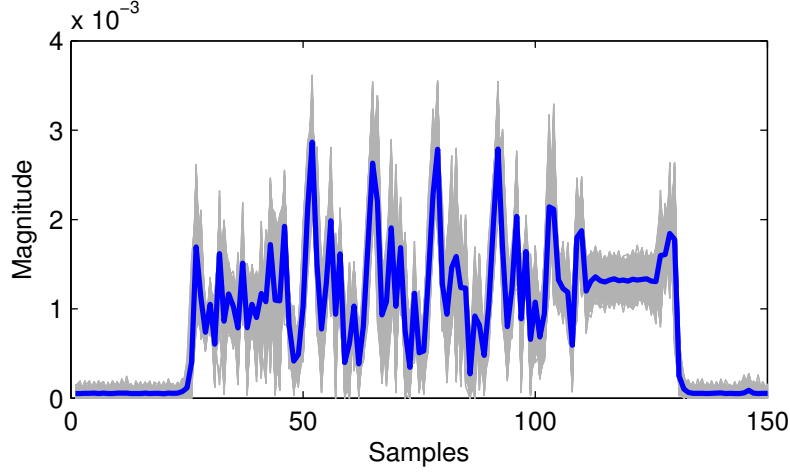
Figure 7.6   Two-hundred and fifty traces collected with the
USRP2 superimposed in grey with the mean of the
samples shown as a dark line for $f_c = 22$ MHz and
$f_s^D = 2$ MSa/s.

collected until number of collected traces reaches or exceeds the desired total number
of traces n_{total}.

The USRP2 and RTL-SDR both contain small memory buffers and samples
must be streamed to the collection PC as they are recorded. If for any reason the
SDR is not able to send the samples fast enough, an overflow occurs and samples
are discarded. As a result, it is possible for portions of an encryption operation
to be missing from the recorded SDR data. Missing samples can cause traces to
contain less than three periods of activity, or for a period of activity in the trace
to be shorter than expected. The USRP2 has an overflow indicator which allows
a collection to be discarded when an overflow occurs. However, collecting with the
USRP at $f_s^D = 2$ MSa/s and $f_s^D = 4$ MSa/s, no overflows occurred. Unfortunately,
the RLT-SDR interface [32] does not report overflows, and overflows do occur via
that interface. To make sure the traces collected with the RTL-SDR are usable,
additional processing steps were required.

7.5.4   Additional Processing for the RTL-SDR.   As discussed in Sec. 7.5.2, the tuner in the RTL-SDR is a single CMOS RF tuner. While the user can set center frequency, sampling frequency and gain before the ADC, the DC offset compensation is performed automatically by the E4000. Testing showed the E4000 does not consistently apply the same DC offset for each collection. Since multiple collections are used together for the CEMA attack, it is necessary to adjust the mean of each RTL-SDR collection to zero. The adjustment is performed on the entire collection, and not on individual traces.

For the target microcontroller there are up to three periods of activity associated with each encryption operation as shown in Figure 7.5. However, if samples are dropped, there may be fewer periods of activity in the retained trace or samples may be missing at one or more unknown point(s) in the trace. Since only the third period of activity contains the AES-128 encryption operation of interest, only samples missing within the last region of activity are of concern.

To allow for traces with less than three periods of activity to be used, the last region of activity is always assumed to be the encryption operation. The samples within this region are aligned using cross-correlation, and the width of the retained region from each trace is evaluated to verify it is consistent with the other traces in the group of $n_g = 250$ traces. If the width of the last period of activity for the encryption operation is greater than 5 standard deviations away from the mean region width, the trace and its corresponding plaintext are excluded from the CEMA attack.

7.6   Software-Defined Radio Results

7.6.1   USRP.   The USRP2 collects side-channel emissions from the ARM Cortex-M4F using $f_c \in \{15, 15.5, ..., 29.5, 30\}$ MHz with $f_s^D = 2$ MSa/s. The gain is fixed on the LFRX daughterboard. Not all center frequencies produced usable traces. The ARM Cortex-M4F has a number of clocks on the device. In addition

to the system clock which is divided to $f_{sys} = 50$ MHz (from 400 MHz) for the implementation of AES with UART communication, there is a Precision Oscillator (PIOSC) with frequency of 16 MHz. There are also strong signals at 16.67 MHz and 25 MHz which are believed to be due to operations that take two or three clock cycles to complete. The spectrum analysis display in HDSDR reveals that both the system clock and PIOSC have substantial clock jitter which makes encryption operation extraction from collections made with center frequencies near $f_c = 16.67$ MHz and $f_c = 25$ MHz more difficult. When the SDR down-converts $f_c = 25$ MHz to baseband the slight variations around $f_c = 25$ MHz become low frequency signals. Although a low pass filter can remove these signals from the collected trace, better results were achieved at frequencies other than $f_c = 16$ MHz or $f_c = 25$ MHz.

With a sampling frequency of $f_s^D = 2$ MSa/s the USRP2 collects $n_{total} = 100,000$ traces at the following center frequencies: $f_c \in \{18, 18.5, ..., 22.5, 23\}$ MHz and $f_c \in \{28, 28.5, 29, 29.5, 30\}$ MHz. For this range of center frequencies with $f_s^D = 2$ MSa/s, the signals are sampled at $1/18 < N_q < 1/30$ the Nyquist rate. CEMA attacks are performed using the first $n_t \in \{5000, 25000, 100000\}$ traces. The confidence $r_{max} \geq r_{next}$ for each of these attacks are shown in Figure 7.7. Higher confidence is indicated by lighter colors. If the CEMA attack yielded an incorrect key byte value, the box corresponding to the attack is marked with an ×.

Next, using a sampling frequency of $f_s^D = 4$ MSa/s the USRP2 is used to collect $n_{total} = 100,000$ traces at the following center frequencies, $f_c \in \{20, 21, 22, 29, 30\}$ MHz. For this range of center frequencies with $f_s^D = 4$ MSa/s, the signals are sampled at $1/10 < N_q < 1/15$ the Nyquist rate. There is a reduced number of center frequencies because the increased sampling rate makes it harder to avoid frequencies with interfering signals. CEMA attacks are performed using the first $n_t \in \{5000, 25000, 100000\}$ traces collected at each center frequency. The confidence $r_{max} \geq r_{next}$ for each of these attacks are shown in Figure 7.8. Increasing the

(a) $n_t = 5,000$ traces

(b) $n_t = 25,000$ traces

(c) $n_t = 100,000$ traces
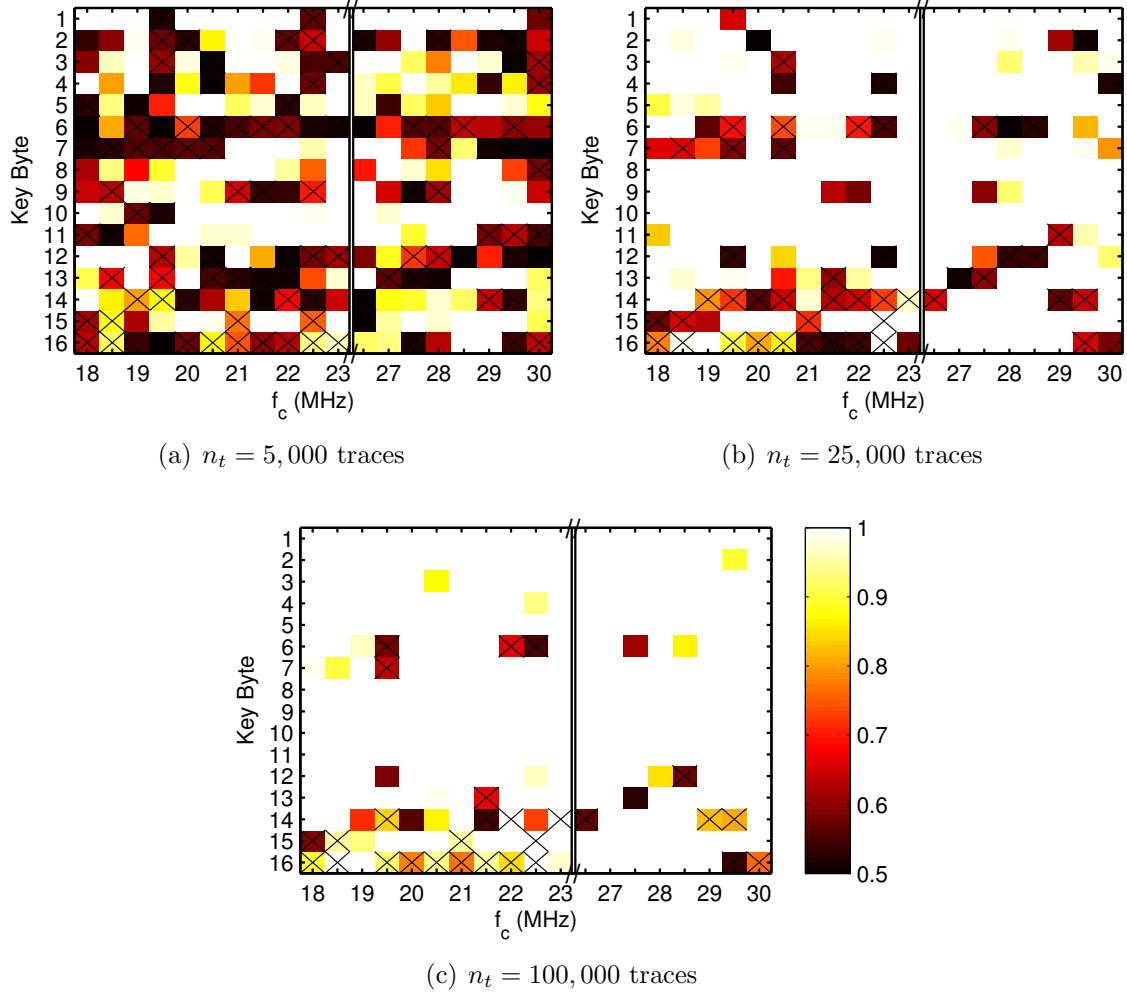
Figure 7.7   Confidence $r_{max} \geq r_{next}$ from CEMA attacks using traces collected with the USRP2. Traces are collected using the indicated center frequency f_c and a sampling rate of $f_s^D = 2$ MSa/s. The signals are sampled at $1/18 < N_q < 1/30$ the Nyquist rate. Attacks are performed for each key byte using the first (a) $n_t = 5,000$, (b) $n_t = 25,000$ or (b) $n_t = 100,000$ traces collected. CEMA attacks that yielded incorrect key byte values are marked with an $\times$.
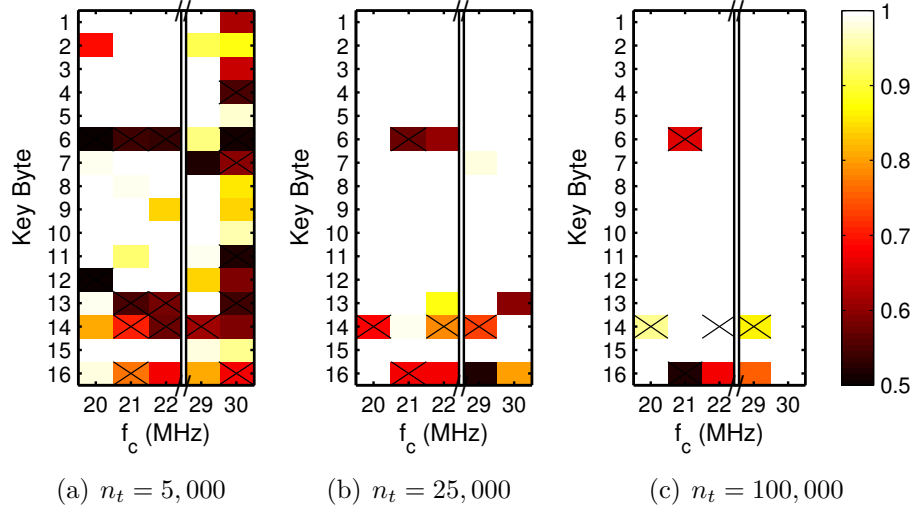
Figure 7.8   Confidence $r_{max} \geq r_{next}$ from CEMA attacks using traces collected with the USRP2. Traces are collected using the indicated center frequency f_c and sampling rate $f_s^D = 4$ MSa/s. The signals are sampled at $1/10 < N_q < 1/15$ the Nyquist rate. Attacks are performed for each key byte using the first (a) $n_t = 5,000$, (b) $n_t = 25,000$ or (b) $n_t = 100,000$ traces collected. CEMA attacks that yielded incorrect key byte values are marked with an ×.

sampling frequency, increases the number of key bytes that can be extracted with high confidence for $f_c \in \{20, 21, 22, 29, 30\}$ MHz.

Even after using all $n_{total} = 100,000$ traces to perform the CEMA attacks, not every key byte can be extracted at every center frequency. As expected, as the number of traces used in the attack increases, confidence $r_{max} \geq r_{next}$ and the number of key bytes correctly identified increase. However, since the confidence calculation in (3.10) is based on pooled standard error, as the number of traces increases small differences in correlation coefficients can result in high confidence levels. Figs. 7.7 and 7.8 both show key byte attacks that fail despite having $\geq 90\%$ confidence for $n_t = 25,000$ and $n_t = 100,000$ traces. Although less key byte values are correctly identified, the confidence results for $n_t = 5,000$ traces in Figs. 7.7(a) and 7.8(a) reliably identify a subset of the key bytes that leak for a given center frequency and sampling rate.

Using the sets of $n_{total} = 100,000$ traces as trace pools from which $n_t = 5,000$ traces are randomly selected, the CEMA attack is repeated $n_r = 1,000$ times for each center frequency and sampling rate. The results are shown in Figure 7.9. Comparing Figs. 7.7(a) and 7.9(a), most bytes that have a high confidence for a given sampling frequency and sampling rate using only the first $n_t = 5,000$ traces also have high success rate when $n_t = 5,000$ traces are chosen randomly from $n_{total} = 100,000$ traces. When the attack using only the first $n_t = 5,000$ traces has a confidence $p > 0.97$, the attack for that byte was successful for at least 80% of trials when repeated $n_r = 1,000$ times.

