COVERT/SIDE CHANNEL ANALYSIS, MODELING AND CAPACITY ESTIMATION

A Dissertation
Presented to
The Academic Faculty

By

Baki Berkay YILMAZ

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2020

COVERT/SIDE CHANNEL ANALYSIS, MODELING AND CAPACITY ESTIMATION

Approved by:

Prof. Alenka Zajic, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Milos Prvulovic, Advisor
School of Computer Science
Georgia Institute of Technology

Prof. Matthieu Ratoslav Bloch
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Angelos D. Keromytis
School of Electrical and Computer Engineering
Georgia Institute of Technology

Prof. Alessandro Orso
School of Computer Science
Georgia Institute of Technology

Prof. Gregory David Durgin
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: March 26, 2020

If one day, my words are against science, choose science.

Mustafa Kemal Atatürk

This thesis is dedicated to my parents Nejla and Erdal Yılmaz, to my sister İrem Yılmaz, and my beloved wife Sena Türk Yılmaz. Without their support, love and patience, I would have given up long ago.

Onur Özkutlu, this is also for you brother. You will always be in our hearts.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and deepest appreciation to my supervisors Prof. Alenka Zajic and Prof. Milos Prvulovic for their support, effort and guidance. Moreover, I would also like to express my indebtedness to Prof. Matthieu Ratoslav Bloch, Prof. Angelos D. Keromytis, Prof. Alessandro Orso and Prof. Gregory David Durgin for their critical reading, insightful comments and appearing in my thesis committee.

In addition, I would like to thank Prof. Alper T. Erdogan, who made me believe in the importance of research and science even your work is underappreciated.

I would also like to offer my sincerest gratitude to my friends in Italy, Switzerland, Germany, Netherlands, UK, US and Turkey. I also feel obliged to express my deepest appreciation to the members of Electromagnetic Measurements in Communications and Computing for their profound impact on my work, motivation and knowledge.

My parents, Erdal Yılmaz and Nejla Yılmaz, and my sister, İrem Yılmaz, desire my heartfelt appreciation for their endless support, camaraderie, encouragement and caring they provided. Without all these, I might have quit long ago.

Finally, I want to express my intense devotion to my wife, Sena Türk Yılmaz. She made me believe in myself and gave me billions of reasons to wake up with hope every single day. Without her, I might have been lost in troubles.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis develops methods to analyze and model covert/side channels and electromagnetic signals emitted during program execution, and estimate their channel capacity to comprehend the severity of information leakage that is a result of hardware and software activities, and provides guidelines to minimize emanations to enable system designs more resilient to side/covert channel attacks.

Side/Covert channels are asynchronous channels which are not designed nor intended to transfer information. These channels are generated as a byproduct of performing legitimate program activities on the hardware of computer systems. Various approaches have been proposed in the literature to model such channels to analyze and estimate information leakage capacity. Main drawbacks of current approaches are that they do not consider 1) asynchronous nature of side/covert channels, 2) variability in execution time of each instruction, and 3) interrupts due to other software activities. Ignoring any of these features can result in underestimating the severity of information leakage, and inaccurate models that can mislead the analysis of these channels.

Likewise, having a tool to analyze the emanated electromagnetic signals in the *design-stage* can reduce the cost of production since designs can be tested before they are manufactured. This tool also helps reducing the flaws that causing information leakages through side/covert channels and securing *sensitive* information of customers. To successfully evaluate the severity of side/covert channels, our research has 1) defined a metric that calculates the emanated signal power difference of two instructions and modeled the covert/side channels as a discrete memoryless channel

with uneven symbol transmission length, 2) defined another metric to calculate the emanated signal power for a single instruction and established a connection with Markov Source Models to model side channels considering the dependency among instructions as a consequence of processor pipeline and program functionality, 3) modeled and analyzed a covert channel, which is deliberately generated with electromagnetic (EM) emanations due to computer activities, by adopting methodologies from conventional communication systems, and proposed bounds for the capacity of these channels, 4) introduced a generalized model for covert channels with different sources (i.e. power, cache, EM, etc.) and an assessment methodology that can be utilized by designers to analyze their systems against attacks based on these channels, and 5) modeled side channel signals emanated while executing instruction sequences on a processor, which leverages *design-stage* investigation of new products. The work provides a deep understanding of side/covert channels generated by program activities which can be utilized to secure devices by optimizing their designs to minimize information leakage.

CHAPTER 1

INTRODUCTION

1.1 Motivation

A side/covert channel is a communication channel that is neither designed nor intended to transfer information [1]. Signals from these channels are generated as a side effect of performing legitimate program activity on the hardware of a device. Since program activity and the resulting hardware activity are dependent on data processed by the program, the resulting side/covert channel signals can (and usually do) carry information about those data values. This allows attackers to obtain sensitive information by analyzing side/covert channel signals that are produced by programs which process this sensitive information.

Covert/side-channel attacks have been acknowledged as a serious security threat [2]. The severity of that threat dramatically increases as computer systems are increasingly getting mobile and may be physically controlled or placed close to potentially malicious entities. For example, previous work [3, 4, 5] confirm that modulated EM emanations from laptop and desktop systems can be created by executing seemingly innocuous code, and EM covert channel transmission of thousands of bits per second has been demonstrated to distances of at least several meters and even through a wall. Because the "transmitter" code for EM covert channel attacks is innocuous-looking, and because reception of the emanations does not require direct contact with the system under attack, numerous EM side/covert channel attacks may be carried out completely undetected.

In information theory, covert channels refer to *low probability of detection communications* [6]. These communications are designed to prevent the existence of transmission in the first place. In these channels, secondary users communicate covertly over the existent communication channels that are utilized by the primary users. Another description for these channels is that users desire to transmit information in secret through a known communication channel to maintain their privacy, etc. The general practice for these channels is to exploit Gaussian channels, pure loss quantum channels and thermal noise quantum channels [7]. Please keep in mind that the covert communication in the information theory literature aims to embed the information into the existent synchronous channel so that the entropy between the transmitted bit sequence and zero sequence is almost zero. In other words, an adversary could not differentiate whether the transmitted bit sequence is either zeros or any other non-zero bit sequence [8]. However, in this thesis, covert channels refer to a means of communication between two parties that are not permitted to communicate. Contrary to covert channels in the information theory, they do not embed information into an already-existent channel. They are covert in the sense that legitimate activities in computer systems can generate them without being aware of doing it. In other words, the covert channels due to software activities are totally separate and individual channels are such that they do not exploit any synchronous channel that already exists.

The severity of a side/covert channel is often measured in terms of bit rates, i.e. how rapidly it can transmit information. Millen was the first to establish a connection between Shannon's information theory and information flow models in computer systems [9] to calculate the capacity of such a covert channel. Considering instructions as the inputs of covert/-

side channels, the proposed method in [9] ignores the variability in execution time of instructions. The model also assumes a synchronous channel which is not a realistic assumption for side/covert channels. Moreover, transmission through covert/side channels is often corrupted by insertion and erroneous transfer of bits due to their unintentional nature.

In the conventional communication literature, there is a large number of papers discussing bounds on the capacity of channels corrupted with synchronization errors [10]-[14]. Also, the work in [15, 16] discusses bounds on the capacity of channels corrupted with synchronization and substitution errors. The main drawbacks of these methods for side/covert channels are ignoring the signaling time deviations due to computer activities, insertions or deletions due to stalls, interrupts, etc., and not having a proper channel model that reflects the characteristics of side/covert channels.

These shortcomings demonstrate that new approaches are required to quantify the actual information leakages through side/covert channels. Having such a leakage quantification reveals the resiliency of a device against attacks based on these *unintended* channels. Therefore, the approaches must be easily applied by system designers to test whether their systems are resilient against side/covert channel attacks. Moreover, the new approaches must consider the worst-case scenarios and motivate minimizing leakages as much as possible to overcome any attacks in the near future. Therefore, these approaches have to provide a detailed analyses of covert/side channels, proper modeling of these channels to quantify worst-case scenarios, a simulation tool to analyze emanated EM signals in design-state and a design assessment methodology that guides systems designers for more secure devices.

1.2 Capacity of the EM Covert/Side-Channel Created by the Execution of Instructions in a Processor

Covert/Side channels are *unintentional* channels that are byproducts of computer activities. These channels can be a source of security flaws because they can enable unauthorized transmission of *secret* information. Here, the question is how to measure severity of these channels. In that respect, Millen was the first to establish a connection between Shannon's information theory and information flow models in computer systems [9]. The covert channel is modeled as a synchronous channel and the channel capacity of the model is calculated by utilizing Shannon's capacity definition. In other words, the paper established a connection between the channel capacity and the severity of the channel. However, the synchronous channel assumption is not realistic for these channels. Although most communication systems are designed to avoid symbol loss and/or insertion with little or no overhead, the covert channel is not designed to transfer information at all and its transmission is often corrupted by insertion, deletion and erroneous transfer of bits.

In conventional communication literature, there are a large number of papers discussing bounds on the capacity of channels corrupted with synchronization errors. There are also some papers discussing the capacity bounds for channels that are corrupted with synchronization and substitution errors. Likewise, in some work, the capacity bounds when codewords have different lengths are derived. However, the main problems with side channels are non-existence of codeword definition and an accurate communication model between source and the attacker to quantify information leakage. Moreover, previous research does not provide the an-

swer to how much information is “transmitted” by execution of particular sequence of instructions that do not have equal timing and are transmitted through erroneous channel. In that respect, a measurement technique is devised to quantify the emanated power that is a result of execution of a pair of instructions. This chapter aims to calculate the information leakage by establishing a connection with Shannon’s capacity and properly modeling the side channel with realistic assumptions that preserve the characteristic of side channels. To achieve our goal, we first derive a mathematical relationship between electromagnetic side channel energy (ESE) of individual instructions and the measured pairwise side-channel signal power. Then, we model the side channel communications as a discrete memoryles channel where instructions are the codewords and transition probabilities are calculated based on ESE. The leakage limit is defined as the channel capacity of the proposed model where the codewords have variable length. Finally, we provide leakage capacities of different devices and demonstrate the severity of possible threats that these channels can cause. We need to note here that the instructions behave as the codewords of the side channel communication, and the goal is to investigate how much information these instructions can convey for the worst-case scenario. The increase in the leakage capacity means better ability to estimate the instruction sequences, hence, predicting the computer activities with higher accuracy.

1.3 Electromagnetic Side Channel Information Leakage Created by Execution of Series of Instructions in a Computer Processor

Side channels are a consequence of program execution in a computer processor. To estimate information leakage and its capacity limits, under-

standing the relationship between code execution and information leakage is a necessary step. The methodology given in the previous section is appropriate for low complex devices where the pipeline has a simpler structure. However, as the systems evolve, the pipeline gets more complex to prevent stalls due to branching operations, etc. One critical observation is that each stage of a pipeline emits EM signals that can carry some information about the executed instruction and we collect the mixture of signals that are emanated from all stages. Moreover, the programs are written to serve a purpose, therefore, the execution order of instructions shows dependency to previously executed instructions. To consider the dependency among instructions due to program functionality and pipeline effect, this chapter proposes a Markov Source Model to relate program execution to electromagnetic side channel emanations, and estimates side channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor. Since the emanated signals are a result of executed instructions, the sources of the model are considered to be instructions. The signal emitted when these instructions are executed is considered as the channel input of the model. Since obtaining the emanated signal of a single instruction is extremely difficult especially in complex devices, we derive a mathematical relationship between the emanated instruction signal power (ESP) and total emanated signal power while running a program. Then, we derive leakage capacity of electromagnetic (EM) side channels created by execution of series of instructions in a processor. We provide experimental results to demonstrate that leakages could be severe and that a dedicated attacker could obtain important information. Finally, we provide an assessment approach that can be utilized by system designers to make their devices more secure

against EM based side channel attacks.

1.4 Communication Model and Capacity Limits of Covert Channels Created by Software Activities

Digital and/or analog characteristics of electronic devices during executing programs can create a side channel. This channel can be exploited by a motivated attacker to extract sensitive information such as cryptographic keys. If an attacker modifies the software application to deliberately exfiltrate sensitive information through a side channel, this channel is called a *covert channel*. Although covert channels are not designed to search for *secret* information within a system, they can be used as a tool to transmit information to outside world, Here, information is provided by another mechanism that is not allowed to communicate with outside world. This work first demonstrates a covert channel based on EM signals that are generated by legitimate program activities. To show the severity of possible attacks with this covert channel, we model this channel to obtain the maximum leakage capacity. Because the covert channels are not designed to transmit information, they are exposed not only to the errors created by the transmission, but also by varying the execution time of computer activities, and/or by insertions from other activities such as interrupts, stalls, etc. Therefore, the model needs to include all the undesired characteristic of the covert channels that is not observed in conventional communication systems. Therefore, to consider insertions and variation in signaling time, we propose to model the covert channel as an insertion channel with random insertion and substitution due to the noise and jitter errors. Since the goal is to transmit information in terms of zeros and ones, we model the transmitted sequence as a pulse amplitude modulated signal with random

pulse positions. We also propose a receiver design that can correctly detect the computer-activity-created signals. Then, we derive capacity bounds of this covert channel by utilizing the proposed model. Finally, we perform experiments with high clock speed devices at some distance to illustrate the severity of leakages. We observe that the theoretical derivations and empirical results show good agreement.

1.5 Covert Channel Information Leakage Capacity: A Generalized Approach

Foreseeing severity of leakages through covert channels is a necessity for designers to minimize security flaws. These vulnerabilities occur because of software activities and are a result of digital and/or analog characteristics of computers. To make these designs more resilient to any covert channel attacks, a judicious approach has to be followed. Having a tool to unveil possible leakage sources of devices in *design-state* can circumvent these attacks because it provides an opportunity for designers to modify their systems to minimize leakages for worst-case scenarios.

In the previous section, a covert channel and its model are proposed to demonstrate how covert channels can cause irreparable damages. However, attackers can exploit every possible source to achieve their goals, hence, a generalized model that is not limited to EM signals, but comprising other covert channels with various sources, i.e., backscattering, cache, power, etc, is required. Having such a generalized model, a detailed analysis of systems can be performed to make them more resilient to covert channel attacks. In that respect, this work proposes a methodology to estimate the worst-case information leakage through various covert channels. The methodology can be adopted for both analog and digital covert

channels. To model the channels, we first define and derive effective channel noise which is shown to be a combination of additive channel noise and jitter noise. We also propose signal and receiver models to mimic the use-cases of covert channels. The model also contains deletions and insertions that are commonly observed for any covert channels. The model helps to utilize Shannon's capacity definition to calculate leakage capacity of these chanels. Based on the model and observations, an assessment technique which exploits the proposed model is presented to help designing more secure systems. Finally, the assumptions to model the covert channels are verified with experimental result.

1.6 A Microarchitecture-Level Modeling Electromagnetic Side-Channel Signals

The concern for information leakage through side channels raises as the chances of attackers to physically access computing systems increase. For Internet-of-Things (IoT) systems, the ability of attackers to be in close proximity becomes a threatening problem since these systems contain sensitive data, such as sensor data, login information for over-the-network management of the system, etc. Therefore, these systems are required to be designed carefully such that they are resilient to any side channel attacks. In other words, these systems have to be quantitatively assessed in the early design stage such that every weak aspect of the design is revealed. This early stage assessment methodology not only secures systems against side channels attacks, but also decreases the cost of production.

Existence of such an assessment methodology can change the perspectives of designers because they can also include information leakages through side channels among their design considerations. Although there

are some assessment methodologies to quantify EM side channel leakages, they are mainly focus on the emanated signal after the system is manufactured. Since the analysis is done after production of the system, previous techniques could not help decreasing the cost of production, and preventing information leakages, but behave as an indicator of the vulnerability level of the designs. In that respect, this work introduces a simulation tool for emanated EM signals on the instruction granularity level to unveil the vulnerable aspects of any given design. To achieve our goal, we first propose a model for the emanated signal when an instruction is executed in any stage of a processor pipeline. Since measuring devices capture a mixture of signals emanated from each pipeline stages, we propose a multiple-input-single-output (MISO) communication system to represent the collected signals. Finally, we present our experiments which demonstrate the accuracy of the proposed model. The experimental results and the theoretical signals show good agreement, hence, the proposed signal model can be used in the *design-stage* to analyze possible flaws in the system.

1.7 Research Contributions

The research contributions of the thesis are

- Definition of ESE (Electromagnetic Side-channel Energy) to calculate the relative signal power difference between two instructions [17],
- A discrete memoryless channel (DMC) model to quantify the information leakages for side channels [18],
- ESP (EM Signal Power), a definition and methodology to calculate the emanated EM signal power while an instruction is executed through

pipeline stages of a processor,

- A Markov source channel model to quantify side channel capacity which considers the dependency among instructions due to pipeline and program functionality [19],
- Definition and derivation of jitter error that is a result of variations in signaling time of transmitted bits in covert channels [20],
- An EM based covert channel and its model to demonstrate its severity [21],
- A generalized model for various covert channels to quantify their channel capacity [22],
- *EMSim*, a tool to simulate emanated EM signals to analyze systems for information leakage at design-stage [23].

1.8 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 provides information on covert/side channels, their use-cases, and previous attempts on qualifying information leakage, definitions that are heavily exploited in both conventional communication and information theory literature, Chapter 3 defines ESE and proposes a DMC channel model to quantify the worst-case information leakage that is available to an attacker. Chapter 4 defines ESP and models the side channel as a Markov Source Model to consider the instruction dependency that is a result of processor pipeline and program functionality. Chapter 5 introduces a covert channel based on EM side channels, defines jitter error, models the communication between the *source* and *sink* as an insertion channel corrupted with noise,

and provides capacity bounds for such channels. Chapter 6 proposes a communication channel model that can be generalized to various covert channels which employ *on-off-keying* modulation scheme. Chapter 7 introduces *EMSim* which is a tool to simulate emanated EM signals while executing a program. Finally, Chapter 8 summarizes the contributions of the thesis and provides possible future directions.

CHAPTER 2

BACKGROUND

Understanding side/covert channels to model, analyze, and estimate leakage capacity requires an interdisciplinary research to accurately expose the severity of these unintentional channels. In this chapter, we provide some important concepts from information and communication theory literature as well as EM based side/covert channel signals that are heavily exploited throughout the thesis.

2.1 Covert/Side Channels

Communication channels are designed carefully to interchange information, resources, etc., between parties. Each party participating in this process can control and limit the extent of shared data to protect company secrets, plans etc. Also, both end-users are required to agree on some standards and/or principles for legitimate transfer of information.

However, in 1967, Dr. Willis H. Wore demonstrated a computer network vulnerability caused by radiations from processors, communication lines, and output devices [24]. Existence of these undesired signals raised question about the possibility of "side channels", which are unintentionally created as a consequence of hardware/software activities in a computer system [25]. Detecting and processing these emanated signals can lead to serious information leakages which could be a significant threat against one of the basic human rights: privacy.

The threat became real when Lampson in [1] exploited a side channel to send some information of customers to the owner of a program even

though this transmission process is not permitted. He called these channels "covert" for the first time, defined them as not intended channels for information transfer, and proposed countermeasures to prevent such attacks. Later, Eck in [26] demonstrated the possibility of eavesdropping on video display units to reconstruct the video content, and claimed the possibility of picking up signals at a distance over 1 kilometer.

Another type of side channel attacks was introduced when Kocher in [27] demonstrated timing attacks to break some cryptosystems, i.e. RSA, DSS. His idea was to exploit slight differences on the amount of time to process different inputs due to branching, conditional statements, etc. Extension of this work was proposed by Schindler in [28] to compromise secret keys of RSA with Chinese Remainder Theorem assuming that the attacker can access to hardware device. The main drawbacks of these timing attacks were that the attacks were performed on simple computing devices, and required a direct access. However, the severity of these attacks gained more attention when practicality of remote timing attacks was illustrated by Bonech in [29]. Bonech et al. broke secret keys of OPENSSL, a cryptosystem commonly used in web servers and SSL applications, remotely.

Attacks based on power analysis made side channel attacks even more menacing when Kocher in [30] demonstrated differential power analysis (DPA) to break various cryptosystems' secret keys on microcontrollers. He collected thousands of power traces to build a high order statistical model to estimate the intermediate secret bits of cryptosystems. He claimed that unanticipated security faults can occur if hardware designers ignore and/or underestimate the attacks based on "side effects" of computing devices. Goubin et al. [31] compared DPA with simple power analysis (SPA),

which aims to deduce parts of secret keys from power consumption values, and revealed that DPA can circumvent the countermeasures against SPA attacks. Messerges et al. extended this work by extracting secret exponent bits from tamper resistant hardware [32]. Comprehending the severity of attacks based on power analysis, some methods have been proposed as countermeasures for these attacks. Chari et al. added randomness while computing the key exponent to prevent the statistical attacks (DPA) [33]. In [34], an application, which first identifies sensitive instructions and then transforms the code to make it more robust to attacks, was proposed to automatically prevent information leakage. Motivated by these approaches, many covert/side channels are investigated and named based on the sources that generate such channels. These channels can be classified into two broad categories of *digital/micro-architectural* and *analog/physical* covert/side channels. These channels can be explained in detail as follows:

i) *Digital/Micro-architectural Covert/Side Channels:* They typically rely on shared hardware resources in the computer between a *source* and a *sink* application. The source application (also called Trojan), has access to sensitive information, however, its communication to outside world (e.g. network or I/O ports) is well-protected by the operating system (OS) and/or other security monitoring modules. Due to these limitations, the source can not "extract" the sensitive information by itself through conventional covert channels without being scrutinized, and hence detected by the existing *protection* mechanisms in the system. Alternatively, to successfully extract the data, the source application can *secretly* communicate to another malicious application

(called sink or spy), via a covert channel. The sink application is a seemingly benign application that has access to the outside world (e.g. through network, I/O ports, etc.) but is not likely to be scrutinized by the OS or other security enforcer modules since it does not have access to any sensitive data in the system. Hence, by receiving the data from the source, the sink can successfully "extract" the data and send it to the outside world. Examples of digital/micro-architectural covert/side channels include caches [35, 36, 37, 38, 39, 40, 41], branch predictors [42, 43], micro-architectural units [44, 45], DRAM and memory bus [46, 47], GPUs [48], etc. In all these cases, the information can be transmitted by modulating certain (micro-architectural) events (e.g. accessing a pre-defined line in cache).

ii) *Analog/Physical Covert/Side Channels:* These are the channels that rely on physical characteristics of a system such as unintentional electromagnetic emanations from different electronic circuits in a computer [3, 49, 50, 51], variation in power consumption [27, 30, 29, 34, 33, 52, 31, 32, 53], sound/acoustic [53, 50], temperature [54, 55], etc. Unlike digital covert/side channels which create a digital link between two applications *within* the processor, physical covert/side channels create an analog *external* link which acts as a wireless communication link that can transmit sensitive data from the computer, through the air or via a pin/cable, to *nearby* systems. For example, it is shown that exfiltrating data from an air-gapped (physically and logically separated from public networks) computer through covert channels is possible [4, 56].

2.1.1 EM Covert/Side Channels

Demonstrating the video content reconstruction from emanated signals of a video display unit on BBC in 1985, Eck [26] opened a new era for side channel attacks. The main result of his work was that unintentional EM signals, emitted while a legitimate activity is running on a device, can cause information leakage if a sophisticated attacker detects and processes these signals. Khun [49] extended the work by reconstructing confidential information displayed on a cathode-ray-tube via an unauthorized access exploiting EM side channel. Moreover, [57] demonstrated attacks based on EM analysis on CMOS chips and smart cards to reveal secret bits of RSA and DES cryptosystems. They exploited the correlation between secret data and corresponding variations in EM field. The main conclusion of their experiments was that EM measurements are as powerful as power analysis. However, Agrawal et al. [25] first described EM emanations as a consequence of current flows due to hardware/software activities, then showed that existence of correlation between the processed data and current flow causes information leakage, and finally made an even stronger argument that EM side channel leakage is even more powerful than the other side channels. Inspired by the information leakage through EM emanations, Zajic et al. [3] generated a covert channel which transmits the morse code of "All data is belong to us." They exploited program activity signals which are modulated by the clock frequency of a device. The achievement with this experiment is that even under close scrutiny, emanated EM signals can be achieved reliably at a distance over 2 meters without raising any suspicion. To measure the signal available to an attacker for instruction-level events, Callan et al. [51] devised a methodology called SAVAT. The method relies on the instruction-level dif-

ferences while running the same code except one instruction. They first tested the methodology in a device with weaker computing ability (FPGA board), then expanded to more complex computer systems. The similar results were obtained across various devices [58]. Later, Monjur et al. [59] broke secret keys of OpenSSL's blinded RSA without relying on the cache organization or timing variations. They first identified the vulnerable part of the program, collected hundreds of training EM traces during modular exponentiation calculations, and then exfiltrated the secret key bits by only measuring one trace.

2.1.2 Applications of Covert/Side Channels

Although covert/side channels can provide an unauthorized access to confidential information through legitimate activities, it is also possible to utilize them to secure computer systems. An example for the use-case of covert side channels is program profiling. Program profiling is an essential tool for monitoring, compiler optimization, performance analysis, user profiling, debugging, software maintenance and paralellization [60]. Although there are many profiling methodologies, the most common approach is to add instrumentation, i.e. logging considered events. These events are generally the paths that are heavily executed and called *hot paths* of the program [61]. Identifying these paths can be very advantageous for efficient code optimization and monitoring. However, instrumentation can cause overhead and change the dynamic behavior of the profiled systems. In that respect, covert/side-channels are utilized to handle the overhead caused by instrumentation and protect the dynamic behavior of the profiled systems. An instruction-level side-channel profiling is proposed in [62], which exploits power side-channels on embedded devices. Similarly,

a classification of hardware-based monitoring techniques and a power-channel based disassembler are provided in [63]. In [64], a method is proposed to detect and monitor a system by processing the power consumption. A zero-overhead profiling method exploiting EM side-channels is proposed in [65]. Although the methodology does not require any modifications to the software/hardware of the profiled device, it fails reporting anomalies with zero false negative rate (it reports benign instead of malware for some cases). By improving these shortcomings, a new input generation technique, called progressive symbolic execution, and an advanced execution-trace-prediction method are proposed in [66]. Likewise, an EM signal model and simulator are proposed in [67, 23], which can be helpful for obtaining signatures of a program even in the design-stage. In [68], a profiling scheme that leverages EM side-channels is introduced. The paper exploits the idea that whenever a loop is executed in a program, an RF signal is generated with a so-called alternation frequency. This frequency corresponds to reciprocal of the average time required to run an iteration of a loop [69]. Since loops are an example of *hot paths* and the program runtime is dominated by the loops, the scheme profiles the program based on these emitted RF signals. Later, this profiling scheme is modified to detect deviations in program execution without any characterization of malware [70]. After creating a statistical model for the deviation of a benign program in terms of loop-timing, a test signal is compared against the model to determine whether the deviations belong to the same distribution.

2.2 Measuring Pairwise Side Channel Signal Power from Processor Instructions

In [51], it was assumed that an attacker has access to a program's source or executable code, and can observe EM emanations from the victim's system while this program is running. The attacker attempts to extract sensitive information by recording EM emanations, using them to infer which instructions are executed, and then infers sensitive data from knowledge of the executed instructions.

The most direct approach to quantifying the EM emanations from side-channel signal created by executing instruction X_1 vs. executing instruction X_2 is to measure the EM emanations while instruction X_1 is active, measure the EM emanations while instruction X_2 is active, and then take the difference between these two signals. This approach is impractical in high performance systems for several reasons. First, equipment capable of recording the $x_1(t)$ and $x_2(t)$ signals at greater than 10 Gsamples/sec (as required to test a processor using a GHz clock) and with thermal, quantization, etc. noise that is low enough to not obscure the (extremely small) difference between the two signals, is prohibitively expensive or non-existent. Second, complex processors heavily optimize the scheduling and execution of instructions, so determining the times where the test instructions X_1 or X_2 are actually active would be problematic. Third, some other instructions must be present around X_1 and X_2 to make the measurement practical (to trigger the measurement, setup the registers and memory used by instructions X_1 and X_2, etc.), and these unrelated components of the received signal can easily obfuscate the signal components created by the X_1 and X_2 instructions themselves.

```
1    for(j=0;j< n_out; i++){
2      // Do some instances of the X1 instruction
3      for(i=0;i< n_inst; i++){
4        ptr1=(ptr1&~mask1)|((ptr1+offset)&mask1);
5        // The X1-instruction, e.g. a load
6        value=*ptr1;
7      }
8      // Do some instances of the X2 instruction
9      for(i=0;i< n_inst2; i++){
10       ptr2=(ptr2&~mask2)|((ptr2+offset)&mask2);
11       // The X2-instruction, e.g. a store
12       *ptr2=value;
13     }
14   }
15
```

Figure 2.1: The X_1/X_2 alternation pseudo-code.

To overcome these problems, a program has been designed in [51] that causes the system to execute X_1 and X_2 instructions in a controlled way that minimizes the effect of all other unrelated system activities, while concentrating the side channel energy into a narrow spectral band to minimize the impact of noise on the measurement. We produced these controlled emanations by choosing a repetition period T_{alt} and then create a small test program (microbenchmark) containing a `for` loop such that the first half of the loop does many repetitions of activity X_1 and the second half does many repetitions of activity X_2. The microbenchmark in Figure 2.1 implements this idea by executing X_1 and X_2 instructions n_{inst} times (denoted as n_{inst} in Figure 2.1) in each iteration of the outer loop. Lines 2 through 7 execute n_{inst} instances of the X_1 instruction, and then lines 8 through 13 execute the same number of instances of the X_2 instruction. Thus lines 2 through 13 represent one X_1/X_2 alternation, and this alterna-

tion is repeated (line 1) until the measurement of the side-channel signal is complete. It is critical to note that the value of T_{alt} is controlled directly by varying n_{inst}. For example, increasing n_{inst} increases the time required to execute one iteration of the outer loop (T_{alt}). The value of T_{alt} can be directly measured using counters available through processor instructions (e.g. the x86 `rdtsc` instruction) or the operating system (e.g. the Windows API $\text{QueryPerformanceCounter}()$ function). We can then select the n_{inst} value that produces the desired alternation frequency ($f_{\text{alt}} = 1/T_{\text{alt}}$).

	Instruction	**Description**
LDM	mov eax,[esi]	Load from main memory
STM	mov [esi],0xFFFFFFFF	Store to main memory
LDL2	mov eax,[esi]	Load from L2 cache
STL2	mov [esi],0xFFFFFFFF	Store to L2 cache
LDL1	mov eax,[esi]	Load from L1 cache
STL1	mov [esi],0xFFFFFFFF	Store to L1 cache
ADD	add eax,173	Add imm to reg
SUB	sub eax,173	Sub imm from reg
MUL	imul eax,173	Integer multiplication
DIV	idiv eax	Integer division
NOP	nop	No operation
NOI		No instruction

Table 2.1: x86 instructions for our X_1/X_2 ESE measurements.

Instructions refers to basic commands that a processor can execute. Some examples of common x86 instructions are listed in Table 2.1, including loads and stores that go to different levels of the cache hierarchy, simple (ADD and SUB) and more complex (MUL and DIV) integer arithmetic, and the "No Instruction" case where a specific line is simply left empty.

Therefore, EM side channel energy available to the attacker is defined as the total power around the alternation frequency which is equivalent to following equation in time domain:

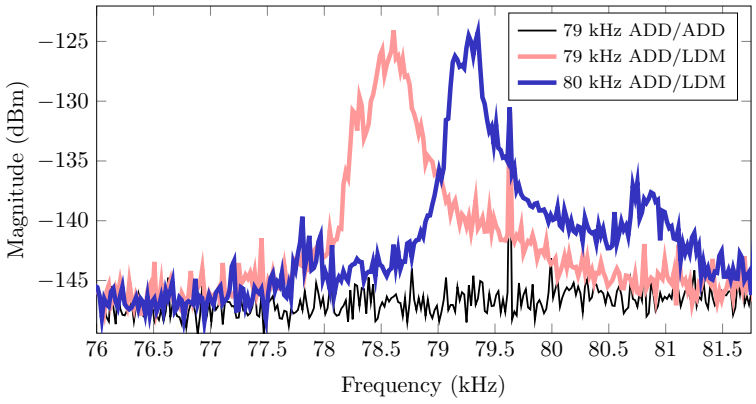


Figure 2.2: Power spectrum of ADD/LDM instruction pair at 79 kHz and 80 kHz.

$$\mathcal{P}_{\mathrm{A}}(s_1(t), s_2(t)) \equiv \frac{1}{R}\int_0^{T_o}(s_1(t) - s_2(t))^2 dt \tag{2.1}$$

where $s_1(t)$ and $s_2(t)$ are voltages measured across a resistance R (typically instruments have $R = 50\Omega$), $t = (0, T_o)$ is the time interval while program execution occurs and can be written as $T_o = T_{\mathrm{alt}} \times n_{\mathrm{out}}$, and n_{out} is the number of repetition of the outer `for` loop. This difference can be an if-then-else statement that causes one path or the other to be executed, a memory access that either does or does not suffer a cache miss, etc.

2.3 Amplitude Modulated Signal Generation by Executing Programs

In this section, we first describe how carrier signals can be created by software activities and then, describe a method to generate modulated signals. To create a carrier, we use repetitive variations in a software activity as described in [3], [17]. We choose T, the period (duration) of each repetition, two types of activities (A and B), and write a small software code (i.e., a microbenchmark) shown in Fig. 2.1 that in each period does activity A in the first half and B in the second half. Please note that the intuition be-

hind this is that if activity A and activity B result in non-identical EM fields around the processor (or the system), repetition of this A-then-B pattern will create oscillations (with period T) in this EM field, i.e., it will result in a "carrier" RF signal at frequency $1/T$. The period T will be selected to correspond to a specific frequency, e.g., to produce a radio signal at 1 MHz, we should set $T = 1\mu s$. This carrier-generation approach is illustrated in Fig. 2.3.

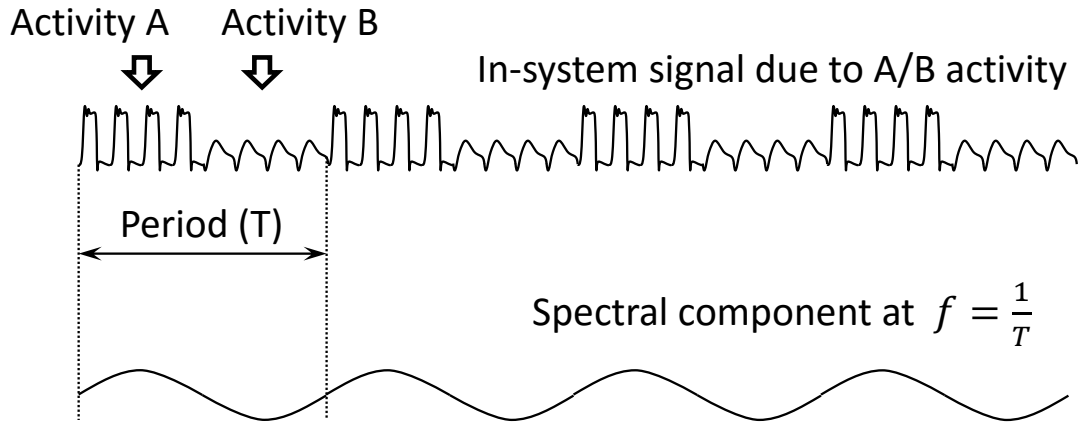


Figure 2.3: Illustration of how microbenchmark induces emanations at a specific radio frequency by alternating half-periods of A and B activity.

Next, the symbols are *amplitude modulated* by inserting intervals during which only activity B is performed in both half-periods which means any carrier signal produced by the differences between A and B should be absent when only B is used, resulting in the simplest form of AM modulation (on-off keying). This approach is illustrated in Fig. 2.4. Note that other modulations (e.g., frequency modulation or even some non-standard modulation) can just as easily be used to create a truly covert transmission. Also note that the assumption here is that the code for generating this software modulation and creating a covert channel is already injected to the system through advanced-persistent threat scenarios [71], or manually entered/created on the target system by a trusted insider (e.g., a

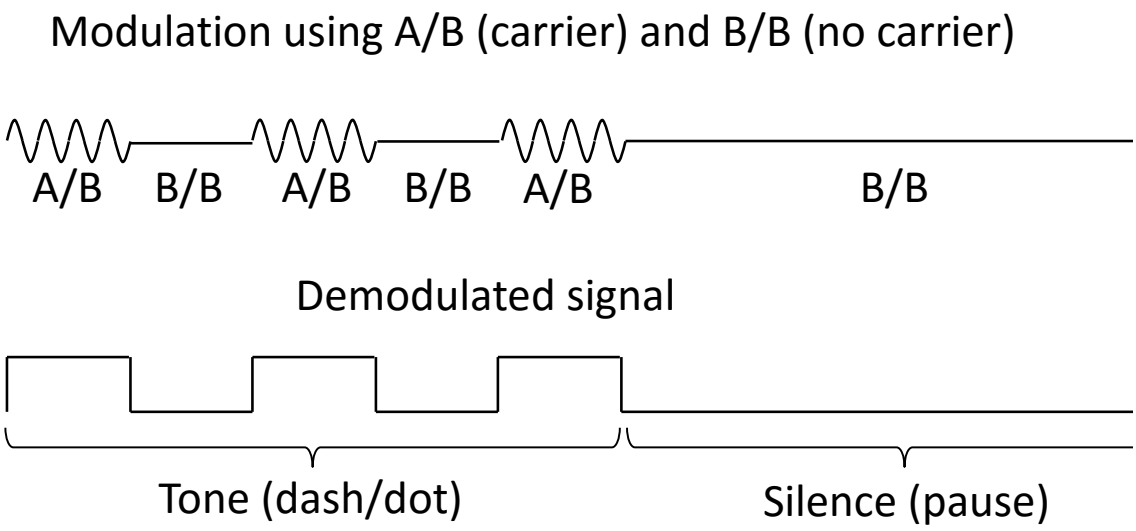


Figure 2.4: Illustration of how microbenchmark modulates the signal into the carrier using on-off keying (bottom).

rogue employee). Moreover, the transmission code itself is not responsible to find the sensitive data, but it is only *a mean of communication for sending the sensitive data from trusted inside to the outside world.* The sensitive information (to this transmitter code) is supplied by an injected/created malware in the system (e.g., a worm that infiltrated to the system and found some sensitive documents in the system). Note that both of these assumptions are realistic and commonly used in the existing literature.

2.4 Channel Capacity

Defining leakage capacity requires deep understanding of channel capacity in conventional communication systems and information theory literature. In this section, we provide essential topics that will be extensively exploited.

2.4.1 Markov Model Capacity over Noisy Channels

Channel capacity provides the limit for a reliable information transmission in a communication system. Assuming Y_1^n and S_1^n represent the channel output and state sequences between $t = 1$ to $t = n$, the capacity of the Markov sources over noisy channels is defined as [72]

$$C = \max_{\substack{P_{ij} \\ (i,j)\in\mathcal{T}}} \lim_{n\to\infty} \frac{1}{n} I\left(S_1^n; Y_1^n | S_0\right) \tag{2.2}$$

where $I(\bullet)$ is the mutual information, P_{ij} is the transition probability from state i to j, and $\mathcal{T}$ is a set of valid state transitions. To maximize the overall mutual information between input and output sequences, we need to find the probability distribution of state transitions under the constraint that state transitions are only possible if $\mathcal{T}$ contains these paths. The equation given in (2.2) can be simplified further by using chain rule, Markov, and stationary properties of the model. In [72], it is shown that the capacity can be simplified as

$$C = \max_{P_{ij}} \sum_{i,j:(i,j)\in\mathcal{T}} \mu_i P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right]. \tag{2.3}$$

where

$$T_{ij} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \left[\log \frac{P_t(i,j|Y_1^n)^{\frac{P_t(i,j|Y_1^n)}{\mu_i P_{ij}}}}{P_t(i|Y_1^n)^{\frac{P_t(i|Y_1^n)}{\mu_i}}}\right], \tag{2.4}$$

and where μ_i is the stationary probability of state i, which satisfies $\mu_i = \sum_{k\in\mathcal{S}} \mu_k P_{ki}, \forall i \in \mathcal{S}$, and $\mathcal{S}$ is the set of states. In this equation, $P_t(i|Y_1^n)$ is the probability that the state at time $t-1$ is i, and $P_t(i,j|Y_1^n)$ is the probability that the states at times $t-1$ and t are i and j respectively, given the received sequence, Y_1^n.

There is no closed form solution to the optimization problem given in (2.3) because the calculation of T_{ij} is still an open problem. However, in [72], a greedy algorithm to calculate C is introduced. The experimental findings show that the performance gap between the actual results and the algorithm's results is small.

2.4.2 Channel Capacity for Insertion & Substitution Channels

One of the main challenges that side/covert channels encounter is insertions and inaccurate acquisition of transmitted information signal. Finding the capacity of channels that suffer from both insertions and substitutions is even challenging for conventional communication systems. To simplify the capacity definitions for such channels, [16], [15] show that a memoryless discrete channel with insertions and substitutions can be decomposed into a cascade of two channels, channel with insertions and channel with substitutions as shown in Figure 2.5.

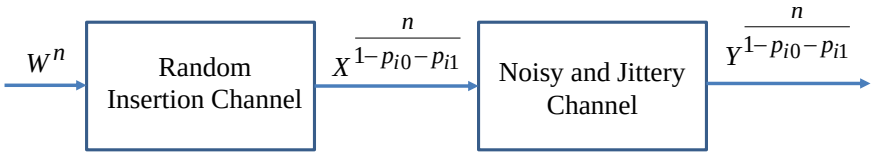


Figure 2.5: Cascaded channels equivalent to the binary discrete memoryless noisy, jittery, synchronization channel with n input symbols.

In [73], it is shown that the capacity of a discrete memoryless synchronization channel exists and is given by

$$C = \sup_{\Xi} \lim_{n\to\infty} \frac{1}{n} \cdot I(W^n; Y^{\bar{N}n}), \tag{2.5}$$

where the supremum is taken over all stationary Markov chains Ξ modeling the input source, n is the number of input bits, W^n and $Y^{\bar{N}n}$ represent the input and observed sequence respectively. Here, $\bar{N}$ is the average num-

ber of received symbols per transmitted symbol.

CHAPTER 3

CAPACITY OF THE EM COVERT/SIDE-CHANNEL CREATED BY THE EXECUTION OF INSTRUCTIONS IN A PROCESSOR

3.1 Overview

The goal of this chapter is to answer how much information is "transmitted" by execution of particular sequence of instructions in a processor [18]. Introducing such a measure would provide quantitative guidance for designing programs and computer hardware that minimizes inadvertent (side channel) information leakage, and would also help detect parts of a program or hardware design that have unusually high leakage (i.e. were designed to function as covert channel "transmitters").

To address this problem, we need to establish relationship between software activity, observed emanations, and side-channel capacity. The first attempt to quantify which instructions have the greatest potential to create side-channel vulnerabilities was reported in [51], where a measurement technique is devised to quantify the emanated power that can be attributed to the difference in execution between a pair of individual instructions.

In this chapter, we derive a mathematical relationship between electromagnetic side-channel energy (ESE) of individual instructions and the measured pairwise side-channel signal power and use this measure to calculate the transition probabilities needed for estimating capacity. Then, we propose a new method to estimate side/covert channel capacity created by the execution of instructions in a processor. Finally, we illustrate how the

proposed method works through evaluation of capacity in several practical systems.

The organization of the section is as follows: Section 3.2 analytically quantifies the difference in energy available to an attacker between two instructions, Section 3.3 describes a new method to quantify covert/side-channel capacity, Section 3.4 analyzes capacity of several systems, Section 3.5 proposes some possible defensive methods for EM based attacks, and Section 3.6 provides the summary.

3.2 Mathematical Relationship Between ESE of Individual Instructions and The Measured Pairwise Side-channel Signal Power

Establishing mathematical relationship between ESE of individual instructions and the measured pairwise side-channel signal power available to an attacker [51] can not only help programmers and computer hardware designers to anticipate the vulnerability of the system, but can also help us relate codeword probabilities with energy levels of particular codewords, i.e., instructions.

Total EM side-channel energy available to the attacker is defined in (2.1). Note that $\mathcal{P}_{\mathrm{A}}$ represents the overall emitted power during the execution of the microbenchmark including many repetition of instructions X_1 and X_2. However, ESE is defined as the available energy for an attacker when two instructions are executed only one time by filtering any other EM signals and noise present in measurements.

Assume $x_1(t)$ and $x_2(t)$ is the characteristic signals belong to execution of instructions X_1 and X_2. Then, ESE is defined as

$$\mathrm{ESE}(X_1, X_2) \equiv \frac{1}{R} \int_0^{T_x} \left(\mathbb{E}[x_1(t) - x_2(t)]\right)^2 dt \tag{3.1}$$

where T_x is the maximum of the execution times of the instructions and $\mathbb{E}$ is the expectation operation.

To make our derivations mathematically traceable, we introduce following notations and assumptions.

1. The for-loops of the microbenchmark given in Fig. 2.1 generate $s_1(t)$ and $s_2(t)$ respectively. Because the emitted signal from each loop depends on the type of instruction used in the loop, to discard the ambiguities regarding the inserted instruction and the taken branch, we denote produced signals as $s_i^{(X_i)}(t)$ which is generated by the execution of the i^{th} inner-for-loop and X_i represents any instruction inserted into that loop.

2. $s_1^{(X_1)}(t)$ and $s_2^{(X_2)}(t)$ are voltages sampled at frequency $1/T_\mathrm{I}$ to create the sequences $s_1^{(X_1)}[n]$ and $s_2^{(X_2)}[n]$ of length $N_s = T_\mathrm{s}/T_\mathrm{I}$.

3. The frequency content of $s_1^{(X_1)}(t)$ and $s_2^{(X_2)}(t)$ above $1/(2T_\mathrm{I})$ is negligible (i.e. $s_1^{(X_1)}(t)$ and $s_2^{(X_2)}(t)$ have bandwidth $1/(2T_\mathrm{I})$).

4. $s_1^{(X_1)}(t)$ and $s_2^{(X_2)}(t)$ are voltages measured across a resistance R.

5. The discrete energy available to the attacker $\mathcal{P}_\mathrm{A}\left[\mathrm{s}_1^{(\mathrm{X}_1)}, \mathrm{s}_2^{(\mathrm{X}_2)}\right]$ is then defined as

$$\mathcal{P}_\mathrm{A}\left[\mathrm{s}_1^{(\mathrm{X}_1)}, \mathrm{s}_2^{(\mathrm{X}_2)}\right] \equiv T_\mathrm{I} \sum_{l=0}^{N_s-1} \frac{\left(s_1^{(X_1)}[l] - s_2^{(X_2)}[l]\right)^2}{R}. \tag{3.2}$$

6. Sampled voltages can be represented as $s_1^{(X_1)}[l] = \mu_1[l] + w_1[l]$ and $s_2^{(X_2)}[l] = \mu_2[l] + w_2[l]$ where $\mu_1[l]$ and $\mu_2[l]$ are the mean voltage values, $w_1[l]$ and $w_2[l]$ are the additive noises which are i.i.d. $\mathcal{N}(0, \sigma_l^2)$. Here, σ_l^2, $\mu_1[l]$, and $\mu_2[l]$ are depended on the instruction. For example, if the

sample is taken during the execution of the first-inner-loop and the embedded instruction is X_1, the sample can be written as

$$s_1[l] = \mu_1[l] + w_1[l] = X_1^v + w_1[l] \tag{3.3}$$

where $w_1[l] \sim \mathcal{N}(0, \sigma_{X_1}^2)$ and X_1^v is the average power instruction X_1 emits. We assume that additive noise describes all the variation in the signal and that noise power is dependent on the executed instruction since the electromagnetic emissions can vary according to the execution location.

7. Finally, the discrete ESE is defined as

$$\mathrm{ESE}\left[\mathrm{X}_1, \mathrm{X}_2\right] \equiv \frac{T_\mathrm{I}}{R} n_X \left(X_1^v - X_2^v\right)^2 . \tag{3.4}$$

where $n_X = T_x/T_\mathrm{I}$ is the number of samples taken during only the execution of the instructions.

The micro-benchmark described in Section 2.2 creates an alternation signal at frequency f_{alt} by repeatedly executing instruction X_1 n_{inst} times, followed by n_{inst} executions of instruction X_2. We then measure $P(f_{\mathrm{alt}})$, the spectral power at frequency f_{alt}. Finally, $\mathrm{ESE}[\mathrm{X}_1, \mathrm{X}_2]$ can be calculated from the spectral power $P(f_{\mathrm{alt}})$ observed at f_{alt} (while running the X_1/X_2 alternation microbenchmark) as follows:

$$\mathrm{ESE}[\mathrm{X}_1, \mathrm{X}_2] \approx \left(\frac{\pi}{2}\right)^2 \frac{P(f_{\mathrm{alt}}) \cdot N}{n_X \cdot n_{\mathrm{inst}} \cdot f_{\mathrm{alt}}} + C(X_1, X_2), \tag{3.5}$$

where N is the number of samples taken during only one inner for-loop, n_X is the maximum of the number of samples taken during the execution of instructions X_1 and X_2 and $C(X_1, X_2)$ is a constant term that depends

on the instruction pair, i.e.,

$$C(X_1, X_2) = -\frac{\mathcal{P}_{\mathrm{A}}\left(\mathrm{s}_1^{(\mathrm{X}_1)}, \mathrm{s}_2^{(\mathrm{X}_1)}\right) + \mathcal{P}_{\mathrm{A}}\left(\mathrm{s}_1^{(\mathrm{X}_2)}, \mathrm{s}_2^{(\mathrm{X}_2)}\right)}{2n_X n_{\mathrm{inst}}}. \tag{3.6}$$

The proof of the relationship in (3.5) is shown in Appendix A. Note that although expectation for $\mathcal{P}_{\mathrm{A}}$ value to be same when the same instruction is inserted into the for-loops given in Fig. 3.7, because of the additive noise given in (3.3), the observed ESE through the execution of the code results in a non-zero value.

3.3 A New Method for Evaluation of EM Side/Covert Channel Capacity Created by the Execution of Instructions in a Processor

In a covert/side channel, identifying the channel capacity is of some interest to attackers who desire to maximize the rate at which information is obtained, but it is of paramount interest to defensive actors (programmers and hardware designers) to gain an insight about how vulnerable a program or a computer system is to an attack and quantify how a potential design change would affect this vulnerability. Thus, in this section, we propose a new method to quantify the amount of information that can be obtained through unintended EM emanations of instructions in a computer system.

3.3.1 Quantifying the Side Channel Leakage

To quantify the leakage capacity through the execution of instruction sequences, we consider each instruction as an independent codeword, and the covert/side channel "transmission" consists of a sequence of codewords that corresponds to the sequence of instructions actually executed

by the program. Note that some hardware events can significantly affect the duration and the emanated signal of specific instructions. We account for this by treating these as distinct instructions. For example, a load (read memory) instruction behaves very differently when it finds the desired value in the first-level cache, in the second level cache, or in main memory, which we view as three separate instructions: LDL1, LDL2, and LDM, respectively.

Fig. 3.1 illustrates the noisy channel model for a covert/side channel where the input codewords are instructions, P_i represents the probability of the i^{th} instruction occurrence and p_{ij} is the transition probability that given i^{th} instruction is used in the code but detected as the j^{th} instruction based on EM emanation observations.

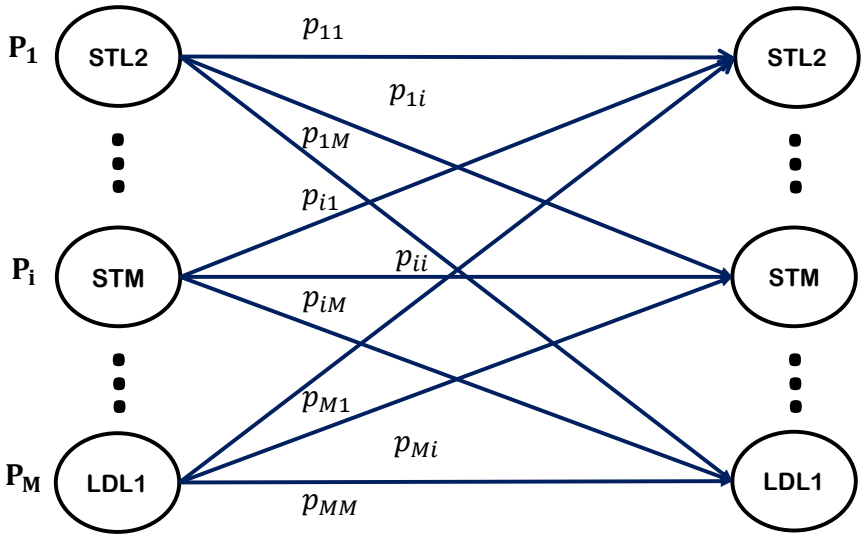


Figure 3.1: Noisy channel model for covert/side channel.

There are several unique properties of this EM side/covert channel that require new methods for evaluating side/covert channel capacity.

First, all codewords are typically equally probable in traditional communication systems, but this is not true for the channel analyzed in this section. Since our codewords are processor instructions, we need to note

that different instructions have different probability of occurrence in a program. For example, a typical program executes on-chip instruction (e.g. arithmetic operations) more often than memory accesses. Furthermore, processor designers put extra-efforts to ensure that among memory accesses the number of LDL1 executions is larger than the number of LDL2 occurrences and much larger than the number of LDM occurrences. Hence, it is realistic to expect that our codewords (i.e. instructions) do not have equal probability of occurrence.

Second, codewords in this channel do not have equal length. For example, it takes more energy and time to execute LDM in comparison to the time to execute ADD instruction.

Finally, this channel is inherently noisy due to variations in activity in the processor (i.e. different data values for a given instruction) and due to noise created by other electronic components near the processor.

All this implies that the Shannon-based channel capacity [16, 74, 75] overestimates the available capacity in the side/covert channel that transmits instructions as codewords. Therefore, we need to compute channel capacity in a way that considers the distinct instruction lengths, variable instruction probabilities, and maximum mutual information at the same time.

If we assume that we have a finite-length sequence where codewords can have different lengths, the goal is to invoke codewords into the sequence such that total information gathered from the sequence is maximized. Therefore, the length of the input and the entropy of this input are the issues critical to consider. Hence, we need to find a distribution for the input set such that total information achievable from the sequence is maximized, i.e., the information per sample is optimized. To achieve that,

we propose the following optimization problem:

Setting I :

$$\text{maximize} \quad \frac{\sum_{i,j} P_i p_{ij} \log\left(\frac{p_{ij}}{\sum_k P_k p_{kj}}\right)}{\sum_i P_i L_i}$$

$$\text{subject to}$$

$$\sum_i P_i = 1$$

$$P_i \geq 0 \text{ for } \forall i, j \in \{1, \cdots, K\}\,.$$

where K is the number of instructions in the input set. Through the optimization process, the variables that maximize the problem are

$$\{P_i \mid i \in \{1, \cdots, K\}\}\,.$$

These variables are the probabilities or frequencies of each instruction such that maximum information leakage occurs and we assume that occurrences of instructions are independent of each other. Transition probabilities p_{ij} are calculated based on ESE measurements as described in Section 3.3.2. The optimization problem is solved using gradient descent approach detailed in Appendix D.

Note that the optimization problem given above simplifies to Shannon's channel capacity if the length of each instruction is the same and all instructions are equally probable.

3.3.2 A Practical Calculation of Transition Probabilities in EM Side/Covert Channel

In this section, we propose a method for finding transition probabilities defined in (3.7) using the measured pairwise side-channel signal power.

We start the process by obtaining ESE values as illustrated in Table 3.1. Each entry in this table is the ESE between the X_1 instruction (row) and X_2 instruction (column) computed using (3.5). The pairwise side-channel signal power needed to compute ESE is measured using method described in Section 2.2. We measure a signal power of each instruction pair in the table 10 times over multiple days and take the mean to minimize the impact of changes in radio signal interference, room temperature, and slight differences in antenna position. For each pair of instructions, X_1 and X_2, we run the X_1/X_2 micro-benchmark and measure the power spectral density from 2.5 kHz above to 2.5 kHz below the alternation frequency. Then we integrate over this band to get the total power $P(f_{\text{alt}})$ generated by the difference between X_1 and X_2. Finally the $\text{ESE}[\text{X}_1, \text{X}_2]$ is calculated using (3.5).

The benchmarks were run as single-threaded Windows 7 32-bit user mode console applications on the Intel Core i7 (L1 Data Cache: 64 KB, 2 way L2 Cache: 1024 KB, 16 way). No other user-mode applications were active and wireless devices were disabled to minimize interference with the intentionally generated signals. Aside from this, the system was operating normally.

As an example, the ESE values for an Intel Core i7 laptop are given in Table 3.1.

Estimating the Voltage of a Specific Instruction

In this part, we provide the steps to estimate the voltage of an instruction. Please note that ESE can be viewed as a metric that measures the Euclidean distance between voltages produced by the execution of the instructions. Here, we introduce our approach to obtain these distances. As

Table 3.1: ESE values (in zJ) for the Core i7 laptop.

	LDM	**STM**	**LDL2**	**STL2**	**LDL1**	**STL1**	**ADD**	**SUB**	**MUL**	**DIV**
LDM	0	128	377	1344	237	246	283	279	247	1392
STM	53	0	944	1389	114	163	153	150	164	1213
LDL2	336	1394	0	24	84	112	149	143	109	736
STL2	1160	1248	21	0	42	56	99	100	114	536
LDL1	463	236	190	76	0	0	0	0	0	169
STL1	464	274	219	97	4	0	0	0	0	135
ADD	509	255	256	136	21	11	0	3	0	107
SUB	487	244	207	145	11	15	5	0	4	131
MUL	487	252	221	143	8	13	3	15	0	116
DIV	1188	812	651	293	82	90	92	93	134	0

the first step, we are required to estimate the voltages generated by the instructions. However, since ESE is considered as an Euclidean distance and only provides the voltage differences, i.e. the distances in generated voltages among instructions, rather than generated actual voltage values of each instruction, we are required to obtain the distances among the instructions. In that respect, since voltage is a one-dimensional quantity, i.e. the EM emanations caused by different instructions differ only in magnitude, we first focus on the locations of these instructions with respect to each other. Therefore, we are only interested in how instructions are ordered on a line with respect to the quantity of leakage they produce. We also need to emphasize that sorting the instructions can be from the instructions which cause more leakages to the ones which generate less leakages, or the other way around. Let us first consider an example of ESE shown in Table 3.1. From the table, we can deduce that

* LDL1, STL1, ADD, MUL, and SUB have similar ESE values and, therefore, we consider them as one instruction which is called as G_6.

* Spacing between DIV and LDM is the largest (e.g., 1392). Therefore, they must be positioned at the corners, and the rest of the instruc-

tions lie between these two instructions.

* STM is the closest to LDM since LDM produces the smallest ESE value with STM among all instructions (e.g., 128).

* The closest instruction to LDL2 is STL2 since it produces the smallest EM leakage with LDL2.

* G_6 generates smaller ESE value with LDM and STM than STL2 and LDL2. In the same manner, ESE produced by the execution of LDM and LDL2 is larger than the one produced by LDM and G_6. Therefore, G_6 is positioned between STM and LDL2.

Based on the observations above, the order of instructions is provided in Fig. 3.2 and we denote this ordered sequence as $\mathbb{S}$.

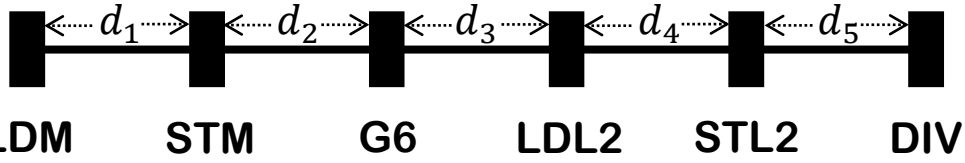


Figure 3.2: An example of instruction ordering based on a measurements in Table 3.1.

Here we note that the order of instructions does not give any intuition about the voltage values of instructions but clarifies the distances among the instructions. We do not give any importance to exact voltage values of each instruction due to the definition of ESE. To calculate the transition probabilities, we only need the distances between each instruction. Therefore, we set the following optimization problem to determine the distances when our input set includes {DIV, MUL, STM, LDM, LDL2, STL2} as follows:

$$\underline{\textit{Setting II :}} \quad \underset{\mathbf{d},\epsilon}{\text{minimize}} \quad \|\epsilon\|_2$$

$$\text{subject to}$$

$$\begin{aligned} \text{ESE}[\text{LDM},\text{STM}] - \kappa d_1^2 &= \epsilon_1 \\ \text{ESE}[\text{LDM},\text{MUL}] - \kappa(d_1+d_2)^2 &= \epsilon_2 \\ \vdots & \\ \text{ESE}[\text{STL2},\text{DIV}] - \kappa d_5^2 &= \epsilon_{15} \end{aligned}$$

where $\mathbf{d}(\mathbb{S}) = [d_1 \ d_2 \ d_3 \ d_4 \ d_5]^T$ is the distance vector required to be revealed, $\epsilon = [\epsilon_1 \ \epsilon_2 \ \cdots \ \epsilon_{15}]^T$ is the estimation error vector needed to be minimized and $\kappa = T_{\text{I}}/R$. We also note that to verify the given order of instructions is the optimum one, we run *Setting II* for different orders and checked the objective value $\|\epsilon\|_2$ is smallest for the order given in Fig. 3.2 among all of the orders of the instructions.

Calculation of Transition Probabilities

After obtaining the relative voltage differences, the next step is to determine the decision boundaries among instructions. Since our inputs lie on the same axis and only ordering of instructions and distances among them are important for transition probability calculations, we assume the mean of the instruction with the leftmost position in the ordered instruction sequence is zero. Then, means of the remaining instructions are calculated based on the obtained distances. Finally, the decision boundaries are calculated using maximum likelihood estimation between two consecutive instructions. For example, if instruction X_1 and X_2 are neighbors, the

decision boundary is the point such that

$$\mathcal{N}(x|\mu_{X_1}, \bar{\sigma}^2_{X_1}) = \mathcal{N}(x|\mu_{X_2}, \bar{\sigma}^2_{X_2}) \tag{3.7}$$

where $\mathcal{N}(\bullet)$ is the normal pdf distribution, $\mu_{X_i} = X_i^v$ is the mean (position) of the i^{th} instruction and

$$\bar{\sigma}^2_{X_i} = \frac{\mathcal{P}_{\mathrm{A}}\left(\mathrm{s}_1^{(\mathrm{X_i})}, \mathrm{s}_2^{(\mathrm{X_i})}\right) - \mathcal{P}_{\mathrm{A}}\left(\mathrm{s}_1^{(\mathrm{NOI})}, \mathrm{s}_2^{(\mathrm{NOI})}\right)}{\kappa}. \tag{3.8}$$

The derivation of $\bar{\sigma}^2_{X_i}$ is given in Appendix B. $\bar{\sigma}^2_{X_i}$ characterizes the additive noise and impact of other program and hardware variabilities present in measurements.

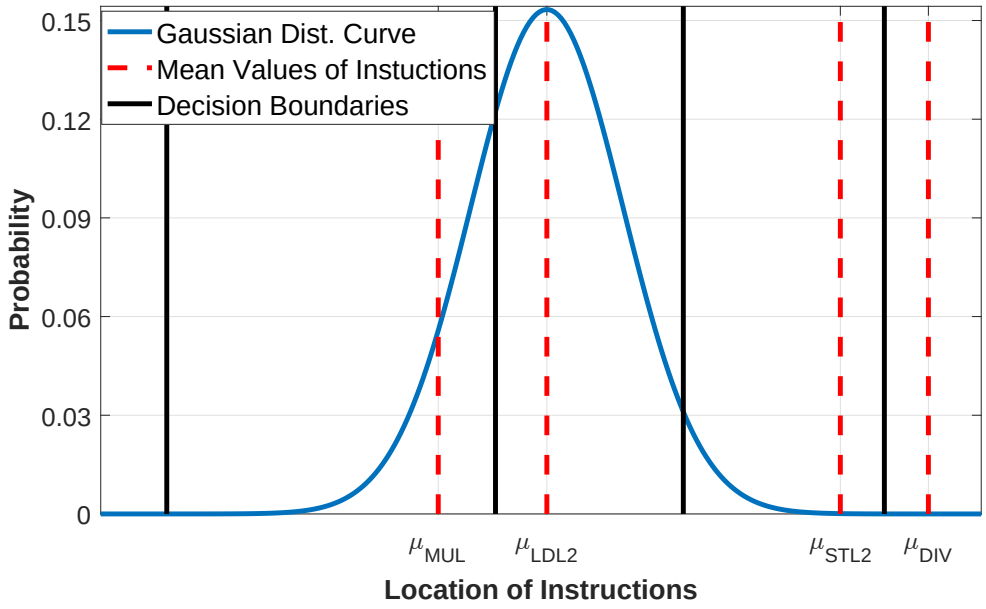


Figure 3.3: An illustration of the process for calculation transition probabilities for a given instruction.

Thereafter, this Gaussian distribution is fit to the graph whose mean equals to the instruction X's mean, and whose standard deviation is $\bar{\sigma}^2_X$. Here we note that scaling mean and the variance of the distances and variances will not change probability distributions, hence we can simplify

the notations by defining $\hat{\sigma}_X^2 = \kappa\bar{\sigma}_X^2$ and

$$\hat{\mathbf{d}}(\mathbb{S}) = \sqrt{\kappa}[d_1 \;\; d_2 \;\; d_3 \;\; d_4 \;\; d_5]^T.$$

Finally, the probability of each interval is assigned to each corresponding instruction as a conditional probability. An illustration of the process is given in Fig. 3.3.

3.4 Experimental Results and Discussions

In the previous section, we have introduced a method to compute side/-covert channel capacity from noisy measurements of ESE. Here we note that the proposed algorithm is general and can be applied to any computational device with any set of instructions. In this section, we give a few examples explaining how to calculate the leakage capacity in (3.7).

Before presenting experimental results, we first summarize the steps for the proposed approach in this chapter:

- To measure the ESE, first install the code given in Fig. 2.1 and measure the power at the frequency f_{alt} which will be depended on n_{inst}.
- By exploiting the equation in (3.5), calculate the alternation energy of every two combinations of instructions.
- Apply *Setting II* as explained in Section 3.3.2 to estimate the emitted EM voltages by the execution of any instructions.
- Estimate the noise power w.r.t. an instruction based on (3.8), and then, calculate the decision boundaries to obtain the cross-transition probabilities of the proposed side/covert channel as described in Section 3.3.2.

- Obtain the probability distribution of instruction occurrences which maximizes mutual information rate of the modeled side/covert channel by exploiting *Setting I*.
- Use (3.7) to compute capacity.

Figure 3.4: Measurement setup.

For the experiments, we use a spectrum analyzer (Agilent MXA N9020A) and a magnetic loop antenna (AOR LA400) as shown in Figure 3.4. This antenna does not use a tuning capacitor and is terminated with a 50Ω load, so it has a flat frequency response between 10 kHz and 1 MHz. Unless otherwise noted, the measurements used an X_1/X_2 alternation frequency of 80 kHz (note that this frequency can be arbitrarily changed) and a measurement distance of 10 cm. A resolution bandwidth of 1 Hz was used to minimize noise.

3.4.1 Experimental Results of Core I7 Laptop

As the first example, we consider the ESE values in Table 3.1. ESE values are from a Core i7 laptop with the 10 cm distance and the intended alternation frequency 80 kHz. We constrain the input set to

$$\{\mathrm{LDM}, \mathrm{STM}, \mathrm{DIV}, \mathrm{MUL}, \mathrm{LDL2}, \mathrm{STL2}\},$$

using MUL as a representative of the "G6" group of instructions

$$\{\mathrm{LDL1}, \mathrm{STL1}, \mathrm{ADD}, \mathrm{SUB}, \mathrm{MUL}\}.$$

Table 3.2: Transition probabilities based on ESE measurement in Fig. 3.1

	LDM	**STM**	**MUL**	**LDL2**	**STL2**	**DIV**
LDM	0.53	0.38	0.09	0	0	0
STM	0.49	0.26	0.19	0.04	0.01	0.01
MUL	0	0	0.94	0.06	0	0
LDL2	0	0	0.11	0.85	0.04	0
STL2	0	0	0	0.06	0.5	0.44
DIV	0	0	0	0	0.39	0.61

As the first step, cross-transition probabilities must be obtained. Therefore, we require to find the distances among the instructions with respect to emitted EM signal powers. The distances are acquired by feeding *Setting II* with Table 3.1. The order of instruction is as given in Section 3.3.2. The resulting distance vector $\hat{\mathrm{d}}$, which contains the interval lengths between two neighbouring instructions in the magnitude space, is $\hat{\mathbf{d}}(\mathbb{S}) = [1 \;\; 18.8 \;\; 5.9 \;\; 9.5 \;\; 1]$.

The next step is to calculate the decision boundaries and noise deviations for each instruction. Noise powers for each instruction are calculated based on (3.8) such that $\overline{\sigma}^2(\mathbb{S}) = [7.87 \;\; 13.64 \;\; 1.47 \;\; 2.6 \;\; 3.21 \;\; 6.38]$. Since both noise deviations and distances in emanated powers are in hand, cross-transition probabilities can be calculated by following instructions given in Section 3.3.2. The resulting transition probabilities for the Core i7 laptop are provided in Table 3.2. Once transition probabilities are computed, it is possible to find the maximum channel rate and achievable capacity. The Intel Core i7 processor is very complex, featuring out of order

execution and many other microarchitectural optimizations. These optimizations often result in multiple instructions being executed in parallel, which results in reduced instruction length in our simplified model. Therefore, we define the length of each instruction as the excess amount of time caused by appending the same instruction to both for-loops given in Fig. 1 with respective to inserting nothing. For example, if T_{NOI} and T_X are the times, which takes to execute the microbenchmark in Fig. 1 when nothing is inserted and instruction X is inserted respectively, the length of the instruction X is

$$L_X = \frac{T_X - T_{NOI}}{2n_{\text{inst}}}. \tag{3.9}$$

Then, the instruction lengths (execution times) are normalized and quantized such that the smallest length is equivalent to one and the lengths of all instructions are integers – this is reasonable because processor activity is in terms of clock cycles, with many instructions taking only one cycle to execute while several instructions take a number of cycles. Therefore, we consider the length vector for *Setting I* as $\mathbf{L}(\mathbb{S}) = [13\ \ 20\ \ 1\ \ 1\ \ 3\ \ 8]^T$. The capacity rate is attained as **0.72 Bits/Quantum**, where **Quantum** is defined as average instruction length per symbol. Although, in the information theory literature, the corresponding capacity is called *Channel Capacity per Unit Cost* [76], we use the term **Quantum** to indicate that our results are respect to both the minimum number of clock cycles to execute an instruction and the clock frequency of the device. The resulting instruction probabilities are

$$\mathbf{P}_{\mathbb{S}} = [0.005\ \ 0\ \ 0.377\ \ 0.354\ \ 0.264\ \ 0]^T.$$

The results are reasonable since the lengths of LDM, STM and DIV are

longer and the deviations of these instructions are high, which increases uncertainty about the outputs of the instructions. On the other hand, although the entropy of STL2 is large, its probability is the largest because its uncertainty is provoked due to existence of DIV. Since the optimal probabilities of occurrence of DIV is zero and the length of STL2 is small, the uncertainty about STL2 is largely eliminated and having a smaller execution time makes it more favourable.

3.4.2 Experimental Results of Core 2 Laptop

In this section, we will provide the experimental results on an Intel Core 2 Duo laptop with 1.8GHz CPU clock, 333MHz DDR2 memory, 32KB 8 way L1 Data Cache and 4096 KB 16 way L2 cache. The considered instruction set is {LDM, LDL2, STL2, LDL1, STL1, ADD, SUB, MUL, DIV}. As the first step, we perform experiments to calculate the ESE power of instruction pairs at 10cm and 80kHz. The measured ESE values are given in Table 3.3.

We observe that the ESE pattern observed in the Core I7 laptop also exists for the ESE values of the Core 2 Duo laptop. For example, all arithmetical instructions except DIV emit similar leakages whereas DIV behaves like a completely different functional instruction. Therefore, for the leakage capacity calculation, one of these will be selected. Another observation is that the maximum leakage occurs when STM and DIV are embedded into the microbenchmark. Moreover, the tableau is almost symmetrical which supports our earlier observation that the instruction order in the microbenchmark does not affect the EM leakage.

To be able to calculate the leakage capacity through *Setting I*, we order the instructions with respect to their emitted voltage potentials. The sorted

Table 3.3: ESE Values (in zJ) for the Core 2 Duo Laptop.

	STM	**LDL2**	**STL2**	**LDL1**	**STL1**	**ADD**	**SUB**	**MUL**	**DIV**
STM	0	61	100	19	25	21	21	21	56
LDL2	62	0	1	79	80	84	82	81	101
STL2	101	1	0	101	104	108	105	105	160
LDL1	19	70	103	0	0	0	0	0	2
STL1	26	79	104	0	0	0	0	0	2
ADD	21	84	109	0	0	0	0	0	1
SUB	20	82	107	0	0	0	0	0	1
MUL	21	83	108	0	0	0	0	0	1
DIV	57	99	161	2	2	1	1	1	0

sequence of instructions which minimizes the objective function of *Setting II* is obtained as

$$\mathbb{S} = \{\mathrm{STL2}, \mathrm{LDL2}, \mathrm{STM}, \mathrm{ADD}, \mathrm{DIV}\}$$

where ADD is chosen to represent the algorithmic operation set. The distances between the emitted voltages are $\hat{\mathbf{d}}(\mathbb{S}) = [2.1\ 7.3\ 1.45\ 1.85]$.