The correct key byte value may be identified with $n_t = 5,000$ traces but not when the number of traces is increased to $n_t = 25,000+$. An example is found in Figure 7.7. The identified value for key byte 14 using USRP traces with $f_s^D = 2$ MSa/s and $f_c \in \{20.5, 21, 23\}$ MHz is correct for $n_t = 5,000$ traces, but incorrect for $n_t = 25,000$ traces. For high quality traces, an attack that is successful with a lower number of traces should be successful when a significantly larger number of traces is used. Since the quality of USRP traces is consistent across the entire trace set, this phenomenon may be due to the poor quality of the traces including the low sampling rate of the SDR. The low sampling rate means multiple operations are performed on the devices between samples.

Figure 7.9(a) shows that, as in the baseline oscilloscope test, bytes 1 and 10 leak at multiple frequencies for $f_s^D = 2$ MSa/s. Although there are multiple key bytes that do not have high extraction success rates for every center frequency, every key byte has at least one center frequency for which the correct key value is selected for all n_r trials. From Figure 7.9(a), all key bytes, with the exception of byte 6 , have at least one frequency interval for which $p \approx 1$. However, even if the remaining bytes cannot be extracted with high confidence, a brute force attack to identify the value of the remaining bytes may be trivial.
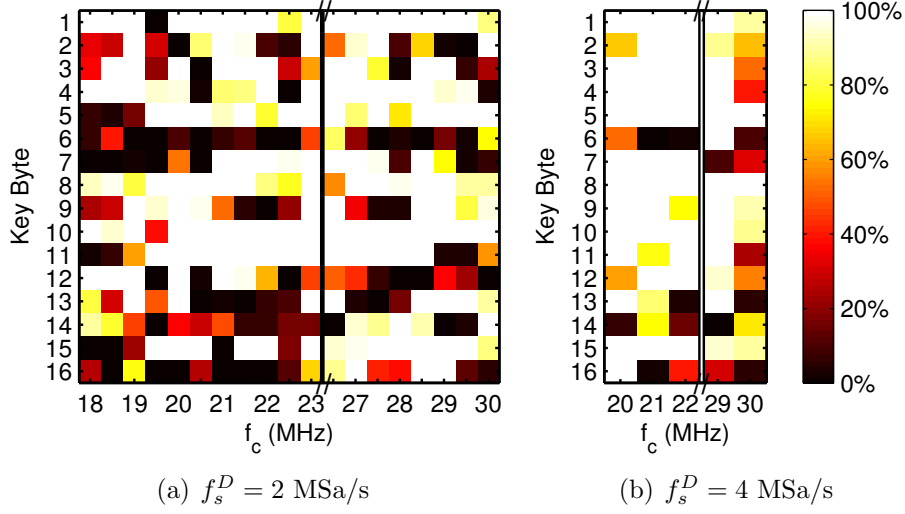
Figure 7.9   Percentage of 1,000 CEMA attacks correct for each key byte using $n_t = 5,000$ traces randomly chosen from the $n_{total} = 100,000$ traces collected with the USRP2 at each center frequency f_c for (a) $f_s^D = 2$ MSa/s ($1/18 < N_q < 1/30$) and (b) $f_s^D = 4$ MSa/s ($1/10 < N_q < 1/20$).

For $f_c = 23$ MHz and $f_s^D = 2$ MS/s, 100% of the $n_r = 1,000$ trials yielded the correct result for attacks on 11 of 16 key bytes. Using only the first $n_t = 5,000$ traces, the CEMA attacks on these 11 key bytes have confidence values greater than 0.97. For $f_c = 28.5$ MHz and $f_s^D = 2$ MSa/s greater than 99.5% of the $n_r = 1,000$ trials yielded the correct result for 12 of 16 key bytes. Using only the first $n_t = 5,000$ traces, the CEMA attacks on 10 of the 12 key bytes with greater than 99.5% of trials correct have confidence values greater than 0.99. Combining the byte values identified at these two frequencies, only two bytes cannot be determined using $n_t = 5,000$ traces at each frequency. When $f_s^D = 4$ MS/s 12 of 16 key bytes can be determined using only the first $n_t = 5,000$ traces collected for $f_c = 20$ MHz.

Two USRP2-based attacks extract all 16 bytes correctly with high confidence ($p \approx 1$). Figure 7.8(a) includes an attack with $f_c = 27$ MHz and $f_s^D = 4$ MSa/s and 7.8(c) shows an attack with $f_c = 30$ MHz and $f_s^D = 4$ MS/s.

7.6.2 RTL-SDR. The lowest center frequency that RTL-SDR can be tuned to is $f_c = 53.5$ MHz. Although the baseline test indicates most leakage is below 63 MHz, collections using at center frequencies up to $f_c = 73$ MHz are made to verify key byte extraction rates and confidence levels are reduce for center frequencies above 65 MHz. With a sampling frequency of $f_s^D = 2$ MSa/s, $n_{total} = 100,000$ traces are collected for $f_c \in \{53.5, 54, ..., 61.5, 62\}$ MHz and $f_c \in \{68, 68.5, ..., 72.5, 73\}$ MHz. For this range of center frequencies with $f_s^D = 2$ MSa/s, the signals are sampled at $1/53.5 < N_q < 1/73$ the Nyquist rate.

The confidence $r_{max} \geq r_{next}$, using the first $n_t \in \{5000, 25000, 100000\}$ traces collected at each center frequency is shown in Figure 7.10. Attacks yielding an incorrect key byte guess are indicated with an $\times$. Using $n_t = 25,000$ and $n_t = 100,000$ the byte-wise CEMA successfully extract all 16 bytes for multiple center frequencies. When the CEMA attack is performed using only the first $n_t = 5,000$ traces there is at least one center frequency at which $p \approx 1$ (with the exception of byte 5 ($p = 0.99$) and byte 9 ($p = 0.98$)).

The CEMA attack is performed using $n_t = 5,000$ randomly selected traces and repeated $n_r = 1,000$ times. The results of these attacks are summarized in Figure 7.11. All attacks with confidence $p > 0.975$ in Figure 7.10(a) are successful for at least 82% of $n_r = 1,000$ trials. As with the USRP2 collections, high confidence for $n_t = 5,000$ traces is a good way to identify a subset of the key bytes that leak information for a frequency interval.

Consistent with the baseline test, center frequencies less than 65 MHz have a lower percentage of successful key byte attacks than center frequencies above 65 MHz. Again, key bytes 1 and 10 have a higher probability of being correctly determined using a CEMA attack for multiple center frequencies. An unknown signal that varies in frequency near 56.9 MHz is observable on a waterfall plot in HDSDR, and causes a reduction in the effectiveness of the attacks using traces collected with $f_c = 57.5$

(a) $n_t = 5,000$ traces

(b) $n_t = 25,000$ traces

(c) $n_t = 100,000$ traces
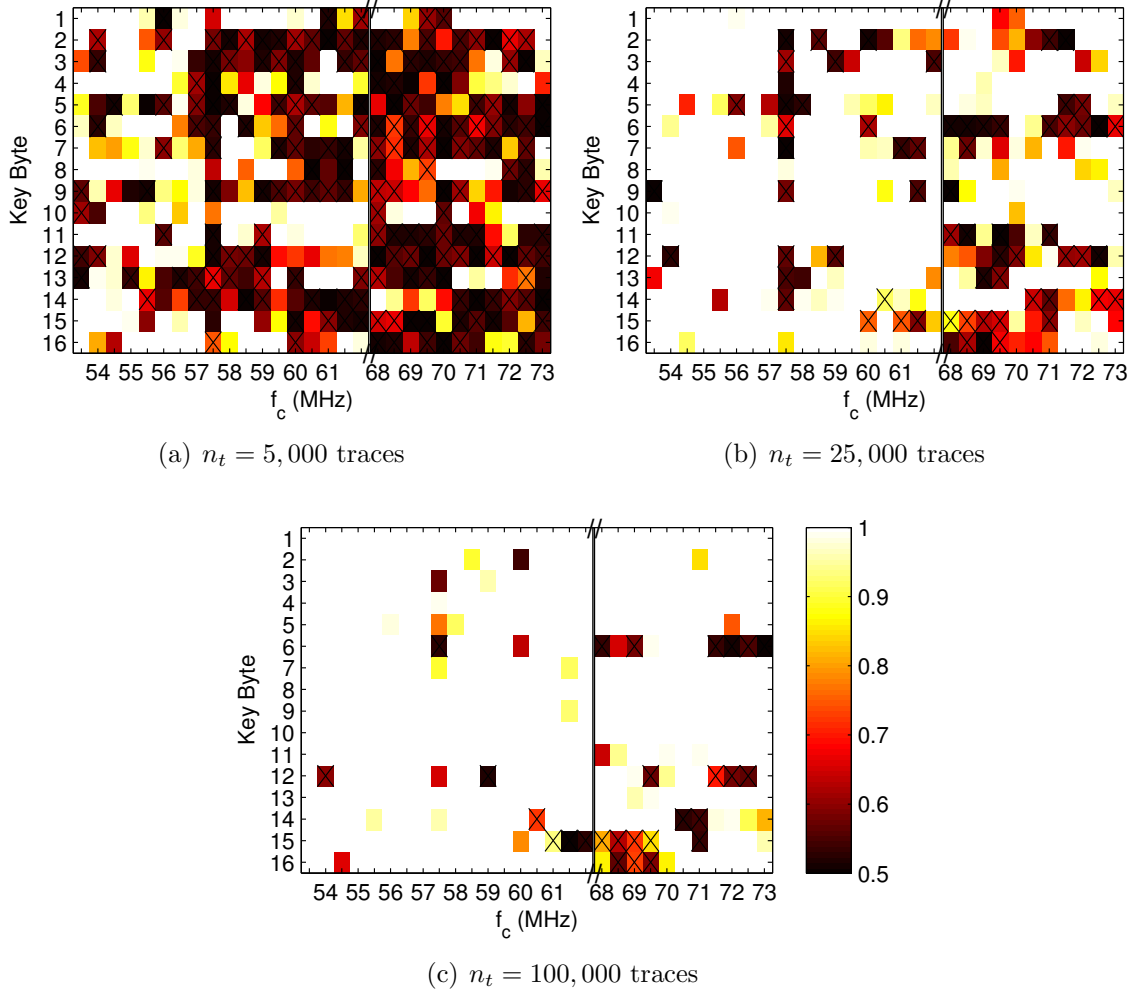
Figure 7.10   Confidence $r_{max} \geq r_{next}$ from CEMA attacks using traces collected with the RTL-SDR. Traces are collected using the indicated center frequency f_c and sampling rate $f_s^D = 2$ MSa/s. The signals are sampled at $1/53.5 < N_q < 1/73$ the Nyquist rate. Attacks are performed for each key byte using the first (a) $n_t = 5,000$, (b) $n_t = 25,000$ or (c) $n_t = 100,000$ traces. CEMA attacks that yielded incorrect key byte values are marked with an $\times$.
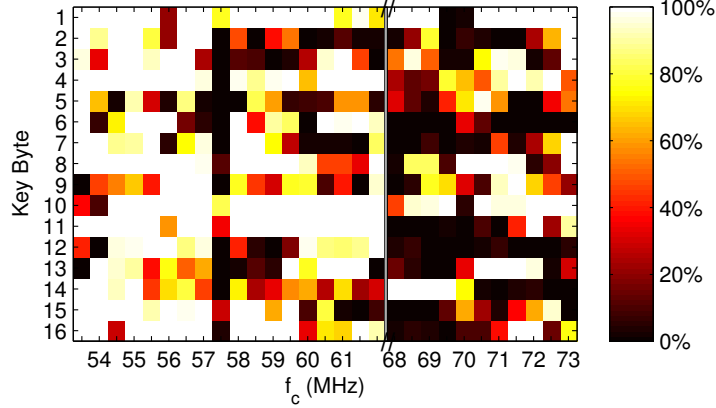
Figure 7.11   Percentage of 1,000 CEMA attacks correct for each key byte using $n_t = 5,000$ traces randomly chosen from the $n_{total} = 100,000$ traces collected with the RTL-SDR at each center frequency f_c and sampling rate $f_s^D = 2$ MSa/s. The signals are sampled at $1/53.5 < N_q < 1/73$ the Nyquist rate.

MHz. The highest number of key bytes successfully attacked in greater than 99.9% of trials is 11 of 16 using traces collected at $f_c = 55$ MHz.

Although the baseline oscilloscope results are filtered to include approximately the same frequencies collected using the SDRs, the baseline results cannot be directly compared to the SDR results. The baseline results are filtered but are not down-converted to baseband, low-pass filtered and decimated. At the sampling rate of $f_s = 250$ MSa/s, a total of 8,450 samples represent each $t_d = 32.18$ μsec encryption operation. At the lower sampling rates of $f_s^D = 2$ MSa/s and $f_s^D = 4$ MSa/s, only 63 and 129 samples respectively make up the entire encryption operation. As a result, calculations performed during multiple clock cycles are included in a single SDR sample.

Comparing the baseline oscilloscope and SDR based attacks, the key bytes that leak for specific center frequencies are not the same in all cases. Key bytes 1 and 10 are easier to extract than other key bytes for the oscilloscope and both SDRs. However, other key bytes which can be easily extracted with high confidence over a wide range of frequencies using filtered oscilloscope traces cannot be identified with

high confidence using SDR traces. For example, key byte 16 can be extracted with high confidence for center frequencies between 18 MHz and 30 MHz using filtered oscilloscope traces, but can only be successfully extracted using the USRP2 for select center frequencies. Key byte 16 can be extracted with a low number of RTL-SDR traces at various center frequencies between 55 MHz and 59 MHz.

In both the oscilloscope and SDR based attacks key bytes leak a different frequencies. If an SDR is used to collect data for a CEMA attack, collections should be performed at multiple frequencies. However, if the target encryption device can be modified to add a trigger, it should be. The SDR is not a replacement for an oscilloscope when the device can be altered to add a trigger.

The center frequency and sampling frequencies for both the USRP and RTL-SDR must to be carefully chosen to avoid clock jitter (from multiple clocks on the device) and various signals on the device with variable frequencies. Including these frequencies degrade the effectiveness of an attack and make it harder to identify the encryption operations. Fortunately, these frequencies can be easily identified by scanning over the potential collection frequencies using SDR spectrum visualization software.

In addition to frequencies near the system clock, frequencies around divisors of the system clock can be attacked. For the ARM Cortex-M4F, frequencies near 16.66 MHz and 25 MHz contain exploitable information. This may be due to some instructions taking multiple clock cycles or certain operations only being performed multiple clock cycles apart.

7.6.3 Additional Observations. Like the baseline attack, both SDR-based attacks indicate key bytes leak at different frequencies. Since the ARM Cortex-M4F is a 32-bit microprocessor with sufficient memory, the T-Box implementation is used to increase the speed of round transformations 1-9. While it is possible to store just one T-Box in memory and implement a byte-wise rotation separately [38], all

four T-Boxes in (2.1) are stored in memory on the target device. Using standard development tools, without programming AES directly in assembly, the designer does not have complete control of when each calculation will be performed on the device and which registers will be used. The compiler transforms the C++ code into object code with corresponding assembly instructions.

Evaluating the assembly code, it is not clear why some bytes are easier to extract than others. There is no correlation with the row of the AES state matrix, which would determine which T-Box is accessed in memory for each byte. One simple way to change key byte leakage is to change the optimization level for the C++ compiler used to program the ARM Cortex-M4F.

To evaluate compiler-dependent byte frequency leakage, traces are collected with the oscilloscope as in Sec. 7.4 with two different optimization levels used to compile the AES code for the microprocessor. The baseline test in Sec. 7.4 used the compiler default optimization level[4] of 2. Immediately after collecting the traces used in the baseline test, the device is reprogrammed using optimization level 0 and a new set of traces using the same set of $n_t = 2,000$ plaintexts is captured.

The baseline attack described in 7.4 is repeated for the trace set with optimization level 0. Figure 7.12 compares the confidence $r_{max} > r_{next}$ for the trace sets with different optimization levels. Changing the optimization level changes the frequencies at which key bytes leak. For example, byte 10 can be extracted with high confidence $r_{max} \geq r_{next}$ for center frequencies between 23 and 28 MHz with optimization level 2, but not for optimization level 0. Optimization to reduce execution time does not necessarily decrease key byte value leakage. Trace sets of $n_t = 2,000$ plaintexts are used to demonstrate that even with a relatively large number of traces some bytes do not leak at certain frequencies.

---

[4]Level 0 includes register optimizations. Level 2 adds local and global optimizations. All SDR collections were performed with optimization level 2.

(a) Optimization level 2

(b) Optimization level 0
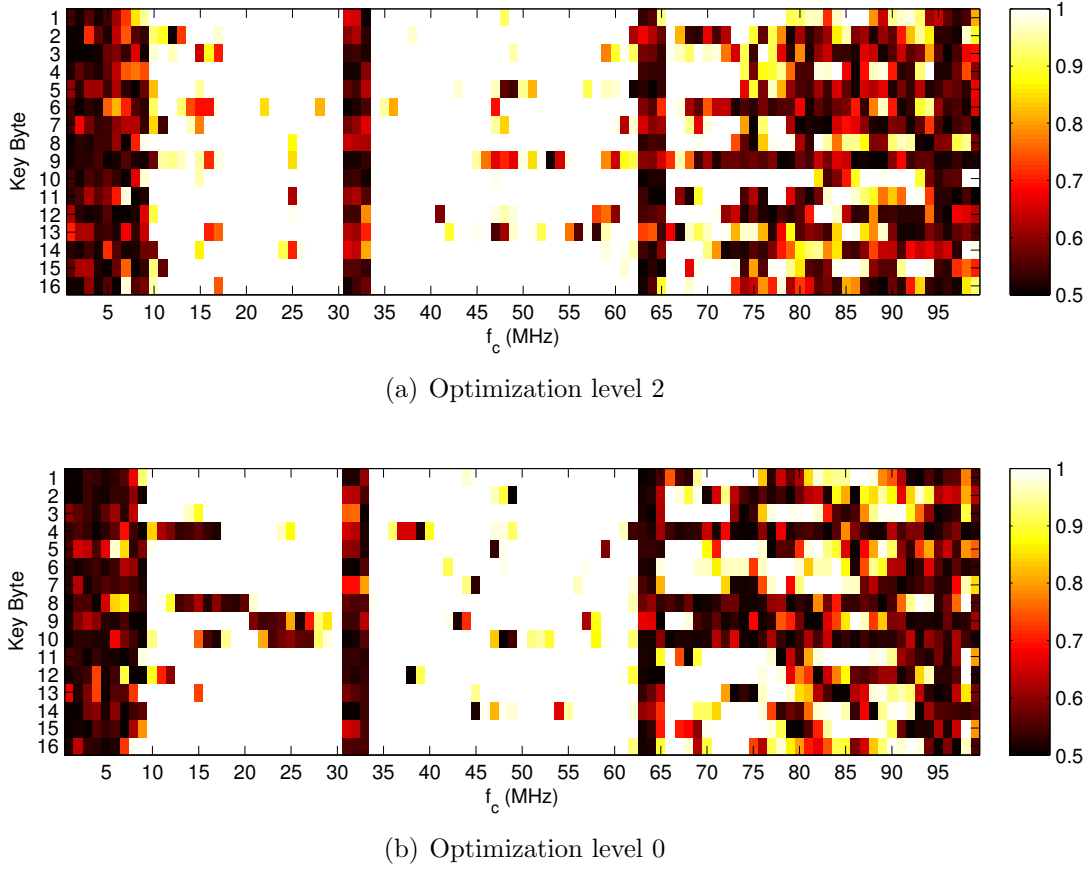
Figure 7.12   Comparison of key byte extraction confidence using $n_t = 2,000$ traces for compiler optimized implementations of AES. The same set of plaintexts are used for each attack.

7.6.4   Comparison of the Baseline and SDR Results.   Based on the duration of the trigger, the encryption operation takes $t_d = 34.89$ μsec for optimization level 0, vs $t_d = 32.18$ μsec for optimization level 2. Since execution time is decreased, the compiler does find some optimizations, but it is not clear what the optimizations are. One way to visualize the difference between the two optimization levels is to display the magnitude of the correlation with the 32-bit HW model for each output of the T-Box, using a technique similar to Cobb [30]. Oscilloscope traces with $f_s^D = 250$ MSa/sec (no bandpass filtering) are used to create the temporal leakage maps for each optimization level shown in Figure 7.13.
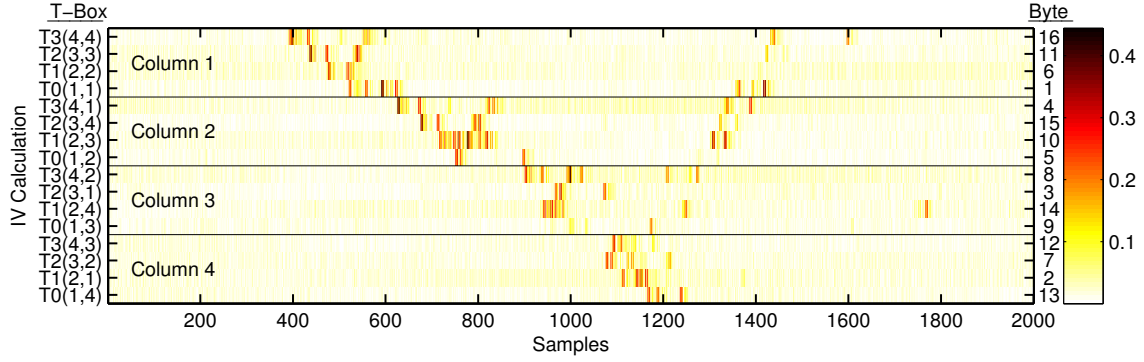
(a) Optimization level 2
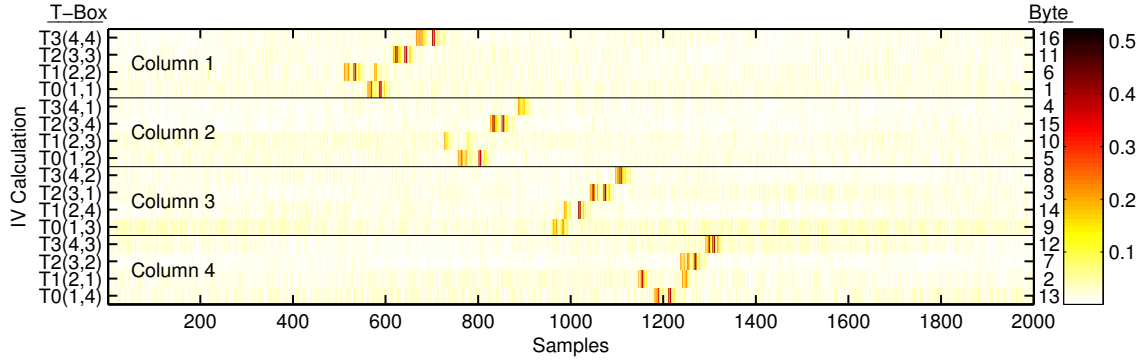
(b) Optimization level 0

Figure 7.13   Comparison of the temporal leakage map for (a) optimization level 2 and (b) optimization level 0. The order the T-Box operations are performed is different depending on the optimization level used by the compiler.

While the columns of the state matrix are processed in order (1-4) for both optimization levels, the order in which the T-Box operations are performed changes for each optimization level, and possibly between columns. The T-Box used and the row and column of the state matrix are indicated on the left y-axis of Figure 7.13. The byte number that corresponds with the state matrix location is list on the right y-axis. The correlation plots for the T-Box operations are arranged in approximate temporal order for optimization level 2 in Figure 7.13(a) and the same order is used for optimization level 0 in Figure 7.13(b) to show that the order in which the T-Box operations are performed changes. While this does not fully explain why the

bytes leak differently, it clearly shows the order intermediate values are calculated is affected by the compiler optimization level used.

7.7 Conclusion and Future Work

SDRs can effectively attack a 32-bit microcontroller running AES-128. Using sampling rates as low as $f_s^D = 2$ MSa/s, both the USRP2 and RTL-SDR, a SDR based on a \$20 USD digital TV tuner, can capture EM emissions from a ARM Cortex-M4F.

Although a low number of samples are collected per encryption operation, the low sampling rate allows RF emissions from an encryption device to be collected continuously. Post-processing extracts encryption operations from a collected EM emission for use in a CEMA attack. This no-trigger, no-profiling approach enables attacks on an unmodified device without external trigger generation hardware. Despite sampling at rates well below the Nyquist rate, the encryption key can be successfully extracted.

Attacks using both SDR and oscilloscope collected traces all found key bytes leak at different frequencies. The correlation-based frequency-dependent leakage mapping technique identified filter parameters which increased the confidence of the CEMA attack and identified frequencies to target with the SDRs. Failing to identify the correct key byte despite high confidence is an acknowledged limitation of the confidence metric when a large number of traces is used in the attack. Despite this limitation, confidence is favored over success rate or guessing entropy for an SDR-based attack because it can be calculated with a small set of test traces, allowing an attacker to determine which bytes likely leak for each center frequency.

To attack a device with an SDR, collections should be made with multiple center frequencies near the clock frequency or divisors of the clock frequency. Using the highest sampling rate possible improves results provided signals on the device such as clocks and signals with variable frequencies can be avoided. SDR spectrum

visualization software can identify frequencies to avoid. To determine the correct 128-bit AES key, an attacker should collect at multiple center frequencies to identify the key bytes that leak strongly for each frequency interval or collect significantly more traces for a single frequency interval.

8.   Conclusion

This chapter summarizes the activities and unique contributions of this doctoral research and makes several recommendations for future research.

8.1   Research Summary

Over the last 15 years many side-channel analysis (SCA) techniques have been developed that work very well in academic laboratory environments. When applying these techniques in operational environments, an attacker may not be able to take actions to improve the quality of the collected traces or have access to high quality collection equipment. This dissertation examined ways to 1) eliminate some assumptions commonly made when performing SCA attacks in the laboratory, and 2) compensate for incomplete assumptions made by others. Ultimately, even if the techniques developed here reduce side-channel attack effectiveness, they remain valuable in so far as the attack can be performed without modifying the device or by using lower cost equipment, thereby improving SCA utility.

The research focused on three primary areas of investigation:

1. An algebraic cryptanalysis-based attack on the AES-128 key schedule,

2. Cross-device template attacks, and

3. Introduction of Software Defined Radios (SDRs) for differential SCA.

   Specific results and contributions in each of these areas are summarized below.

8.1.1   Algebraic Cryptanalysis.   The Key Schedule Redundancy Attack (KSRA) developed here reconciles uncertainty in the classification stage of template attacks using a SAT solver [82]. A system of equations for the AES-128 key schedule was generated and constrained based on the results of template attacks to create a new unknown-plaintext, unknown-ciphertext attack. By attacking the key schedule,

traces from multiple encryption operations can be used without knowledge of the plaintext.

Previous work in algebraic side-channel analysis used a fixed number of guesses for each targeted intermediate value [81, 89, 102, 137], but the KSRA uses a novel thresholding technique to gradually increase the maximum number of guesses per key byte [82]. This approach prevents key byte guesses with low probabilities from being included when one or more guesses are assigned a high probability by the classifier, reducing solve time and the probability of identifying an incorrect key schedule.