The next step is to obtain the cross-transition probabilities of the proposed model. In that respect, the noise powers conditioned on the execution locations of instructions are calculated as

$$\bar{\sigma}^2(\mathbb{S}) = [0.23\ 0.23\ 17.64\ 0.23\ 0.23]$$

by the equation given in (3.8). Since we have both the noise powers and distances among the emitted powers resulting from the execution of instructions, it is time to calculate the cross translation probabilities of the model as described in Section 3.3.2. The transition probabilities are given in Table 3.4. We observe that the transition probability matrix is diagonally dominated expect the instruction STM as in the case of the Core I7

laptop.

Table 3.4: Transition probabilities based on ESE measurement in Table 3.3

	STL2	**LDL2**	**STM**	**ADD**	**DIV**
STL2	0.99	0.01	0	0	0
LDL2	0.01	0.98	0.01	0	0
STM	0.02	0.05	0.49	0.15	0.29
ADD	0	0	0.07	0.9	0.03
DIV	0	0	0	0.03	0.97

.

For this processor, we obtain the relative instruction times as $\mathbf{L}(\mathbb{S}) = [3\ 1\ 31\ 1\ 8]$ by employing (3.9). We attain the capacity rate as **1.09 Bits/Qauntum** with the instruction occurrence probabilities

$$\mathbf{P}_{\mathbb{S}} = [0.09\ 0.44\ 0\ 0.47\ 0].$$

We can observe again that the probabilities of longer instructions in terms of execution times are set to zero although these instructions have larger ESE values. Since execution times of these instructions are longer, the information, they provide at each quantum interval, is much smaller than the instruction with shorter execution times.

3.4.3 Experimental Results for Turion X2 Laptop

In this section, we provide the experimental results for AMD Turion X2 processor with 64 KB, 2 way L1 Data Cache and 1024 KB, 16 way L2 Cache. As usual, the first step is to measure the ESE values of instruction pairs which are given in Table 3.5.

The order which minimizes the objective function of *Setting II* is same as for the Core 2 Duo laptop. As the side product of *Setting II*, the dis-

Table 3.5: ESE Values (in zJ) for the AMD Turion X2 Laptop.

	STM	**LDL2**	**STL2**	**LDL1**	**STL1**	**ADD**	**SUB**	**MUL**	**DIV**
STM	0	16	26	5	8	5	5	2	18
LDL2	14	0	0	31	33	34	33	34	53
STL2	30	0	0	40	40	42	41	42	70
LDL1	4	32	40	0	0	0	0	0	1
STL1	4	32	40	0	0	0	0	0	0
ADD	6	35	43	0	0	0	0	0	0
SUB	5	34	41	0	0	0	0	0	0
MUL	6	34	42	0	0	0	0	0	0
DIV	25	54	71	1	0	0	0	0	0

tance vector providing the distances between the neighbouring instruction voltages is $\hat{\mathbf{d}}(\mathbb{S}) = [1.01\ 3.74\ 1.97\ 1.74]$. The next step is to calculate the cross-transition probabilities of instructions. Thus, we calculate noise powers conditioned on instructions as $\bar{\sigma}^2(\mathbb{S}) = [0.4\ 0.34\ 24.22\ 0.34\ 0.34]$. By following the steps given in Section 3.3.2, we calculated the cross transition probabilities which are given in Table 3.6. The final step is to obtain the optimal solution for *Setting I*. Relative instruction execution times are $\mathbf{L}(\mathbb{S}) = [3\ 1\ 30\ 1\ 8]$. The rate is attained as **0.97 Bits/Quantum** with instruction occurrence probabilities $\mathbf{P}_{\mathbb{S}} = [0\ 0.498\ 0\ 0.502\ 0]$. If we compare the results with the Core 2 Duo laptop, this time we observe STL2 is set to zero. The reason can be justified as follows: We observe that the existence of STL2 and LDL2 creates more uncertainty, therefore, instead of keeping these two instructions in the codebook, removing one of these instructions increases the reliability of the communication channel. Since the execution time of STL2 is longer, forcing its occurrence probability to be zero enhances the overall rate obtained through *Setting I*.

Table 3.6: Transition probabilities based on ESE measurement in Table 3.3

	STL2	**LDL2**	**STM**	**ADD**	**DIV**
STL2	0.79	0.21	0	0	0
LDL2	0.19	0.79	0.02	0	0
STM	0.19	0.11	0.27	0.14	0.28
ADD	0	0	0.04	0.89	0.07
DIV	0	0	0	0.07	0.93

3.4.4 Experimental Results for NIOS Processor on the DEI FPGA

As our last example, we use the ESE values from [58]. The measurements are done 10 cm above the NIOS processor on the DE1 FPGA board. Assuming that an instruction could not have smaller magnitude than NOI, i.e.

$$\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X})}, \mathrm{s}_2^{(\mathrm{X})}\right] > \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{NOI})}, \mathrm{s}_2^{(\mathrm{NOI})}\right]$$

where X is any instruction rather than NOI, we assume that minimum ESE measurement with an instruction is 0.015 zJ. The main difference between FPGA and Core i7 measurements is that arithmetic operations and LDL1 are distinguishable in the FPGA case. This time, the order of instructions is $\mathbb{S}$ ={LDM-DIV-LDL1-MUL-ADD-SUB}. As the first step, we find the interval lengths among instructions which is $\hat{\mathbf{d}}(\mathbb{S}) = [2.29\ \ 0.39\ \ 0.27\ \ 3.39\ \ 0.91]$. Furthermore, noise power is calculated as

$$\overline{\sigma}^2(\mathbb{S}) = [0.008\ \ 0.0075\ \ 0.0075\ \ 0.0075\ \ 0.0075\ \ 0.0075].$$

Then, the transition probabilities for FPGA are calculated and given in Fig. 3.7. For the *Setting I*, we assume $\mathbf{L}(\mathbb{S}) = [7\ \ 5\ \ 4\ \ 4\ \ 1\ \ 1]^T$ and obtain **1.14 Bits/Quantum** channel capacity rate where the probabilities of instructions

are assigned as

$$\mathbf{P}_{\mathbb{S}} = [0.004 \;\; 0.018 \;\; 0.032 \;\; 0.035 \;\; 0.455 \;\; 0.456]^T.$$

Observe that although the entropy of LDM and DIV is similar to ADD and SUB, assigned probabilities are much higher for ADD and SUB because their lengths are much smaller than other instructions, therefore, they can transmit more information in a second. Likewise, the optimum occurrence probabilities also explain which instructions create more vulnerability when an alternation created with them. For example, if a script generates an alternation due to ADD with another instruction, the leakage caused by this alternation will be huge presumably.

Table 3.7: Transition probabilities based on ESE measurement in [58]

	LDM	**DIV**	**LDL1**	**MUL**	**ADD**	**SUB**
LDM	1	0	0	0	0	0
DIV	0	0.99	0.01	0	0	0
LDL1	0	0.01	0.93	0.06	0	0
MUL	0	0	0.06	0.94	0	0
ADD	0	0	0	0	1	0
SUB	0	0	0	0	0	1

Since a modern processor executes several billion instructions per second, the computed EM side/covert channel capacity of 1.14 Bits/Quantum implies that the attacker might obtain several gigabytes of information per second. Although this is an extremely high information leakage rate, the rate actually achieved by practically demonstrated side channel attacks on cryptographic implementations is much lower. This apparent discrepancy is primarily caused by different assumptions about how the program is designed and different definitions of what constitutes infor-

mation. Our capacity derivations are for the worst-case scenario where the program is specifically designed to leak information, whereas cryptographic implementations are designed to have significant resilience to side channel attacks. Furthermore, cryptographic attacks only consider the rate of leakage for encryption keys, whereas our capacity derivations account for any information about program execution.

3.4.5 Effect of Alternation Time T_{alt} on ESE

In this section, the effect of measurement frequencies on leakage capacity is investigated. The measurements are done at the side of the NIOS processor on the DE1 FPGA board with 10 cm distance. For each frequency, we follow the procedure given in the beginning of Section 3.4. Occurrence probabilities of instructions are provided in Table 3.8. Furthermore, $\mathbf{R}(\mathbb{S}) = [1.12 \quad 1.14 \quad 1.16 \quad 1.14]$ Bits/Quantum is the vector containing resulting rates of the setups with varying frequencies where $\mathbb{S} =$ [40 kHz, 50 kHz, 70 kHz, 80 kHz]. These results indicate that as long as the same setup is considered but different values of T_{alt} are used, the change on the capacity and probability distribution of instructions are minimal.

Table 3.8: Occurrence Probabilities of Instructions From Measurements Collected at Different Frequencies

	LDM	**DIV**	**LDL1**	**MUL**	**ADD**	**SUB**
40 kHz	0.005	0.019	0.024	0.034	0.459	0.459
50 kHz	0.004	0.017	0.035	0.036	0.454	0.454
70 kHz	0.004	0.018	0.041	0.041	0.447	0.449
80 kHz	0.004	0.018	0.032	0.035	0.455	0.456

3.4.6 Justification of the Proposed Model

In this part, we compare our results with classical Shannon capacity and provide the underlying reasons why the proposed methodology is needed. Table 3.9 provides the capacities for different platforms. Note that Shannon capacity presumes that the codewords have the same length which means the number of channel uses by each codeword is same.

In Table 3.9, PS, SC and USC represent the capacity results obtained by the proposed framework, by assuming equal codeword length and exploiting Shannon's capacity, and by Shannon's capacity divided by the average code-length calculated via the occurrence probabilities which are utilized to obtain Shannon's capacity. These results clarify why the proposed method is more informative in terms of leakage capacity.

Table 3.9: Capacity comparison with classical Shannon's capacity

	AMD Turion	**Core 2 Duo**	**Core I7**	**NIOS**
PS (Bits/Quantum)	0.97	1.09	0.72	1.14
SC (Bits/Symbol)	1.46	1.86	1.64	2.46
USC (Bits/Quatum)	0.42	0.48	0.27	0.68

Executed instructions take different amount of time, therefore, the number of channel uses for each instruction varies which could not be reflected by the classical Shannon capacity. On the other hand, if we normalize the Shannon's capacity with the average channel uses, we obtain the information per channel uses. If we compare the results, which are USC and the proposed method, we observe that the proposed method increases the effectiveness of the channel per use. Therefore, we can claim that the proposed method provides better information on the severity of the leakage.

3.5 Potential Defense Mechanisms

Our technique can help defensive efforts for both covert and side-channels, although in somewhat different ways. For covert channels, the assumption is that the developer of the program is crafting the code so as to maximize the leakage. Since our technique allows potential leakage to be computed given the instruction frequencies (how often each type of instruction executes in the program), one defensive approach would be to determine the instruction mix (there are a number of program execution profiling tools that can do that), compute the potential leakage and use that to drive the decision about what to do (e.g. subject high-potential-leakage programs to additional scrutiny, "sandbox" them to limit their access to potentially sensitive information, etc.).

For side-channels, the assumption is that the developer is interested in reducing the leakage produced by the code they are developing. Our technique allows the programmer to quantify the potential leakage of various parts of the program (or the program as a whole), identify parts of the program whose leakage may be higher than tolerable, and then make sure that these parts of the program do not operate on sensitive data, change instruction mix in a way that reduces the potential leakage (e.g. reduce the use of high-ESE instructions, or even use our technique to quantify the reduction in leakage that would be obtained through each potential code change), or surgically apply one of the known techniques for reducing information leakage through side-channels.

In terms of insight that we can offer to programmers and hardware designers, our results indicate that most of the potential information leakage is a result of using a very small number of instructions that are much

easier to correctly distinguish. For software designers, this means that a program's use of these instructions should not be dependent on sensitive data values. For hardware designers, this means that reduction of a hardware design's overall vulnerability to EM side channel attacks largely depends on addressing the EM side channel signals produced by this very small subset of the processor's overall instruction set.

The goal of this chapter is to find out which instructions are the most promising to be used as codewords in a covert channel or to be checked for in side-channel. One may assume that instructions that have the strongest EM signatures (e.g. cache misses) are the best to be used as codewords for information transmission. However, as our analysis shows, that is not the case because cache misses also take the longest time to be executed, hence carrying much smaller Bits/Quantum information. The future work will be on how to design a code-book that consists of these codewords and achieves capacity limits derived in this chapter. Some simple examples of an attack based on the selection of instructions are shown in [50] and [4], but more systematic approach is needed and is part of our future work.

3.6 Summary

This section offers an answer to how much information is "transmitted" by execution of particular sequence of instructions in a processor. We first propose a new method to estimate the maximum information leakage through EM signals generated by execution of instructions in the processor. Then, we derive a mathematical relationship between electromagnetic side-channel energy (ESE) of individual instructions and the measured pairwise side-channel signal power. Furthermore, we use this measure to

calculate the transition probabilities needed for estimating capacity, and propose a new method to estimate side/covert channel capacity created by the execution of instructions in a processor. Finally, we illustrate how the proposed method works on several practical examples. The reason behind naming the proposed structure as covert/side channel can be claimed as follows: Covert channels are the channels can be exploited for a hidden communication. Therefore, by taking advantage of side channels generated by EM leakage, a covert channel communication system can be created. However, how to exploit this side channel as a communication system is still an open question we will address in following chapters.

CHAPTER 4
ELECTROMAGNETIC SIDE CHANNEL INFORMATION LEAKAGE CREATED BY EXECUTION OF SERIES OF INSTRUCTIONS IN A COMPUTER PROCESSOR

4.1 Overview

This chapter introduces a methodology to relate program execution to electromagnetic side-channel emanations, and estimates side-channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor. To model dependence among program instructions in a code, we propose to use Markov Source model, which includes the dependencies among sequence of instructions as well as dependencies among instructions as they pass through a pipeline of the processor.

We know that side channels are not designed to transfer information at all, and its transmission is often corrupted by insertion, deletion and erroneous transfer of bits. While there is a large number of papers discussing 1) bounds on the capacity of channels corrupted with synchronization errors, 2) bounds on the capacity of channels corrupted with synchronization and substitution errors, or 3) bounds on the capacity when codewords have variable length but no errors in the channel, none of them provides the answer to how much information is "transmitted" by execution of particular sequence of instructions that do not have equal timing and are transmitted through erroneous channel. The first attempts to answer this question were presented in [77, 20], where covert channels are generated,

and upper and lower leakage capacities were derived. In Chapter 3, a side-channel leakage capacity is derived for a discrete memoryless channel where the assumption is that each transmitted quantum of information (i.e. instruction in the code) is mutually independent but do not have equal length. Although all these efforts make an important step toward assessing information leakage from side-channels, they fall short of considering the relationship among sequence of instructions, which is a result of program functionality as well as a processor pipeline depth, which impacts how much signal energy will be emanated.

This chapter addresses these problems by

- Deriving side-channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor,
- Using Markov Source model to model dependence among program instructions in a code, which includes the dependencies that exist in instruction sequence since each program code is written systematically to perform a specific task,
- Deriving a mathematical relationship between the emanated instruction signal power (ESP) as it passes through processor pipeline and total emanated signal power while running a program (This is in contrast to work in Chapter 3 where all energy emanated through side-channels is assigned to an instruction, without taking into account effect of processor pipeline depth, which significantly impacts the emanated signal),
- Considering sources for channel inputs are emitted EM signals during instruction executions,

- Providing experimental results to demonstrate that leakages could be severe and that a dedicated attacker could obtain important information.

Moreover, the work in this chapter considers processors as the transmitters of a communication system with multiple antennas. The antennas correspond to different pipeline stages of any processor. Also, inputs of the transmitter show dependency based on a Markov model which reflects the practicality of a program. Therefore, the goal in this chapter is to obtain the channel capacity of a communication system, or the severity of the side channels.

The rest of the chapter is organized as follows: Section 4.2 defines the proposed leakage capacity and introduces the Markov Source model. Section 4.3 derives a mathematical relationship between the emanated instruction power (ESP) as it passes through processor pipeline and total emanated signal power while running a program. Section 4.4 provides experimental results and leakage capacities for various devices. Finally, Section 4.5 provides a recipe for the leakage capacity calculation, and Section 4.6 provides the summary of the chapter.

4.2 Modeling Information Leakage from a Computer Program as a Markov Source Over a Noisy Channel

In this section, we propose a Markov source model whose states are series of instructions in a pipeline. We assume that channel inputs at each state are the emanated signal powers produced as combination of different instructions in a pipeline, and the channel outputs are the noise corrupted versions of the emitted signals. The reason for considering such a Markov model is that individual instructions are not independent from each other

in the code as well as that ordering of instructions as they pass through pipeline significantly impacts emitted signal patterns.

4.2.1 Proposed Markov Source Model for Modeling Information Leakage from a Sequence of Instructions

Here, we describe a Markov source model that characterizes relationship among sequence of instructions as they pass through pipeline stages in a processor. Note that a processor pipeline is an assembly line for computing, and contains groups of activities related to computational tasks, i.e. fetching, decoding, executing, etc. [78]. We assume that channel inputs at each state are the emanated signal powers obtained as a combination of different power levels that instructions experience as passing through a pipeline, and the channel outputs are the noise corrupted versions of the emitted signals. To include the effect of pipeline depth, states are assumed to be all possible instruction combinations because each stage performs an operation on the instruction in the queue. For example, if a pipeline has a depth of m, and the cardinality of $\mathcal{S}$ is q, the number of states will be q^m.

To illustrate how the proposed Markov Model works, Fig. 4.1 shows an example of Markov Source Model for the instruction execution when the pipeline depth is m, and the cardinality of the considered instruction set is three. In the figure, $P_{i,j}$ represents the state transition probability from state i to state j, and circles denote the states of the model. The instruction set used in the example is {D, S, M}, which corresponds to division, subtraction, and multiplication, respectively. We utilize trellis diagram to explain the model explicitly although transitions are time invariant, i.e. $P_{i,j}$ does not vary in time. Moreover, the labels of the states are chosen

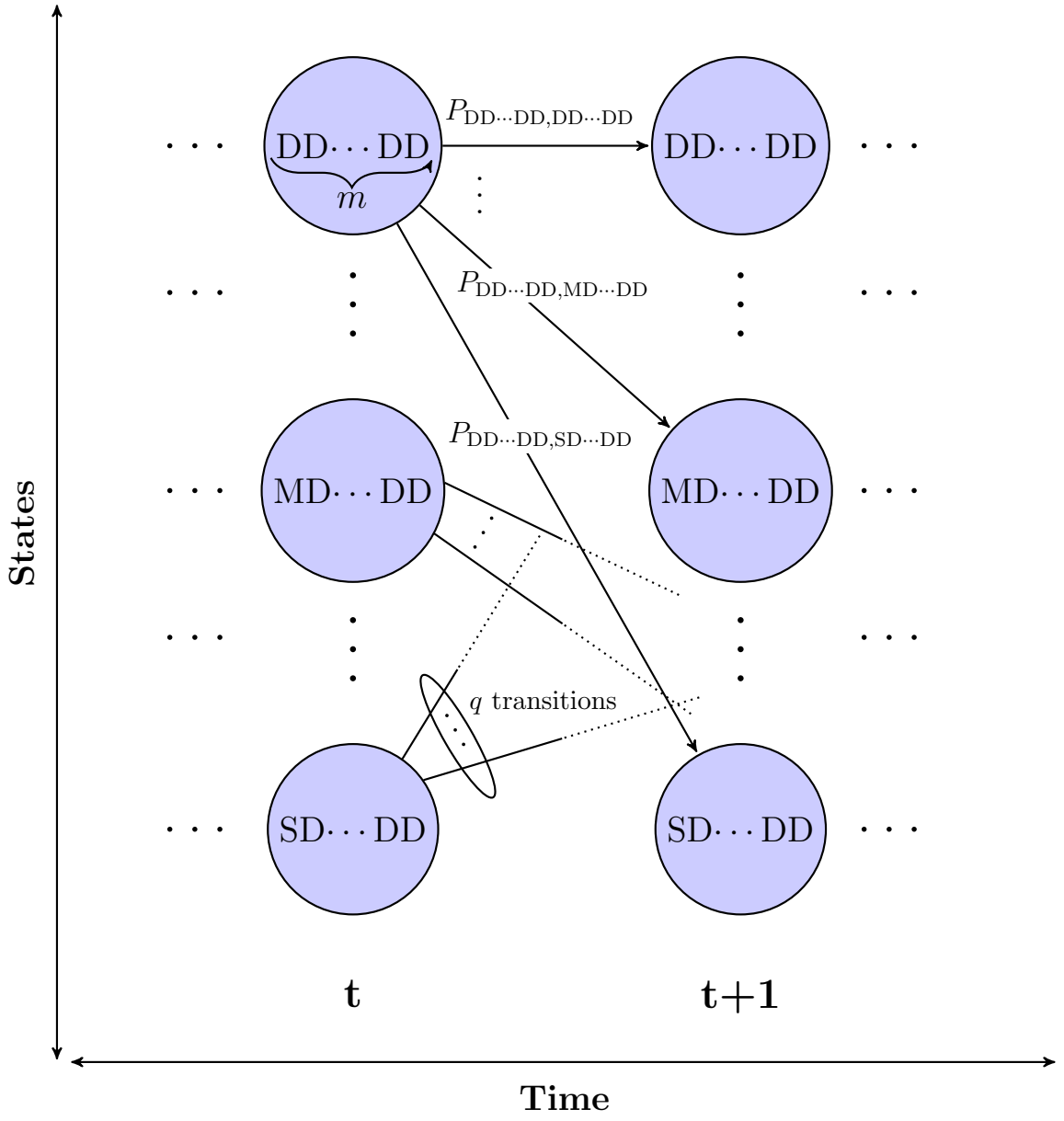


Figure 4.1: Markov Source Model for the instruction execution when the pipeline depth is m, and the cardinality of the considered instruction set is three.

as the combination of letters representing the instructions in the pipeline. Considering these three instructions, one of the states can be labeled as "DDI$_S$DD" where $\mathbf{I}_S$ is a sequence of instructions whose length is $m-4$. Interpretation of the state corresponding to the label is that instructions in the 1^{th}, 2^{nd}, ..., $m-1^{th}$ and m^{th} stages of the pipeline are D, D, ..., D, and

D, respectively.

For each state, the number of possible paths is q, i.e. it is equal to the number of instructions in the set. For example, for the considered example, there exist only three paths from each state since the instruction set contains only three elements. For example, the possible states after "DDI$_S$DD" could be "DDDI$_S$D", "MDDI$_S$D" or "SDDI$_S$D". Furthermore, we assume that any instruction can be followed by any other instruction. This assumption helps the proposed model to be an indecomposable channel, therefore, the mutual information definition given in (2.3) is applicable to the proposed scheme.

We need to note that by considering the Markov source model, we can successfully capture the pipeline effect because it puts constraints on the state transitions. Moreover, $P_{i,j}$ explains the frequency of the instruction order encountered in the program. Therefore, the capacity of the proposed model provides the worst instruction sequence distribution which leaks information the most.

4.2.2 Introducing Information Leakage Capacity for the Proposed Markov Source Model

The capacity definition given in (2.3) is well suited for Markov source models if the states take the same amount of time. In other words, the definition is valid for the models where the transitions last equal amount of time, and the transition time is not dependent on a given state. Unfortunately, applying the same capacity definition to the proposed scheme is not appropriate because different instructions take different time to execute. Therefore, we need a capacity definition which also accounts for instruction execution times. Hence, we propose a method to quantify the

information leakage, which considers both execution time of each state and the mutual information between input and output sequences.

Definition 1. *Assuming varying execution time of instructions, maximum possible information leakage through a processor is defined as*

$$C = \max_{\substack{P_{ij} \\ (i,j)\in\mathcal{T}}} \lim_{n\to\infty} \frac{I\left(S_1^n; Y_1^n | S_0\right)}{\sum_{i=1}^{n} \mathbf{L}(i)} \tag{4.1}$$

where $\mathbf{L}(i)$ *is the length of the state executed at the* i^{th} *transition.*

Following the analogy between equations (2.2) and (2.3), we can rearrange the equation in (4.1) as follows

$$\lim_{n\to\infty} \frac{I\left(S_1^n; Y_1^n | S_0\right)}{\sum_{i=1}^{n} \mathbf{L}(i)} = \frac{\lim_{n\to\infty} \frac{1}{n} I\left(S_1^n; Y_1^n | S_0\right)}{\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{L}(i)} \tag{4.2}$$

$$= \frac{\sum_{i,j:(i,j)\in\mathcal{T}} \mu_i P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right]}{\sum_{i\in\mathcal{S}} \mu_i L_i} \tag{4.3}$$

where $\mathcal{S}$ is the set containing all existing states, i.e. all instruction combinations, and L_i is the execution length of the state i. Therefore, our definition can also be written as

$$C = \max_{\substack{P_{ij} \\ (i,j)\in\mathcal{T}}} \frac{\sum_{i,j:(i,j)\in\mathcal{T}} \mu_i P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right]}{\sum_{i\in\mathcal{S}} \mu_i L_i}. \tag{4.4}$$

The result of this optimization provides the possible information leakage in bits per smallest number of clock cycles required to execute a state in $\mathcal{S}$ (which we call Bits/Quantum), not bits per second. The reason is that each instruction takes at least one clock cycle for any device, but clock frequencies can vary from one device to another. Since the goal is to an-

alyze the leakage capacity on instruction level, we provide our results in Bits/Quantum (similar to previous chapter, we use the term Quantum instead of *Capacity per Unit Cost* [76]). Please note here that even the leakage capacity of a device is small, the number of bits, a device can transmit in a second, could be large. Therefore, while examining the vulnerability of any device against side channel attacks, combining the leakage capacity with clock frequency leads to the most accurate results.

4.2.3 Reducing the Size of the Markov Source Model

The main problem of the proposed Markov source model is the number of possible states and transitions. As the depth of the pipeline and the number of considered instructions increase, the number of states increases exponentially. This increase causes the iterative algorithm given in [72] to be more complex. Choosing states as individual instructions will simplify the proposed scheme. For these states, the channel input signal is assigned as the emanated EM signal while executing the corresponding instruction through all pipeline stages. With this approach, the number of states increases linearly, not exponentially, as the number of instructions increases.

In Fig. 4.2, we provide an example of the state diagram when the instruction set is {D, M, S}. This model is still indecomposable based on the assumption that each instruction can follow any other instruction. Therefore, the capacity definition given in (4.3) can be used to calculate leakage capacity limits. However, this definition also does not have a closed form solution, and an empirical algorithm similar to expectation-maximization (ExMa) algorithm in [72] is needed to solve the problem.

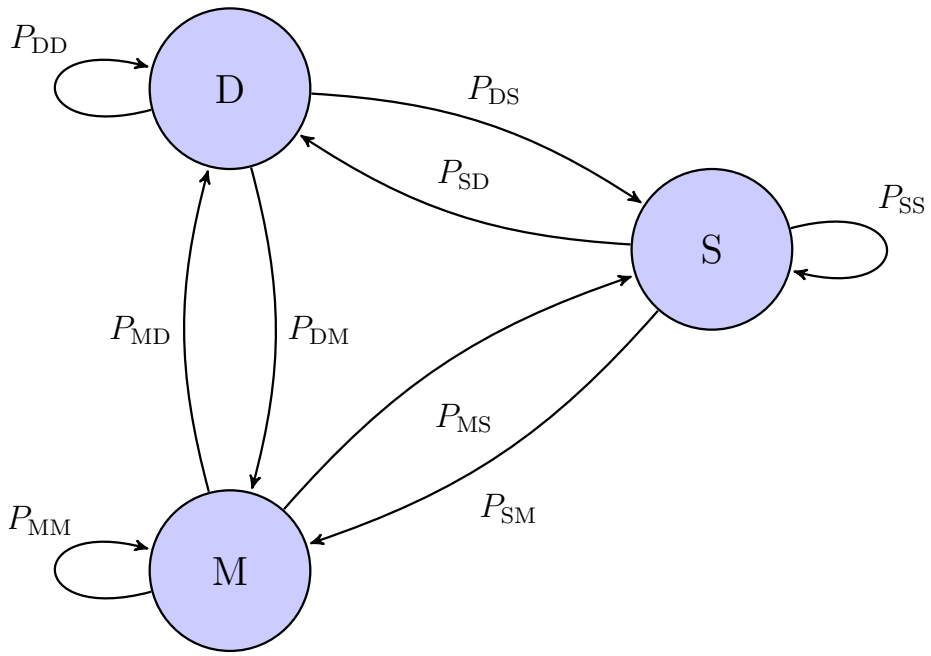


Figure 4.2: Simplified version of Markov Source Model for the instruction execution when the cardinality of the considered instruction set is three.

4.2.4 An Empirical Algorithm to Evaluate the Leakage Capacity

To utilize the ExMa algorithm, we have to adjust the proposed model given in the previous section to remove the execution time of the instructions from the optimization problem. To achieve this goal, we propose to split the instructions into unit length sections, i.e., one clock cycle segments, and treat each of these segments as an individual state. To protect the overall framework and instruction sequence, we have to introduce some constraints for possible state transitions.

Let $K \in \mathcal{S}$ be a state whose length is $L_K > 1$. For the proposed model, we divide it into L_K different states, where the states are named as K_i where $i \in \{1, \cdots, L_K\}$. Each sub-state is called:

- *Initial state* if $i = 1$, i.e. K_1,
- *Exit state* if $i = L_K$, i.e. K_{L_K},

- *Intra-state* if $i \in \{2, \cdots, L_K - 1\}$

of an instruction K. However, if the length of the instruction K equals to one, we keep the instruction set unmodified. Note that the initial and exit states of K will refer to full set K for the scenario when K takes only one clock cycle. Let $\mathcal{S}_M$ and $\mathcal{T}_M$ be the set of states and state transitions, respectively, after splitting the states to have a new instruction set whose members take same amount of time. Therefore, we can rewrite (4.3) as

$$C = \max_{\substack{\mathbb{P}_{ij} \\ (i,j)\in\mathcal{T}_M}} \sum_{(i,j)\in\mathcal{T}_M} \mathfrak{u}_i \mathbb{P}_{ij} \left[\log \frac{1}{\mathbb{P}_{ij}} + \mathbb{T}_{ij} \right] \tag{4.5}$$

where $\mathbb{P}_{ij}$ refers the modified state transition probabilities, $\mathfrak{u}_i$ is the stationary distribution of the new states, and $\mathbb{T}_{ij}$ is defined as in (2.4) in the new model.

Dividing the original states into substates is not enough to protect the duality between the optimization settings given in (4.3) and (4.5). We also have to make sure that the state transitions occur in a way that the instruction sequences for both settings follow the same path. For example, let L_K be equal to 2. To ensure the duality, $\mathbb{P}_{K_1 j}$ must be nonzero only if j is K_2. More formally, to guarantee the duality between the equations (4.3) and (4.5), we employ constraints on transitions which only allow state transitions in the following scenarios:

$\mathbf{R}_1$. An exit state of any instruction to an initial state of any instruction,

$\mathbf{R}_2$. K_i to K_{i+1} of instruction K where $i \in \{1, \cdots, L_K - 1\}$.

Fig. 4.3 illustrates the proposed framework. This figure is a transformed version of the Markov source model given in Fig. 4.2 based on the rules imposed by $\mathbf{R}_1$ and $\mathbf{R}_2$. We assume that D and M take four and

three times of the execution time of S, respectively. Here, M_1 and D_1 are the initial states, M_3 and D_4 are the exit states of M and D, respectively. D_2 and D_3 are the intra-states of DIV, and M_2 corresponds the intra-state of M. Note that these values are chosen arbitrarily, and only given as an illustration.

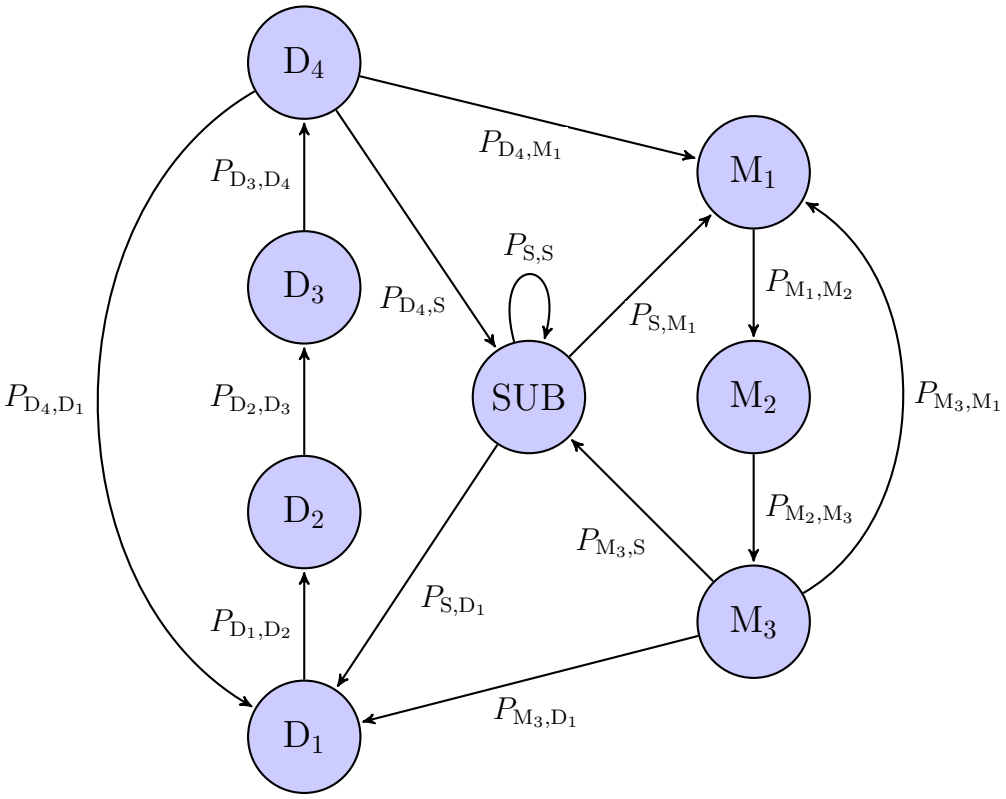


Figure 4.3: Markov Model for the instruction execution as it goes through sub-states that take equal amount of time.

By applying the transformations introduced above, we have removed the problem of variable time of execution per instruction. The following theorem proves the models given in Section 4.2.3 and Section 4.2.4 are dual, and will lead to the same capacity results.

Theorem 1 (Duality)**.** *The optimization settings given in (4.3) and (4.5) are dual problems if the constraints imposed by* $\mathbf{R}_1$ *and* $\mathbf{R}_2$ *are satisfied.*

Proof. Please see Appendix E. □

Figure 4.3 illustrates that although we pose some constraints on the possible state transitions, the state transition diagram is still indecomposable. Therefore, the capacity definition and corresponding iterative algorithms given in [72] can be utilized. However, to apply the algorithm, the channel inputs have to be known. In the following section, we introduce a methodology to calculate the channel input power, i.e., emitted signal power while processing an instruction through the pipeline.

4.3 Estimating Channel Input Power in the Proposed Markov Model

To obtain the channel inputs for the proposed model, in this section, we derive a mathematical relationship between the emanated instruction power as it passes through processor pipeline and total emanated signal power while running a program. This is in contrast to work in [18] where all energy emanated through side-channel is assigned to an instruction, without taking into account effect of processor pipeline, which significantly impacts the emanated signal. Another advantage of this approach to calculate emanated energy per instruction is that capacity can be directly related to signal to noise ratio (SNR).

4.3.1 Definition for Emanated Signal Power (ESP) of Individual Instructions as They Pass Through Pipeline

In this section, we define **E**manated **S**ignal **P**ower (ESP) which is the channel input power for the proposed Markov source model.

For activity $\mathcal{A}_1$, let assume $\mathtt{T}_{\mathcal{A}_1}$ is the execution time, $\mathtt{T}^{\mathtt{P}}_{\mathcal{A}_1}$ is the total time spent in the pipeline except the execution stage, $a_{\mathcal{A}_1}(t)$ is the characteristic

signal emanated only when $\mathcal{A}_1$ is executed, and $a^P_{\mathcal{A}_1}(t)$ is the signal emanated as a consequence of processing the activity throughout the pipeline excluding the execution stage. We define $\mathrm{ESP}(\mathcal{A}_1)$ as:

$$\mathrm{ESP}(\mathcal{A}_1) = \frac{\int_0^{\mathtt{T}^{\mathtt{P}}_{\mathcal{A}_1}} |a^P_{\mathcal{A}_1}(t)|^2 dt + \int_0^{\mathtt{T}_{\mathcal{A}_1}} |a_{\mathcal{A}_1}(t)|^2 dt}{R} \tag{4.6}$$

where we assume the activity $\mathcal{A}_1$ stays in the pipeline for the time interval $(0, \mathtt{T}_{\mathcal{A}_1} + \mathtt{T}^{\mathtt{P}}_{\mathcal{A}_1})$ only once, R is the resistance of the measuring instrument, and the execution step is the last step of the pipeline. Here, we need to emphasize that $a_{\mathcal{A}_1}(t)$ and $a^P_{\mathcal{A}_1}(t)$ are desired signals emanated while processing activity $\mathcal{A}_1$ through the pipeline only. They do not contain any components from any other signals and interrupts ideally. We also need to note that although we assume that the execution of an instruction happens at the very end of the pipeline, it is only for better illustration of the equation given in (4.6), and the execution could be done at any stage of a pipeline. We need to note that ESP provides the mean available power while executing an instruction, therefore, we assume that the noise term comprises all variations in the emanated power.