The strength of the attack comes from the redundancy of the key schedule, allowing 40 key schedule-bytes to be attacked rather than just 16 key-bytes normally targeted in a SubBytes-based attack and from the ability to used traces from multiple encryption operations. Since the redundancy exists, the SAT solver can be used to identify working key schedules that meet the constraints. Even if the key schedule can only be observed once, the KSRA yields much better performance (100% of 500 trials) than a SubBytes attack (16.8% of 500 trials) [82].

Incorporating multiple traces into the attack phase dramatically improves attack performance for poor quality traces. For traces collected at $h = 5$ mm using only one trace, the key schedule was not recovered in any of the 500 attempts. When 50 traces were used, the Satisfiability solver identified the correct key schedule in 97.6% of the trials [82].

The ability to perform the KSRA without knowledge of the plaintext or ciphertext, and its robust performance using poor quality traces, may enable an attacker without placing a near-field probe directly on the device and/or matching each collected trace with its corresponding plaintext or ciphertext to be successful.

Although Renauld et al. note that up to 200 Hamming weights (HWs) can be recovered from a single power trace from one encryption operation, they do not use

actual data to perform their attack [102]. All their data is simulated, and although they may randomly determine which intermediate values are included, all HWs used to constrain the SAT solver result are correct. The Pseudo-Boolean optimization approaches by Oren et al. both use simulated data as well [89, 91]. The KSRA is the first known algebraic side-channel attack demonstrated using actual collected data, and the first side-channel attack method to demonstrate robustness by intentionally degrading the quality of the collected traces [82].

8.1.2 Cross-Device Template Attacks. The assumption that side-channel emissions from two similar devices produce similar emissions, as made by Chari et al. [24] and adopted by others [3, 9, 53, 92, 102, 122] is challenged here for the first time. It was shown that while template attacks based on mean and covariance matrices work well for attacking the same device on which the training traces are collected, the slight differences in emissions from similar devices may be sufficient to cause a template attack to fail [83]. The process of identifying distinguishing features and the distribution of training and test data at each of the distinguishing features were analyzed to identify differences between devices.

The simple technique of mapping both the test data and the training data to the standard normal, or zero-mean and unit-variance normalization (MVN), was developed here to improve the effectiveness of cross-device template attacks [83]. Same part number attacks are improved from 65.1% to 100%, and attacks against similar devices in the same device family are also improved [83]. For the PIC microcontrollers, only a small number of traces (approximately 15) are needed to estimate the mean and variance for a cross-device attack. Although the MVN technique was shown to reduce the effectiveness of same-device attacks using a small number of traces, an attacker can always perform a standard template attack since both attacks are based on the same collected data.

The distinguishing features selected may be different from device to device. While the goal for a same-device attack is to reduce the number of distinguishing features to make the templates easier to create, increasing the number of distinguishing features improves the cross-device attack success rate [83]. A master template can be created for a family of devices by combining the distinguishing features for each type of device and building templates from a combined training set. Training data from each training device is pre-processed with the MVN technique before being combined into a larger training set. While the resulting attack did not perfectly identify every byte for all 40 PIC devices, it provided the best performance of a single set of templates, achieving an average byte extraction success rate of 99.95%.

The MVN technique was also shown to effectively compensate for changes in probe placement on larger more complex devices such as the ARM Cortex-M4F. Combined with the negative MVN technique which compensates for negatively-correlated EM emissions, the MVN technique increases the number of locations above the device where template attack can be performed successfully for all 16 bytes by 226% [83]. Calculating the power spectral density (PSD) variance was found to be a simple, yet powerful, way to identify signal frequency components that change in power between collected traces and have an adverse effect on cross-device template attacks. Using notch filtering to attenuate these frequencies in both training and test traces reduced the average number of traces needed to perform a successful template attack by 85.8% [83].

The ability to use a different device for training, rather than the device being attacked is one assumed benefit of template attacks. This research identified ways to increase the effectiveness of template attacks when training and test data are collected on different devices. Ultimately, the original assumption that training and target devices have sufficiently similar side-channel emissions in [24] is validated with an added caveat that device-dependent differences in sample means and variances must be compensated for before performing the template attack [85]. Additionally,

if signals unrelated to the encryption operation being performed can be identified, notch filtering to reduce the contribution of theses signals may improve the effectiveness of the template attack [83].

8.1.3 *Software Defined Radios (SDR).* SDRs can be used to effectively attack a 32-bit microcontroller running AES-128 [84]. Two SDRs were used to passively collect traces from a ARM Cortex-M4F at sampling rates as low as $f_s^D = 2$ MSa/s. The RTL-SDR is based on a commercial digital TV tuner that can be purchased for \$20 USD. This research is the first known use of SDRs for differential side-channel analysis.

Due to their limited sampling frequencies, a low number of samples are collected for each encryption operation using an SDR. The low sampling rates allow the traces to be collected continuously and eliminate the need for a trigger. This no-trigger, no-profiling approach allows for attacks to be performed on an unmodified device without external trigger generation hardware and greatly reduces the equipment needed to perform a side-channel attack [84].

Attacks using both SDR and oscilloscope collected traces found key-byte leakage at different frequencies. Since previous research had focused on a single key byte [13,14], the research here is the first to identify this phenomenon [84]. Key-byte leakage can also be changed by reprogramming the target device using a different optimization level.

To attack a device with an SDR, traces should be collected at multiple center frequencies and with the highest sampling rate possible without loosing samples due to overruns [84]. The center frequencies and bandwidths must be chosen carefully to avoid clock frequencies (including jitter) and signals on the device that vary in frequency. These frequencies can easily be identified using SDR spectrum visualization software or the variance of trace PSD technique for traces collected with an oscilloscope [83].

Although more traces are required to perform an attack with SDR-collected traces than with oscilloscope-collected traces, the fact SDR traces can be collected without modifying the device or additional hardware makes SDR-based collection a powerful tool for operational side-channel attacks. However, since oscilloscope based attacks are more effective, if an oscilloscope is available and the cryptographic device can be modified without losing the key being attacked, it should be [84].

8.2   Recommendations for Future Research

8.2.1   Algebraic Cryptanalysis.

Since the KSRA uses 40 key schedule byte values when only 16 bytes are required, the attack would be possible using fewer intermediate values. It may be possible to identify which intermediate values to include based on the posterior probabilities for each attacked byte. Since the correct key schedule can not be found unless the correct byte value is included in the list of possible values for each targeted byte, eliminating a key-schedule-byte from the list of constraints if it has a larger number of possible byte values than other key-schedule-bytes may improve attack performance and reduce solving time.

Optimizer-based approaches that incorporate the posterior probabilities for each targeted intermediate value into a goal function, have been demonstrated to be more tolerant of errors than SAT-solver based approaches using simulated data [89]. However, using an optimizer rather than a SAT solver dramatically increases the solve time and memory requirements. A comparison of an optimizer-based approach, and a SAT solver-based approach with constraints defined using the thresholding technique developed for KRSA should be performed.

Finally, a more powerful attack might be created by combining SDR-based trace collection, MVN technique-based template attacks and KSRA. Since the KSRA is a no-plaintext/no-ciphertext attack an SDR could be used to collect traces from the test device without needing to identify the plaintext or ciphertext. Template attacks could be performed using training and test data pre-processed with the

MVN technique. This attack could be performed without device modification and would likely require fewer traces than the correlation-based electro-magnetic analysis (CEMA) based attack in Chapter 7. However, due to the poor quality of individual traces when collecting with the SDR, this attack would likely only be successful if the key schedule is calculated on the fly with each encryption operation. These attack methodologies can not be combined for every attack and their utility depends on the implementation being attacked.

8.2.2 Cross-Device Template Attacks. As microprocessors and Field Programmable Gate Arrays (FPGAs) continue to become more complex, and cross-device template attacks are used to attack these devices, additional steps may be required to address differences between devices. As device features size and power consumption is reduced, probe placement will be more important and there may be greater differences in the leakage distribution for each type of device. While techniques developed in this dissertation will continue to be valuable, additional device specific techniques may need to be developed.

It was demonstrated in this research that a single set of templates can be created to attack multiple devices with minor differences in memory and on-board peripherals. This set of templates was created by identifying distinguishing features from each device before performing the MVN technique on training data from each training device to form a combined training set. Further research is required to determine the most effective method of identifying distinguishing features from multiple training devices. Additional methods for transforming and combining data should also be explored.

The MVN technique may also be able to compensate for differences in the operating conditions of the device. For example, it was shown the power consumption of a SASEBO-GII FPGA platform changes as the ambient temperature changes. This is addressed using a stochastic approach that compares the differences in consecutive

power traces in place of the power traces themselves in [56], but it may be possible to use the MVN technique to pre-process groups of data collected during short periods of time when the ambient temperature is relatively constant. The processed groups would be recombined into a new training set, and the test set would be similarly processed.

This research found that notch-filtering improved the performance of template attacks but reduced the effectiveness of CEMA attacks. Conversely, bandpass filtering was found to improve CEMA attacks, but was not shown to improve template attacks. Further research is needed to verify this phenomenon is true for other devices and understand why each type of attack is enhanced or degraded by each filtering method, and identify additional pre-processing methods to improve SCA attack performance.

8.2.3 Software Defined Radios. Further research is needed to understand why key-bytes leak at different frequencies, and how changing the compiler optimization level changes which key bytes leak at each frequency. Speed optimization may have made the implementation easier to attack. If it can be determined why compiler-dependent byte frequency leakage differences exist, it may be possible to program the device in assembly to reduce the devices vulnerability to side-channel attacks. Once well understood, it may be possible to design a compiler that minimizes leakage from a device. Unfortunately, leakage may change from device-to-device, reducing the utility of such a compiler.

One way to enhance the attack using the USRP2 is to collect at multiple frequencies simultaneously. The USRP2 can collect at two center frequencies with the same sampling rate. The data from both center frequencies could be used in the attack, or one frequency could be used to identify and align the encryption operations collected at another center frequency. This would allow attacks to be performed for center frequencies where the encryption operations are not easy to

identify. Additionally, if the same external clock is used on both the cryptographic device and the SDR, attacks with a center frequency equal to the clock frequency of the cryptographic device may be more effective.

Given the capabilities of modern FPGAs, it may be possible to create a single device that performs all aspects of a side-channel attack. Many FPGAs include analog-to-digital converters which can digitize the side-channel information, and perform demodulation. The FPGA can be used to identify traces in the collected side-channel and perform a CEMA-based attack. The device could read ciphertext from the a network connection to simulate encrypted packets being captured on a network. If the attacker knows which samples are highly correlated with the targeted intermediate value, only these samples need to be retained for each trace. Although this would require a profiling step, the amount of memory required to perform the attack would be dramatically reduced.

Appendix A.   Constructing and Solving Systems of Equations

This appendix provides a brief introduction to Conjunctive Normal Form (CNF), Satisfiability (SAT) solvers, and how to present a system of equations written in CNF to a SAT solver.

A.1   Conjunctive Normal Form

To use a SAT solver to determine a solution to a set of equations, the problem must be transformed into CNF. A brief introduction to CNF, based on [127], is presented below.

A Boolean variable x can take on two values, '1' for true, and '0' for false. The basic Boolean functions are negation (NOT), conjunction (AND) and disjunction (OR). The Boolean negation function is,

$$\bar{x} = \begin{cases} 1 & \text{if } x \text{ is false,} \\ 0 & \text{otherwise.} \end{cases}$$

A *literal* is a Boolean variable or its negation. A conjunction of a collection of literals $x_1,...,x_n$ is,

$$x_1 \wedge x_2 \wedge \cdots \wedge x_n = \begin{cases} 1 & \text{if all of the } x_i \text{ are true,} \\ 0 & \text{otherwise.} \end{cases}$$

A *clause* is a disjunction of a collection of literals. For example a *disjunction* of $x_1,...,x_n$ is,

$$x_1 \vee x_2 \vee \cdots \vee x_n = \begin{cases} 1 & \text{if any of the } x_i \text{ are true,} \\ 0 & \text{otherwise.} \end{cases}$$

A Boolean function, f, is said to be in conjunctive normal form (CNF) if it is written as,

$$f(x_1, ..., x_n) = \bigwedge_{k=1}^{m} C_k$$

There each C_k is a conjunction of literals. The following function is in CNF,

$$f(x_1, ..., x_n) = (x_1 \vee x_2 \vee \bar{x}_5) \wedge (\bar{x}_3 \vee x_6) \wedge (x_4 \vee \bar{x}_6).$$

A *truth assignment* assigns values to the variable of the Boolean function such that $\mathbf{x} = (x_1, ...x_n) \in \{0, 1\}^n$. If the set of variables satisfies the function such that $f(\mathbf{x}) = 1$, the set $\{\mathbf{x} \in \{0, 1\}^n : f(\mathbf{x}) = 1\}$ is a *satisfying truth assignment*. If a set of variable assignment exist such that $f(\mathbf{x}) = 1$, then f is *satisfiable*.

A.2 SAT Solvers

SAT solvers attempt to answer the question, "for a given Boolean function $f(x_1, ..., x_n) = \bigwedge_{k=1}^{m} C_k$, is f satisfiable?"

SAT solvers use an encoding scheme with the alphabet $\Sigma = \{0, 1, \vee, \wedge, \neg\}$. Each variable x_i is denoted by the binary representation of i and $\bar{x}$ is written as $\neg x$. The CNF Boolean function written as

$$f(x_1, ..., x_6) = (x_1 \vee x_2 \vee \bar{x}_5) \wedge (\bar{x}_3 \vee x_6) \wedge (x_4 \vee \bar{x}_6)$$

would be encoded using Σ as

$$1 \vee 10 \vee \neg 101 \wedge \neg 011 \vee 110 \wedge 100 \vee \neg 110.$$

A.3 Converting MQ to SAT

Although a number of articles outline the steps to represent a system of multivariate polynomials as a SAT problem, the best step-by-step process is found in [11].

The following section is based on Bard's technique and includes many of his examples.

A.3.1 Step 1: Convert the Polynomial System to a Linear System.   Every polynomial is a sum of linear and higher degree terms. Quadratic and higher degree terms must be rewritten. For example, the logical expression

$$(w \vee \overline{a})(x \vee \overline{a})(y \vee \overline{a})(z \vee \overline{a})(a \vee \overline{w} \vee \overline{x} \vee \overline{y} \vee \overline{z})$$

can we rewritten as $a \iff (w \wedge x \wedge y \wedge z)$, or in $\mathbb{GF}(2)$ as $a = wxyz$ [11]. A similar equation of the form $a = w_1 w_2 \ldots w_r$, for any $r > 1$ can be written for any monomial of degree $d > 1$. A dummy variable represents each monomial with degree $d > 1$. The number of clauses required to represent the monomial is $d + 1$. If the monomial appears more than once, the dummy variable should be used instead of added additional equations. Care must be taken to avoid encoding the same monomial twice.

CNF does not have constants. A work around to include a 1 or 0 in a CNF clause is to add a separate clause consisting of T or equivalently $(T \vee T \vee \ldots \vee T)$. Since this statement must be true in any satisfying solution, T can be used in place of a **1** and $\overline{T}$ can be used in place of **0**. As a results, the constant term 0 or zero can be treated as a variable.

A.3.2 Step 2: Linear System to CNF Expression.   Each polynomial is now a sum of variables, which can be represented using logical-XORs. For example, the sum $(a \oplus b \oplus c \oplus d = 0)$ is equivalent to

$$(a \vee b \vee c \vee d)(a \vee b \vee \overline{c} \vee \overline{d})(a \vee \overline{b} \vee c \vee \overline{d})(a \vee \overline{b} \vee \overline{c} \vee d)$$
$$(\overline{a} \vee b \vee c \vee \overline{d})(\overline{a} \vee b \vee \overline{c} \vee d)(\overline{a} \vee \overline{b} \vee c \vee d)(\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{d}) \tag{A.1}$$

The length of the CNF equation grows exponentially with the number of variables in the XOR statements. The statement must equal one whenever there is an even number of ones. The CNF must include every arrangement of variables that can produce a zero. XORs statements with many variables require long CNF clauses which are more difficult for SAT solvers to solve.

For a sum of length l, where $2\lfloor l/2 \rfloor = j$, this requires

$$\binom{l}{0} + \binom{l}{2} + \binom{l}{4} + \ldots + \binom{l}{j} = 2^{l-1} \tag{A.2}$$

clauses.

To prevent this exponential increase in clauses, the XOR sum can be cut into subsums of length c. This is referred to as the cutting number. For example, the equation $x_1 \oplus x_2 \oplus \ldots \oplus x_l = 0$ can be written as the set of equations

$$x_1 \oplus x_2 \oplus x_3 \oplus y_l = 0$$

$$y_1 \oplus x_6 \oplus x_7 \oplus y_2 = 0$$

$$\vdots$$

$$x_i \oplus x_{4i+2} \oplus x_{4i+3} \oplus y_{i+1} = 0$$

$$\vdots$$

$$x_h \oplus x_{l-2} \oplus x_{l-1} \oplus y_l = 0$$

if $l = 2 \pmod c$. If $l \neq 2 \pmod c$ then the final sum is shorter than l. Since multiple shorter XOR statements produce fewer CNF clauses, this is a more efficient way to represent the original XOR. This method produced $h + 1$ subsums, where $h = \lceil l/c \rceil - 2$. There will be $h + 1$ subsums and each will require 2^{c-1} clauses of length c each as shown in (A.2).

A.3.3 Step 3: DIMACS CNF Form. The Center for Discrete Mathematics & Theoretical Computer Science (DIMACS) at Rutgers University proposed a standard graph format in 1993 for satisfiability problems in CNF. CNF file format is an ASCII file with the following structure.

- Comments are indicated by making the first character of each comment line a lower case letter 'c'. Comments are typically placed at the beginning of the file, but are allowed through the file.

- The "problem" line should be placed after comment lines at the beginning of the file. This line begins with 'p' followed by the problem type. For CNF files, 'cnf' should be followed by the number of variables and the number of clauses.

- The clauses appear immediately after the problem line and make up the remainder of the file. The variables are assumed to be numbered 1 to n. Each clause is a sequence of numbers, separated by a space, tab or a new line character. The non-negated variable is represented by i and the negated variable is represented by $-i$. Each clause is terminated by the value 0.

  The CNF equation in (A.1) can be written in DIMACS CNF file format as,

```
c A XOR B XOR C XOR D = 0
c 4 variables, 8 clauses
p cnf 4 8
1 2 3 4 0
1 2 -3 -4 0
1 -2 3 -4 0
1 -2 -3 4 0
-1 2 3 -4 0
-1 2 -3 4 0
-1 -2 3 4 0
-1 -2 -3 -4 0
```

A.4 Methods for Solving Non-linear Multivariate Systems of Equations

A number of methods have been demonstrated or proposed for solving non-linear multivariate systems of equations. One applies a technique from computational algebra called Gröbner basis algorithms. Gröbner basis can be used to triangulate a polynomial system and can be found using the Buchberger Algorithm, which is an exact algorithm. More efficient $F4$ or $F5$ algorithms can also find the Gröbner basis [44], [45]. A number of mathematical tools use Gröbner basis techniques to solve systems of equations, but for complex problems they crash due to memory requirements. MAGMA and SINGULAR tools are recommended for solving systems of polynomial equations [11].

The eXtended Linearization (XL) algorithm developed by Courtois builds on the concept of relinearization [112]. *Relinearization* takes a given system of linear equations and adds non-linear equations which capture the fact that these variables are related rather than independent, making the system of equations easier to solve. XL is a combination of bounded degree Gröbner basis and relinearization techniques [35].

Courtois recommends that Gröbner basis methods should be avoided because they expand the system of equations to a larger degree (e.g., 4 or 5) to solve them, resulting in time and memory-consuming expansion. He recommends using linear algebra and known elimination techniques to take advantage of and, if possible, preserve sparsity [37]. The ElimLin function incorporates this philosophy.

The ElimLin function, also developed by Courtois [37], is another algebraic attack. Starting with an initial system (i.e., degree 2 or 3), the algorithm looks for linear equations in the linear span of the equations. If equations in the span of equations are identified, several variables can be eliminated using simple substitution by a linear expression. When new linear equations are found they are added to the system. The process is repeated until no more linear equations are found. The variables that appear in the smallest number of equations are eliminated first, helping

to preserve sparsity, while key variables are eliminated last. As a results, ElimLin is able to solve systems were Gröbner basis techniques fail due to lack of memory.

Courtois and Pieprzyk introduced the idea of describing ciphers with S-boxes as an over-defined system of algebraic equations. They found the quadratic equations for AES are both sparse and over-defined and proposed a new method for solving over-defined systems of equations called eXtended Sparse Linearization (XSL), which takes advantage of the sparsity and structure of the system [36] . XSL is designed to work with ciphers that have XOR, S-Box and Linear diffusion layers. After Courtois and Bard published their results in [36], the efficiency of an XSL-attack on AES was challenged in [26].

Raddum and Semaev present an alternative approach [96]. Rather than represent the system equations as polynomials, the equations are represented as lists of bit-strings. Each string is a value assignment of a variable that satisfies the equation. Raddium and Semaev's algorithm is more efficient than classical methods which represent the system as polynomials [96].

Murphy and Robshaw describe the essential algebraic structure within AES using a new block cipher [86]. They introduce the Big Encryption System (BES), that uses only simple algebraic operations in $\mathbb{GF}(2^8)$ and show that AES, which performs operations in $\mathbb{GF}(2)^8$, can be implemented as a form of BES with a restricted message and key space. BES uses very simple operations in $\mathbb{GF}(2^8)$, and as a result AES can be described without operations in $\mathbb{GF}(2)^8$. A round of AES can be described in BES as an inversion, a matrix multiplication, and a key addition in $\mathbb{GF}(2^8)$. Thus, AES can be described as a "very simple" and extremely spare system of multivariate quadratic equations over $\mathbb{GF}(2^8)$. Using $\mathbb{GF}(2^8)$ to describe AES S-Box equations has the benefit of much sparser systems of equations with a reduced number of free terms. However, working in a larger field may increase the complexity of the solving algorithm [18]. Note that this embedding technique is not applicable to all block ciphers, but takes advantage of the particular AES algebraic structure.

Appendix B. Writing AES-128 for a SAT Solver

Implementing an AES-128 SAT solver tool from a system of equations required a number of steps, which are performed by different programs. The system of equations generated by the SR polynomial generator are in algebraic normal form (ANF). A ANF to CNF converter is used to convert the system of equations to conjunctive normal form (CNF). Constraints for known values can be added by including additional polynomials in the system, by adding additional lines to the DIMACS code, or by using Limboole. The DIMACS file with constraints is the input to the SAT solver. Finally, the output of the SAT solver must be converted back to the original variables. These steps are shown in Figure B.1 and are explained in the following sections.

B.1 SR Polynomial Generator

Small scale variants of the AES were defined to inherit the design features of AES and to provide a framework for comparing cryptographic methods [25]. In addition to implementing the small scale variants described in [25] and [27], a full scale implementation of AES-128 can be constructed. The SR generator is based in Sage Mathematical Software, a free and open source software tool created with the goal of being an "open source alternative to Magma, Maple, Mathematica, and MATLAB" [124].

Since the SR generator is designed to implement all small scale versions specified in [25] and [27], various parameters must be set. The size of the variant is specified using `sr = mq.SR(n,r,c,e)` where `n` is the number of rounds, `r` is the number of rows in the state array, `c` is the number of columns and `e` is the exponent of the finite extension field. For the Rijndael polynomial used in AES-128: sr = mq.SR(10,4,4,8).
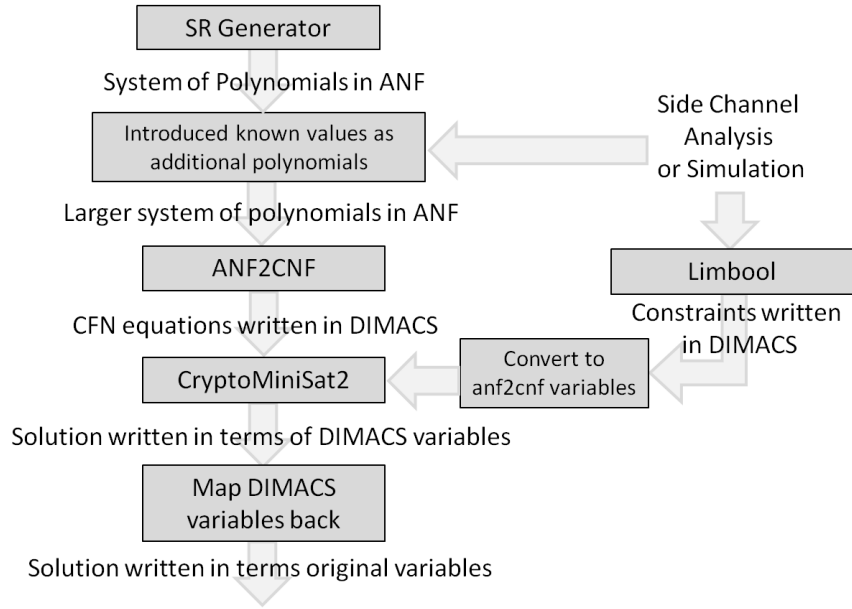
Figure B.1   Data flow from the system of polynomial produced by the SR polynomial generator to a results in terms of the original variables.

To use the anf2cnf converter, the output of the SR generator must be a system of equations specified over a Boolean Polynomial Ring. The `polybori=True` option tells the SR generator to used the PolyBoRi package [21] included in Sage to create a system of polynomials over the boolean ring defined by SR. The `star=True` and `gf2=True` options tell the SR generator to use the AES key schedule and omit the MixColumns transformation during round ten. Finally, the option `correct_only=True` specifies that only inversion polynomials that are correct for all SubBytes inputs should be used. If this option is not specified, plaintext/key pairs that result in a `00` as the input to any SubBytes transformation will result in an inconsistent system of equations.

The following options specify the correct SR generator for generating a system of equation in $\mathbb{GF}(2)$ for AES-128:

```
sr = mq.SR(10,4,4,8,star=True,gf2=True,polybori=True, ...
correct_only = True)
```

For a full AES-128 encryption operation this system included 7288 polynomials in algebraic normal form (ANF) and 4544 variables. The variables are specified at the bit level for the inputs to each of AES-128 round, the output of each SubBytes inversion and each bit of each round key. Variables were added to the system to represent the plaintext and ciphertext. The polynomials define the relationship between each round, the inversion in SubBytes, and the key schedule. As a result, the system of equations fully defines the relationship between each of the variables defined by the AES-128 block cipher. If the values of enough of the variables are known, the key can be determined. The value of the some variables may be known from plaintext/ciphertext pairs, ciphertext and intermediate values found using side-channel analysis.

B.1.1   Variable Names.   The SR generator uses the following naming convention for intermediate variables. Inversion refers to the inversion step in the SubBytes operation. Depending how the SubBytes transformation is implemented in a device, this intermediate value may not be calculated.

- $k_{i,j,l}$ subkey: round i, word j, bit l

- $w_{i,j,l}$ inversion input: round i, word j, bit l

- $x_{i,j,l}$ inversion output: round i, word j, bit l

- $P_{j,l}$ plaintext: word j, bit l

- $C_{j,l}$ ciphertext: word j, bit l

For AES-128 the round, word and bit values are always represented as two-character-wide numbers. For example, the variable name for the starting round value for round 1, work 2, bit 10 is written as `w010210`. Note that, although they are not part of a round, both the plaintext and ciphertext use `00` as the round number.