Although ESP is defined in continuous time domain, we have to alter this equation to cope with discrete time analysis since measurements are done on digital devices. Let assume sampling frequency of the measuring instrument is $f_s = 1/T_s$. We also assume that the number of samples taken during the execution of the instruction $\mathcal{A}_1$ is $N_I = \mathtt{T}_{\mathcal{A}_1}/T_s$, and the number of samples taken, when the instruction $\mathcal{A}_1$ is processed in a pipeline except for execution stage, is $P_S = \mathtt{T}^{\mathtt{P}}_{\mathcal{A}_1}/T_s$. Then, ESP in discrete time can be written as

$$\mathrm{ESP}[\mathcal{A}_1] = \frac{\sum_{m=0}^{P_S-1} |a^P_{\mathcal{A}_1}[m]|^2 + \sum_{m=0}^{N_I-1} |a_{\mathcal{A}_1}[m]|^2}{R/T_s}. \tag{4.7}$$

4.3.2 Estimating ESP From The Total Emanated EM Signal Power Created by a Program

Measuring ESP is not a trivial task. Execution of any instruction is overlapped with execution of other instruction in the code as well as other activities in the other stages of the pipeline. Therefore, we need a method to separate signal components that do not belong to the considered instruction from the desired signals related to a particular instruction. In [51], a program is designed to calculate the emanated energy difference between two instructions.

In this section, we modify the work in [51] to evaluate energy emanated by a single instruction. For ease of explanation, we show the code from [51] in Fig. 2.1. The code has two inner for-loops such that the first for-loop repeats the execution of Activity A, and the second for-loop repeats the execution of Activity B. Work in [51] shows that given the activities in the inner for-loops are non-identical, a spectral component at the alternation frequency, $f_{\text{alt}} = 1/T_{\text{alt}}$, is generated where T_{alt} is the one period of outer for-loop.

Instead of inserting two different activities into for-loops of the code, we insert instruction under observation in the first for-loop of the code, and NOP instruction into the second for-loop of the code. We note here that NOP instruction keeps the processor idle for one clock cycle. Hence, if the execution time of the activity in the first for-loop takes more than one clock cycle, the number of NOPs in the second for-loop has to be chosen carefully so that both loops take equal amount of time. In other words, the number of iterations of the first for-loop, `n_inst`, has to be equal to number of iterations of the second for-loop `n_inst2`=n_{inst}. Here, we assume the emitted signal power at all stages of a pipeline for NOP forms

the baseline that we use to normalize the power consumption of other instructions relative to NOP. Therefore, for the mathematical tractability of the derivations given in Appendix F, we assume that the signal measured while execution of NOP is a consequence of additive Gaussian white noise.

After running the modified code in [51] and measuring the power at the alternation frequency, the next step is to derive the relationship between the total emitted power and ESP. Let $s(t)$ be the emanated signal when the outer loop iterates for one time. We assume that the frequency content of $s(t)$ is negligible for the frequencies above $f_s/2$, and lasts for $\mathtt{T_E}$ seconds. Therefore, the total number of samples taken during the experiment is equal to $N_T = \mathtt{T_E}/T_s$. Let T_L be the execution time of any inner for-loop only for one period. Then, the number of samples taken in a period can be written as $N_L = T_L/T_s$. Therefore, the relationship between N_T and N_L becomes $N_T = 2 \times n_{inst} \times N_L$.

Now, let the power measured around this frequency be $\mathcal{P}_{\mathcal{A}_1}(f_{\text{alt}})$ while executing the code under the assumptions stated above. The following theorem gives the relationship between the total emanated signal power and the instruction power.

Theorem 2 (ESP). *Let $\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\text{alt}})$ be the normalized emanated power which is defined as*

$$\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\text{alt}}) = \mathcal{P}_{\mathcal{A}_1}(f_{\text{alt}}) - \mathcal{P}_{\text{NOP}}(f_{\text{alt}}) \tag{4.8}$$

where $\mathcal{P}_{\text{NOP}}(f_{\text{alt}})$ is the measured emanated power when both for-loops of the code are employed with NOP*. The mathematical relationship between* $\text{ESP}[\mathcal{A}_1]$ *and $\mathcal{P}_{\mathcal{A}_1}(f_{\text{alt}})$ while running the activity $\mathcal{A}_1$ in the first for-loop can be*

written as:

$$\mathrm{ESP}[\mathcal{A}_1] \;=\; \left(\frac{\pi}{2}\right)^2 \frac{\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\mathrm{alt}}) \cdot N_L}{(N_I + P_S) \cdot f_{\mathrm{alt}} \cdot n_{inst}}. \tag{4.9}$$

Proof. Please see Appendix F. □

4.4 Experimental Results and Information Leakage Analysis

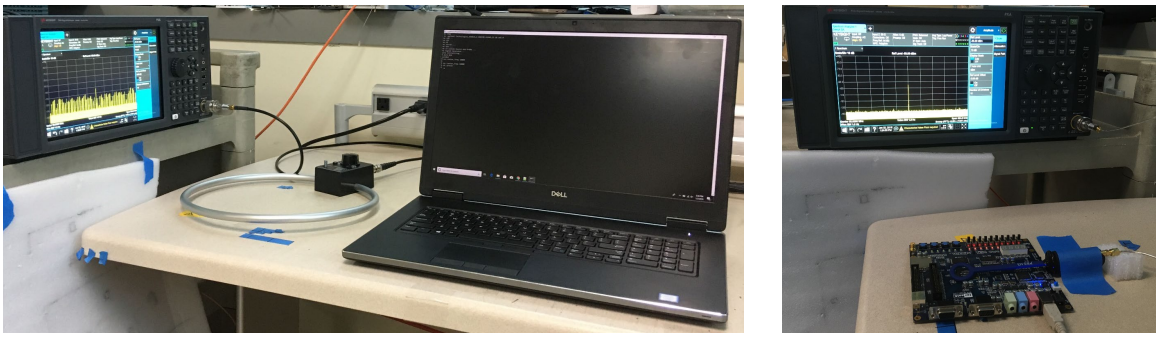


Figure 4.4: Measurement setups used in the experiments.

In this section, we provide the experimental results for emanated signal power of each instruction, and evaluate leakage capacity of various computer platforms.

The experimental setup is shown in Fig. 4.4. We used a spectrum analyzer (Agilent MXA N9020A), and magnetic loop probe (AAronia H field probe PBS-H3) for FPGA board and a magnetic loop antenna (AOR LA400) for other devices. We performed our measurements by setting the alternation frequency, f_{alt}, to 80 kHz. We keep the distance as close as possible to the processor since our goal is to reveal the input powers of the transmitter, i.e. ESP. The activities used in this section correspond to x86 instructions given in Fig. 2.1.

To obtain the experimental results, the steps we follow are:

- Run the program given in Fig. 2.1 as described in Section 4.3.2 to measure the available total signal power at the alternation frequency.

- Calculate ESP of each instruction for all available devices based on the equation given in (4.9).

- Transform the Markov Chain of instructions, and define the new constraints for the new model in terms of allowable paths as in Section 4.2.4.

- Define the signal to noise ratio (SNR) as:

$$\mathrm{SNR} = \frac{\sum_{i \in \mathcal{S}} \left(\mathrm{ESP}[i]\right)^2}{|\mathcal{S}| \times N_0/2} \tag{4.10}$$

  where $|\mathcal{S}|$ is the cardinality of instruction set $\mathcal{S}$.

- For a given SNR, run the algorithm given in [72] to obtain the stationary probabilities of each sub-state and corresponding leakage capacities.

- If the stationary probability of instructions is required, solve the following equations

$$\mu_i = \mathtt{L} \times \mathfrak{u}_1^i, \quad \forall i \in \mathcal{S} \tag{4.11}$$

  where μ_i is the stationary probability of i^{th} instruction for the original case, and $\mathfrak{u}_1^i$ is the initial sub-state of i^{th} instruction for the transformed scenario, and $\mathtt{L}$ is a constant which can be written as

$$\mathtt{L} = \left(\sum_{k \in \mathcal{S}} \mathfrak{u}_1^k \right)^{-1}. \tag{4.12}$$

- Define "Quantum" as the ratio between number of required clock cy-

cles to execute an instruction and minimum number of clock cycles to execute at least one instruction.

Please note that for some experiments, Quantum is equivalent to a clock cycle, but for some experiments, it can correspond to a couple of clock cycles. Additionally, abbreviations used in this section can be listed as follows:

- $\boldsymbol{C}_P$: Capacity in Bits/Quantum obtained with the proposed scheme.
- $\boldsymbol{C}_0$: Capacity in Bits/Instruction obtained by assuming execution time of all instruction takes only one clock cycle and using capacity definition given in (2.2). We also assume that the optimal stationary distribution for this capacity definition is denoted as $\boldsymbol{\mu}_0$.
- $\boldsymbol{C}_N$: Capacity in Bits/Quantum which is calculated as

$$\boldsymbol{C}_N = \frac{\boldsymbol{C}_0}{\sum\limits_{i \in \mathcal{S}} \boldsymbol{\mu}_0[i] L_i}. \tag{4.13}$$

  This capacity definition maps $\boldsymbol{C}_0$ into Bits/Quantum for a fair comparison.

- $\boldsymbol{C}_\infty$: Capacity in Bits/Quantum obtained by setting $\mathrm{SNR} = \infty$ and exploiting the proposed scheme to obtain the maximum possible leakage.

4.4.1 Experimental Results and Leakage Capacity for FPGA

This section presents the experimental results and leakage capacity for NIOS Processor on DE1 FPGA board. The ESP and corresponding execution length of each instruction are provided in Table 4.1. Please note that

length of an instruction means total execution time of each instruction in terms of Quantum.

Table 4.1: ESP values (in zJ) for DE1 FPGA board.

	LDM	**LDL1**	**DIV**	**ADD**	**SUB**	**MUL**
ESP	139.38	69.98	87.60	0.32	6.10	55.14
Length	7	4	5	1	1	4

In Fig. 4.5, we plot the leakage capacity for FPGA as a function of SNR. We observe that C_0 exceeds C_P because C_0 considers that each instruction takes only one clock cycle. However, if we normalize C_0 to obtain C_N, we can observe that applying traditional Shannon theory underestimates available leakage capacity and that proposed leakage capacity estimation C_P is needed to establish relationship between sequence of instructions as they pass through pipeline and leakage capacity.

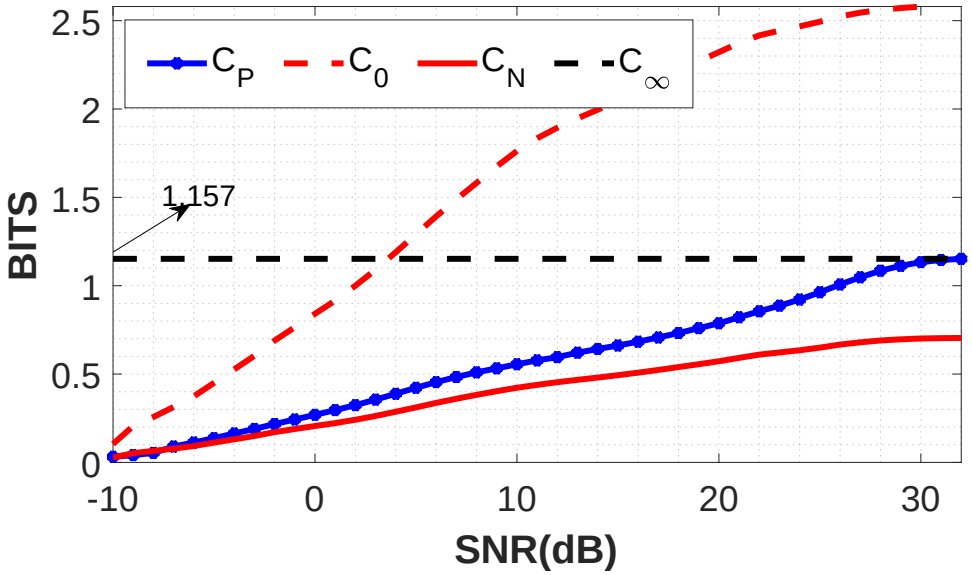


Figure 4.5: Leakage Capacity for NIOS Processor on the DEI FPGA.

Additionally, we observe that leakage capacity for SNR = 59.96 dB in [18] is 1.14 Bits/Quantum. Please note that the method in [18] does not allow for capacity calculation as a function of SNR. On the other hand,

with the proposed scheme, the estimated leakage capacity is higher and reaches 1.157 Bits/Quantum when SNR is around 30 dB. This result indicates that considering the pipeline depth and the dependence between instructions, which are not included in [18], more realistically estimates leakage capacity. We also note that the leakage capacity is high even for low SNR regimes allowing for transmission of thousands of bits per second because the clock frequencies of the current devices are high. Therefore, software and hardware designers need to consider side-channels and devise countermeasures to decrease side-channel leakages as much as possible.

4.4.2 Experimental Results and Leakage Capacity for AMD Turion X2 Laptop

This section provides the leakage capacity for a laptop with AMD Turion X2. It has 64 KB 2 way L1 Cache and 1024 KB 16 way L2 Cache. ESP values and execution lengths are given in Table 4.2.

Table 4.2: ESP values (in zJ) for AMD Turion X2 Laptop.

	LDL2	**LDM**	**STM**	**STL2**	**STL1**	**MUL**	**DIV**
ESP	150.08	84.66	64.74	188.17	0.49	0.21	7.26
Length	1	26	30	3	1	1	8

We need to note here that LDL1, ADD, and SUB are not included into our analysis because ESP values and execution lengths of these instructions are almost equal to STL1. Therefore, including these instructions does not affect overall leakage capacity. However, if we consider STL1, LDL1, ADD, and SUB as a sub-instruction set whose members are almost identical, STL1 could be thought as a representative of this set.

We observe that the deviation of the execution length of instructions is

much larger compared to FPGA. The effect of having such a deviation can be seen from Fig. 4.6 where the gap between C_0 and C_P is significantly larger. Additionally, the leakage capacity given in [18] is 0.97 Bits/Quantum when SNR is 23.78 dB, but the new proposed leakage capacity C_P shows that the leakage can be up to 1.36 Bits/Quantum for the same SNR region. This result indicates that all signals emanated from all stages of a pipeline carry some information, therefore, ignoring these signals can cause underestimation of the leakages.

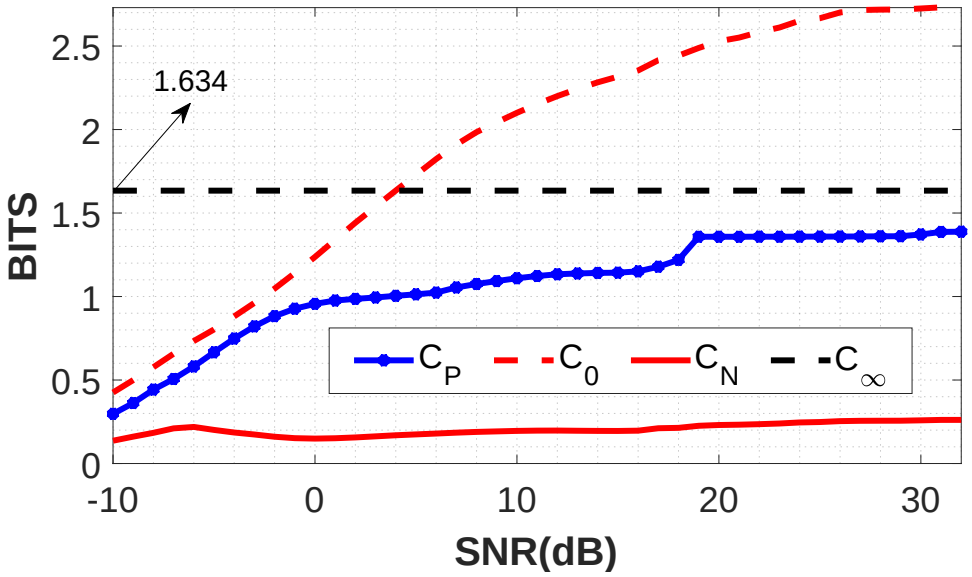


Figure 4.6: Leakage Capacity for AMD Turion X2 Laptop.

The results also show that the capacity of the laptop is moderately high even for low SNR regimes. For example, we observe that the leakage capacity of this system is approximately 1 Bits/Quantum around 0 dB SNR. Unfortunately, if the attacker is in very close proximity, and has the ideal decoder to reveal the secret information, C_P could raise up to 1.634 Bits/Quantum, which corresponds to $1.634*10^9$ bits/second for a processor with 1 GHz processor clock and all instructions taking one clock cycle. We also observe that C_P could not achieve the data rate of C_∞ in the given

SNR regime. For C_P to achieve maximum rate, it requires about 57 dB SNR. However, for the consistency among figures, we keep the considered SNR regime same for each plot.

4.4.3 Experimental Results and Leakage Capacity for Core 2 DUO Laptop

In this section, we provide the results for Core 2 DUO laptop. It has 1.8 GHz CPU clock, 32 KB 8 way L1 and 4096 KB 16 way L2 caches. ESP values and lengths of instructions are given in Table 4.3. Similar to AMD laptop, the deviation of the instruction length is large, which causes the capacity gap between the proposed and Shannon based methods to be larger.

Table 4.3: ESP values (in zJ) for Core 2 DUO Laptop.

	STL2	**LDM**	**STM**	**LDL2**	**LDL1**	**MUL**	**DIV**
ESP	422.16	181.58	79.94	320.48	0.75	0.06	7.02
Length	1	26	31	3	1	1	8

We do not consider the results for STL1, SUB and ADD because the lengths and ESP values of these instructions are almost same with LDL1. For this device, we assume that LDL1, STL1, SUB and ADD form a sub-instruction set, and LDL1 as the representative of this set. We observe that C_P can be up to 1.634 Bits/Quantum if the attacker can find a way to capture emanated signals with high SNR. Furthermore, at 23.82 dB SNR, C_P is 1.36 Bits/Quantum, again higher then 1.09 Bits/Quantum capacity predicted in [18]. The difference between these results reveals the importance of considering both pipeline depth and ordering of instructions.

We also observe that the required SNR for C_P to achieve C_∞ must be at least 56 dB. However, with a moderate gain antenna and proximity to

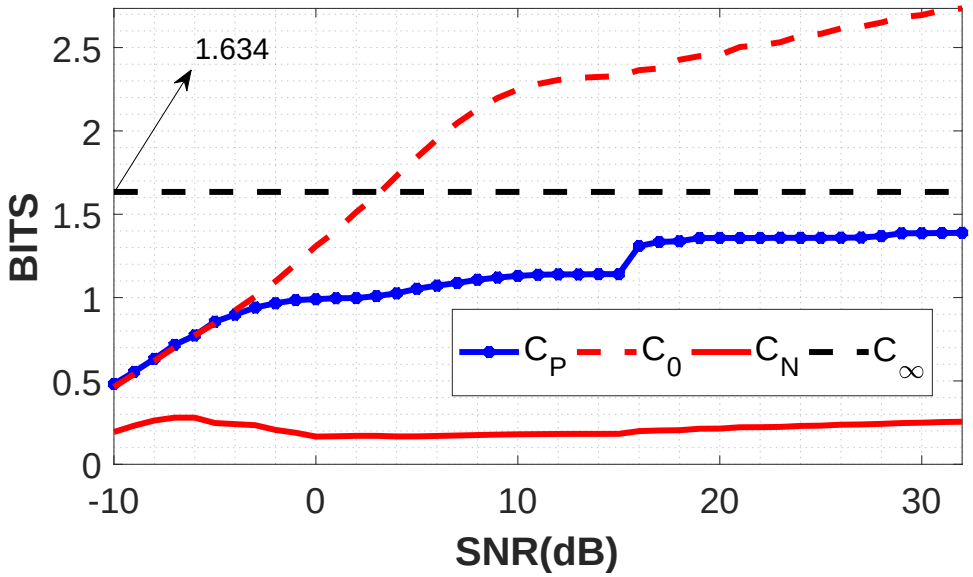


Figure 4.7: Leakage Capacity for Core 2 DUO Laptop

the laptop, the attacker can steal sensitive information since the leakage capacity is 0.5 Bits/Quantum when SNR is around -10dB. Considering the clock frequency of the computer, the side channel can have a transmission rate of thousand of bits per second under ideal circumstances.

4.4.4 Experimental Results and Leakage Capacity for Core I7 Laptop

The last example we provide is for Core I7 laptop which has 3.4 GHz CPU clock with 64 KB 2 way L1 Data and 1024 KB 16 way L2 caches. Table 4.4 provides ESP and execution length of each instruction. The first observation here is that the deviation of the execution length of instructions is not as large as the other laptops, which causes the gap between C_P and C_0 results to decrease as given in Fig. 4.8.

Table 4.4: ESP values (in aJ) for Core I7 Laptop.

	LDL2	**LDM**	**STM**	**STL2**	**SUB**	**STL1**	**ADD**	**MUL**	**DIV**
ESP	1.03	1.38	1.23	0.56	0.05	0.09	0.08	0.06	0.54
Length	1	12	15	4	1	1	1	1	8

We observe that LDL1 and SUB have approximately same ESP. There-

fore, SUB is considered as the representative of the group of these instructions. For ideal scenarios, C_P can go up to 2.32 Bits/Quantum. To achieve this rate, the setup must ensure at least 47 dB SNR. In addition, when SNR is 23.84 dB, the leakage capacity with the model in [18] is 0.72 Bits/Quantum, although it is obtained as 1.65 Bits/Quantum with the proposed model. Hence, including both pipeline depth and dependencies between instructions helps better quantification of leakage capacity.

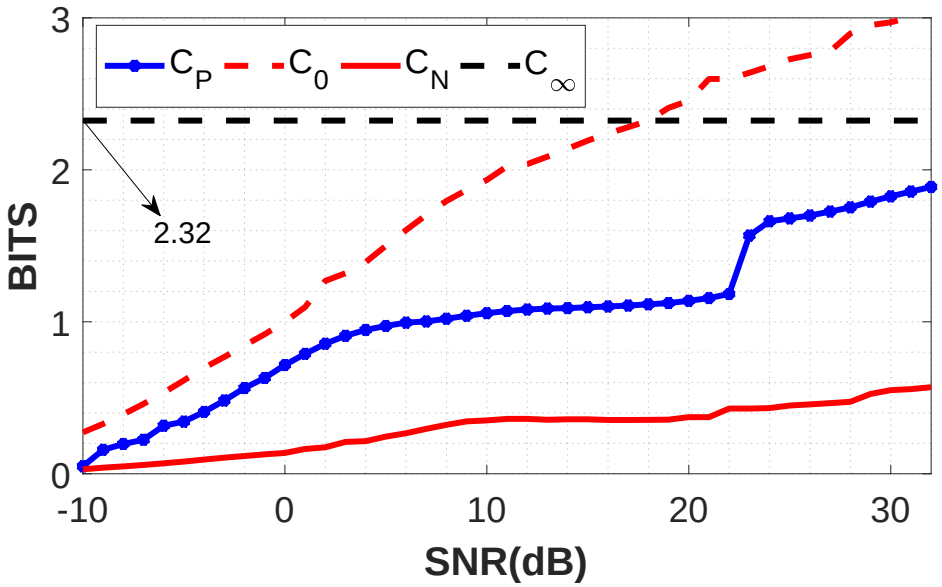


Figure 4.8: Leakage Capacity for Core I7 Laptop

Also, for the low SNR scenarios, C_P is high enough, i.e. 0.7 Bits/Quantum around 0 dB. Considering the clock frequency of the laptop, the capacity values given in Fig. 4.8 could be a messenger to warn any users about the possible vulnerabilities that computer systems might have.

Another evaluation methodology to assess the severity of side channels is given in [79]. They define success rate to demonstrate the performance of an adversary attack. It is possible to establish a connection between the success rate and the proposed information leakage. This connection is achieved if the probabilities which lead to maximum information leakages

are utilized to calculate the success rate. As a simple example, if the goal of an attack is to reconstruct instructions (although this chapter does not consider a specific attack, but aims to provide a universal upper bound for EM side channels), we can define the success rate as

$$\mathbf{Succ}^{sc-ir-1,\mathbf{I}}_{A_{\mathbf{I},\mathrm{ESP}}}(\mathbf{D},\sigma) = \sum_{i\in\mathbf{I}} \left[\mu[i] \int_{d_{i_L}}^{d_{i_U}} f(x|\mathrm{ESP}(i),\sigma)dx \right] \tag{4.14}$$

where **D** is the set of decision boundaries for all instructions, **I** is the set containing all considered instructions, d_{i_U} and d_{i_L} are the upper and lower decision boundaries for the i^{th} instruction, $f(x|\alpha,\sigma)$ is the pdf of white Gaussian noise distribution with mean α and standard deviation σ, $\mu[i]$ is the stationary probability of i^{th} instruction that is the result of the optimization problem given in (7). The decision boundaries are calculated based on ESP values of neighboring instructions. Therefore, if the target of an attack is known, it is possible to provide success rate of an attack by exploiting the parameters which optimize (7).

Welch's T-test is an evaluation methodology which is heavily exploited in the security assessment of cryptographic implementations against side channel attacks [80, 81, 82]. The proposed framework can be also associated with T-test assessment methodology if we assume there exists an attack which can separate emitted signals of different pipeline stages, and depends on the emitted signal power of individual instructions when the same instruction is executed successively. If these assumptions hold, the attacker will receive samples which will be the noise added version of emitted signal power while performing activity i, i.e, $y^i_j = \mathrm{ESP(i)} + noise$, where

y_j^i denotes the j^{th} successive execution of the instruction i. Let $\mathbf{y}_i$ be

$$\mathbf{y}_i = \left[y_1^i \; y_2^i \; \cdots \; y_{N_i}^i\right] \tag{4.15}$$

where N_i is the number of successive execution of the instruction i. Let also $\Delta_{m,n}$ be the T statistic of instruction m and n, which is given as

$$\Delta_{m,n} = \frac{\mathbb{E}\left(\mathbf{y}_m\right) - \mathbb{E}\left(\mathbf{y}_n\right)}{\sqrt{\frac{\text{var}(\mathbf{y}_m)}{N_m} + \frac{\text{var}(\mathbf{y}_n)}{N_n}}} \tag{4.16}$$

where $\mathbb{E}(\bullet)$ is the expectation operation, and $var(\bullet)$ gives the variance of its input. The T statistic for the instruction m will be significant only if $\Delta_{m,n}$ is above a threshold for any $n \in \mathbf{I}$. Therefore, the T statistic could be an empirical methodology, which can provide required repetition of an instruction for a successful side channel attack.

4.5 Utilizing the Proposed Framework for Security Assessment

The leakage capacity definition given in this section provides the maximum leakage amount that any EM side/covert channel can achieve on a given device. This capacity can help designers to predict possible vulnerabilities of their products at the design-stage and provide the opportunity to design countermeasures, or to redesign their systems to prevent possible side-channel attacks. Comparing with the evaluation method based on success rate, which quantifies accurate retrieval rate of an attack's target (i.e., secret key bit estimation), leakage capacity defines the maximum information leakage through side channels without specifying the attack itself. Therefore, it provides a universal upper bound for EM side channels. This section provides a recipe to check whether the considered system is

secure enough against side channel attacks, and explains steps to justify why they are required. The procedure for the assessment is given in Figure 4.9, and can be explained as follows:

- The first step is to collect emanated EM signal power available to an attacker while executing an instruction. Considering the clock frequency of modern computer systems, measuring the single instruction power could be problematic because of synchronization, complex pipeline structure, etc. To handle these problems, the designed microbenchmark given in Figure 4 is run to obtain both $\mathcal{P}_{\mathcal{A}_1}(f_{\text{alt}})$ and $\mathcal{P}_{\text{NOP}}(f_{\text{alt}})$ where $\mathcal{P}_i(f_{\text{alt}})$ is the total emanated signal power when instruction i and NOP are inserted into the first and second inner-for-loops, respectively.

- The measurements to obtain $\mathcal{P}_i(f_{\text{alt}})$ are done from near-field because the goal is to capture all emanated signal as much as possible. This approach helps to have close empirical results for signal power because actual emanated instruction power is not available. Then, ESP of each considered instruction is calculated based on the formula given in (12).

- Because of the functionality of a program, a script, etc., and the complex pipeline structure of modern computer systems, instructions shows dependency to each other. To consider the dependency among instructions, a Markov Model is created as given in Section II.

- The next step is to calculate the limit for information leakage. To obtain the limit, the algorithm given in [72] will be exploited. For the algorithm, it is required that the channel inputs from each source have to last for the same amount of time. However, instructions can

take different number of clock cycles to execute. Therefore, the Quantum length of each instruction has to be revealed. Please note that the Quantum length is defined as the ratio between actual execution time of an instruction and the minimum execution time within instruction set.

- After having the execution length and utilizing the duality given in Theorem 1, the next step is to apply the transformation in Section II-E. This transformation makes sure that each channel input takes same amount of time so that the algorithm given in [72] can be utilized to calculate the leakage capacity for a targeted SNR regime.

- The result of the algorithm provides the leakage capacity which is denoted as C_P. We use this number later as the baseline to compare with the leakages of designs to understand the relative resistance of them against any possible side channel attack.

- To find the leakage of any code, program, design, etc., the number of transitions from i^{th} to j^{th} instructions is counted. These numbers are normalized to calculate P_{ij} for the inspected source code.

- Having the state transition probabilities, P_{ij}, our next goal is to reveal the available mutual information for the test code. Please note that our goal is to find the mutual information with the given P_{ij}, therefore, we do not need to update the state transition probabilities. Hence, we run the algorithm given in [72] only once without updating the state transition probabilities. The mutual information obtained as a result of the algorithm is denoted as M_C.

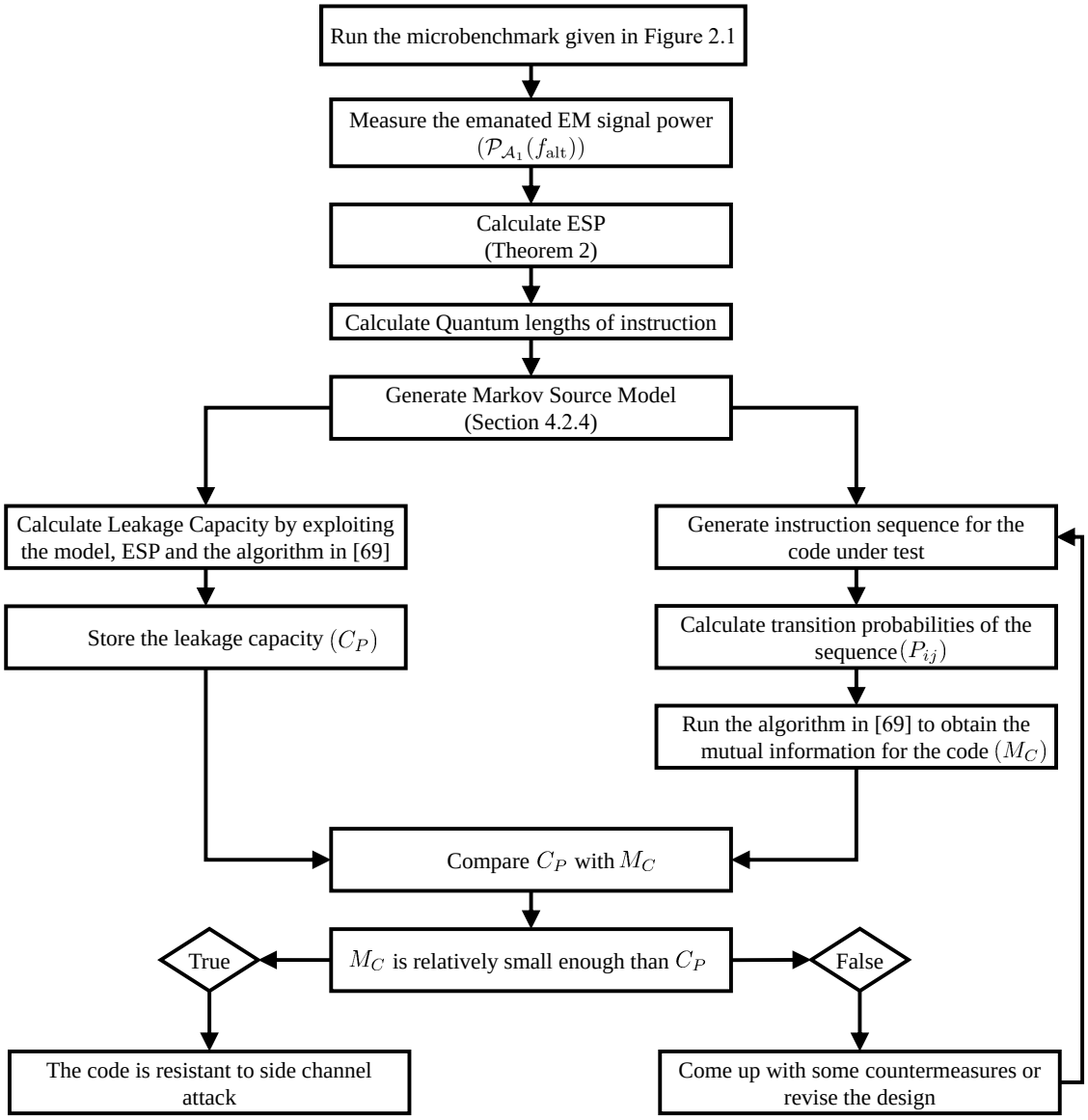


Figure 4.9: The methodology to assess information leakage.

- As the last step, we compare C_P with M_C. If M_C is much smaller then C_P, and very close to zero, the designer can conclude that the source code is secure. Otherwise, a new design or some countermeasures, i.e. shielding, etc., has to be considered. Then, the same steps given in this section have to be followed again until achieving $M_C \ll C_P$.

Please note that the procedure given here does not specify the attack methodology, but provides the worst case scenario for a victim in terms of information leakage. It is still an ongoing research to have an attack that achieve the limits given in this chapter. However, designers can utilize the procedure to prevent any future attacks.

4.6 Summary

This chapter proposed a methodology to relate program execution to EM side-channel emanations and estimate side-channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor. To model dependence between program instructions in a code, we have proposed to use Markov Source model, which includes the dependencies that exist in instruction sequence since each program code is written systematically to perform a specific task. The sources for channel inputs are considered as the emitted EM signals during instruction executions. To obtain the channel inputs for the proposed model, we derive a mathematical relationship between the emanated IP and total emanated signal power while running a program. Then, we have derived leakage capacity of EM side channels created by execution of series of instructions in a processor. Finally, we have provided experimental results to demonstrate that leakages could be severe enough for a dedicated attacker to obtain some prominent information.

CHAPTER 5

COMMUNICATION MODEL AND CAPACITY LIMITS OF COVERT CHANNELS CREATED BY SOFTWARE ACTIVITIES

5.1 Overview

In this chapter, we model an electromagnetic covert channel as a communication channel and derive upper and lower capacity bounds [21]. Covert channels can be used to communicate sensitive data between two processes inside a processor - typically a *privileged* process that has access to secret data but no/limited access to the outside world and a *non-privileged* process with no access to the data but connected to the outside world, or alternatively, they can be used to "exfiltrate" data from an air-gapped computer which is physically and logically separated from public networks [4, 56]. In both cases, the secret data can be *secretly* transferred to the outside world through a *wireless channel* which, in turn, breaks the existing assumptions about the security of sensitive data inside a system.

Covert channels are considered as a serious security threat [2] since they can circumvent and break existing defense mechanisms (e.g., memory isolation, partitioning, etc.) for protecting secrets inside a computer. Typically, wireless communication is a carefully designed process that encompasses the coordinated design of transmitter and receiver and usually, the transmitted and received signals are well-synchronized. In contrast, covert channels lack these characteristics. Moreover, contrary to most communication systems, which are designed to avoid symbol loss and/or insertion with little or no overhead, covert channels are not designed to transfer

information at all and their transmission is often corrupted by insertion, deletion, and erroneous transfer of bits. While there is a large number of papers discussing bounds on the capacity of channels corrupted with synchronization errors [10], [11], [12], [13], [14] and more recently, papers discussing bounds on the capacity of channels corrupted with synchronization and substitution errors [15], [16], none of them provide bounds for the capacity of the wireless covert channel which can be modeled as a cascaded insertion-substitution channel that suffers from random pulse position shifts, and that insertions occur with different probabilities for zero and one.

Similar to traditional wireless communications, some errors in the covert channel occur due to variations in the propagation environment. However, in addition to channel errors, the software activity "transmitter" lacks precise synchronization, causing jitter that reduces the signal's effective bandwidth *and* increases the noise level. Also, the "transmitter" gets interrupted with other (system) activities, and the transmitted signal may go through a channel obstructed by metal, plastic, etc. To capture all effects of the observed behavior, we have modeled the transmitted sequence as a pulse amplitude modulated (PAM) signal with randomly varying pulse positions. In summary, the main contributions of the chapter can be listed as follows:

- Model the transmitted signal as a pulse amplitude modulated signal with random pulse positions to avoid problems due to variation in the execution time of computer activities,
- Model the covert channel as an insertion channel to consider other activities such as interrupts, stalls, etc.,

- Derive power spectral density and the bit error rate (BER) of the transmitted signal with only substitution errors,
- Derive capacity bounds with random insertion and substitution due to the noise and jitter errors,
- Provide a receiver design that can correctly detect the computer-activity-created signals,
- Perform experiments with high clock speed devices at some distance.

The organization of the chapter is as follows: Section 5.3 describes the transmission model for a covert channel caused by EM emanations of the processor, Section 5.4 explains reception model and BER for two case studies, Section 5.5 derives lower and upper bounds of the covert channel capacity, Section 5.6 presents the experimental results to validate the proposed framework, and Section 5.7 summarizes the chapter.

5.2 Wireless Transmission via Covert Channels

Even an idle computer system produces RF signals that, after AM demodulation, result in clicking, whining, and other sounds (this can be confirmed by placing an AM radio receiver close to a computer). To confirm that the received communication sequence is indeed the transmitted message, we performed two experiments. First, we modulated our transmitted signal with the A5 note (880 Hz), and turned this tone on/off to transmit Morse code for "All your data belong to us" [3]. Second, we placed our micro-benchmark code around the keyboard driver, which allowed us to transmit keystrokes wirelessly [83]. In both cases, we were able to correctly receive and demodulate transmitted signals.

However, we observed that the timing of the instructions was not perfectly synchronized. This issue confirmed that the baseband pulses generated with on-off keying do not have equal timing, and the created carrier is spread over several kilohertz in contrast to traditional communications where the carrier is well concentrated around a single frequency. This lack of synchronization in the transmitter causes significant jitter and has to be carefully modeled, as described in the following sections.

5.3 Transmission Model for Software-Activity-Created Signals

In this section, we propose a model for covert channel communication systems. Before introducing the proposed model, we briefly review the baseband PAM signal and corresponding notations used in the rest of the chapter.

The baseband PAM signal with a period of $\mathcal{T}$ can be written as [84]

$$x_p(t) = \sum_k x_k \delta(t - k\mathcal{T}) * p(t), \tag{5.1}$$

where $\delta(\bullet)$ is Dirac delta function, $*$ is the convolution operator, $\mathbf{x}_k = (x_k, x_{k-1}, x_{k-2}, \ldots)$ is the sequence of data symbols that are chosen from a finite alphabet, and $p(t)$ is a shaping pulse. The power spectral density (PSD) of $x_p(t)$ can be written as [84]

$$S_{xp}(f) = \frac{|P(f)|^2}{\mathcal{T}} S_x(f), \tag{5.2}$$

where $P(f)$ is the Fourier transform of the shaping pulse,

$$S_x(f) = \sum_{k=-\infty}^{\infty} R_x[k] e^{-j2\pi f k\mathcal{T}} \tag{5.3}$$

is PSD of the stationary sequence x_k, and $R_x[k]$ is the autocorrelation function of sequence x_k. Furthermore, if an impulse function is used as the shaping pulse, the power spectral density can be simply written as $S_{xp}(f) = S_x(f)/\mathcal{T}$.

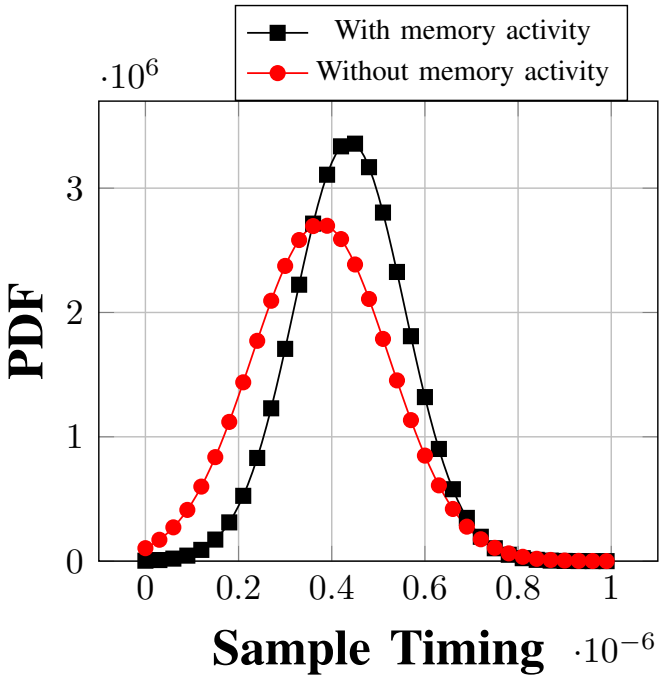


Figure 5.1: Illustration of two timing distributions of symbols for an EM covert channel, one when memory activity is used and one with on-chip instructions is used.

The baseband signal shown in (5.1) assumes perfect symbol timing. However, the transmitted signals created by computer software activities are exposed to synchronization problems due to variations in symbol timing. As an example, Fig. 5.1 illustrates how the mean and the variance of the symbol timing vary with software activities when a covert channel is created based on emanated EM signals [3]. While the on-chip activities have a more concentrated distribution with a smaller variation, off-chip activities such as memory create more variations in symbol timing.

To deal with the pulse width variations and establish a connection us-

ing conventional communication theories, we assume the pulse width is fixed, but the center of the pulse changes due to the non-synchronous nature of the channel. Therefore, we propose to model the baseband signal as a pulse amplitude modulated (PAM) signal with a random pulse position. Then, the baseband received signal can be written as

$$y_p(t) = \sum_k x_k p(t - k\mathcal{T} - \mathbf{T_k}), \tag{5.4}$$

where $\mathbf{T}_k$ is a random shift associated with a particular pulse for the transmitted symbol, x_k, whose probability density function (pdf) is denoted by $f_{\mathbf{T}_k}(t_k)$. As illustrated in Fig. 2.4, the pulses are assumed to have a 50% duty cycle and the neighboring pulses do not overlap. Here, we need to note that although symbol timing varies as given in Fig. 5.1, our model contains a pulse function whose width is fixed and whose position is randomized. Also, the position of the pulse is chosen as the mid-point of the actual pulse width. This can be explained as follows: in traditional PAM modulation, pulse duty cycles are assumed to be fixed and do not vary. However, in a covert communication system, duty cycles generated by software activities can vary from one execution to another. Hence, to capture the variation in symbol timing in software activities and link the covert communication with the existing approaches, the transmitted signal is modeled as a PAM signal with a fixed pulse duty cycle.

To ensure that neighboring pulses do not overlap, the support set of $f_{\mathbf{T}_k}(t_k)$ is set to $\{t_k \in [-\mathcal{T}/4, \mathcal{T}/4)\}$. We also assume probability density functions,

$$\{f_{\mathbf{T}_k}(\bullet) \mid \forall k \in \{-\infty, \infty\}\},$$

are identical and independent distributions (i.i.d.). As an example, Fig. 5.2

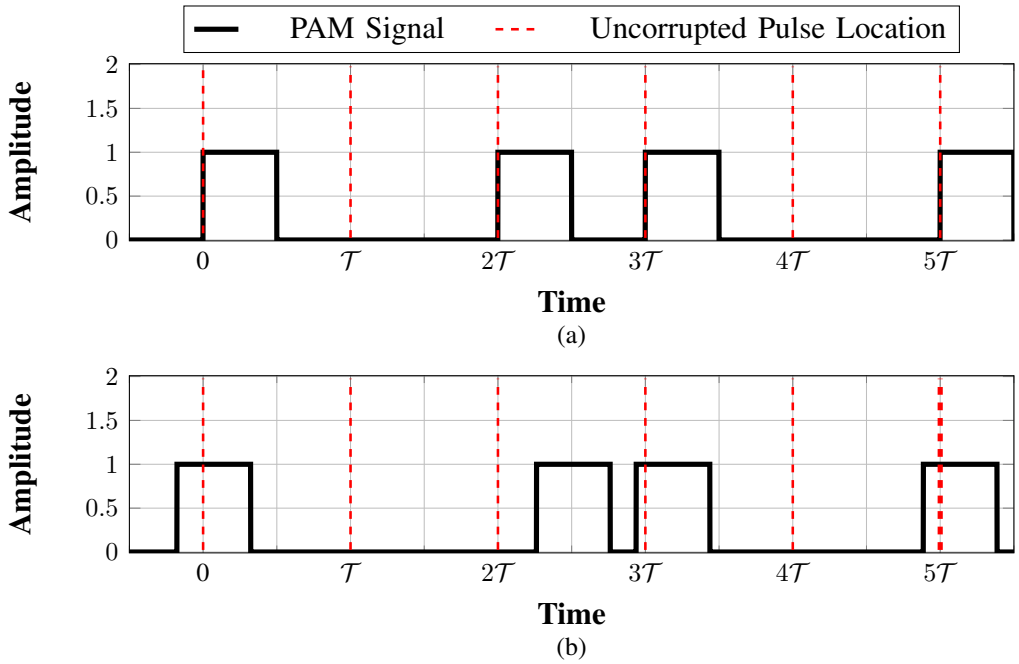


Figure 5.2: (a) PAM with sequence x_k and (b) distribution of pulses perturbed randomly in time and modulated in amplitude when the shaping pulse is a square wave.

illustrates a typical PAM signal with 50% duty cycle and its randomly shifted version. We can observe that the time difference between neighboring pulses can increase or decrease, which mimics the variations in software activities and reflects the lack of synchronization.

To simplify (5.4) further, we will assume that $p(t)$ is an impulse function, $\delta(t)$, and the modulated baseband signal can be written as

$$y(t) = \sum_k x_k \delta(t - k\mathcal{T} - \mathbf{T_k}). \tag{5.5}$$

To evaluate the impact which the jitter introduces to the system due to the variation in symbol timing, we need to find PSD of baseband PAM signal with a random pulse position. The following theorem provides PSD of the signal with a random pulse position:

Theorem 3. *Let $\Phi(f)$ be the Fourier transform of $\phi(\tau)$ and*

$$\phi(\tau) = \int f_{\mathbf{T}}(\tau + t) f_{\mathbf{T}}(t) dt = f_{\mathbf{T}}(\tau) * f_{\mathbf{T}}(-\tau), \tag{5.6}$$

where the subscript k is removed to represent the random position distribution of the k^{th} pulse since all distributions are assumed to be i.i.d. Then, PSD of the received signal, $y(t)$, in (5.5) can be written as

$$S_y(f) = \frac{1}{\mathcal{T}} S_x(f) \Phi(f) + \frac{R_x[0]}{\mathcal{T}} (1 - \Phi(f)). \tag{5.7}$$

Proof. Please see Appendix H. □

Furthermore, for an arbitrary pulse shape, PSD of PAM signal with a random pulse position becomes

$$S_{y_p}(f) = \frac{|P(f)|^2}{\mathcal{T}} \left(S_x(f) \Phi(f) + R_x[0] \Big(1 - \Phi(f) \Big) \right). \tag{5.8}$$

This result shows that PAM with a random pulse position is equivalent to passing the PAM signal through a filter with power spectral density $\Phi(f)$, and having a jitter noise whose power is redistributed as a continuous wideband noise.

The characteristics of the filter and the noise are completely determined by the probability distribution of the pulse positions. By fitting the measured samples of symbol duration into different probability distributions, we have found that the best fit for the pulse position variations is a Gaussian distribution with the mean μ and the standard deviation σ. Although fitting Gaussian distribution for the pulse positions contradicts our previous assumption that pulses can be only between $-\mathcal{T}/4$ and $\mathcal{T}/4$, we can still approximate the pulse positions with a Gaussian random distribu-

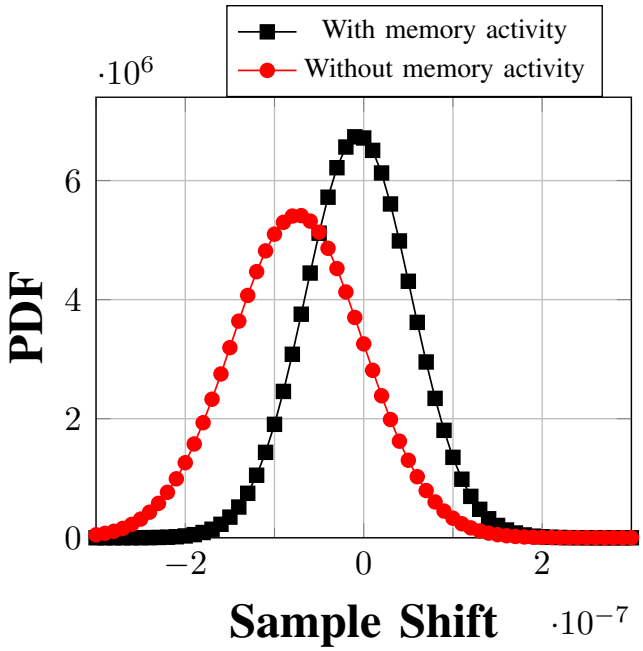


Figure 5.3: Illustration of two distributions of pulse shift for an EM covert channel, one when memory activity is used and one with on-chip instructions is used.

tion by assuming that the tail probability beyond $-\mathcal{T}/4$ and $\mathcal{T}/4$ is almost zero. Moreover, for the tractability of the derivations, we will assume that the means of these Gaussian distributions are equal. Fig. 5.3 plots the shift distributions of these pulse positions. We can observe that the pulse shift distributions are concentrated around zero. Therefore, (5.6) can be specified further for the memory and non-memory activities.

Given that Gaussian distribution has a Fourier transform

$$\mathfrak{F}\{f(t)\} = e^{-2\pi j f \mu} e^{-2\pi^2 \sigma^2 f^2}, \tag{5.9}$$

where $\mathfrak{F}\{\bullet\}$ takes Fourier transform of its argument and $f(t)$ is any Gaussian distribution with mean μ and standard deviation σ, we then combine

(5.9) with (5.6) and obtain

$$\Phi(f) = e^{-2\pi^2 f^2(\sqrt{2}\sigma)^2}. \tag{5.10}$$

Finally, inserting (5.10) into (5.7), we calculate PSD of PAM signal with Gaussian distributed pulse positions as

$$\begin{aligned} S_y(f) &= \frac{1}{\mathcal{T}} S_x(f) e^{-2\pi^2 f^2(\sqrt{2}\sigma)^2} \\ &\quad + \frac{R_x(0)}{\mathcal{T}} \left(1 - e^{-2\pi^2 f^2(\sqrt{2}\sigma)^2}\right) \\ &= S_{xt}(f) + S_{nt}(f), \end{aligned} \tag{5.11}$$

where $S_{xt}(f)$ denotes the spectrum of the transmitted sequence and $S_{nt}(f)$ denotes the noise spectrum due to random pulse position.

5.4 Quantifying the Information Leakage of Covert Channel Software-Activity-Created Signals

Traditionally, the performance of a communication system is evaluated by estimating symbol error rate or BER of the system. The error probability of pulse amplitude modulated signal (PAM) can be written as [84]

$$P_{PAM} = Q\left(\sqrt{\frac{P_s}{2P_n}}\right), \tag{5.12}$$

where $Q(\cdot)$ function denotes the tail probability of the standard normal distribution, P_s is the averaged transmitted power of a symbol, and P_n is the averaged noise power.

For the proposed scenario, BER represents the severity of the covert channel. Quantifying the information leakage in terms of BER reveals how fast we can transmit the information by establishing a reliable communi-

cation link using the computer systems. To estimate BER, we need PSD of the signal derived in (5.11). Then, we assume that the transmitted signal, $y(t)$, defined in (5.5) has been affected by the channel noise, and that received signal can be written as

$$r(t) = y(t) + n(t), \tag{5.13}$$

where $n(t)$ denotes the white Gaussian noise with zero mean and standard deviation, σ_n. We also assume that the noise and the transmitted sequence are independent. By observing that a communication system based on the covert channels described in Section 5.2 typically occurs at low frequencies ($\sim$ 1 MHz) where the multi-path effect does not play a significant role, it is reasonable to assume that the received signal is mostly impacted by noise and that inter-symbol interference (ISI) has almost negligible effect on the reliability of the covert communication. Here, we assume that the noise component contains both additive channel noise and all corruptive signals due to other activities in the system. Then, utilizing (5.11), the power spectral density of the received signal can be written as

$$S_r(f) = S_{xt}(f) + S_{nt}(f) + N_0/2, \tag{5.14}$$

where $N_0/2$ denotes the power spectral density of the additive white noise.

Obtaining PSD of the transmitted symbols facilitates the calculation of BER. To utilize (5.12), we need to know the signal and noise powers. Since our transmitted signal experiences jitter due to the variations in the symbol position, we start by calculating PSD of the jitter noise and the signal.

Corollary. *Considering the variation in the symbol position, PSD of the*

transmitted sequence for on-off keying (OOK) is given as

$$\begin{aligned} S_y(f) &= \frac{R_x[0]}{\mathcal{T}}\left(\bar{S}_{xt}(f)+\bar{S}_{nt}(f)\right) \\ &= \frac{R_x[0]}{\mathcal{T}}\left(\underbrace{\left(\frac{1}{2}+\frac{1}{2\mathcal{T}}\sum_m \delta(f-m/\mathcal{T})\right)\Phi(f)}_{\bar{S}_{xt}(f)}+\underbrace{(1-\Phi(f))}_{\bar{S}_{nt}(f)}\right) \end{aligned} \quad (5.15)$$

where $\bar{S}_{xt}(f)$ and $\bar{S}_{nt}(f)$ are the normalized signal and jitter noise powers.

Proof. Please see Appendix H.1. □

Fig. 5.4 illustrates the behavior of the normalized signal and noise power when $\mathcal{T} \approx 15\sigma$.

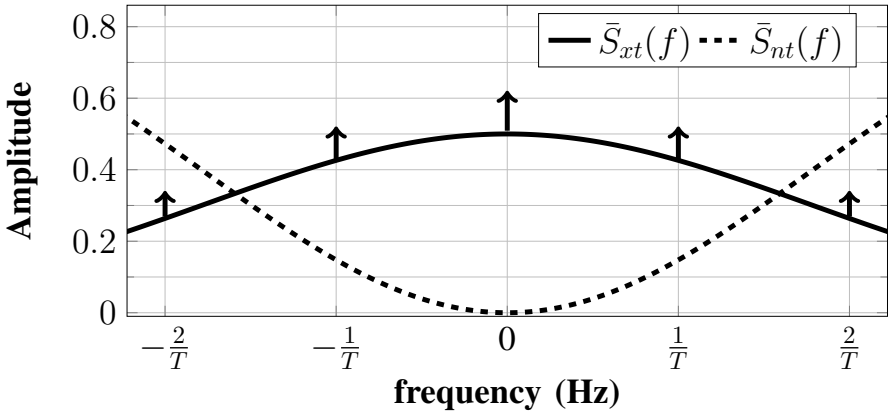


Figure 5.4: PSD of normalized signal and jitter noise due to random pulse position when $\mathcal{T} \approx 15\sigma$.

Since the noise power due to jitter behaves like a white noise when $\Phi(f) \approx 0,$ at the receiver front-end, we employ a low-pass filter whose bandwidth and magnitude are $1/2T$ and 1, respectively. The total signal power can be obtained as

$$\begin{aligned} P_{st} &= \frac{R_x[0]}{\mathcal{T}} \cdot \int_{-\frac{1}{2\mathcal{T}}}^{\frac{1}{2\mathcal{T}}} \bar{S}_{xt}(f)df \\ &= \frac{R_x[0]}{\mathcal{T}} \cdot \left(\frac{1}{2\mathcal{T}}+\frac{\sqrt{\pi}}{4\mathcal{T}}\mathrm{erf}\left(\pi\sigma/\mathcal{T}\right)/(\pi\sigma/\mathcal{T})\right), \end{aligned} \quad (5.16)$$

and total noise power due to jitter is equivalent to

$$\begin{aligned} P_{nt} &= \frac{R_x[0]}{\mathcal{T}} \cdot \int_{-\frac{1}{2\mathcal{T}}}^{\frac{1}{2\mathcal{T}}} \bar{S}_{nt}(f)df \\ &= \frac{R_x[0]}{\mathcal{T}} \cdot \left(\frac{1}{\mathcal{T}} - \frac{\sqrt{\pi}}{2\mathcal{T}} \mathrm{erf}\left(\pi\sigma/\mathcal{T}\right)/(\pi\sigma/\mathcal{T}) \right), \end{aligned} \tag{5.17}$$

where erf(•) is the error function.

Having the power for both jitter noise and the received signal, we can estimate BER by using (5.12) assuming we have a channel without synchronization problems. However, this is not the case for a software-activity-based covert channels because synchronization can hurt the stealthy nature of these covert channels. Fortunately, after filtering the received signal at the receiver side, the jitter power becomes flat and behaves like an extra power source for the channel noise. Therefore, the receiver sees the channel noise with power $\hat{N}_0/2 = N_0/2 + \mathcal{T}P_{nt}$. With that approximation, we can treat our communication system as a synchronized system with extra channel noise power. Hence, BER for the system can be approximated as

$$\mathrm{BER} = Q\left(\sqrt{\frac{P_{xt}/\mathcal{T}}{\hat{N}_0}} \right). \tag{5.18}$$

The effect of varying jitter noise on BER is given in Fig. 5.5 where SNR_i is defined as

$$\mathrm{SNR}_i = R_x[0]/(N_0/2). \tag{5.19}$$

As the power of additive channel noise increases, the effect of the lack of synchronization on the erroneous transfer of bits becomes negligible. However, while the channel noise power decreases, the impact of jitter

noise can be observed explicitly. In Section 5.6, we will demonstrate that assuming the jitter as an another source for the channel additive noise is a proper assumption to model the characteristics of BER of a covert channel.

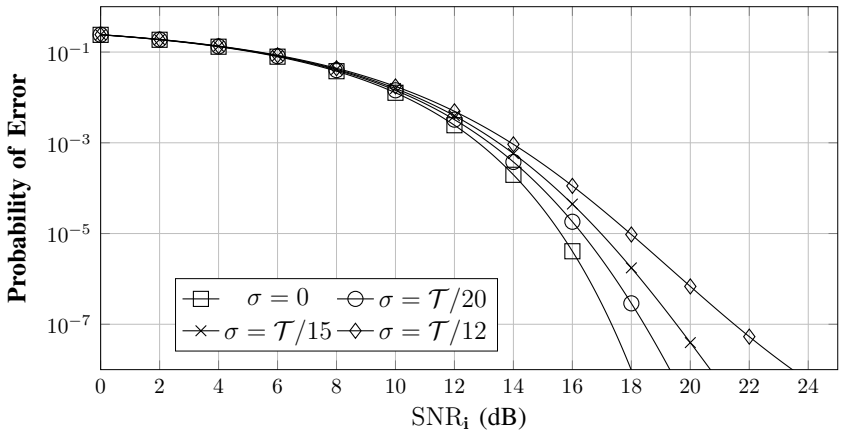


Figure 5.5: BER for the covert wireless communication system with varying jitter noise power.

Finally, we investigate how the variation in the symbol position corrupts the transmitted sequence. The signal to jitter noise power ratio, SNR_{jitter}, at the transmitter side, due to random pulse positioning, can be written as

$$\mathrm{SNR}_{jitter} = \frac{P_{st}}{P_{nt}} = \frac{\frac{1}{2} + \frac{\sqrt{\pi}}{4}\mathrm{erf}\left(\pi\sigma/\mathcal{T}\right)/(\pi\sigma/\mathcal{T})}{1 - \frac{\sqrt{\pi}}{2}\mathrm{erf}\left(\pi\sigma/\mathcal{T}\right)/(\pi\sigma/\mathcal{T})}. \tag{5.20}$$

Fig. 5.6 depicts how SNR_{jitter} changes with respect to $\sigma/\mathcal{T}$. Since we assume $12\sigma \leq \mathcal{T}$, we limit $\sigma/\mathcal{T}$ to be between 0 and $1/12$ (due to the assumption that the distribution of the pulse shift has non-zero probability in the region $[-\mathcal{T}/4, \mathcal{T}/4)$ and considering three-sigma rule). As expected, as the variation in pulse position decreases, the distortion in the transmitted signal, due to jitter noise, decreases.

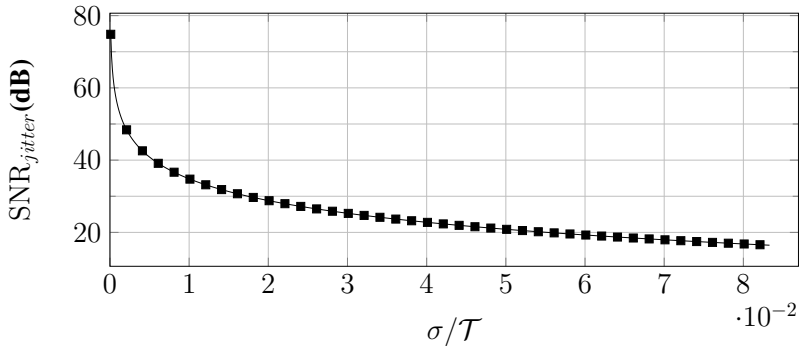


Figure 5.6: SNR_{jitter} vs. $\sigma/\mathcal{T}$.

5.5 Capacity of the Covert Channel Created By a Computer Software Activity

The conventional method to assess the capacity of a communication system over Gaussian channels is to employ Shannon's capacity definition [84]. However, in addition to errors due to the Gaussian channel and pulse position, covert communication channels are exposed to insertion errors due to random software activities. We assume covert transmission occurs continuously, and the insertions are due to processor optimization, stalls, cache misses, queues, etc. These activities in a computer system can stall the covert channel communication and produce unintended signals. The receiver interprets these signals as ones or zeros, and these received bits are considered as the inserted symbols of the covert wireless communication. To model this covert channel, we assume binary discrete memoryless channel for the random insertions, as illustrated in Fig. 5.7. The channel parameters are (p_{i0}, p_{i1}, p_e), where p_{i0} denotes the probability that the random inserted symbol is 0, p_{i1} denotes the probability that the random inserted symbol is 1, and p_e denotes the probability of substitution error during transmission due to channel noise and jitter. To be able to calculate p_e, we follow the procedure given in Section 5.4, where we assume the jitter behaves like another power source for the additive channel noise.

This model is a modified version of the channel model used by Davey and MacKay [11], where we additionally account for the fact that symbols for zero and one do not have equal probabilities of insertion, and the channel is noisy and jittery.

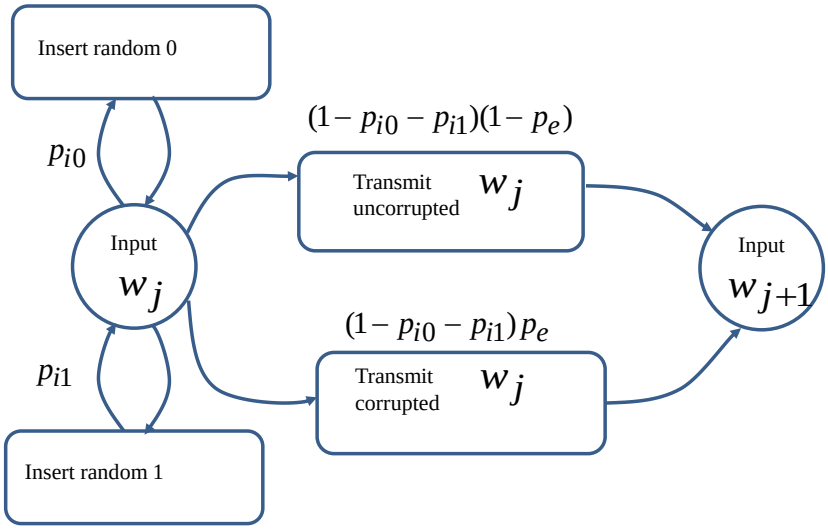


Figure 5.7: Binary discrete memoryless noisy, jittery, synchronization channel.

With the model and the parameters defined above, the following theorem provides the upper and lower bounds of the covert channels which exhibit an OOK structure:

Theorem 4. *The covert channel capacity with probabilities* (p_{i0}, p_{i1}, p_e) *and* $(p_{i0} + p_{i1}) < 0.5$ *is upper bounded by*

$$C \leq 1 - H_b(p_e), \tag{5.21}$$

and lower bounded by

$$C \geq \max\left(0, \frac{1 - H_b(p_e) - H_b(p_{i0} + p_{i1})}{1 - p_{i0} - p_{i1}}\right), \tag{5.22}$$

where $H_b(\bullet)$ *is the entropy of a binary source.*

Proof. Please see Appendix I. □

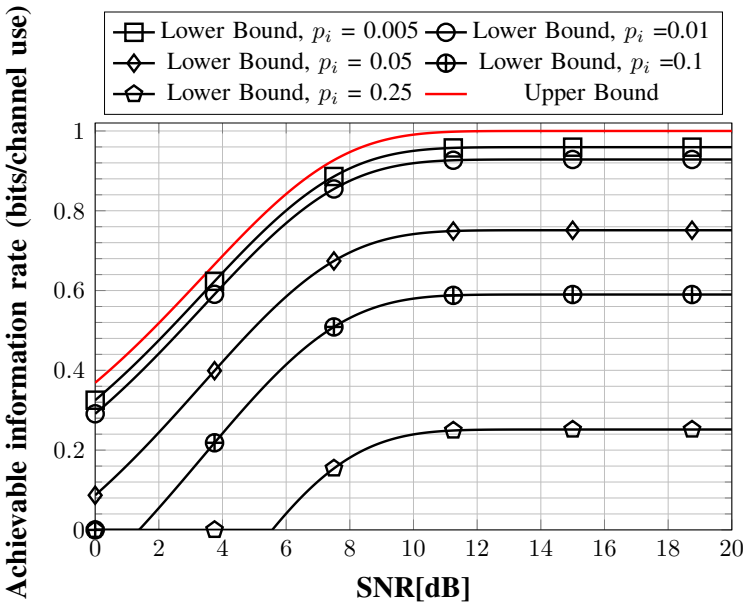


Figure 5.8: Upper and lower bounds of information rates for deliberate side channel with several probabilities of insertion for a synchronized channel.

Fig. 5.8 shows achievable information rates for the covert channel with various insertion probabilities. Here, we plot the upper bound derived in (5.21) and lower bound in (5.22) for several different insertion probabilities. We can observe that unless $p_{i0} = p_{i1} = p_i = 0$, the existence of insertions greatly limits the transmission capabilities, and reliable communication is not possible without channel coding even when SNR is high.

Moreover, Fig. 5.9 compares the lower bound derived in (5.22) with a prior method [15] for several insertion probabilities. We observe that when the system is in the proper SNR regime for a reliable communication, the proposed lower bound is tighter than the one given in [15] for the capacity of a channel with insertion and substitution. We compare our results with work in [15] because it is the most similar channel scenario to the covert channels analyzed in this paper and their lower bound is shown to be

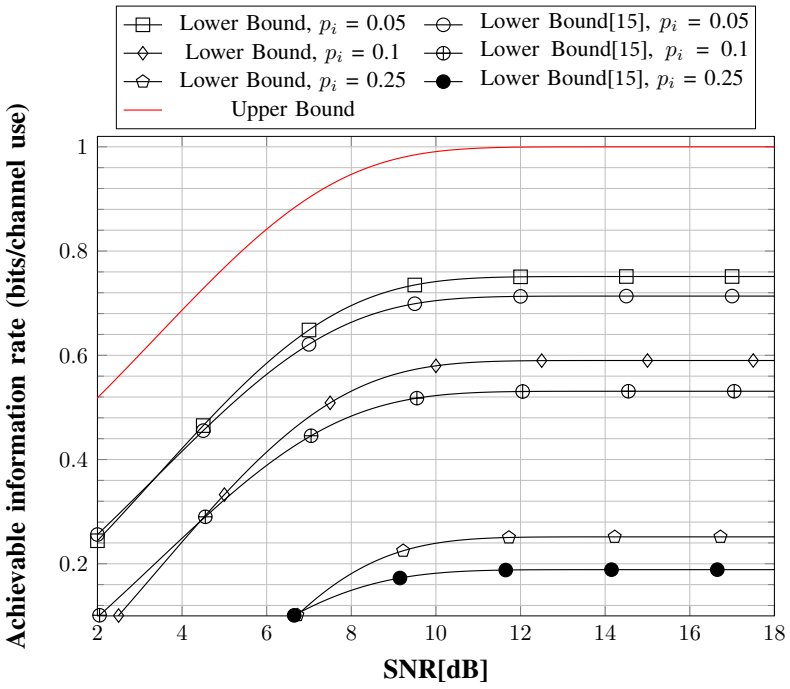


Figure 5.9: Comparison of the lower bound of information rates for covert channel with no jitter and AWGN channel with insertions with the lower bound of information rate derived in [15].

tighter than all previously reported lower bounds for this type of a channel. We need to note that for Fig. 5.8 and Fig. 5.9, the jitter variance is set to zero for a fair comparison of the proposed bounds with the previous lower bound results because they are derived without considering the channels with jitter. However, assuming no jitter for a covert channel is not proper. Fig. 5.10 illustrates the relation between achievable information rates and jitter variance. Here, we assume the insertion probability is $p_i = 0.05$. We can observe that the achievable information rate increases with SNR and decreases with jitter variance. While the existence of jitter limits the transmission capabilities, with high enough SNR, reliable communication can be achieved.

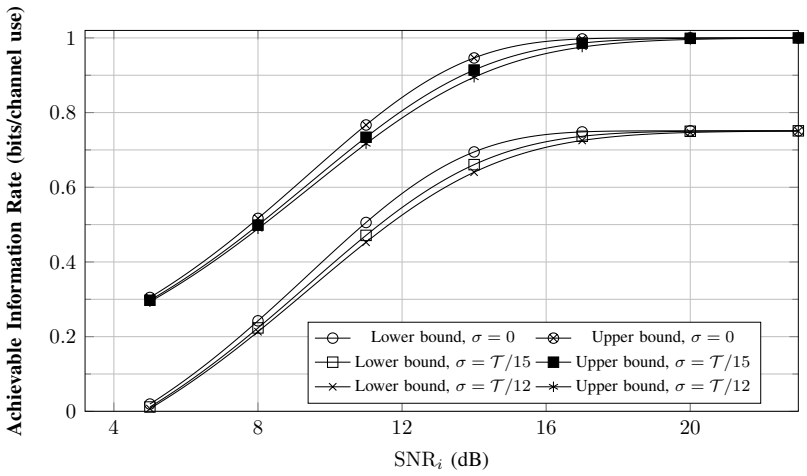


Figure 5.10: Upper and lower bounds of information rates of the covert channel with different jitter variances when $p_i = 0.05$.

5.6 Experimental Validation of the Proposed Model

In this section, we first demonstrate the existence of physical/analog covert channels generated by EM emanations, and then analyze this covert channel based on the proposed model, and also justify the assumption that the received signal is a PAM signal with insertions and substitutions for a variety of devices, i.e., FPGA, IoTs, laptops.

Analysis of the Covert Channel

To create a covert channel, we ran a microbenchmark (i.e., a *spy* application to transmit the sensitive data outside of the device) shown in Section 5.2 (also described in [17, 18]) on an Altera NIOS-II (soft) processor using a commercial Terassic DE1 SoC board [85]. This board is equipped with an Altera/Intel Cyclone-II FPGA chip and a variety of I/O protocols such as VGA, Serial, etc., and represents a popular class of embedded systems commonly used in the market. The application was written in standard C language and was compiled using the publicly available NIOS-II toolchain.

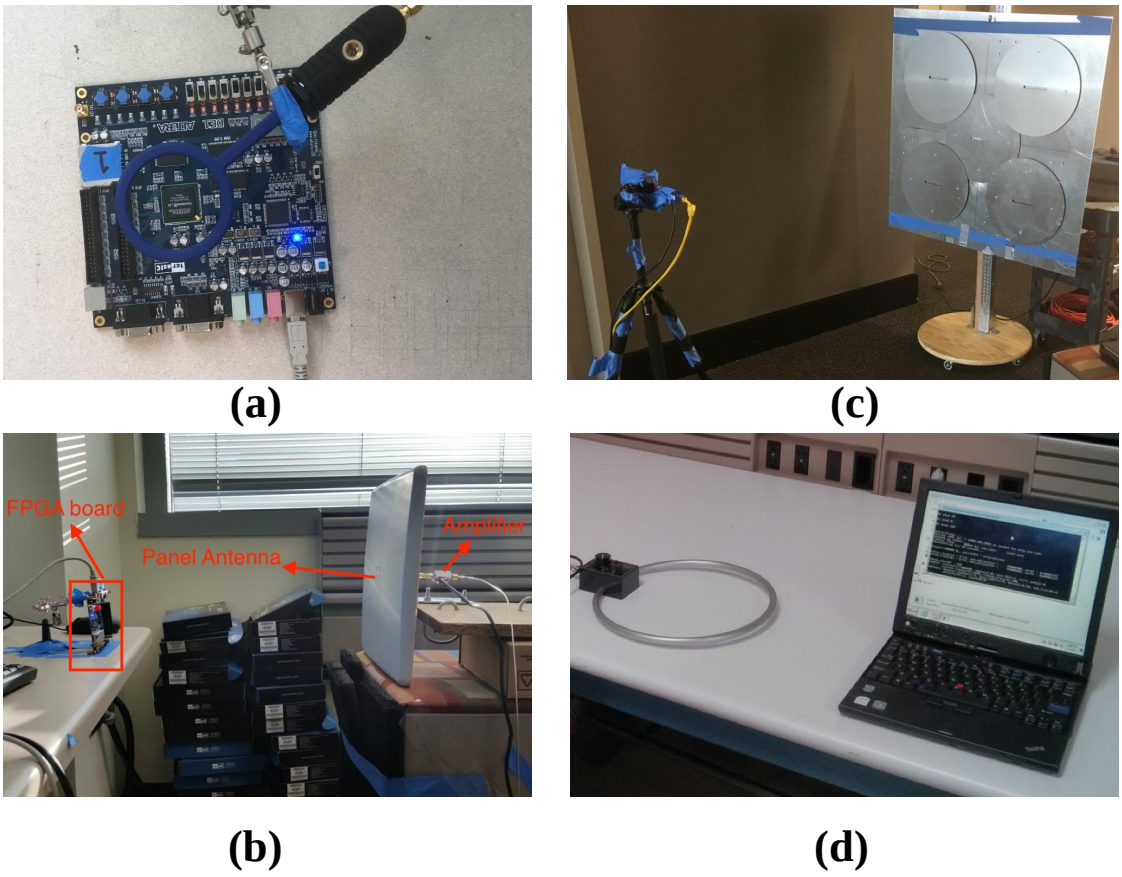


Figure 5.11: The meausrement setup for devices: a) FPGA, b) FPGA, c) OLinuXino, d) Laptops with distance.