B.2   ANF to CNF Converter

The ANF to CNF converter (anf2cnf) is written for Sage and requires the system of polynomials to be written as a list of Boolean Polynomials, a specific class in Sage. The SR polynomial generator creates a list of Boolean Polynomials over $\mathbb{GF}(2)$ if the option `polybri=True` is used. When creating an instance using the ANFSatSolver() class, the Boolean Ring over which the Boolean Polynomials are defined in Sage must be specified. If `sr` is the name of the SR generator instance, `sr.R` refers to the Boolean Ring used to create the boolean polynomials. An instance of the anf2cnf converter called `anf2cnf` can be created in Sage by typing `anf2cnf` `= ANF2CNF(sr.R)`. If F is the system of equations produced by the SR polynomial generator, `anf2cnf.cnf(F)` will print the DIMACS to the screen.

When a polynomial is in conjunctive normal form each formula is a conjunction of clauses and each clause is a disjunction of literals. In DIMACS format, each line represents a clause and each literal is represented as an integer. A disjunction is represented as multiple integers on the same line. For the DIMACS to be satisfiable, a set of variable assignments must be found to make each line equal to 1.

B.2.1   Specifying Known Values.   The anf2cnf converter assigns integers to variables in the order they are encountered when parsing the system of polynomials. To allow the integers to be mapped back to the original variables, the anf2cnf converter builds a dictionary mapping the variable names to the assigned integer. Constraints for known values can be added by including additional polynomials in the system, by adding additional lines to the DIMACS code, or by using Limboole. If the system of equations is augmented by adding polynomials to the system which represent the known values, the anf2cnf converter will automatically created DIMACS entries for the known values. Constraints can also be added to the DIMACS file created by the anf2cnf converter by generating (a) DIMACS clause(s) for each known value(s). To write DIMACS constraints, the correct integer representation

of the variable must be found using the variable name to integer mapping. This dictionary is also required to use the equations for the constraints generated using Limboole.

B.2.1.1 Limboole. Limboole, a simple Boolean calculator, reads a Boolean formula, checks if it is valid and converts it to a CNF formula in DIMACS format [60]. A number of operators are allowed in the Limboole syntax, including & (and), | (or), and not (!). The order of operations can be specified using parenthesis.

Suppose through SCA it is determined that Hamming Weight of the byte 0 for key round 1 is equal to 1. Although, the values of individual bits are not known, the relationship between the 8 bits in the byte is known. Only one of the bits can be equal to 1, and all others must be zero. In decimal the possible values of the byte are 1, 2, 4, 8, 16, 32, 64, and 128. If the byte is represented as a,b,c,d,e,f,g, the Limboole input is found in Code Listing B.1.

Code Listing B.1   Limboole Code

```
1  !((a&!b&!c&!d&!e&!f&!g&!h) | (!a&b&!c&!d&!e&!f&!g&!h) |
2    (!a&!b&c&!d&!e&!f&!g&!h) | (!a&!b&!c&d&!e&!f&!g&!h) |
3    (!a&!b&!c&!d&e&!f&!g&!h) | (!a&!b&!c&!d&!e&f&!g&!h) |
4    (!a&!b&!c&!d&!e&!f&g&!h) | (!a&!b&!c&!d&!e&!f&!g&h))
```

Although there are only 8 variables in the Limboole input, Limboole produces a DIMACS file with 65 variables and 163 clauses. Many additional intermediate variables are introduced by the conversion process. Like the anf2cnf converter, Limboole assigns integers to the original and intermediate variables created during the conversion from Boolean formula into DIMACS. To make the two DIMACS files compatible, the Limboole DIMACS file must use the same integers to represent the original variables. A Python script was written to translate the Limboole DIMACS integers into the integers used by the anf2cnf converter. Using the Limboole variable assignments listed in the comments of the Limboole DIMACS code, and the dictio-

nary created by the anf2cnf converter, Limboole integers can be substitute with the anf2cnf integer. Any integers from the Limboole DIMACS not associated with one of the original variables are intermediate values, and are assigned to unused integers in the anf2cnf DIMACS file.

B.2.2 SAT Solver. The SAT solver chosen for this research is CryptoMiniSat2 [118]. CryptoMiniSat2 is optimized for working with cryptographic instances allowing for XOR clauses to be reconstructed from the DIMACS input. XOR clauses are treated differently allowing the solver to handle them faster in most scenarios. Since SAT solvers use a standard input format, another SAT solver could easily be substituted instead of CryptoMiniSat2.

B.3 Example Code

B.3.1 Full System of Equations. The code listed in Code Listing B.2 creates a system of polynomials for the AES-128 key with symbolic plaintext and ciphertext variables. Additional equations for known values are imported from a file using the `ImportKnownValuesFromFile()` function list in Code Listing B.5 and the additional polynomials are created by the `KnownValuesPolynomials()` function and added to the system. Finally, the combined system of equations is converted to DIMACS format. This code is based on the instructions at [7] and contained in the `anf2cnf` converter source code.

Code Listing B.2   fullsystem.py

```
1   output_filename = 'DIMACS.cnf'
2   # A symbolic representation for the plaintext (P) and the
3   # cipher text to create equations for fully symbolic AES
4   sr = mq.SR(10,4,4,8,star=True,gf2=True,polybori=True, correct_only = True)
5   R = sr.R
6   vn = sr.varstrs("P", 0, 16, 8) + R.variable_names() + sr.varstrs("C", 0, 16, 8)
7   R = BooleanPolynomialRing(len(vn),vn)
8   sr.R = R
9   C = sr.vars("C",0);
10  P = sr.vars("P",0);
```

```
11  # Generate system of equations (F is the system of equations)
12  F,s = sr.polynomial_system(P=P,C=C);
13  data_filename = 'known_values_format.txt'
14  known_values, actual_key = ImportKnownValuesFromFile(data_filename)
15  KnownValuePolys = KnownValuePolynomials(known_values, sr)
16  # Add the known value polynomials to the system
17  for each_polynomial in KnownValuePolys:
18          F.append(each_polynomial)
19  print('With Known Values: ' + str(F))
20  # Find and save the DIMACS
21  print('\nRunning ANF to CNF...')
22  anf2cnf = ANFSatSolver(sr.R)
23  o = open(output_filename,'w')
24  o.write(anf2cnf.cnf(F))  # DIMACS is output to file
25  o.close()
```

B.3.2 Key Schedule Only System of Equations. The code listed in Code Listing B.3 creates a system of polynomials for the AES-128 key schedule, adds additional equations for known values using the `ImportKnownValuesFromFile()` function in Code Listing B.5, creates the polynomials from the known values using `KnownValuePolynomials()` function in Code List B.7 and converts the combined system of equations into DIMACS format.

Code Listing B.3   keyschsystem.py

```
1   data_filename = "data.txt"
2   sr = mq.SR(10,4,4,8,star=True,gf2=True,polybori=True, correct_only = True, ←
        AES_mode = True)
3   full_system = []
4   for rnd_idx in range(11):
5           key_polys += sr.key_schedule_polynomials(rnd_idx)
6   known_values, actual_key = ImportKnownValuesFromFile(data_filename, ←
        key_values_only = True)
7   KnownValuePolys = KnownValuePolynomials(known_values, sr)
8   full_system = key_polys + KnownValuePolys
9   anf2cnf = ANFSatSolver(sr.R)
10  o = open(inputfile,'w')
11  o.write(anf2cnf.cnf(full_system))  # DIMACS is written to file
12  o.close()
```

B.3.3   Known Values Format.   To simplify testing, the data import format show in Code Listing B.5 was created. All known values are represented by hexadecimal numbers and unknown values are listed as 'X'. This format is parsed using the `ImportKnownValuesFromFile()` function in Code Listing B.5.

Code Listing B.4   knownvaluesformat.sage

```
1   # Note: actual_key is not used as an input to the SAT solver
2   # for testing you can specify the actual key in the R00_k_sch
3
4   actual_key = D810B8F5649A78C08D6E15A80EAE2398
5
6   plaintext =  dec0dedfab1edec0dedfab1edec0dedf
7   R00_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
8   R01_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
9   R01_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXda6cb0ae
10  R02_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
11  R02_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXX38d3bf0f
12  R03_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
13  R03_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXe1c84037
14  R04_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15  R04_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXeea7b960
16  R05_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
17  R05_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
18  R06_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
19  R06_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20  R07_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
21  R07_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
22  R08_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
23  R08_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
24  R09_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
25  R09_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
26  R10_start =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
27  R10_k_sch =  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
28  ciphertext = 07881c0ec03c192a9b6c553e5cfe1b65
```

B.3.4   Helper Functions.   The process of creating a system of equations for AES, introducing known data, converting to a SAT problem and interpreting the output of the SAT solver involved various tools and data formats. A number

of functions were written in Python to import and convert data between types and formats.

To import the known values file, the `ImportKnownValuesFromFile()` function was written. The result is a list of variable names and their values.

Code Listing B.5   ImportKnownValuesFromFile()

```python
def ImportKnownValuesFromFile(known_values_filename, key_values_only = False):
  try:
   file = open(known_values_filename)
   data = file.readlines()
  except IOError as err:
   print('File Error:' + str(err))
  finally:
   if 'file' in locals():
   file.close()

# Import data list. All lines without equal signs will be ignored
  data_list = []
  for each_line in data:
   if each_line.find("=")> -1:
    (row_name, row_values) = each_line.split('=', 1)
    row_name = row_name.strip()
    row_values = row_values.strip()
    data_list.append((row_name, row_values))

# Process each set of data in data_list
  known_list = [];
  for data_pair in data_list:
   (data_name, data_values) = data_pair
   if data_name == 'actual_key':
    actual_key = data_values
   elif data_name == 'plaintext' and not key_values_only:
    prefix = 'P'
    rnd_str = '00'
    known_list += BuildKnownValuesPairs(prefix, rnd_str, data_values)
   elif data_name == 'ciphertext' and not key_values_only:
    prefix = 'C'
    rnd_str = '00'
    known_list += BuildKnownValuesPairs(prefix, rnd_str, data_values)
   elif data_name[:1] == 'R' and data_name[4:] == 'start' and not key_values_only:
    prefix = 'w'
```

```
36        rnd_str = data_name[1:3]
37        known_list += BuildKnownValuesPairs(prefix, rnd_str, data_values)
38      elif data_name[:1] == 'R' and data_name[4:] == 'k_sch':
39        prefix = 'k'
40        rnd_str = data_name[1:3]
41        known_list += BuildKnownValuesPairs(prefix, rnd_str, data_values)
42
43    return known_list, actual_key
```

The `BuildKnownValuesPair()` function builds the name of the variables from the prefix, round, and each of the string of values specified by the input file. The function hex2binX() is a simple hexadecimal to binary converter but also converts a 'X' in the hex string to a 'XXXX' in the binary string. The output is a paired list of the variables with known binary values.

<div align="center">Code Listing B.6   BuildKnownValuesPair()</div>

```
1   def BuildKnownValuesPairs(prefix, rnd_str, data_values_hex):
2           known_list = []
3           data_value_binary = hex2binX(data_values_hex)
4           for word_num in range(16):
5                   word_str = str(word_num).zfill(2)
6                   for bit_num in range(8):
7                           bit_str = str(bit_num).zfill(2)
8                           bit_idx = 8 * word_num + bit_num
9                           var_name = prefix + rnd_str + word_str + bit_str
10                          if not data_value_binary[bit_idx] == 'X':
11                                  var_value = Integer(data_value_binary[bit_idx])
12                                  known_list.append((var_name, var_value))
13          return known_list
```

The `KnownValuePolynomial()` function takes the paired list of variables and known binary values and creates polynomials to represent the known values using the Boolean Polynomial class required by the anf2cnf converter. The SR generator's `variable_dict()` function produces a dictionary that relates the variable name with its corresponding Boolean Monomial. The Boolean Monomial must be used in the polynomial for the polynomial to be a Boolean Polynomial class. Since each polynomial in ANF is equal to 0 if the variable is equal to 1, 1 is added to the variable

to create a polynomial equal to zero over $\mathbb{GF}(2)$. If the variable is equal to 0, the Boolean Monomial is simply appended to the list of polynomials.

Code Listing B.7   KnownValuePolynomials()

```
def KnownValuePolynomials(inputList, srGenerator):
        # get the boolean polynomial to integer dictionary
        PBDict = srGenerator.variable_dict()
        # create an empty list
        IntValuePolynomials = [];
        for each_item in inputList:
                varstr = each_item[0]    # variable name
                myBP = PBDict[varstr]    # BooleanPolynomial for that name
                if each_item[1]:
                        IntValuePolynomials.append(myBP + 1)
                else:
                        IntValuePolynomials.append(myBP)
        return tuple(IntValuePolynomials)
```

Appendix C. List of Acronyms

Acronym	Definition
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
AES-128	Advanced Encryption Standard (128-bit variant)
ANF	Algebraic Normal Form
ARK	AddRoundKey
CBC	Cipher Block Chaining
CEMA	Correlation-based Electro-Magnetic Analysis
CFB	Cipher Feedback
CMOS	Complementary Metal Oxide Semiconductor
CNF	Conjunctive Normal Form
CTR	Counter
DAB	Digital Audio Broadcasting
DDC	Digital Down-Converter
DEMA	Differential Electro-Magnetic Analysis
DES	Data Encryption Standard
DFT	Discrete Fourier Transform
DIMACS	Discrete Mathematics & Theoretical Computer Science
DPA	Differential Power Analysis
DRAM	Dynamic Random Access Memory

Acronym	Definition
DSCA	Differential Side-Channel Analysis
DVB-T	Digital Video Broadcasting-Terrestrial
DoM	Difference of Means
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
EM	Electro-Magnetic
EMA	Electro-Magnetic Analysis
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GNU	GNU's Not Unix (recursive)
HD	Hamming Distance
HDSDR	High Definition
HW	Hamming Weight
I/Q	In-phase Quadrature
IF	Intermediate Frequency
KSRA	Key Schedule Redundancy Attack
LNA	Low Noise Amplifier
MC	SubBytes
ML	Maximum-Likelihood
MVN	Mean and Variance Normalization

Acronym	Definition
NCO	Numerically-Controlled Oscillator
NIST	National Institute of Standards and Technology
OFB	Output Feedback
PBOPT	Pseudo-Boolean Optimization Problem
PCA	Principal Component Analysis
PIOSC	Precision Oscillator
PSD	Power Spectral Density
PoSSo	Polynomial System Solving
RAM	Random Access Memory
RF	Radio Frequency
RTL-SDR	Realtek RLT2832U-based Software Defined Radio
S-box	Substitution Box
SASEBO	Side-channel Attack Standard Evaluation Board
SAT	SATisfiability
SB	SubBytes
SCA	Side-Channel Analysis
SDR	Software Defined Radio
SEMA	Simple Electro-Magnetic Analysis
SNR	Signal-to-Noise Ratio
SPA	Simple Power Analysis

Acronym	Definition
SR	ShiftRows (Figure 2.1 only)
SR	Small Scale Variants of the AES
SSCA	Simple Side-Channel Analysis
SYMAES	Symbolic AES
TASCA	Tolerant Algebraic Side-Channel Analysis
UART	Universal Asynchronous Receiver/Transmitter
UNSAT	Unsatisfiable
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
XOR	eXclusive-OR
ZIF	Zero Insertion Force

Bibliography

1. Agrawal, Dakshi, et al. "Advances in Side-Channel Cryptanalysis Electromagnetic Analysis and Template Attacks," *RSA Laboratories Cryptobytes*, *6*:20–32 (2003).