To receive the transmitted EM signals created by the *spy* application inside the system, we placed a magnetic probe, PBS-M [85], about 10 cm above the board so that it covers the processor area as shown in Fig. 5.11(a). We intentionally put the probe very close to the board to receive the EM signal with the highest achievable SNR to avoid the limitations due to low SNR. In the later sections, we will show how this covert channel performs under more realistic scenarios where the receiver is placed several meters away from the device.

The EM signals were recorded using a spectrum analyzer (Agilent MXA N9020A). We set the sampling rate to 10 MHz, and set the spectrum an-

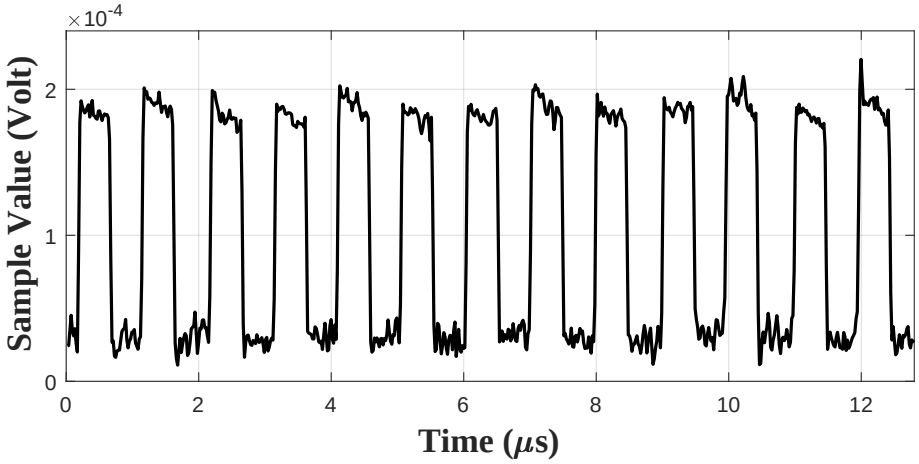


Figure 5.12: The received baseband signal with period $\mathcal{T} = 1\mu s$.

alyzer's center frequency to 50 MHz (i.e., the clock frequency of the FPGA chip), and the span to 4 MHz (i.e., 2 MHz for each side-band) since the designed microbenchmark creates periodic activities (i.e., A/B or A/A alternations) at 1 MHz so the device can pick up spikes from this periodic activity and its multiple harmonics.

The received signal is shown in Fig. 5.12 when the activity A and activity B of the microbenchmark are chosen as a load from main memory and load from an L1 cache, respectively. We can observe that the received signal has a shape of a PAM signal whose pulse width fluctuates due to uncertainties in the execution times of computer-software activities.

Table 5.1: Comparison of experimental and theoretical results in terms of BER for NIOS processor on the DE1 FPGA board.

	SNR (dB)	**Experimental BER**	**Theoretical BER**
1	6	0.096	0.0829
2	13	0.0013	0.0016
3	14	0.0008	0.0009
4	24	0	0

To compare the theoretical results with the experimental results, we

perform experiments with different activities and bandwidth. The estimated SNRs and the corresponding experimental and theoretical results are given in Table 5.1. The activities and the corresponding bandwidth used for the experiments can be listed as follows: 1) Addition-Multiplication with 4 MHz bandwidth, 2) Addition-Multiplication with 2 MHz bandwidth, 3) Load from Main Memory-Load from L1 cache with 4 MHz bandwidth, and 4) Load from Main Memory-Load from L1 cache with 2 MHz bandwidth. The sampling frequency for all these experiments is 10 MHz. We need to note that we only provide these results because we do not control SNR of the communication, therefore, generation of a plot with various SNR values is hard and unreliable. These results illustrate that proposed model is a realistic model for covert channels and can be used as a simulation tool.

The second example illustrates the presence of jitter and why our assumption that the jitter power can be added as an extra power source of white noise to calculate the BER, i.e., why assumption in (5.18) is valid. The following discussion considers the jitter distributions given in Fig 5.3. First, we plot PSD of the information signal and jitter noise. Here, we study the scenario without memory activity and the baseband transmitted pulses are sent with period $\mathcal{T} = 1\ \mu s$. The standard deviation of the jitter noise is calculated as $\sigma = 5.91 \times 10^{-8}$. The theoretical PSD of the transmitted signal and the jitter noise are given in Fig. 5.13(a), and PSD of the filtered signal at the receiver side is given in Fig. 5.13(b). We observe that the jitter noise power dominates the transmitted signal for higher frequencies. Filtering the received signal removes this redundancy and helps to retrieve the information signal.

As the final step, to verify BER estimation given in (5.18) based on

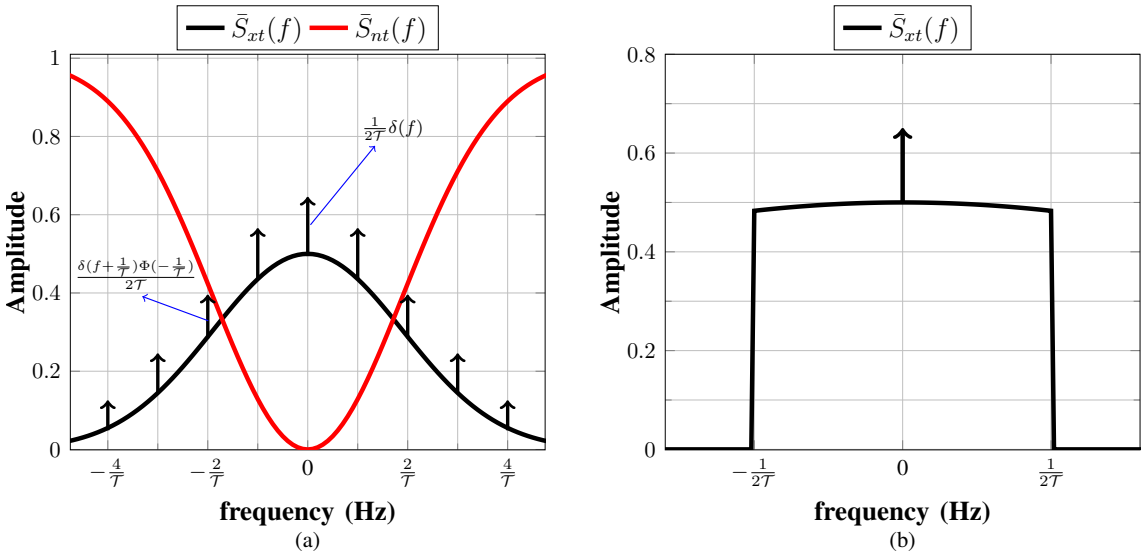


Figure 5.13: PSD of a) the transmitted signal and b) its filtered version at the receiver side for the symbol without memory activity.

PSD of the transmitted signal and jitter noise, we design the following experiment: We create an impulse train with period $\mathcal{T}$ and apply jitter noise by altering the location of pulses based on the normal distribution with variance σ^2. Then, we disturb the signal further by adding white noise, whose distribution can be given as $\mathcal{N}(0, N_0/2)$. The received signal is filtered with an ideal low pass filter on the receiver side and sampled with frequency $1/\mathcal{T}$. Finally, the sampled outputs are thresholded to estimate the transmitted inputs. The results are shown in Fig. 5.14. Here, we plot the simulation and theoretical BER results for the cases with and without memory activity. The results assert that simulated results agree with theoretically derived BER and verify the intuition that as the jitter variance increases, BER also increases.

Please note that not only PAM, but also frequency shift keying (FSK) modulation scheme can be generated with the microbenchmark. However, the detection of FSK signals can be harder since the variations in execution time of the microbenchmark will cause scrambling of different frequencies

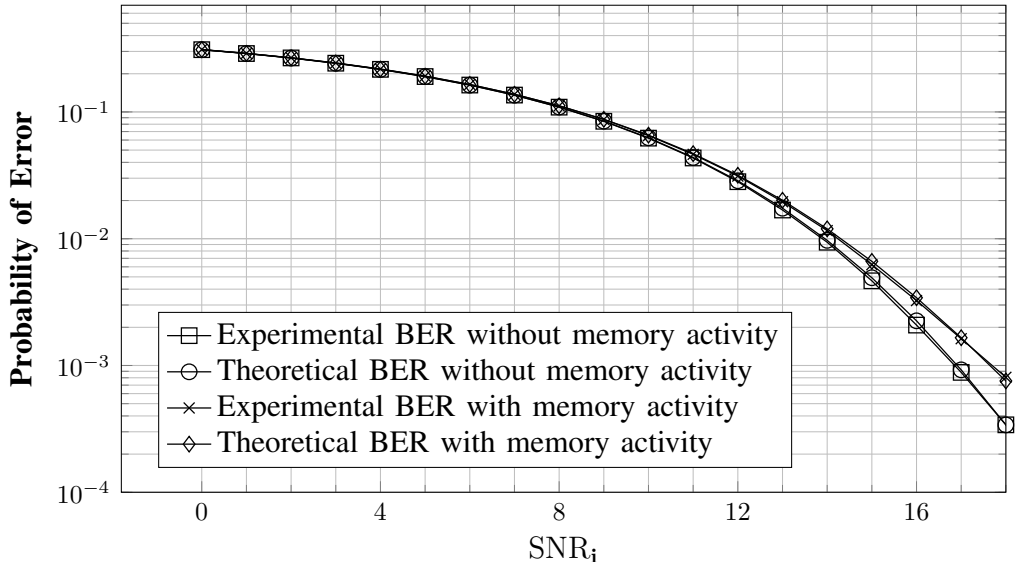


Figure 5.14: Theoretical and experimental BER for the symbols with and without memory activity.

and increase in BER. Therefore, more sophisticated receiver designs and more complex mathematical derivations are needed to achieve the same performance levels with the PAM case.

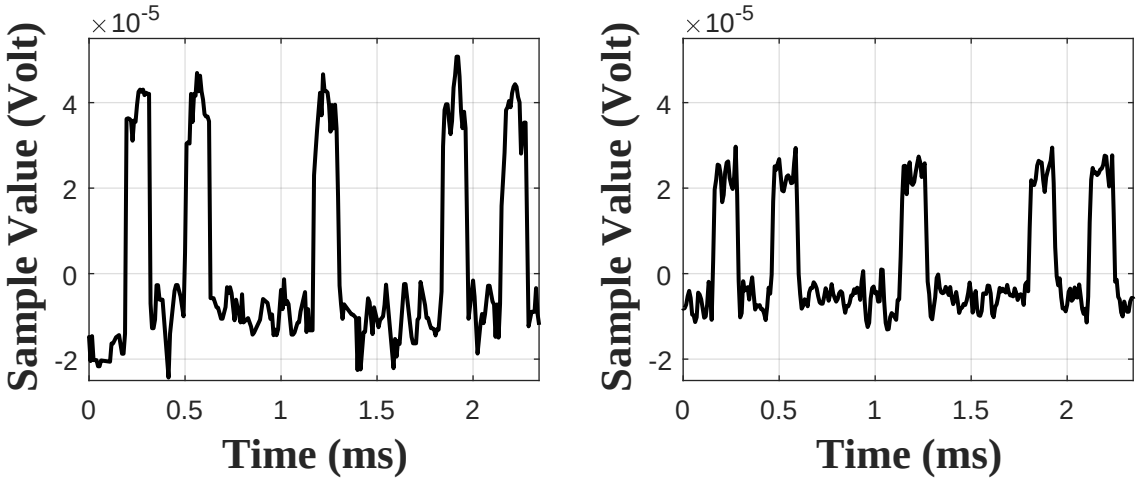
Demonstration of the Analog Covert Channel on More Complex Systems

In this section, we provide examples to show the practicality of the EM covert channel on more complex devices and more realistic distances.

We first study the impact of distance (i.e., the position of the probe/antenna and the receiving signal's SNR) on the bit-error-rate (BER). To measure the EM signals, we used a panel antenna [86]. We set the spectrum analyzer's center frequency to 2.3 GHz (i.e., the 46th harmonic of the FPGA's clock frequency, 50 MHz). This frequency is chosen to maximize the antenna's gain. We placed the board 50 cm and 1 m away from the board. The setup is shown in Fig. 5.11(b).

Fig. 5.15 shows the signal received from these distances. Please note

that received signals preserve the square-wave-structure like the signals in Fig. 5.12. These experiments confirm that the proposed EM covert channel is not sensitive to the position of the probe, and can be exploited from longer distances.

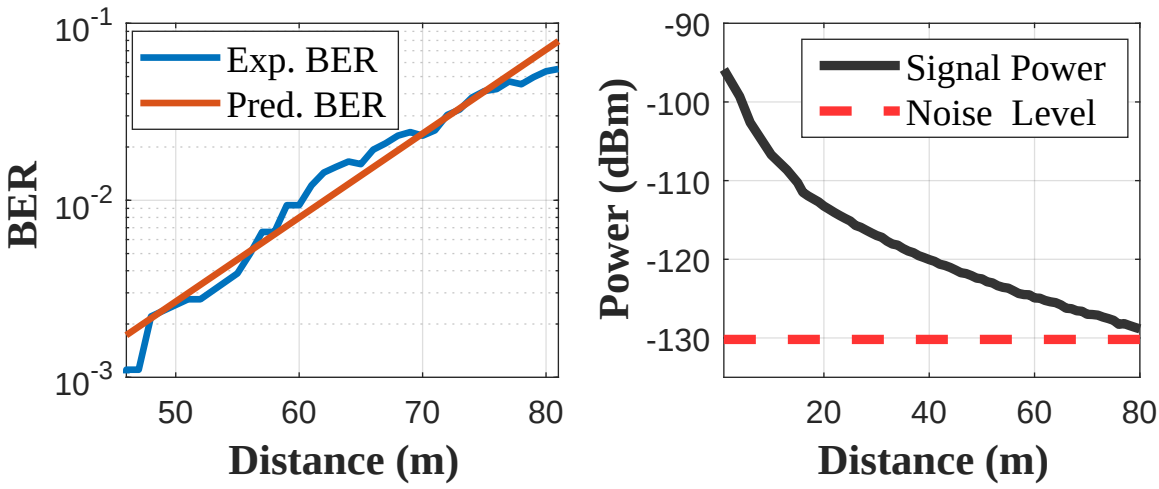


(a) (b)

Figure 5.15: The received signal at distance of a) 50 cm, b) 1 m.

To further study the possible range of an EM covert-channel attack and investigate its possibility on other (more complex) types of devices, we perform another experiment, this time using an embedded single-board computer called OlinuXino [87]. This board is equipped with a modern Cortex A8 ARM core with two levels of caches, 4 MB main memory, and runs a Debian Linux operating system. OlinuXino represents a popular class of single-board computers widely used in the market to control a variety of critical and commercial tasks in factory lines, hospitals, etc.

To receive the EM signals, we leveraged two different antennas: a commercially available horn antenna [88], and a high-gain custom-made disk-array based antenna [89]. Similar to the FPGA, we use our microbenchmark (written in C and compiled with Gnu-gcc tool) to establish the covert channel. We use the same spectrum analyzer for recording the signals with



(a) (b)

Figure 5.16: a) BER vs. distance, b) The received signal power vs. distance where the noise level of the instrument sensitivity level is about -130dBm.

a center frequency set to 1 GHz (i.e., the clock frequency of the ARM core), and span of 5 MHz, while setting the microbenchmark to generate alternations at 2 MHz. Please note that Fig. 5.11(c) shows our measurement setup and demonstrates that communication is conducted in a realistic indoor environment.

The results for BER and the signal power are given in Fig. 5.16. In Fig. 5.16(a), we plot the BER of measurements when the distance is more than 45 m. For closer distances, the success rate of the measurements is almost 100% (i.e., BER $< 10^{-3}$). As can be seen from this figure, the success rate (BER) linearly decreases as the distance increases (as shown by the fitted line). From the results we note that covert channel can achieve more than 99.9% of success rate if the distance is less than 45 m and the signal level is at least 8 dB above the noise level of the measuring device (i.e., -130 dBm in this experiment).

Finally, to show that this EM covert channel can be created for complex computing systems such as laptops, we performed experiments on four

different laptops with different processors, namely: AMD Turion X2 Ultra, Intel Core Duo T2600, Intel I7 2620M, and Intel Core 2 Extreme X9650.

In all these measurements, we use the same experimental setup given in [69] which utilizes a magnetic loop antenna with a radius of 30 cm [90] as shown in Fig. 5.11(d). The center frequency for the measurements is set to 1.024 MHz. The results are shown in Table 5.2. In this table, we

Table 5.2: Experimental results for computer systems with distance.

Platform	**CPU**	**Distance**	**BER**
AMD Turion	2.1 GHz	2.5 m	10^{-3}
Intel Core DUO	2.16 GHz	0.81 m	10^{-3}
Intel i7	2.7 GHz	1.75 m	10^{-3}
Intel Core 2	3 GHz	1.17 m	10^{-3}

provide the maximum distances that we achieve a reliable communication (i.e., BER $\approx 10^{-3}$) when the transmission rate for the covert channel is 800 bits per second (bps). We observed that the signal power leaked from different platforms shows variation (depending on the packaging, board, etc.), and that affects the range of the covert channel. Compared to the state-of-the-art [4, 56, 91], our studied covert channel provides up to 5x higher data-rate and 5x lower bit-error-rate.

5.7 Summary

A covert channel generated by program activities in a computer system is described and modeled. These covert channels experience jitter errors in addition to channel errors due to noise. This is a result of the computer activity "transmitter" which lacks precise synchronization. Also, the "transmitter" gets interrupted by other (system) activities, and the trans-

mitted signal goes through a channel obstructed by metal, plastic, etc. To capture all these effects, we have modeled the transmitted sequence as a pulse amplitude modulated (PAM) signal with random varying pulse position. From the model, we have derived the power spectral density and the bit error rate of the transmitted signal with insertion and substitution errors. We have also derived capacity bounds of these covert channels with insertion and substitution errors due to interrupts, noise, and jitter. The theoretical derivations are compared to empirical results and show good agreements.

CHAPTER 6
A GENERALIZED APPROACH TO ESTIMATION OF COVERT CHANNEL INFORMATION LEAKAGE CAPACITY

6.1 Overview

In this chapter, we propose a methodology to estimate the worst-case information leakage through various covert channels which can be adopted for both analog and digital covert channels. In that respect, we first model the communication channel as a deletion-insertion channel to mimic the possible losses due to software activities. Unlike conventional communication systems where the noise is assumed to be Additive White Gaussian Noise (AWGN), covert channels also suffer from changes in signaling time of transmitted bits. We show that the noise caused by signaling time variation can be combined with AWGN to explain the overall effective noise on the covert channel communication system. Secondly, based on the effective noise, we model the communication channel between the receiver and the transmitter. Then, we define the channel capacity as the maximum leakage for a given covert channel. Finally, we provide experimental results for various covert channels to show that the proposed model is an effective and a general method to attain the resilience of a given system.

In the communication literature, many papers discuss bounds on the capacity of channels corrupted by synchronization errors [10, 11, 12, 13, 14]. These papers are followed by capacities of channels corrupted by synchronization and substitution errors [15], [16]. Applying these capacity definitions to measure covert channel leakage for the worst-case scenarios

is not valid since they do not consider the variation in signaling time. A *micro-level* investigation on side channel capacities is considered in [18, 19] assuming instructions (which are the lowest level order to computer processor) are the transmitted symbols between the transmitter (source) and the receiver (sink). These papers exploit emanated EM signal power while executing instructions, and model the channel based on the differences in signal power levels of different instructions. However, these papers overestimate the covert channel capacities because they do not consider insertions and deletions that are encountered in *macro-level* (program level) scenarios. In that respect, a program-level channel model is proposed including insertions and deviation in signaling time while transmitting signals [77, 21]. Then, leakage capacity bounds are defined for only EM covert channels under the assumption that signaling time deviation can be modeled by changing the position of the pulses and by keeping the pulse width fixed. However, the scope of these papers is limited to EM based covert channels, ignores the losses due to deletions, fixes the pulse width while modeling the channel, and provides capacity bounds instead of exact values.

The proposed model in this chapter is a generalized approach for various covert channels and overcomes the issues regarding insertions, deletions, and asynchronous nature of the considered channels. The main contributions of the chapter can be listed as follows:

- We propose a communication model for covert channels including insertions and deletions to comply with software activities.
- We experimentally demonstrate that the distribution of signaling time variation exhibits a normal behavior, therefore, it can be fitted to a normal distribution with mean μ and standard deviation σ.

- Effective channel noise is introduced after deriving that the jitter error (error due to variation in signaling time) can be combined with additive channel noise. The behavior of the effective channel noise can change for different symbols.

- Based on the communication model and combined effective channel noise, we model channel to calculate the worst-case leakage through a covert channel. With this channel model, exact leakage capacity is obtained instead of providing capacity bounds.

- The proposed model is generalized for various covert channels, therefore, the same framework can be utilized by system designers to assess security of their systems against different types of *already existing* covert channels.

The rest of the chapter is organized as follows: In Section 6.2, we introduce the model for transmitted signal, receiver, and communication channel, Section 6.3 provides the derivation for effective channel noise and lekage capacity, Section 6.4 demonstrates how the proposed model can be utilized for various covert channel analysis, Section 6.5 provides experimental setup and results, and Section 6.6 is the conclusion.

6.2 Overall Communication Model

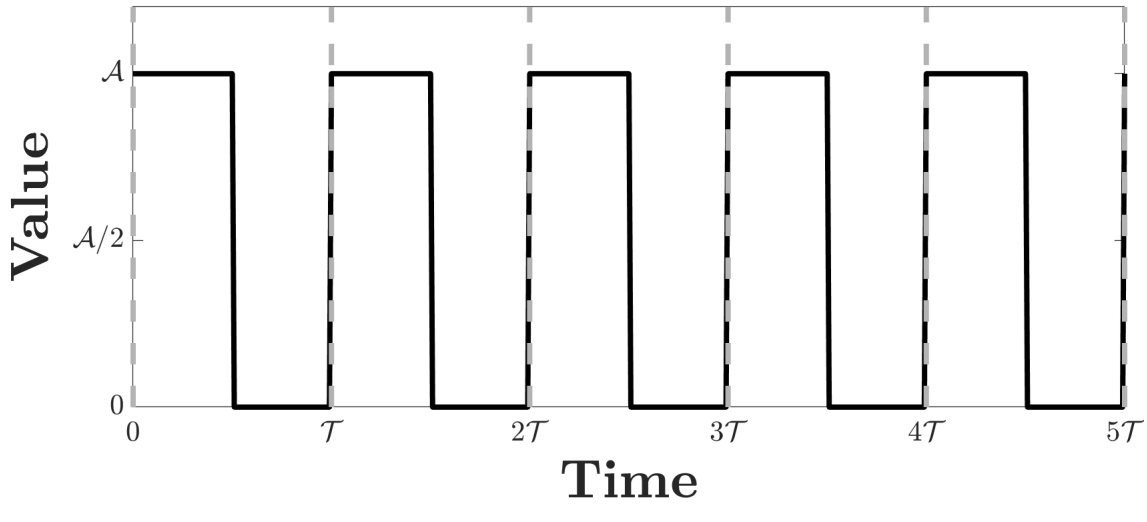
In this section, we first describe the proposed model for the transmitter of a covert channel considering its asynchronous nature and the variation in signaling time. Then, we analyze the design of a receiver by describing how a pulse-shape filter, similar to conventional communication systems, is used for receiving the data. This is followed by deriving effective channel noise which is a combination of additive and jitter noise caused by

the variation in signaling time. Finally, we propose the channel model by considering the effective channel noise, insertions, and deletions.

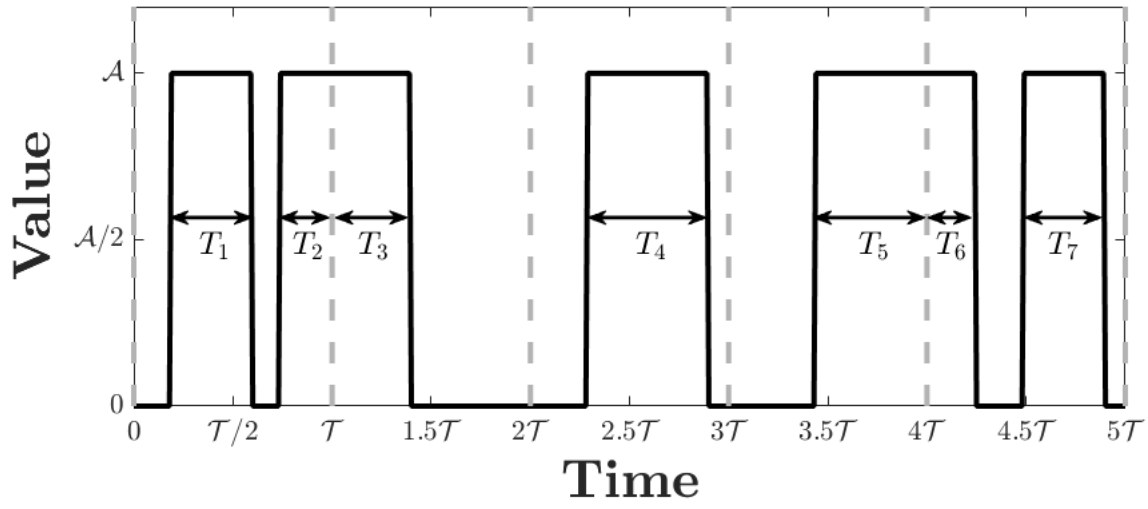
To obtain the worst-case information leakage that can be achieved by covert channels, we first need to model the transmitter and emanated signal. Note that the covert channels considered in this section do not seek for relevant information in the system. A trusted insider or a Trojan provides this information to these channels to transmit to an outsider, which is not an allowed action. In other words, the proposed method builds a bridge between the trusted insider and adversarial outsider by creating a covert channel. Also note that this section does not propose a new covert channel. The goal of the section is to analyze *already existing* covert channels and evaluate their severity.

6.2.1 Transmitted Signal and Receiver Model

Transmitted Signal: In conventional communication systems, transmitters and receivers are designed carefully to prevent any synchronization problems. However, covert channels are not designed to communicate at all [1]. Considering a connection between covert and conventional communication systems, the question is whether different modulation schemes (e.g., pulse amplitude-width modulation, pulse width modulation, and pulse amplitude modulation) are suitable for covert channels as well. However, employing such modulation schemes with various width and amplitude choices are not practical for covert channels since the width and amplitude of the transmitted signals deviate due to other program activities [65]. To avoid these difficulties arising because of these deviations, the general practice in covert channel community is to employ a modulation scheme that can transmit zeros and ones [56, 21, 5]. Therefore, we make



(a)



(b)

Figure 6.1: The received signal for a) ideal conventional communication system, b) covert channel communication system.

the following assumptions and introduce notations for the model of the transmitted signal:

A1: The Transmitted Signal Assumptions and Notations

- The receiver samples the signal at every $\mathcal{T}$ seconds under the assumption that the transmission time of the covert channel is $\mathcal{T}$.
- On-Off-Keying (OOK) is considered as the modulation scheme to

transmit information signal (i.e., the source exploits a specific side-channel such as cache to transmit bits).

- The targeted duty cycle changes based on the implementation of the covert channel. In other words, the ratio between the width of the pulse and the transmission time $\mathcal{T}$ can vary for different channels.
- There is no overlapping among the bits transmitted by the covert channel transmitter. Therefore, a bit is transmitted only if the transmission of the previous bit is complete.
- The signal is on for T^0 and T^1 seconds if the transmitted bit is zero and one, respectively. Please note that T^0 could be also zero, which represents the transmission period when nothing is transmitted.

In Fig. 6.1(a), an ideal OOK modulated signal is shown when the transmitted bit sequence is all ones (Although the behavior of any bit sequence is the same, all-one-bit-sequence is used for better explanation of the process). Unfortunately, obtaining such a signal is not possible with an unintentional channel. Because of the delays and synchronization problems in covert channels, the system experiences shifts in transmitted signals as shown in Fig. 6.1(b). This undesired behavior appears in almost all of the covert channels, and needs to be modeled to understand and estimate leakage capacity.

Receiver Model: For the receiver model, we need to combine the knowledge of conventional communication theory and all the practices utilized in the security community. The common approach in conventional sys-

tems is to apply a match filter under the assumption that the system is well-synchronized [84]. Motivated by this approach, we make the following assumptions and observations:

A2: The Receiver Model Assumptions

- The receiver employs a filter which mimics a match filter to capture the transmitted information.
- Since the modulation is a result of software-hardware activities, the transmitted keys encounter problems with changes in duty cycle and non-synchronization, i.e., delays while transmitting modulated signal.

Since synchronization problems limit the capability of the match filter for covert channels, the first step to design the receiver is to relax these synchronization requirements in order to handle shifts in signaling-time. To capture the transmitted bit sequence, the receiver employs a *matched* filter with 100% duty cycle (irrespective of the actual duty cycle of the transmission) to capture the changes in signaling time as

$$m_c(t) = \frac{1}{\sqrt{\mathcal{T}}}\texttt{rect}\left(\frac{t}{\mathcal{T}}\right), \tag{6.1}$$

where `rect`(t) is a function with amplitude one, and has only nonzero values between -0.5 and 0.5. Assuming $r(t)$ is the received signal, we can write the match filtered sequence after sampling with period $\mathcal{T}$ as

$$\begin{aligned} y(n\mathcal{T}) &= r(t) * m_c(t - n\mathcal{T})|_{t=n\mathcal{T}} \\ &= \frac{1}{\sqrt{\mathcal{T}}} \int_{(n-1)\mathcal{T}}^{n\mathcal{T}} r(n\mathcal{T} - \tau)\texttt{rect}\left(\frac{\tau}{\mathcal{T}} - (n - 0.5)\right) d\tau \\ &= y_n \end{aligned} \tag{6.2}$$

where $*$ is the convolution operation and n is the sample index. Under this receiver implementation, the received signal can be considered as a signal which only experiences variation in duty cycles. Please observe that raising time of the received signals can be altered such that the total area under the signaling period stays the same. Therefore, an equivalent version of the original signal in Fig. 6.1(b) can be presented as in Fig. 6.2. The equivalence of these two signal models stems from the fact that the receiver does not consider the shift within the sampling period, but the existence of the signal components.

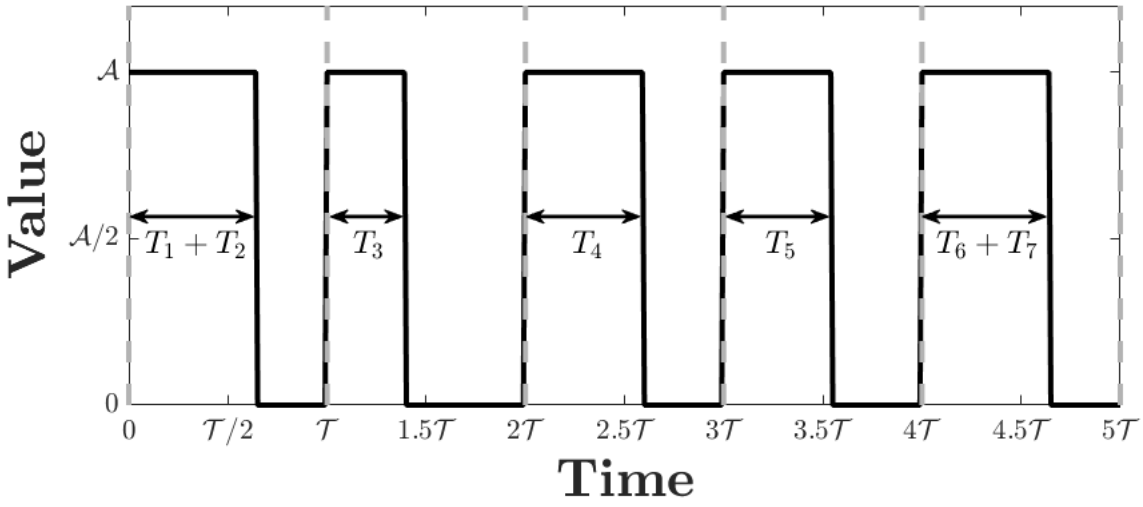


Figure 6.2: The equivalent version of the received signal under the assumption that the receiver employs a matched filter in (6.1).

Let us consider a noiseless environment where the received signals only experience asynchronous nature of the covert channel. Fig. 6.3 shows the received signal between $(n-1)\mathcal{T}$ and $n\mathcal{T}$ when, without loss of generality, the transmitted bit is one after modifying the raising time as given above (the same discussion can be made for bit-zero).

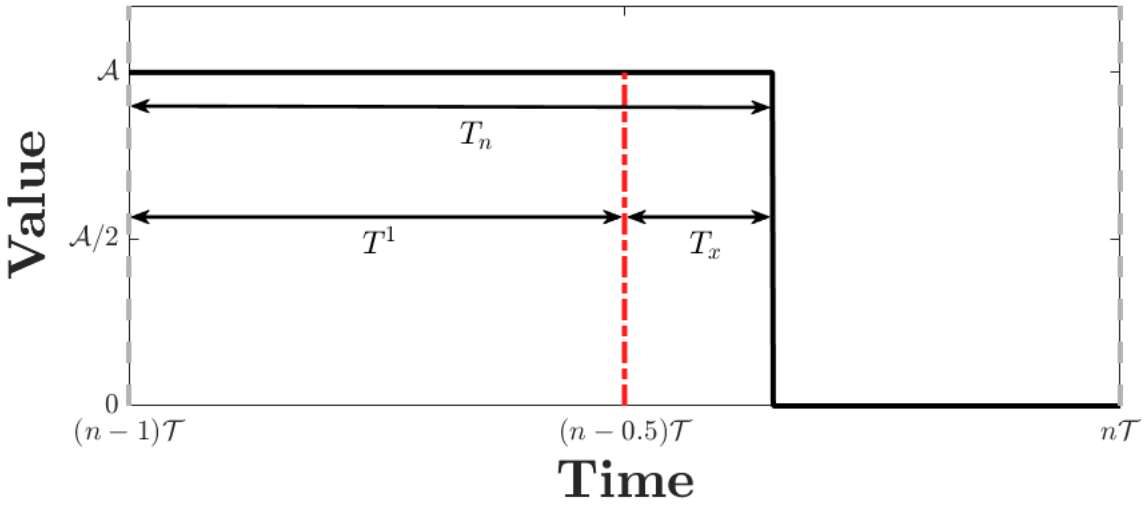


Figure 6.3: One cycle corrupted received signal that was modified by signaling time variation, and modified such that the raising time is equivalent to $(n-1)\mathcal{T}$.

The received sample for this time slot can be written as

$$\begin{aligned} y_n &= \frac{1}{\sqrt{\mathcal{T}}} \int_{(n-1)\mathcal{T}}^{n\mathcal{T}} r(n\mathcal{T}-\tau)\texttt{rect}\left(\frac{\tau}{\mathcal{T}}-(n-0.5)\right) d\tau \\ &= \frac{\mathcal{A}}{\sqrt{\mathcal{T}}} T_n = \frac{\mathcal{A}}{\sqrt{\mathcal{T}}}\left(T^1+T_x\right) \end{aligned} \tag{6.3}$$

where T_x is a random variable for the width variation of the received signal. For the rest of the section, we refer to T_x as the *effective variation*. Please note that T_x can also be negative and the outputs of the match filter can be smaller or larger than the expected output value.

6.2.2 Channel Model

In this section, we introduce our discrete memoryless channel model for the covert communication. Having such a model is essential for establishing connection with information theory which allows us to calculate the channel capacity.

For the channel model, we make the following assumptions:

A3: Bit-Deletion

The covert channel transmitter continuously sends information bits unless it encounters interrupts, stalls, etc. Due to other program activities that can run in parallel with the covert channel source, the received signal is masked and can be randomized because of the constructive and destructive interference. From the receiver perspective, the received bit has the highest entropy, hence, this scenario is considered as the deletion.

A4: Bit-Insertion

When stalls, interrupts, etc., force the source of covert channel to stop transmitting information, the insertion of random bits occurs. Since the receiver is not aware of such an interrupt, it keeps interpreting the sampled symbols as the actually transmitted symbols.

A5: Additive Gaussian White Noise (AWGN)

The transmitted signals are also corrupted by additive white Gaussian noise. We assume that this noise covers all unrelated signals that are produced by the environment and the system.

Under these assumptions, signal-to-noise-ratio (SNR) for the conventional communication systems can be written as

$$\mathrm{SNR} = \frac{P_s}{\sigma_n^2} \tag{6.4}$$

where P_s is the signal power and σ_n is the standard deviation of the noise after sampling. However, the conventional SNR definition does not reflect the variation in the width of the transmitted signal. Therefore, the first goal is to combine the channel noise and the noise due to width variation, which is called jitter noise. First we consider the noiseless scenario given

in (6.3). If we define the ideally received sampled symbol, y_o, as

$$y_o = T^i \frac{\mathcal{A}}{\sqrt{\mathcal{T}}} \tag{6.5}$$

and the jitter noise term, n_x as

$$n_x = T_x \frac{\mathcal{A}}{\sqrt{\mathcal{T}}}, \tag{6.6}$$

the received sampled symbol can be written as

$$y_n = y_o + n_x \tag{6.7}$$

where $i \in \{0,\ 1\}$. Let T_x be normally distributed as

$$T_x \sim \mathcal{N}\left(\mu_x, \sigma_x^2 | \text{Bit-}i \text{ is transmitted}\right).$$

This equation exposes two main intuitions about the characteristics of covert channels: 1) the mean and standard deviation of the signaling time variation could be different for different bits, and 2) the distributions can exhibit differences for each system since they can correspond to non-identical program activities. Hence, different jitter-noise schemes need to be considered. With this assumption, the distribution of n_x can be written as

$$\mathcal{N}\left(\frac{\mu_x \mathcal{A}}{\sqrt{\mathcal{T}}}, \frac{(\sigma_x \mathcal{A})^2}{\mathcal{T}} \middle| \text{Bit-}i \text{ is transmitted}\right) = \mathcal{N}\left(\mu_{x,i}, \sigma_{x,i}^2\right).$$

Equation 6.7 reveals that even with noiseless scenario assumption, the covert channel system still encounters noise due to jitter in the system. Including the additive channel noise (AWGN), the received symbol can be

written as

$$y_n = y_o + n_x + n_o \tag{6.8}$$

where $n_o \sim \mathcal{N}(0, \sigma_n^2)$ is the channel noise sample. Here, jitter and channel noise can be combined as a single random variable because both have Gaussian distribution. Let's define n_c as the effective noise component which is given as

$$n_c = n_x + n_o.$$

Therefore, the distribution of n_c can be written as

$$n_c \sim \mathcal{N}\left(\mu_{x,i}, \sigma_n^2 + \sigma_{x,i}^2\right). \tag{6.9}$$

Moreover, it is also possible that bit-zero signal can also be nonzero for a while if $T^0 > 0$. Therefore, the average transmitted signal can be written as

$$P_s = \frac{\mathcal{A}^2}{\mathcal{T}}\left(p_0\left(T^0\right)^2 + p_1\left(T^1\right)^2\right) \tag{6.10}$$

where p_j $(\{j \in \{0,1\}\})$ represents the probability that bit-j is transmitted without deletion. Likewise, average effective noise power can be written as

$$P_n = \sigma_n^2 + p_0\left(\mu_{x,0}^2 + \sigma_{x,0}^2\right) + p_1\left(\mu_{x,1}^2 + \sigma_{x,1}^2\right) \tag{6.11}$$

Therefore, the effective SNR ($\mathrm{SNR}_{\mathrm{eff}}$) in these covert channels can be defined as

$$\mathrm{SNR}_{\mathrm{eff}} = P_s/P_n. \tag{6.12}$$

These equations show that covert channels suffer not only from channel noise but also variations in signaling time. With the modeling assumptions from Section 6.2, we can observe that signaling time variation behaves like

an extra source of channel noise. Therefore, for the simplicity of discussion, we assume additive channel noise has two components which are independent of each other: jitter and additive channel noise.

Having the model for the noise term in the system, the received samples given in (6.8) can be written as

$$y_n = y_o + n_c. \tag{6.13}$$

Therefore, overall channel between the transmitter and receiver can be modeled as a discrete memoryless channel since the transmitted bits show no dependency to other bits in the sequence. Another important point is that if the communication is insertion/deletion-free, the overall channel can be considered as a binary channel. However, because of other activities in the computer system, ignoring insertions/deletions in the covert communication overestimates the possible information leakage through these systems.

The channel model based on these assumptions is given in Fig. 6.4. To simplify explanations, we divide the model into two parts *Bit-Generation* and *Communication Channel*. The *Bit-Generation* shows the probabilities of different types of signals that exist in the system. In this part, x_i represents the transmitter (the actual source of the covert channel). I represents the other activities that can cause insertions in the channel. The probabilities p_0 and $p_1 (= 1 - p_0)$ are the probabilities to send bit zero and one, p_d is the deletion probability, p_i is the insertion probability, and p_c is equivalent to $p_i + p_d$. The second part, *Communication Channel*, presents the transition probabilities of different symbols. Here, s_0, s_1 and $s_?$ represent the transmitted bit zero, one, or an insertion/deletion. The received

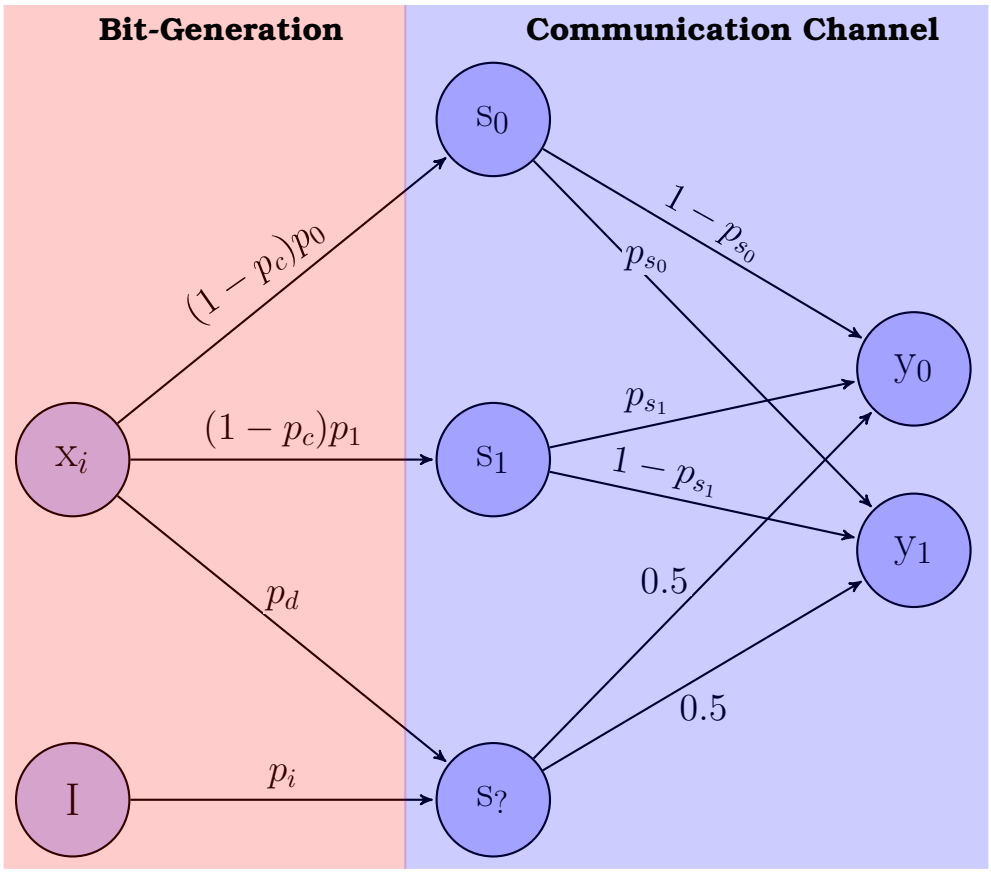


Figure 6.4: Channel Model for the communication system.

symbols corresponding to bits zero and one are denoted by y_0 and y_1, respectively. We need to note here that based on the assumptions **A3** and **A4** given in Section 6.2, whenever a deletion/insertion occurs, the receiver only guesses whether the received signal corresponds to one or zero.

The transmitter behaves like a ternary source, which generates zero, one or an insertion(or deletion), however, the receiver always interprets the received symbols as zero or one since it is unaware of insertion (or deletion) locations. Another observation is that the timing variation while transmitting bit-zero or bit-one could be different, which leads to differences in T_x distribution. This means the channel does not exhibit a binary-symmetric feature even if there are no insertions (or deletions). To incorporate this asymmetrical feature of the system, the substitution probabilities for bit-

zero and bit-one are represented as p_{s_0} and p_{s_1}, respectively.

6.3 Leakage Capacity

Having a model for the covert channels enables calculation of leakage capacity because leakage capacity corresponds to channel capacity of the model. In conventional communication systems, the channel capacity is calculated based on Shannon's theorem [75]. The channel capacity is defined as

$$C = \sup_{p(x)} I(X;Y) \tag{6.14}$$

where X and Y are the random variables for inputs and outputs, and $p(x)$ is the probability distribution for the inputs. In our scenario, X and Y represent the symbols sent from the transmitter (a *source*), and the received symbols (by an *sink*), respectively.

To calculate the leakage capacity, the first step is to obtain the transition probabilities, p_{si} where $i \in \{0,1\}$. We know that the threshold is calculated based on the posterior distribution of the inputs [84], and that the asymmetric nature of the system affects the threshold while calculating substitution probability. Combining these information, the threshold has to fulfill the following equations:

$$\begin{aligned} p_0 f\left(z_{thr} \left| \hat{\mu}_0, \hat{\sigma}^2_{n,0}\right.\right) &= p_1 f\left(z_{thr} \left| \hat{\mu}_1, \hat{\sigma}^2_{n,1}\right.\right) \\ p_0 f_0(z_{thr}) &= p_1 f_1(z_{thr}) \end{aligned} \tag{6.15}$$

where

$$\hat{\mu}_i = T^i \frac{\mathcal{A}}{\sqrt{T}} + \mu_{x,i} \text{ and } \hat{\sigma}^2_{n|i} = \sigma^2_n + \sigma^2_{x,i},$$

which represent the effective symbol mean power and the effective noise

variation when bit-i is transmitted, respectively, $f(x|\mu,\sigma^2)$ is the probability density function (pdf) for Gaussian distribution with mean μ and standard deviation σ, and z_{thr} is the threshold value to calculate substitution probabilities which preserve the equality in (6.15). Therefore, considering all these features of a covert channel, we define the leakage capacity as

$$\begin{aligned} &\underset{p(x)}{\text{maximize}} && I(X;Y) \\ &\text{subject to} \\ &&p(x \text{ is insertion}) &= p_i \\ &&p_0 f_0(z_{thr}) &= p_1 f_1(z_{thr}) \\ &&p_{s_0} &= P_0(Y > z_{thr}) \\ &&p_{s_1} &= P_1(Y \leq z_{thr}) \end{aligned} \tag{6.16}$$

where $P_i(\bullet)$ provides the probability of its event with respect to the pdf of the corresponding bit, $f_i(\bullet)$. The solution to this optimization problem provides the worst-case information leakage through covert channels.

We need to note here that the insertion (or the deletion) probability depends on the targeted computer system. Therefore, p_i and p_d have to be kept fixed while calculating the leakage capacity. A further analysis can be done to find the effect of insertion/deletion on the leakage capacity. We can observe that increase in these probabilities decreases the channel capacity. This means that systems can be designed more chaotic (randomly pumping power to the systems, activating some random components, etc.) to increase insertion/deletion probability (which can be thought as a shielding strategy). Hence, having an investigation on the

effect of deletion/insertion on the channel capacity gives insight into the required level of this chaotic regime.

6.4 Establishing Connection between the Proposed Model and Covert Channels

In this section, we explain how the proposed framework can be used to model various covert channels. By establishing such a connection, we demonstrate that the model can be utilized to calculate leakage capacity of these channels, and define a metric measuring the resilience of any system to covert channel attacks.

6.4.1 Power Based Covert Channels

The connection between covert channels based on Simple Power Analysis (SPA) and the proposed model can be established explicitly because power covert channel attacks utilize total power consumption of the system. For example, they exploit variation in power consumption while executing bits for signing operation in crypto-systems [30, 92]. The main goal is to measure the total power, and to estimate whether the signed bit is zero or one, therefore, the system can be represented by OOK modulation.

To calculate the leakage capacity of power covert channel, we can assume that $\mathcal{T}$ is the average time required to sign a bit for a cryptosystem, or processing one bit of information. However, processing this information can take different amount of time due to other software activities, optimization, etc. Therefore, it can cause some shifts in time, and variation in processing time, which can be explained by the proposed model as long as the distribution of the effective variation is known. Also, due to stalls, interrupts, etc., some of the bits correspond to deletions or insertions. Both

of these issues are covered by the proposed model, therefore, the power based covert channel can be analyzed theoretically.

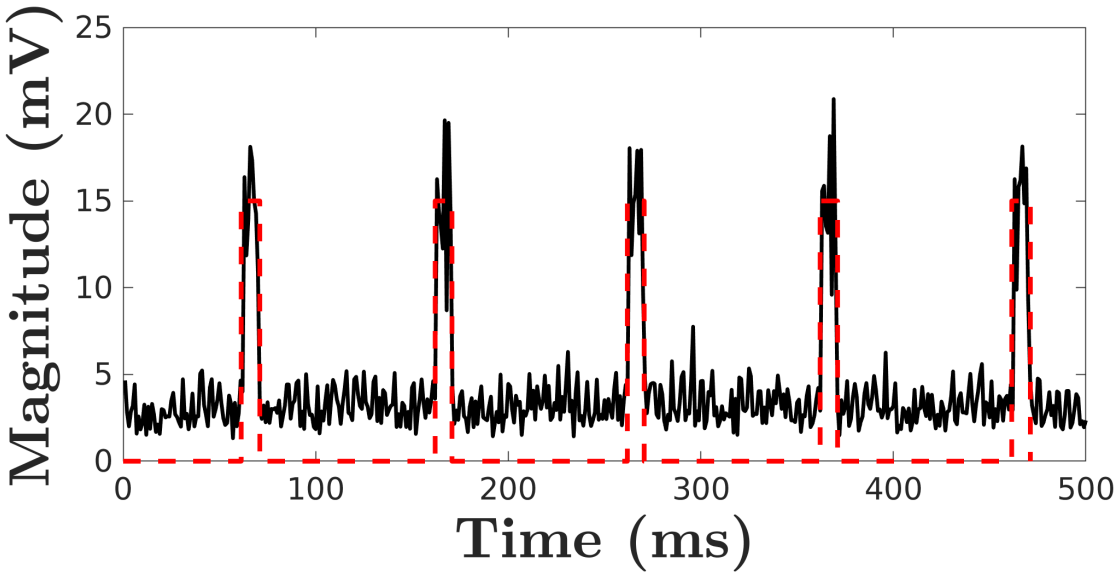


Figure 6.5: Received signal generated by the covert channel in [29]. The black (solid) curve represents the measured signal and the red (dotted) curve is for the modeled signal.

An example of the received signal for power analysis is given in Fig. 6.5 when the microbenchmark in [51] is executed to transmit 0 - 1 sequence repetitively. Because power channels are very noisy channels, we filter the signal with move-median filter that helps exposing the OOK structure of the received signal. Since the proposed framework is flexible to model any OOK signal with any duty cycle, it is possible to define and obtain the leakage capacity by collecting the statistics about timing variation, insertions, and deletions.

6.4.2 EM-Based Covert Channels

EM covert channels are a consequence of computer activities and their effects on EM fields. By measuring the variation in the EM field, it is possible to steal information from a distance [5, 21, 59, 93]. Requiring no direct ac-

cess to the system and having larger available frequency band can be listed as the main advantages of these channels over other covert channels. In this section, we consider two channel types that exploit the variation in EM field caused by different units of a modern computer system.

EM-Based Covert Channels Due to Processor Activities

It is already shown that a covert channel can be generated by running a microbenchmark that causes systematic changes in the surrounding EM field, and a motivated attacker can monitor these changes to infer the transmitted bits [3, 21]. An example of the generated signal is given in Fig. 6.6. The main observation is that similar to the power covert channels, the received signal displays OOK structure, but suffers from variation in signaling time.

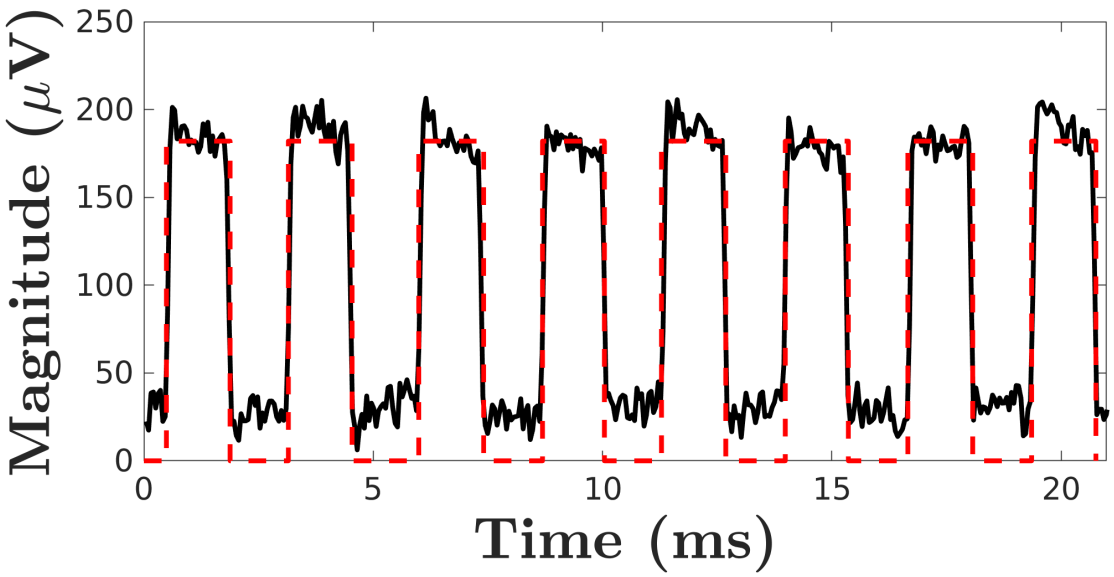


Figure 6.6: Received signal generated by the covert channel in [21]. The black (solid) curve represents the measured signal and the red (dotted) curve is for the modeled signal.

Considering the same arguments with power based covert channels, we can see that the proposed model can explain the leakage in the worst-case

scenario for a given design. To achieve our goal, the critical step is to obtain the variable values for deletion, insertion, $\mathcal{T}$, and the distribution for the effective variation. Then, the proposed methodology can be utilized to assess the resilience of a system against EM-based covert channels.

EM-Covert Channels Based on Power Management Units

These covert channels are generated by exploiting power management units (PMU) and voltage regulator module (VRM) of modern computers. PMU is responsible for power alteration to optimize the power consumption of the system. Since the priority of designers is to minimize the consumption of power, not security, they do not put enough effort on the security aspect of their design. By leveraging such a security flaw, a covert channel that transmits sensitive information from *air-gapped* computers is generated in [5].

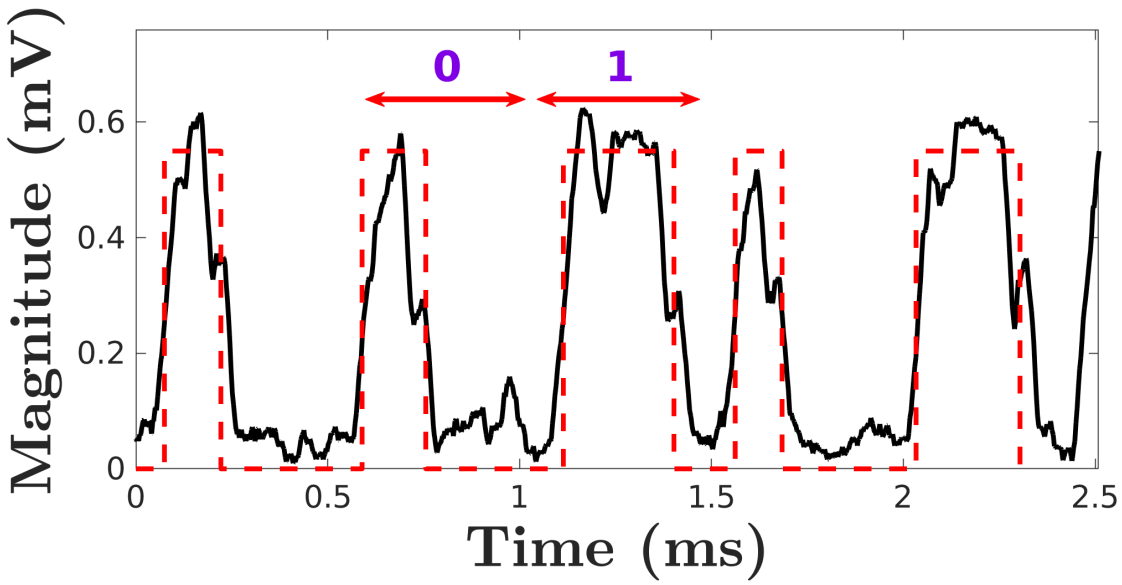


Figure 6.7: Received signal generated by the covert channel in [5]. The black (solid) curve represents the measured signal and the red (dotted) curve is for the modeled signal.

To transmit information, a microbenchmark is designed causing changes

in the power state of a system. An example of the demodulated signal for the covert channel is given in Fig. 6.7. The main observation here is that the received signals are active for a while even for the *off* case. However, this is also included in the proposed model since we do not restrict T^0 to be zero. Furthermore, this channel suffers from insertions, deletions and signaling time variation as previously considered covert channels. Although the received signal is not a perfect square signal, the proposed methodology can be exploited to calculate the leakage limit assuming distortions are due to additive channel noise.

6.4.3 Backscattering Covert Channels

This covert channel is created by deliberately exploiting the recently introduced backscattering side-channel. It exploits circuits as a semi-passive RFID and relies on switching activities on the level of transistor gates (between low and high states). The switching activities change the impedance of circuits, hence, the circuit behaves as an RFID tag. For example, by utilizing this channel, circuits with Trojans are identified because the backscattered signals show different characteristics due to change in the impedance of the circuits [94]. Although there is no registered attack, or a paper investigating attack scenarios based on these channels, we still consider this channel to obtain its capacity because impedance change can result in exfiltration of some sensitive information.

An example of the received signal for the backscattering covert channel is given in Fig. 6.8. The same characteristic features with other covert channels, i.e., insertions, deletions, timing variation, are observed as well. Therefore, the proposed methodology can calculate the maximum leakage (or information transfer) given that the statistics about deletion, insertion,

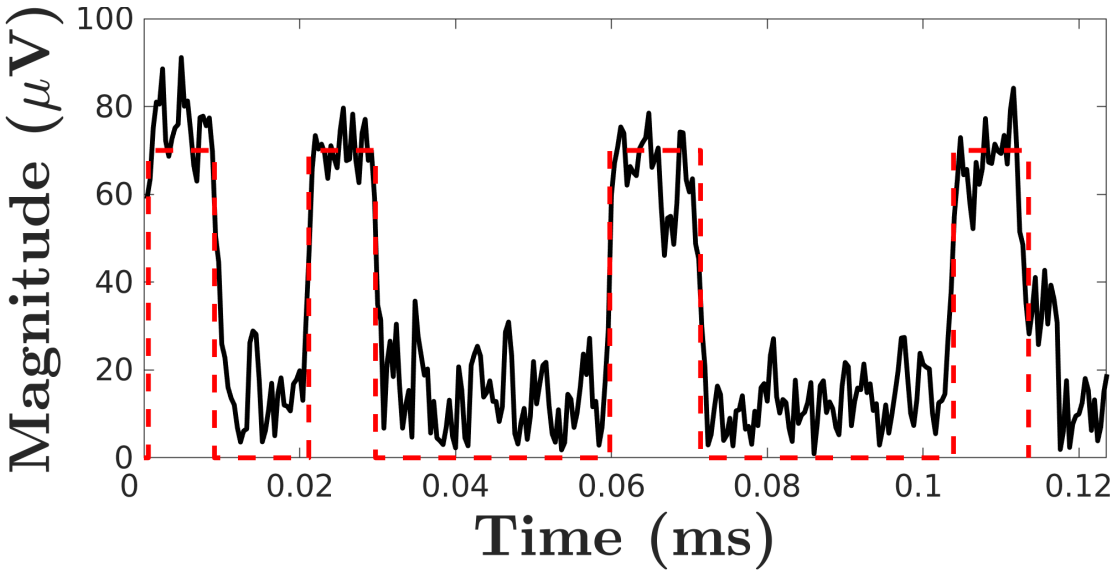


Figure 6.8: Received signal generated by the covert channel in [95]. The black (solid) curve represents the measured signal and the red (dotted) curve is for the modeled signal.

and timing distribution are known.

6.4.4 Cache-Based Covert Channels

To improve the performance of a computer system by reducing the effective main memory latencies originating from data accesses, faster hardware caches are used for storing the frequently used data. Based on the speed of the caches, they are divided into levels i.e., L1, L2, etc., where the highest cache level i.e. L1 is the smallest, fastest and closest to the processor, and subsequent lower levels are placed closer to the main memory with varying higher latencies. Thus, more recent and/or frequent the data is, the higher it will placed in the cache levels.

The cache-based covert channel attacks exploit this difference in data access time to steal sensitive information of a victim. For example, based on the time differences in recalling cache entries, secret keys of different cryptosystems are broken [37, 38]. We need to note here that the recall

time can be used not only for *evil* purposes. For example, a methodology is proposed in [96] to profile the memory access that does not cause any overhead on the system. The method exploits emanated EM signals for performance analysis, and provides statistical information on the recall time of the system.

The question here is that how the proposed method can model the cache based covert channels since the only data collected during these attacks is the recall time. Let us start with the following observation: These attacks request recall time at every pre-defined time interval. In our case, this pre-defined interval is equivalent to $\mathcal{T}$. Moreover, the recall time can be considered as the output of the receiver after filtering with $m_c(t)$. Let T_R be the current recall time of an experiment or attack. If we assume the transmitted signal, $T_S(t)$, is a pulse function whose width and amplitude are equivalent to the recall time and $\sqrt{\mathcal{T}}$ with a random shift, we have

$$\begin{aligned} y(n\mathcal{T}) &= T_S(t) * m_c(t - n\mathcal{T})|_{t=n\mathcal{T}} \\ &= \int_{(n-1)\mathcal{T}}^{n\mathcal{T}} T_S(n\mathcal{T} - \tau)\frac{1}{\sqrt{\mathcal{T}}}\texttt{rect}\left(\frac{\tau}{\mathcal{T}} - (n - 0.5)\right)d\tau \\ &= \int_{t'}^{t'+T_R} \sqrt{\mathcal{T}}\frac{1}{\sqrt{\mathcal{T}}}\texttt{rect}\left(\frac{\tau}{\mathcal{T}} - (n - 0.5)\right)d\tau \\ &= \int_{t'}^{t'+T_R} \texttt{rect}\left(\frac{\tau}{\mathcal{T}} - (n - 0.5)\right)d\tau \\ &= T_R \end{aligned} \tag{6.17}$$

where

$$t' = \max((n-1)\mathcal{T}, (n-1)\mathcal{T} + t_d)$$

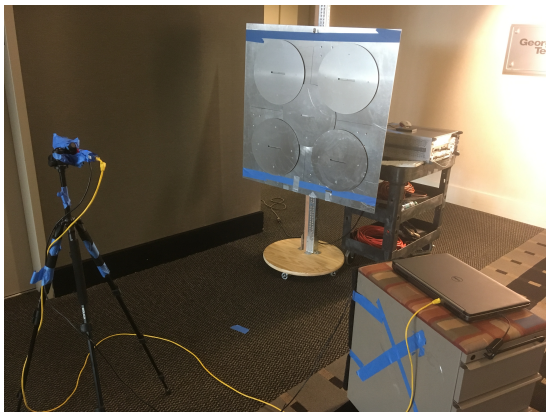
and t_d is the time shift. Since the output is equivalent to T_R, our model can represent the cache based covert channels with the assumed transmitter and receiver. Please also observe that the recall time varies at each operation, and that is represented by the timing distribution in the model. Another observation is that based on these transmitter and receiver assumptions, the additive channel noise power is equivalent to zero, and all of the effective noise is represented by the jitter noise. Finally, cache-based covert channels generally suffer from deletions and insertions, which is also considered in the proposed model. Therefore, the proposed methodology can calculate the leakage capacity of these channels if the required statistics are available.

6.5 Experimental Setups and Results

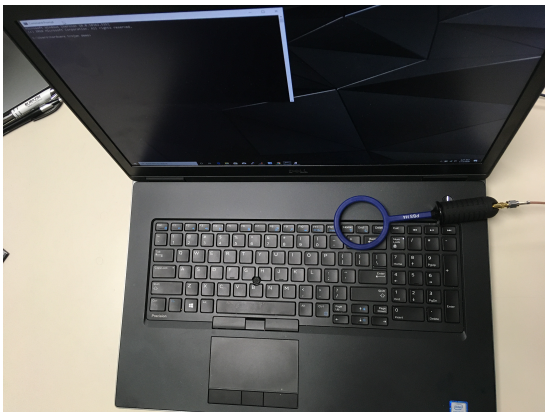
In this section, we first provide signaling time distributions for various covert channels to demonstrate that assuming normal distribution with a specified mean and standard deviation is a valid assumption, and then provide the capacity results for these channels.

For the experiments, the devices we consider are an Altera NIOS-II processor with a commercial Terassic DE1 SoC board [85], an OlinuXino board [87] which has a modern Cortex A8 ARM core with two levels of caches, 4 MB main memory that is commonly used in factory lines, etc., and a Dell Precision 7730 laptop [97]. The antennas to collect emanated signals can be listed as a high-gain custom-made disk-array based antenna [89], near EM field probes [98], a power rail probe [99], a horn antenna [88] and a lab-made near field probe. We record the signals using a spectrum analyzer (Agilent MXA N9020A) [100].

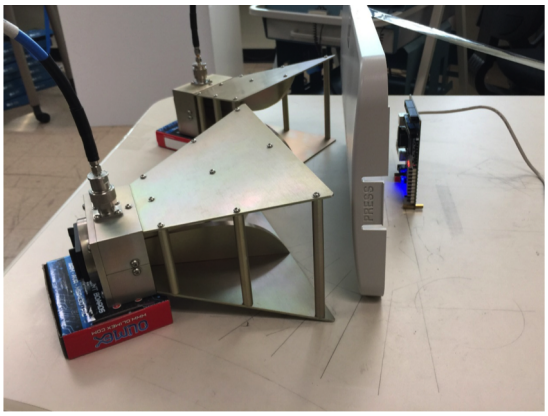
The first goal of these experiments is to collect data while transmitting



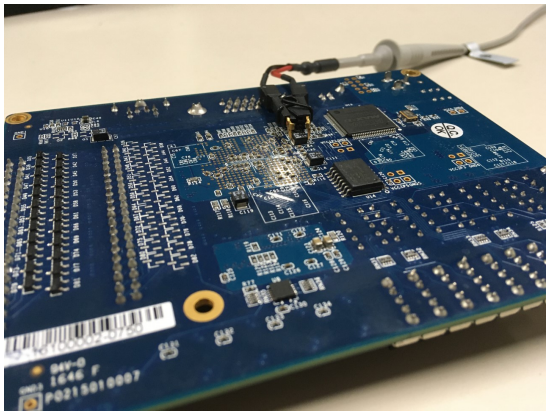
(a) EM based covert channel [19].



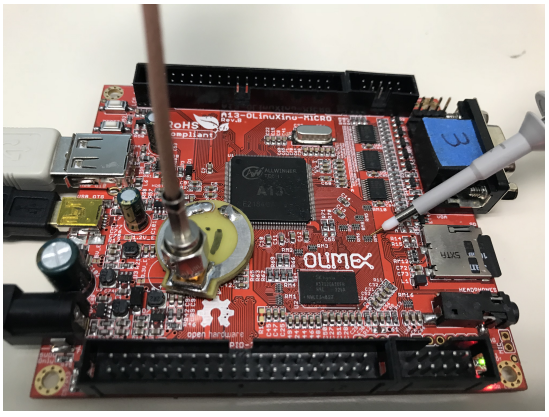
(b) Based on power unit management [5].



(c) Backscatter covert channels [95].



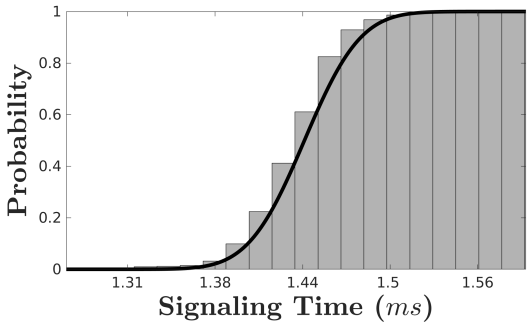
(d) Power covert channels [29].



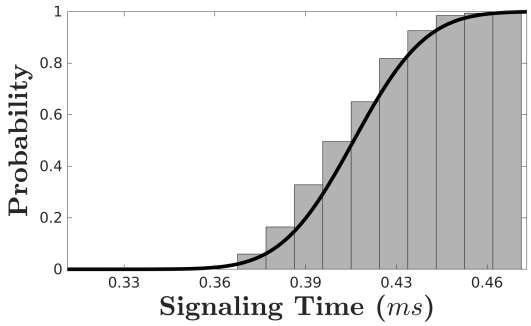
(e) Cache based covert channels [37, 96].

Figure 6.9: Experimental setups for the measurements.

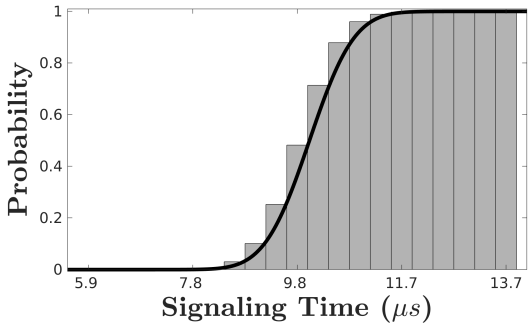
bits to experimentally obtain the distribution of T_x for both bits. In that respect, we follow the experiments done in [3] for EM, [5] for power unit, [95] for backscattering, and [37, 96] for cache-based covert channels. For the



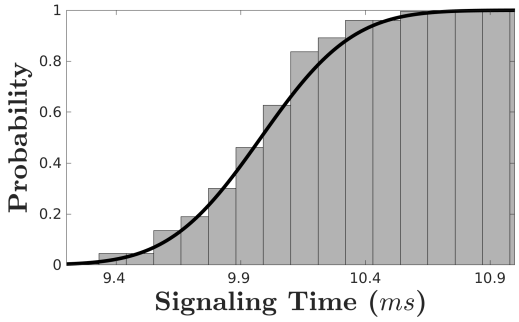
(a) EM based covert channel [19].



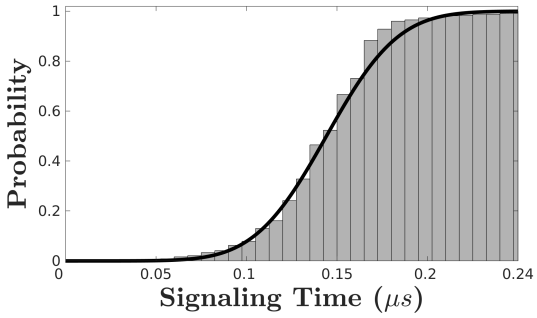
(b) Based on power unit management [5].



(c) Backscatter covert channels [95].



(d) Power covert channels [29].



(e) Cache based covert channels [37, 96].

Figure 6.10: Distributions for the signaling time for various covert channels.

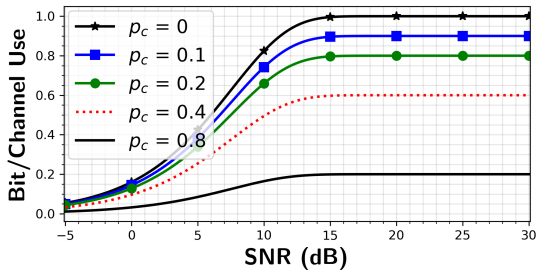
power covert channel, we collect the signal from a capacitor while running the code given in [69] and following the attack scenario given in [29]. The setup for all these measurements are given in Fig. 6.9. For more accurate distribution results of T_x, we collect signals closer to the device under inspection, and then perform an edge detection to obtain the width of the pulses. Empirical cumulative distribution functions (CDF) of bit-1 for vari-

Table 6.1: Parameters utilized for the leakage capacities for covert channels.