2. Agrawal, Dakshi, et al. "The EM Side–Channel(s)." *Cryptographic Hardware and Embedded Systems - CHES 2002 2523*. Lecture Notes in Computer Science, edited by Burton Kaliski, et al., 29–45, Springer Berlin / Heidelberg, 2003.

3. Agrawal, Dakshi, et al. "Templates as Master Keys." *Cryptographic Hardware and Embedded Systems CHES 2005 3659*. Lecture Notes in Computer Science, edited by Josyula Rao and Berk Sunar, 15–29, Springer Berlin / Heidelberg, 2005.

4. Agrawal, Dakshi, et al. *Multi-channel Attacks*, *2779*. Lecture notes in computer science, 2–16. Berlin, Heidelberg: Springer, 2003.

5. Albrecht, Martin and Carlos Cid. "Algebraic Techniques in Differential Cryptanalysis." *Fast Software Encryption 5665*. Lecture Notes in Computer Science, edited by Orr Dunkelman, 193–208, Springer Berlin / Heidelberg, 2009.

6. Albrecht, Martin and Carlos Cid, "Cold Boot Key Recovery by Solving Polynomial Systems with Noise." Cryptology ePrint Archive, Report 2011/038, 2011.

7. Albrecht, Martin and Niles Johnson, "Small Scale Variants of the AES (SR) Polynomial System Generator," 2011.

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

9. Archambeau, C., et al. "Template Attacks in Principal Subspaces." *Cryptographic Hardware and Embedded Systems - CHES 2006 4249*. Lecture Notes in Computer Science, edited by Louis Goubin and Mitsuru Matsui, 1–14, Springer Berlin / Heidelberg, 2006.

10. Asonov, Dmitri and Rakesh Agrawal. "Keyboard Acoustic Emanations." *IEEE Symposium on Security and Privacy*. 3–11. 2004.

11. Bard, Gregory V. *Algebraic Cryptanalysis*. Springer, 2009.

12. Bard, Gregory V., et al., "Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers." Available at, 2007.

13. Barenghi, Alessandro, et al. "Improving first order differential power attacks through digital signal processing." *Proceedings of the 3rd international conference on Security of information and networks*. SIN '10. 124–133. New York, NY, USA: ACM, 2010.

14. Barenghi, Alessandro, et al. "Information Leakage Discovery Techniques to Enhance Secure Chip Design." *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication 6633*. Lecture Notes in Computer Science, edited by Claudio Ardagna and Jianying Zhou, 128–143, Springer Berlin / Heidelberg, 2011.

15. Batina, Lejla, et al. "Comparative Evaluation of Rank Correlation Based DPA on an AES Prototype Chip." *Information Security 5222*. Lecture Notes in Computer Science, edited by Tzong-Chen Wu, et al., 341–354, Springer Berlin / Heidelberg, 2008.

16. Biham, Eli and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology*, *4*:3–72 (1991).

17. Biham, Eli and Adi Shamir. "Differential fault analysis of secret key cryptosystems." *Advances in Cryptology CRYPTO '97 1294*. Lecture Notes in Computer Science, edited by Burton Kaliski, 513–525, Springer Berlin / Heidelberg, 1997.

18. Biryukov, Alex and Christophe De Cannire. "Block Ciphers and Systems of Quadratic Equations." *Fast Software Encryption 2887*. Lecture Notes in Computer Science, edited by Thomas Johansson, 274–289, Springer Berlin / Heidelberg, 2003.

19. Blossom, Eric. "GNU radio: tools for exploring the radio frequency spectrum," *Linux Journal*, *2004*(122):4 (June 2004).

20. Boak, David G., "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`.

21. Brickenstein, Michael and Alexander Dreyer. "PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials," *Journal of Symbolic Computation*, *44*(9):1326 – 1345 (2009). Effective Methods in Algebraic Geometry.

22. Brier, Eric, et al. "Correlation Power Analysis with a Leakage Model." *CHES*. 16–29. 2004.

23. Chari, Suresh, et al. "Towards Sound Approaches to Counteract Power-Analysis Attacks." *Advances in Cryptology CRYPTO 99 1666*. Lecture Notes in Computer Science, edited by Michael Wiener, 791–791, Springer Berlin / Heidelberg, 1999.

24. Chari, Suresh, et al. "Template Attacks." *Cryptographic Hardware and Embedded Systems - CHES 2002 2523*. Lecture Notes in Computer Science, edited by Burton Kaliski, et al., 51–62, Springer Berlin / Heidelberg, 2003.

25. Cid, C., et al. "Small Scale Variants of the AES." *Fast Software Encryption 3557*. Lecture Notes in Computer Science, edited by Henri Gilbert and Helena Handschuh, 145–162, Springer Berlin / Heidelberg, 2005.

26. Cid, Carlos. "Some Algebraic Aspects of the Advanced Encryption Standard." *Advanced Encryption Standard AESgro 3373*. Lecture Notes in Computer Science, edited by Hans Dobbertin, et al., 571–571, Springer Berlin / Heidelberg, 2005.

27. Cid, Carlos, et al. *Algebraic Aspects of the Advanced Encryption Standard*. Springer Verlag, 2006.

28. Cid, Carlos and Ralf-Philipp Weinmann. "Block Ciphers: Algebraic Cryptanalysis and Grobner Bases." *Grobner Bases, Coding, and Cryptography* 307–327, Springer Berlin Heidelberg, 2009.

29. Cobb, W., et al. "Intrinsic Physical Layer Authentication of Integrated Circuits," *Information Forensics and Security, IEEE Transactions on*, $7(99){:}14$–24 (2011).

30. Cobb, William. *Exploition of the Unintentional Information Leakage of Integrated Circuits*. PhD dissertation, Air Force Institute of Technology, 2011.

31. Cobb, William E., et al. "Physical layer identification of embedded devices using RF-DNA fingerprinting." *Military Communications Conference, 2010 – MILCOM 2010*. 2168 –2173. November 2010.

32. Communication Engineering Lab (CEL) at the Karlsruhe Institute of Technology (KIT), Germany, "Simulink-RTL-SDR: A Simulink interface for rtl-sdr."

33. Cormen, Thomas H, et al. *Introduction to algorithms*. MIT press, 2001.

34. Coron, Jean-Sbasticn, et al. "Statistics and Secret Leakage." *Financial Cryptography 1962*. Lecture Notes in Computer Science, edited by Yair Frankel, 157–173, Springer Berlin / Heidelberg, 2001.

35. Courtois, Nicolas, et al. "Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations." *Advances in Cryptology EUROCRYPT 2000 1807*. Lecture Notes in Computer Science, edited by Bart Preneel, 392–407, Springer Berlin / Heidelberg, 2000.

36. Courtois, Nicolas and Josef Pieprzyk. "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations." *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*. ASIACRYPT '02. 267–287. London, UK, UK: Springer-Verlag, 2002.

37. Courtois, Nicolas T. and Gregory V. Bard. "Algebraic cryptanalysis of the data encryption standard." *Proceedings of the 11th IMA international conference on*

Cryptography and coding. Cryptography and Coding'07. 152–169. Berlin, Heidelberg: Springer-Verlag, 2007.

38. Daemen, Joan and Vincent Rijmen, "The Rijendael Block Cipher. Version 2," 1999.

39. Daemen, Joan and Vincent Rijmen. *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

40. De Mulder, Elke. *Electromagnetic Techniques and Probes for Side-Channel Analysis on Cryptographic Devices*. PhD dissertation, Arenberg Doctoral School of Science, Engineering & Technology, 2010.

41. Elaabid, M. and Sylvain Guilley. "Portability of templates," *Journal of Cryptographic Engineering*, *2*:63–74 (2012).

42. Elonics Ltd, "Elonics E4000 product page," Retrieved October 2, 2012.

43. Ettus Research, "LFRX Daughterboard 1-250 MHz Rx," 2012.

44. Faugère, Jean Charles, "A New Efficient algorithm for Computing Gröbner Basis, F4," 1999.

45. Faugère, Jean Charles. "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)." *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*. ISSAC '02. 75–83. New York, NY, USA: ACM, 2002.

46. Fisher, R.A. "Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population," *Biometrika*, *10*(4):507–521 (1915).

47. Gandolfi, Karine, et al. "Electromagnetic Analysis: Concrete Results." *Cryptographic Hardware and Embedded Systems CHES 2001 2162*. Lecture Notes in Computer Science, edited by Cetin Koç, et al., 251–261, Springer Berlin / Heidelberg, 2001.

48. Gebotys, Catherine, et al. "EM Analysis of Rijndael and ECC on a Wireless Java-based PDA." *Cryptographic Hardware and Embedded Systems CHES 2005 3659*. Lecture Notes in Computer Science, edited by Josyula Rao and Berk Sunar, 250–264, Springer Berlin / Heidelberg, 2005.

49. Gebotys, Catherine and Brian White. "EM alignment using phase for secure embedded systems," *Design Automation for Embedded Systems*, *12*:185–206 (2008).

50. Gebotys, Catherine H. and Brian A. White. "EM analysis of a wireless Java-based PDA," *ACM Trans. Embed. Comput. Syst.*, *7*:44:1–44:28 (August 2008).

51. Gibbs, W. "How to Steal Secrets," *Scientific American Magazine*, *300*(5):58–63 (2009).

52. Gierlichs, Benedikt, et al. "Mutual Information Analysis." *Cryptographic Hardware and Embedded Systems CHES 2008 5154*. Lecture Notes in Computer Science, edited by Elisabeth Oswald and Pankaj Rohatgi, 426–442, Springer Berlin / Heidelberg, 2008.

53. Gierlichs, Benedikt, et al. "Templates vs. Stochastic Methods." *Cryptographic Hardware and Embedded Systems - CHES 2006 4249*. Lecture Notes in Computer Science, edited by Louis Goubin and Mitsuru Matsui, 15–29, Springer Berlin / Heidelberg, 2006.

54. Goubin, Louis and Jacques Patarin. "DES and Differential Power Analysis The Duplication Method." *Cryptographic Hardware and Embedded Systems 1717*. Lecture Notes in Computer Science, edited by Cetin Koç and Christof Paar, 728–728, Springer Berlin / Heidelberg, 1999.

55. Halderman, J. Alex, et al. "Lest we remember: cold boot attacks on encryption keys." *Proceedings of the 17th conference on Security symposium*. 45–60. Berkeley, CA, USA: USENIX Association, 2008.

56. Heuser, Annelie, et al. "A New Difference Method for Side-Channel Analysis with High-Dimensional Leakage Models." *Topics in Cryptology – CT-RSA 2012 7178*. Lecture Notes in Computer Science, edited by Orr Dunkelman, 365–382, Springer Berlin Heidelberg, 2012.

57. Hintze, Jerry L and Ray D Nelson. "Violin plots: a box plot-density trace synergism," *The American Statistician*, *52*(2):181–184 (1998).

58. Hodgers, Philip, et al. "Power Spectral Density Side Channel Attack Overlapping Window Method." *14th Euromicro Conference on Digital System Design*. 274 –278. 2011.

59. Homma, Naofumi, et al. "High-Resolution Side-Channel Attack Using Phase-Based Waveform Matching." *Cryptographic Hardware and Embedded Systems - CHES 2006 4249*. Lecture Notes in Computer Science, edited by Louis Goubin and Mitsuru Matsui, 187–200, Springer Berlin / Heidelberg, 2006. Includes POC method from Java PDA paper.

60. Institute for Formal Models and Verification at the Johannes Kepler University in Linz, Austria. *Limboole*, 2003. `http://fmv.jku.at/limboole/`.

61. Jun, Benjamin and Gary Kenworth, "Is Your Mobile Device Radiating Keys?." RSA Conference Presentation, 2012.

62. Kamal, Abdel Alim and Amr M. Youssef, "Applications of SAT Solvers to AES key Recovery from Decayed Key Schedule Images," 2010.

63. Karlof, Chris and David Wagner. "Hidden Markov Model Cryptanalysis." *Cryptographic Hardware and Embedded Systems - CHES 2003 2779*. Lecture Notes

in Computer Science, edited by Colin Walter, et al., 17–34, Springer Berlin / Heidelberg, 2003.

64. Kim, ChanKyun, et al. "Differential Side Channel Analysis Attacks on FPGA Implementations of ARIA," *ETRI journal*, *30*:315–325 (2008).

65. Kim, Chong Hee. "Improved Differential Fault Analysis on AES Key Schedule," *Information Forensics and Security, IEEE Transactions on*, *7*(1):41 –50 (feb. 2012).

66. Kocher, Paul. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems." *Advances in Cryptology CRYPTO 96 1109*. Lecture Notes in Computer Science, edited by Neal Koblitz, 104–113, Springer Berlin / Heidelberg, 1996.

67. Kocher, Paul, et al. "Differential Power Analysis." *Advances in Cryptology CRYPTO 99 1666*. Lecture Notes in Computer Science, edited by Michael Wiener, 789–789, Springer Berlin / Heidelberg, 1999.

68. Kocher, Paul, et al. "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, 1–23 (2011).

69. Maes, R. and P. Tuyls. *Secure integrated circuits and systems*. Springer, New York, 2010.

70. Mangard, Stefan. "Exploiting Radiated Emissions - EM Attacks on Cryptographic ICs." *Processings of Austrochip*. 2003.

71. Mangard, Stefan. "A simple power-analysis (SPA) attack on implementations of the AES key expansion." *Proceedings of the 5th international conference on Information security and cryptology*. ICISC'02. 343–358. Berlin, Heidelberg: Springer-Verlag, 2003.

72. Mangard, Stefan. "Hardware Countermeasures against DPA A Statistical Analysis of Their Effectiveness." *Topics in Cryptology CT-RSA 2004 2964*. Lecture Notes in Computer Science, edited by Tatsuaki Okamoto, 1998–1998, Springer Berlin / Heidelberg, 2004.

73. Mangard, Stefan, et al. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

74. Markgraf, Steve and Dimitri Stolnikov, "rtl-sdr OsmoSDR," 2012.

75. Massacci, Fabio and Laura Marraro. "Logical Cryptanalysis as a SAT Problem," *Journal of Automated Reasoning*, *24*:165–203 (2000).

76. Matsui, Mitsuru. "Linear Cryptanalysis Method for DES Cipher." *Advances in Cryptology EUROCRYPT 93 765*. Lecture Notes in Computer Science, edited by Tor Helleseth, 386–397, Springer Berlin / Heidelberg, 1994.

77. Messerges, Thomas S., et al. "Investigations of Power Analysis Attacks on Smartcards." *Usenix Workshop on Smartcard Technology 1999*. 1999.

78. Messerges, T.S., et al. "Examining smart-card security under the threat of power analysis attacks," *Computers, IEEE Transactions on*, *51*(5):541 –552 (may 2002).

79. Meynard, O., et al. "Enhancement of simple electro-magnetic attacks by pre-characterization in frequency domain and demodulation techniques." *Design, Automation Test in Europe Conference Exhibition (DATE) 2011*. 1 –6. march 2011.

80. Mishali, M. and Y.C. Eldar. "Sub-Nyquist Sampling," *Signal Processing Magazine, IEEE*, *28*(6):98 –124 (nov. 2011).

81. Mohamed, Mohamed Saied Emam, et al., "Improved Algebraic Side-Channel Attack on AES." Cryptology ePrint Archive, Report 2012/084, 2012. `http://eprint.iacr.org/`.

82. Montminy, David P., et al. "An algebraic side-channel attack on the AES key schedule," *International Journal of Applied Cryptography* (2013). Manuscript submitted for publication.

83. Montminy, David P., et al. "Cross-device attacks on complex microprocessors," *Journal of Cryptographic Engineering* (2013). Manuscript submitted for publication.

84. Montminy, David P., et al. "Differential Electromagnetic Attacks on a 32-bit Microprocessor Using Software Defined Radios," *Information Forensics and Security, IEEE Transactions on* (2013). In Review.

85. Montminy, David P., et al. "Improving cross-device attacks using zero-mean unit-variance normalization," *Journal of Cryptographic Engineering*, 1–12 (2012). Manuscript submitted for publication.

86. Murphy, Sean and Matthew Robshaw. "Essential Algebraic Structure within the AES." *Advances in Cryptology CRYPTO 2002 2442*. Lecture Notes in Computer Science, edited by Moti Yung, 1–16, Springer Berlin / Heidelberg, 2002.

87. NIST, "NIST Special Publication 800-38A." Online.

88. NIST, "FIPS 197," November 26 2001.

89. Oren, Y., et al. "Algebraic Side-Channel Analysis Beyond the Hamming Weight Leakage Model." *Cryptographic Hardware and Embedded Systems, CHES 2012 7428*. Lecture Notes in Computer Science, 140–154, Springer Berlin / Heidelberg, 2012.

90. Oren, Yossef, et al. "Algebraic Side-Channel Analysis in the Presence of Errors." *Cryptographic Hardware and Embedded Systems, CHES 2010 6225*. Lecture Notes in Computer Science, edited by Stefan Mangard and Franois-Xavier Standaert, 428–442, Springer Berlin / Heidelberg, 2010.

91. Oren, Yossef and Avishai Wool, "Tolerant Algebraic Side-Channel Analysis of AES." Cryptology ePrint Archive, Report 2012/092, 2012. `http://eprint.iacr.org/`.

92. Oswald, Elisabeth E. and Stefan Mangard. *Template Attacks on Masking - Resistance Is Futile*, *4377*. Lecture notes in computer science, 243–256. Berlin, Heidelberg: Springer, 2007.

93. Ott, Henry W. *Electromagnetic compatibility engineering*. John Wiley & Sons, Inc., 2009.

94. Quisquater, Jean-Jacques and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards." *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*. E-SMART '01. 200–210. London, UK, UK: Springer-Verlag, 2001.

95. Quisquater, Jean-Jacques and David Samyde. "Automatic code recognition for smart cards using a kohonen neural network." *Proceedings of the 5th conference on Smart Card Research and Advanced Application Conference-Volume 5*. 6–6. 2002.

96. Raddum, Håvard and Igor Semaev, "New Technique for Solving Spase Equation Systems." Cryptology ePrint Archive, 2006.

97. Realtek, "Realtek RTL2832U," Retrieved 11 Dec 2012.

98. Rechberger, C. and E. Oswald. "Stream ciphers and side-channel analysis." *Proceedings of the ECRYPT Workshop, SASC-The State of the Art of Stream Ciphers*. 320–326. 2004.

99. Rechberger, Christian and Elisabeth Oswald. "Practical Template Attacks." *Information Security Applications 3325*. Lecture Notes in Computer Science, edited by Chae Lim and Moti Yung, 440–456, Springer Berlin / Heidelberg, 2005.

100. Renauld, Mathieu and François-Xavier Standaert. "Combining Algebraic and Side-Channel Cryptanalysis against Block Ciphers." *30th Symposium on Information Theory in the Benelux*. May 2009.

101. Renauld, Mathieu and François-Xavier Standaert. "Algebraic side-channel attacks." *Proceedings of the 5th international conference on Information security and cryptology*. Inscrypt'09. 393–410. Berlin, Heidelberg: Springer-Verlag, 2010.

102. Renauld, Mathieu, et al. "Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA." *Cryptographic Hardware and Embedded Systems - CHES 2009 5747*. Lecture Notes in Computer Science, edited by Christophe Clavier and Kris Gaj, 97–111, Springer Berlin / Heidelberg, 2009.

103. Renauld, Mathieu, et al. "A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices." *Advances in Cryptology EURO-CRYPT 2011 6632*. Lecture Notes in Computer Science, edited by Kenneth Paterson, 109–128, Springer Berlin / Heidelberg, 2011.

104. Riscure, "Inspector Data Sheet."

105. Riscure, "Inspector - The Side Channel Test Platform." Online, 2009. Retrieved on 25 July, 2011, from `http://www.riscure.com/inspector/product-description.html`.

106. Riscure, "icWaves, Inspector Data Sheet," 2011.

107. Sauvage, Laurent, et al. "ElectroMagnetic Radiations of FPGAs: High Spatial Resolution Cartography and Attack of a Cryptographic Module," *ACM Transactions on Reconfigurable Technology and Systems* (2008).

108. Schindler, Werner, et al. "A Stochastic Model for Differential Side Channel Cryptanalysis." *Cryptographic Hardware and Embedded Systems CHES 2005 3659*. Lecture Notes in Computer Science, edited by Josyula Rao and Berk Sunar, 30–46, Springer Berlin / Heidelberg, 2005.

109. Schlösser, Alexander, et al. "Simple Photonic Emission Analysis of AES," *Cryptographic Hardware and Embedded Systems–CHES 2012*, 41–57 (2012).

110. Schneier, Bruce. "A self-study course in block-cipher cryptanalysis," *Cryptologia*, 1833 (2000).

111. Shamir, A. 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/.

112. Shamir, Adi and Aviad Kipnis. "Cryptanalysis of the HFE Public Key Cryptosystem." *Advances in Cryptography, Proceedings in Crypto '99*. Springer-Verlag, LNCS, 1999. Could not find the PDF.

113. Shannon, C. E. "Communication in the presence of noise," *Proc. IRE*, *37*:10–21 (1949).

114. Shannon, Claude Elwood. "Communication theory of secrecy systems," *Bell System Technical Journal*, *28*:656–715 (1949).

115. Sklar, B. *Digital communications: fundamentals and applications*. Prentice Hall Communications Engineering and Emerging Technologies Series, Prentice-Hall PTR, 2001.

116. Skorobogatov, S. "Using Optical Emission Analysis for Estimating Contribution to Power Analysis." *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on*. 111 –119. sept. 2009.

117. Skorobogatov, Sergei P. *Semi-invasive attacks - A new approach to hardware security analysis*. PhD dissertation, University of Cambridge, 2005.

118. Soos, Mate, "Limits of SAT Solvers in Cryptography." Presented at CASED, July 2011.

119. Soos, Mate and Martin Albrecht, "INRIAGForge: anf2cnf: Project Info.," 2010.

120. Soos, Mate, et al. "Extending SAT Solvers to Cryptographic Problems." *Theory and Applications of Satisfiability Testing - SAT 2009 5584*. Lecture Notes in Computer Science, edited by Oliver Kullmann, 244–257, Springer Berlin / Heidelberg, 2009.

121. Standaert, François-Xavier, et al. "Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices." *Information Security and Cryptology ICISC 2008 5461*. Lecture Notes in Computer Science, edited by Pil Lee and Jung Cheon, 253–267, Springer Berlin / Heidelberg, 2009.

122. Standaert, Franois-Xavier and Cedric Archambeau. "Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages." *Cryptographic Hardware and Embedded Systems CHES 2008 5154*. Lecture Notes in Computer Science, edited by Elisabeth Oswald and Pankaj Rohatgi, 411–425, Springer Berlin / Heidelberg, 2008.

123. Standaert, O.-X., et al. "An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays," *Proceedings of the IEEE*, *94*(2):383 –394 (Feb. 2006).

124. Stein, W. A. and others. *Sage Mathematics Software (Version 4.7b)*. The Sage Development Team, 2011. `http://www.sagemath.org`.

125. Stinson, Douglas Robert. *Cryptography: theory and practice*. CRC press, 2006.

126. Taeubel, Mario, "High Definition Software Defined Radio (HDSDR)."

127. Talbot, John and Dominic Welsh. *Complexity and Cryptography: An Introduction*. New York, NY, USA: Cambridge University Press, 2006.

128. Texas Instruments, "Stellari ARM Cortex-M4F Microcontrollers Applications."

129. Texas Instruments, "Stellaris LM4F232 Evaluation Board User's Manual," 2011.

130. Texas Instruments, "Stellaris LM4F232H5QD Microcontroller Data Sheet," 2012.

131. Tiri, Kris and Ingrid Verbauwhede, "Synthesis of Secure FPGA Implementations." Cryptology ePrint Archive, Report 2004/068, 2004. `http://eprint.iacr.org/`.

132. Trappe, Wade and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005.

133. Tuttlebee, Wally H. W., editor. *Software defined radio: Origins, drivers, and international perspectives*. West Sussex, England: John Wiley, 2002.

134. Velichkov, Vesselin, et al. "SYMAES: A Fully Symbolic Polynomial System Generator for AES-128." *Proceedings of the ECRYPT Workshop on Tools for Cryptanalysis 2010*, edited by Francois-Xavier Standaert. June 2010.

135. Vuagnoux, M. and S. Pasini. "An improved technique to discover compromising electromagnetic emanations." *2010 IEEE International Symposium on Electromagnetic Compatibility (EMC)*. 121–126. July 2010.

136. Vuagnoux, Martin and Sylvain Pasini. "Compromising electromagnetic emanations of wired and wireless keyboards." *Proceedings of the 18th conference on USENIX security symposium*. SSYM'09. 1–16. Berkeley, CA, USA: USENIX Association, 2009.

137. Zhao, X., et al. "SAT based Error Tolerant Algebraic Side-Channel Attacks." *Submitted to SCIENCE CHINA Information Sciences*. 2011.

138. Zhuang, Li, et al. "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security*, *13*:3:1–3:26 (November 2009).

REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
15–09–2013	Doctoral Dissertation	Sep 2010-Sep 2013

4. TITLE AND SUBTITLE

Enhancing Electromagnetic Side-Channel Analysis
in an Operational Environment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Montminy, David P., 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-ENG-DS-13-S-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Side-channel attacks exploit the unintentional emissions from cryptographic devices to determine the secret encryption key. This research identifies methods to make attacks demonstrated in an academic environment more operationally relevant. Algebraic cryptanalysis is used to reconcile redundant information extracted from side-channel attacks on the AES key schedule. A novel thresholding technique is used to select key byte guesses for a satisfiability solver resulting in a 97.5% success rate despite failing for 100% of attacks using standard methods. Two techniques are developed to compensate for differences in emissions from training and test devices dramatically improving the effectiveness of cross device template attacks. Mean and variance normalization improves same part number attack success rates from 65.1% to 100%, and increases the number of locations an attack can be performed by 226%. When normalization is combined with a novel technique to identify and filter signals in collected traces not related to the encryption operation, the number of traces required to perform a successful attack is reduced by 85.8% on average. Finally, software-defined radios are shown to be an effective low-cost method for collecting side-channel emissions in real-time, eliminating the need to modify or profile the target encryption device to gain precise timing information.

15. SUBJECT TERMS

Side-Channel Analysis, Cross-Device, Mean & Variance Normalization, Software Defined Radios, Algebraic Cryptanalysis

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	247	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4445; email:rusty.baldwin@afit.edu