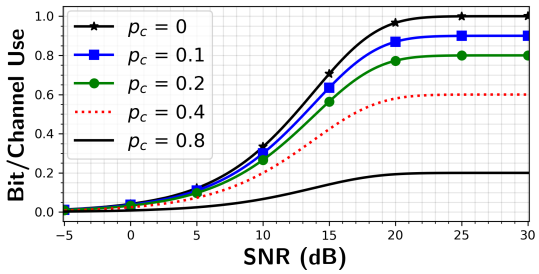
Parameter	**Power**	**Power Unit**	**EM**	**Backscatter**	**Cache**
$\mathcal{T}$	50	0.5	3	20	2
T^1	10	0.4	1.5	10	0.15
T^0	0	0.2	0	0	0
μ_0	0.12	0.03	0.14	1.02	≈ 0
μ_1	-0.02	0.02	-0.05	-0.01	-0.1
σ_0	0.05	0.01	0.03	0.66	≈ 0
σ_1	0.29	0.02	0.02	0.66	0.04
Unit	ms	ms	ms	μs	μs

ous covert channels are given in Fig. 6.10. Furthermore, in this figure, we provide CDF of normal distributions that are fitted to these data. As seen from these figures, the signaling time distributions have a Gaussian characteristic, which means the assumption in Section 6.2 holds. Actually, the assumption is also supported by Law of Large numbers [101] because of ubiquitous software activities.

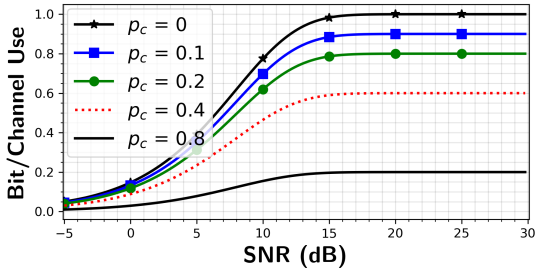
The experimental variables required for the leakage capacity calculation for the experiments are given in Table 6.1. Here, σ_0 and σ_1 represent standard deviations of the signaling time (σ_x) when bit-0 and bit-1 are transmitted, respectively. Since the performance of these covert channels, in terms of bandwidth, noise characteristics, etc., are different from each other, we choose different transmission time for more reliable results. For example, in the literature, the reported transmission rates for various covert channel attacks vary from 5 bit/s to a couple of kbits/s [102, 4, 56, 5].



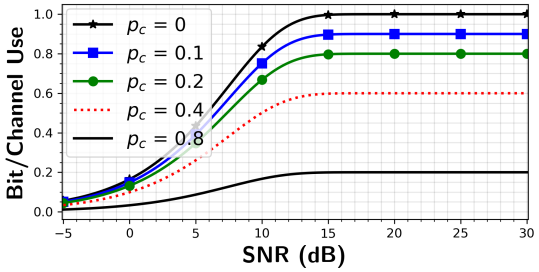
(a) EM based covert channel [19].



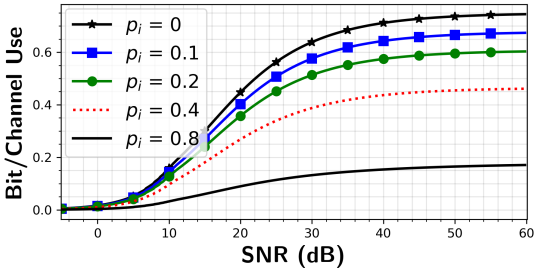
(b) Based on power unit management [5].



(c) Backscatter covert channels [95].



(d) Power covert channels [29].

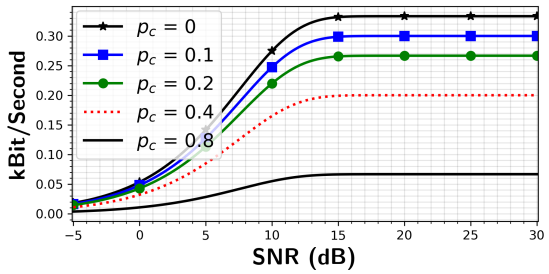


(e) Cache based covert channels [37, 96].

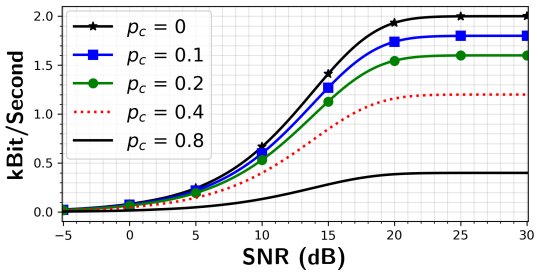
Figure 6.11: Bit/Channel Use for various covert channels.

The leakage capacities as the result of the optimization problem given in Section 6.3 are provided in Fig. 6.11 and Fig. 6.12. In the first figure, we provide the results in terms of Bit/Channel-Use [15] to show whether maximum gain from each bit transmission can be achieved at any SNR. The figures contain behavior of maximum leakage as the sum of deletion and insertion, p_c, changes. For a fair comparison of channel capacity for various deletion and insertion probabilities, SNR is defined as

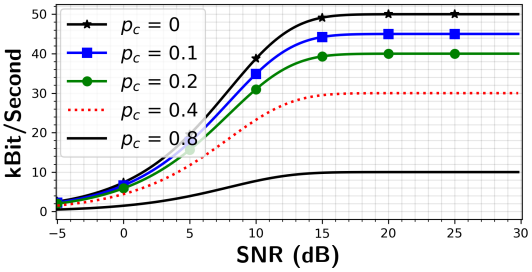
$$\text{SNR} = \frac{\mathcal{A}^2 \left((T^0)^2 + (T^1)^2 \right)}{2\mathcal{T}\sigma_n^2}. \tag{6.18}$$



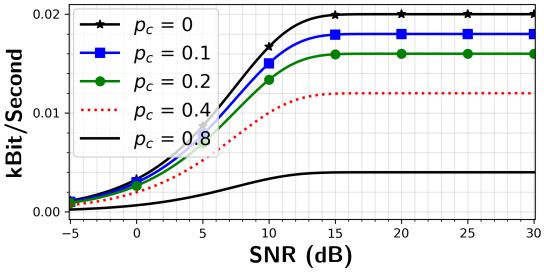
(a) EM based covert channel [19].



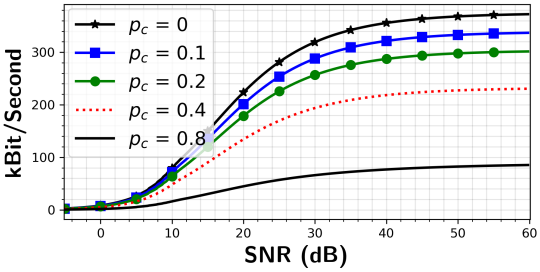
(b) Based on power unit management [5].



(c) Backscatter covert channels [95].



(d) Power covert channels [29].



(e) Cache based covert channels [37, 96].

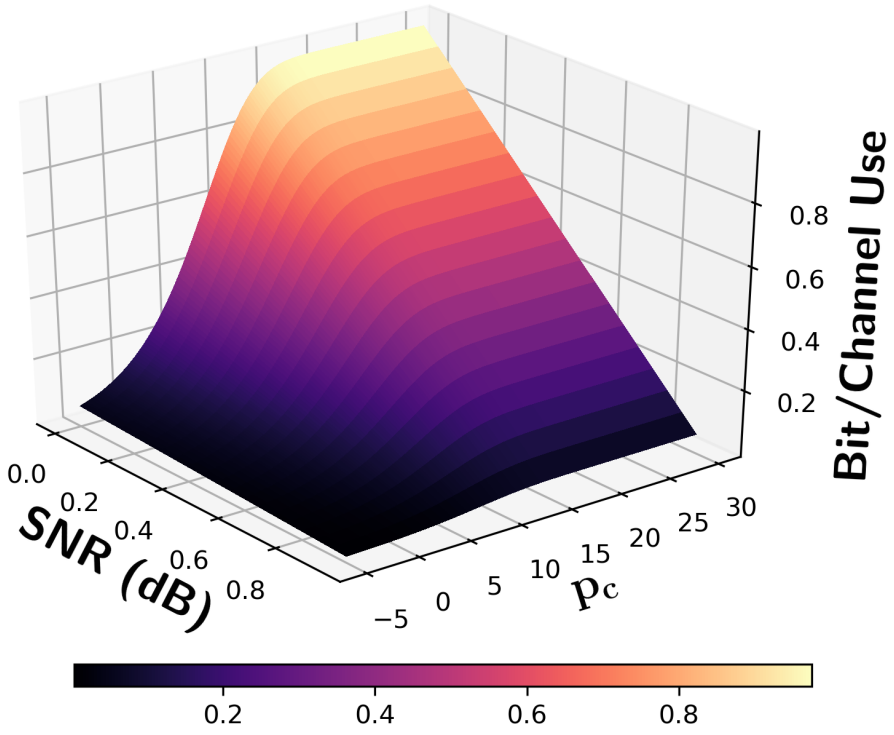
Figure 6.12: Bit per second (Bps) for various covert channels.

We observe that the maximum gain is only possible if there is no insertion and deletion, and the communication takes place in high SNR, which is an unrealistic scenario because of *unintentional* nature of covert channels. However, it does not mean that systems are secure enough against these channels. If the attacker can establish a longer connection, even transmission with slow data rate could be a disaster. For example, if SNR is 5 dB, and $p_c = 0.8$, a communication with at least 0.1 Bit/Channel-Use could be possible for any given covert channel. Considering the attacker aims to steal some passwords, credit card information, etc., even this rate

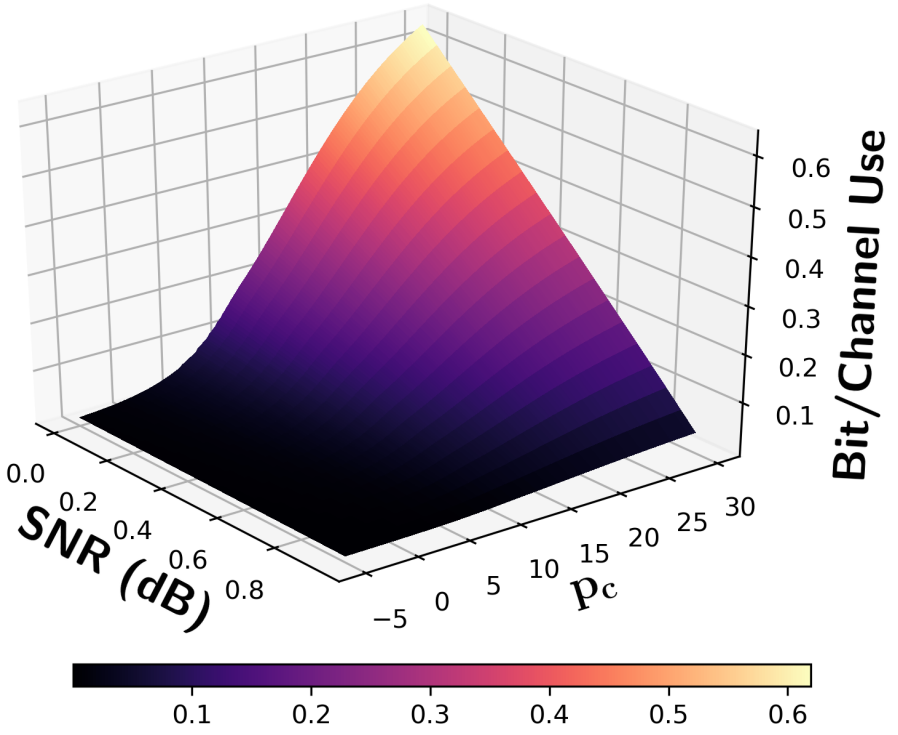
could be severe enough. In the second figure, we provide the results in terms of Bit-per-second (Bps). The goal of providing this result is to show that although leakage capacity is small in terms of Bit/Channel-Use, hundreds of information bits can be transmitted in a second through these channels. For example, when Fig. 6.11(e) is compared with the rest of covert channels, we can conclude that it is the most inefficient channel for an attacker. However, Fig. 6.12(e) demonstrates that this channel can achieve higher data rates than others.

Another interesting observation here is that although all but cache-based covert channels achieve almost the maximum gain in terms of Bit/Channel-Use for high SNR, the cache-based covert channel converges to 0.6 Bit/Channel-Use. The reason is higher signaling time deviation when a cache-miss occurs. This introduces powerful jitter noise to the system, which could not be removed even the attacker measures the signal when SNR=∞. This result shows that the signaling time variation causes an additive noise which decreases the transmission rate further.

Finally, in Fig. 6.13, we provide the same results with Fig. 6.11 for backscattering and cache-based covert channels to observe the behavior in a 3-D surface plot. These figures reveal the general behavior of the worst-case leakage scenario through different covert channels. As said before, we observe that the characteristic of the cache-based covert channel is different than other covert channels. Here, as the representative of other covert channels, we provide the result for backscattering covert channels. Our observation here is that the decrease in the leakage capacity of cache-based covert channel is sharper in both p_c and SNR directions than the backscattering channel due to jitter noise. However, both figures illustrate the possibility of severe information leakages through covert channels.



(a) Backscatter covert channels [95].



(b) Cache based covert channels [37, 96].

Figure 6.13: Bit/Channel Use while p_c and SNR vary.

Finally, we compare our capacity results with the results given in [21]. In that paper, capacity bounds for the leakage capacity are provided. To compare our results with these bounds, we need to assume there is no deletion, the signal deviation for both bits are same with zero mean because the bounds are obtained when there is no deletion, and the signaling variation has zero mean with fixed standard deviation irrespective of bits. In Fig. 6.14, we provide the upper and lower bounds in [21], and the proposed leakage capacity for EM covert channel. We observe that our leakage capacity lies within the region defined by the bounds, and closer to the upper bound. We conclude that the decrease in the leakage capacity is much less than the decrease in the lower bound in [21] as the insertion probability increases.

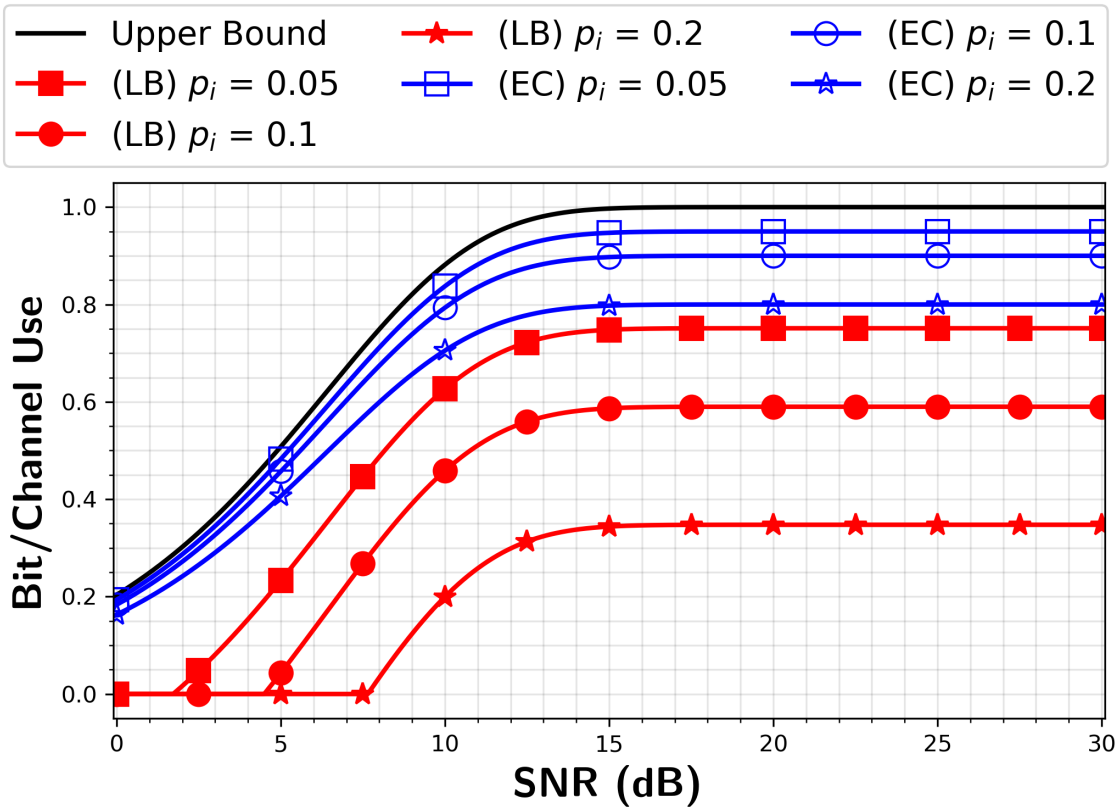


Figure 6.14: The proposed leakage capacity and bounds given in [21] while SNR changes.

Although all these results demonstrate the possible threat through these covert channels, the methodology in this chapter can be utilized by designers to make their systems more resilient to covert channels. In the *design-stage*, designers can collect the statistics for p_c, and estimate signal power that can be generated by any covert channel attack. Then, SNR vs. leakage capacity analysis can be done by solving the optimization problem given in Section 6.3. If the leakage is zero or very close to zero at the targeted SNR, they can conclude their system is secure enough. Otherwise, they need to modify their design to protect privacy of their customer.

6.6 Summary

As the systems get more mobile, the likelihood that an attacker and a victim to be in the same place increases. Although there are many defense mechanisms, attacks based on covert channels can circumvent and break the existing protection because these channels are a consequence of computing, and not intended for a conventional communication.

In this chapter, we proposed a methodology to estimate the worst-case information leakage through various covert channels. We showed that the method can be adopted to both analog and digital covert channels. To mimic the losses due to software activities, we first modeled the communication channel as a deletion-insertion channel. Then, we introduced the jitter noise that is an extra source for additive white noise. This jitter noise is a result of signaling time variation due to stalls, interrupts, optimization, etc., which conventional communication systems do not suffer. We showed that these noise sources can be combined and called effective additive noise. Secondly, based on the effective noise, we modeled the communication channel between the receiver (an attacker) and transmitter (a

victim). Then, we defined the channel capacity as the maximum leakage for a given covert channel. Finally, we provide experimental results for various covert channels to show that the proposed model is an effective and a general method to attain the resilience of a given system.

CHAPTER 7

A MICROARCHITECTURE-LEVEL MODELING ELECTROMAGNETIC SIDE-CHANNEL SIGNALS

7.1 Overview

Side-channel attacks have become a serious security concern for computing systems, especially for embedded devices, where the device is often located in, or in close proximity to, a public place, and yet the system contains sensitive information. To design systems that are highly resilient to such attacks, an accurate and efficient design-stage quantitative analysis of side-channel leakage is needed. For many system properties (e.g., performance,power, etc.), cycle-accurate simulation can provide such an efficient-yet-accurate design-stage estimate. Unfortunately, for an important class of side-channels, electromagnetic emanations, such a model does not exist, and there has not even been much quantitative evidence about what level of modeling detail (e.g., hardware, micro-architecture, etc.) would be needed for high accuracy. Please remember that EM side-channel signals are created due to *bit-flips* at the transistor-level [26, 3]. In principle, all transistors and metal-layer interconnect components contribute to the signal, thus the signal could be modeled using all transistors and on-chip wires as predictor variables, which *should* be highly accurate but is practically infeasible. As a result, the main challenge in model-building is to select (or discard) potential predictors in a systematic manner, to achieve a trade-off where feasibility (or even efficiency) is achieved without a major sacrifice in accuracy.

To achieve a simple yet accurate model, existing methods are mainly focused on individual instructions and their operands, and they attribute an "average" behavior to each of the instructions, rather than model its cycle-by-cycle effect on the processor's hardware. In effect, these methods model a simplified *one-instruction-at-a-time* implementation, essentially ignoring pipeline effects and other important aspects of the micro-architecture.

In this chapter, we introduce *EMSim* which is a tool to simulate emanated EM signals from devices [23]. For more accurate EM signal simulation, we model micro-architectural components as ***independent*** sources of EM emanations and then further group these units in each pipeline stage as an individual source. We used pipeline stages as the sources mainly because we observed that each instruction has different footprint in each cycle, and the side-channel generated at each cycle is a combination of these activities in *ALL* stages. Using this methodology, we model a multi-input (pipeline stages), single-output (EM signal) system (MISO).

Leveraging this approach, the main contributions of this chapter are to

- Model the signal for individual sources,
- Establish a connection between MISO systems and mixture of signals due to execution of instructions at pipeline stages.

In that respect, we first describe the experimental methodology for obtaining EM signals generated by actual hardware, introduce the single instruction signal model, provide the MISO model for the signal mixtures, and finally demonstrate experimental results which show great consistency with the theoretical work.

7.2 Experimental Methodology for Signal Acquisition

While signals can be collected from an actual off-the-shelf processor, such a processor would be difficult to model in detail because many of its microarchtiectural details (of even entire microarchitectrual blocks) are not publicly available, and unceratinty about how well the model used in *EM-Sim* matches the actual microarchitecture would make direct comparison of real and simulated signals difficult. Therefore, we implement a RISC-V [103] processor on an FPGA (using Verilog), giving us full knowledge of the actual microarchitecture of the processor [23]. The designed processor has five pipeline stages namely: `Fetch`, `Decode`, `Execute`, `Memory`, and `Writeback`.

Using the implemented processor, we then use a magnetic probe and a signal acquisition device (an oscilloscope) to receive and record the EM signals (more details on Section 7.5.1).

7.2.1 Signal Acquisition

Capturing the emanated signal is the first step to properly modeling the signal. Unfortunately, measuring the ideal emanated signal (i.e., not corrupted by additive channel white noise) is not possible if only one-time-run of any instruction sequence is considered. One option is to collect many one-time-run signals and take the average. The problem with this approach is capturing synchronized signals, i.e., the starting points of captured signals may not correspond to the same processing-time of the given instruction sequence. To address this problem, we use a novel signal-processing method called "*modulo operation*" [67] to create a highly accurate estimate of the ideal emanated signal, i.e., to remove most of the

noise and distortion that was present in the actual signal due to under-sampling, synchronization, noise, and other imperfections that are unavoidable during practical collection of signals (Please see Appendix G).

The main parameters for the *modulo operation* are the number of clock cycles to execute a given sequence, `noc`, the sampling-rate of the instrument, and the clock frequency of the device. After having these parameters, the next step is to collect the emanated signal. For that, a given sequence is executed several times (1000 times in our measurements). Each set of measurements consists of a sequence of instructions (called `sequence`). The goal is to retrieve the emanated signal for the `sequence`.

The next step is to utilize the *modulo operation* to map each received samples to average these many measurements. Assuming T_s is the total time to execute the sequence once (i.e., $T_s = \texttt{noc} \times T_{clk}$), it applies the following operation to the sampling time of each sample to map each sample to its fundamental period:

$$\Delta_m = mod(T_m, T_s), \tag{7.1}$$

where T_{clk} is the clock time, Δ_m is called the *modular offset*, and T_m is the sampling time of m^{th} sample. After obtaining the modular offsets for each sample, the *modulo operation* takes the mean of the samples that have same modular offset, to produce the desired signal. Further signal-processing techniques such as moving average, Gaussian filtering, etc., can be applied to this generated signal to obtain smoother reference signals.

7.3 Signal Reconstruction

Simulating an analog signal can be considered a signal-processing reconstruction problem where a continuous signal (i.e., EM in this case) needs to be determined from a sequence of equally-spaced samples with sampling-rate T, where T is preferably much smaller than T_{clk}. Such a signal can be ideally reconstructed using Whittaker-Shannon interpolation formula [104], however, it is well-known that such a method is not feasible in practice. Instead, a popular method for signal reconstruction is zero-order hold (ZOH) technique where a continuous signal, y, can be reconstructed from a sample sequence $x[n]$, assuming one sample per time interval is T:

$$y(t) = \sum_{n=-\infty}^{+\infty} x[n] \times rect\left(\frac{t - T/2 - nT}{T}\right). \tag{7.2}$$

To improve the ZOH accuracy, in this chapter we make an observation that *switching activity in a processor is synchronized to its clock* and most of the switching happens right after the positive/negative edge of the clock. Thus, instead of using a rectangular function (which implies that activity is evenly spread over a cycle), a decaying function can be used:

$$f_1(t) = e^{-\theta t}\mathtt{u}(t), \tag{7.3}$$

where θ is a positive normalization factor that changes the width of the signal, and $\mathtt{u}(t)$ is the unit step function. Substituting $rect()$ in Equ. 7.2 by the exponential we get:

$$y(t) = \sum_{n=0}^{+\infty} x[n] \times e^{-\theta(t-nT)} \times \mathtt{u}(t - nT). \tag{7.4}$$

However, we observed that the received signal is also exposed to oscilla-

tions with decreasing magnitude. To meet the requirements for both oscillations and decreasing amplitude, combining sinusoidal with exponential can increase the accuracy further:

$$f(t) = \sin(2\pi t/T_0) \times e^{-\theta t} \times \mathtt{u}(t), \tag{7.5}$$

where T_0 is the periodicity of the sinus function. Again, substituting $rect()$ in Equ. 7.2 by $f(t)$, we will have:

$$y(t) = \sum_{n=0}^{+\infty} x[n] \sin(\frac{2\pi(t-nT)}{T_0}) e^{-\theta(t-nT)} \mathtt{u}(t-nT). \tag{7.6}$$

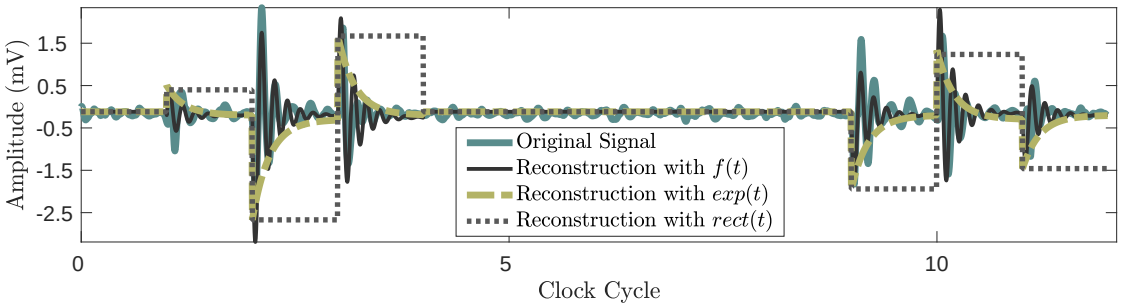


Figure 7.1: Reconstructing the original signal using three different approaches. Using a combination of a sinusoidal and an exponential function ($f(t)$ in Equ. 7.5) can achieve the best accuracy.

In Figure 7.1, we plot a measured signal and its reconstructions with these options. We observe that $f(t)$ explains the behavior of the received signal much better. Thus, by finding $x[n]$ for each cycle and using Equ. 7.6, the analog side-channel signal can be modeled.

7.4 EMSim Modeling

7.4.1 Signal Amplitude for Individual Sources

In practice, there are two contributors in creating EM side-channel signals for each pipeline stage. The first group of contributors, which we call *instruction-dependent* activities, are caused by the switching activities of micro-architectural units (e.g., register-file, ALU, etc.) that are utilized in that stage (e.g., whether the register-file is being written or not).

The second group, *data-dependent* activities, are created due to bit-flips on the data-bus, address-bus, and any other registers that hold operand's values. These bit-flips are independent from the instruction-type but are dependent to the previous state of the bus. In the following, we will describe how we *independently* measure each of these two groups.

Instruction-Dependent Activities. To independently measure these groups, we first minimize the effect of the *data-dependent* activities by setting all the operands, addresses, and immediate values to zero. This approach enables us to measure the *baseline* signal for each stage which is *only* created by the switching activities of the micro-architectural units used in that stage.

After decoupling the data-related activities from the signal, the second challenge is to minimize the effects of other stages on the generated EM signal. Recall that we mentioned *ALL* pipeline stages contribute to the overall signal, however, ideally we want to be able to measure the effect of each stage *separately* so that we can use them as the basic-blocks to reconstruct the overall signal. To achieve that, we use `NOP` instruction as the *baseline* since it has the minimum possible switching activity, and

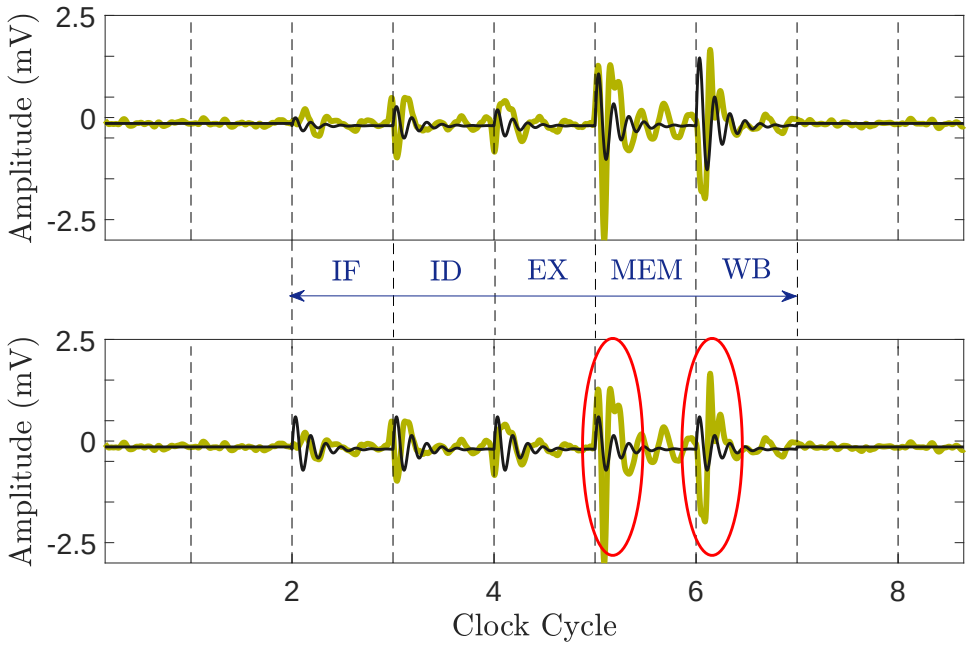


Figure 7.2: The signal amplitude for an ADD as it progress in the pipeline (while all other instructions are NOP). The actual signal is shown in light color (green). Darker color (black) shows the simulated signal when considering each pipeline stage as a separate source (top), and when considering the entire processor as a single source (bottom), and the largest differences between the two are pointed out using red ellipses.

then create NOP $\rightarrow$ inst $\rightarrow$ NOP instruction sequence (for all instructions), while operands for inst are all set to $r1$ (and $r1 = 0$). Using this method, no data/operand-dependent bit-flips are created, but register-file, ALU, etc. may be used (depending on the instruction type). We then measure the signal amplitude for all instructions and every pipeline stages. We call this ***baseline hardware amplitude*** or A.

Figure 7.2 shows how the (actual) EM signal (shown in green/light color) changes as an ADD instruction progresses through the pipeline while all the other instructions are NOP. Using Equ. 7.6 and NOP$\rightarrow$ inst$\rightarrow$ NOP instruction sequence, we used our simulator to generate the signal. Further, to show why individual stages should be modeled separately, Figure 7.2 (bottom) shows the simulated signal when the "average" amplitude

is used for all stages. As can be seen, failing to model each stage individually (as used in previous work [105]) can lead to significant inaccuracies in some stages (note that using *max* instead of *average* also leads to similar inaccuracies).

Data-Dependent Activities. Once the baseline amplitude is measured, the next step is to find how this amplitude changes as the number of bit-flips changes due to value/operand used in the instruction and the previous state of the bus/register. Intuitively, the more bit-flips, the higher the amplitude should be thus we define *activity-factor*, α, as a ***scaling factor*** to the baseline activity, A. To find α, we first treat each bit-flip *equally*, and assume that each bit-flip has similar effect on the signal amplitude. We then calculate α as:

$$\alpha = 1 + \frac{(\mathit{flips}_{new} - \mathit{flips}_{base})}{\mathit{flips}_{total}}, \tag{7.7}$$

where flips_{new} is the total number of flips for the current instruction, flips_{base} is the total number of flips when previous instruction is `NOP`, and flips_{total} is the maximum possible number of flips for the current instruction. Using this equation, we then define $A' = \alpha \times A$, and use it to simulate the signal. Figure 7.3 (bottom) shows the original signal (shown in light green), and the simulated signal using this approach (shown in black) for the similar `NOP`$\rightarrow$ `inst`$\rightarrow$ `NOP` instruction sequence discussed in the previous section. As can be seen in the figure (bottom), this *"averaging"* modeling can not accurately predict the amplitude of the signal which indicates that *not all the bit-flips have the similar impact on the amplitude*. Our further investigation confirmed this theory. Particularly, we found that flips in the output of

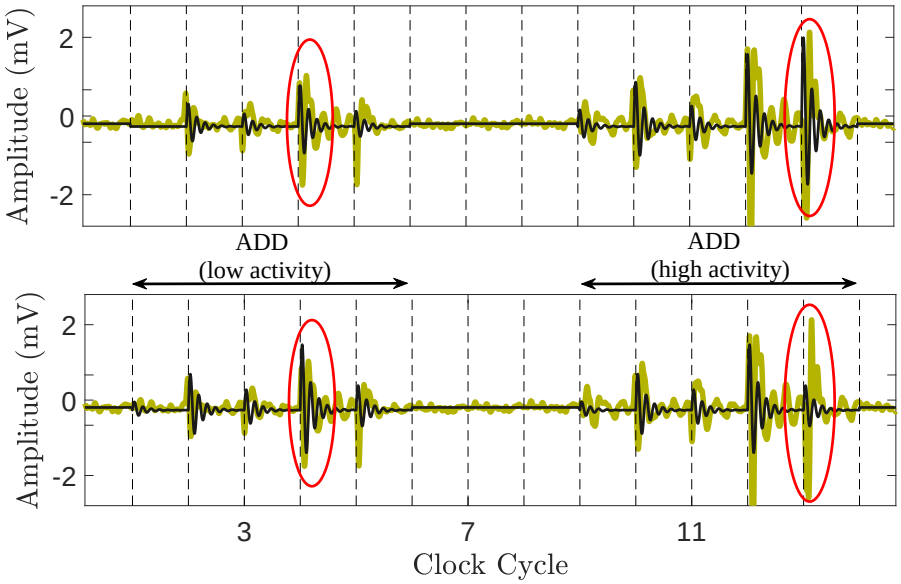


Figure 7.3: Effect of the *activity factor* on the amplitude. The actual signal shown in green. The simulation is shown in black when activity factor is modeled using a linear regression model (top) and when an *average* activity is used (bottom).

the ALU and memory have the most significant impacts on the signal. We believe this difference is mainly due to the different physical parameters of transistors and/or lengths of the connecting wires.

Using this observation, to systematically calculate the activity factors, we use a *linear regression* model:

$$\alpha = \delta + \mathcal{T} \times c + \epsilon, \tag{7.8}$$

where $\mathcal{T}$ is a vector of transition bits across all the existing registers in the targeted pipeline stage, δ and ϵ are the vector of scalar intercept and error terms respectively, and c is the vector of activity factors to be predicted by the model. As mentioned before, α is the scaling factor for the baseline amplitude, A, thus $\alpha = A_{meas}/A_{simul}$. Note that to find $\mathcal{T}$, a detailed micro-architecture model is needed to track all the bit-flips for every

gate in the processor (except cache/memory). However, to significantly reduce the complexity and simulation time, the size of $\mathcal{T}$ can be reduced using the step-wise regression method [106] where, iteratively, the size of the fitted model (i.e., α and $\mathcal{T}$ in our case) is reduced using standard statistical metrics such as F-tests [106]. In other words, since *not all the bit-flips have statistically significant impact on the emanated signal*, the non-contributing factors can/should be removed from the model. In our processor, using this method we managed to reduce the size of $\mathcal{T}$ by more than 65%.

Figure 7.3 (top) shows the simulated signals when the linear regression (LR) model is used for activity factors. Compared to the averaging method (bottom), using LR has significantly improved the simulation accuracy.

7.4.2 Multi-Input Modeling

Once the signal amplitude for individual sources are calculated, the next step is to combine the signals generated by these individual sources to create the simulated EM signal. In principle, the generated EM signal is the *superposition* of individual waves thus depending on each source's phase, the superposition of each pair can be either *constructive* or *destructive*. Using this fact, the overall signal can be approximated as a *linear* combination of these individual sources where the coefficients may vary between ± 1, depending on the phases.

Due to the complex nature of the generated EM signals, accurately modeling each and every source mathematically is significantly time-consuming and often infeasible in practice. To tackle this problem and find coefficients for each source, a *model-fitting* approach can be used. We use a *linear-regression* model to find (predict) the overall EM signal. Specifically,

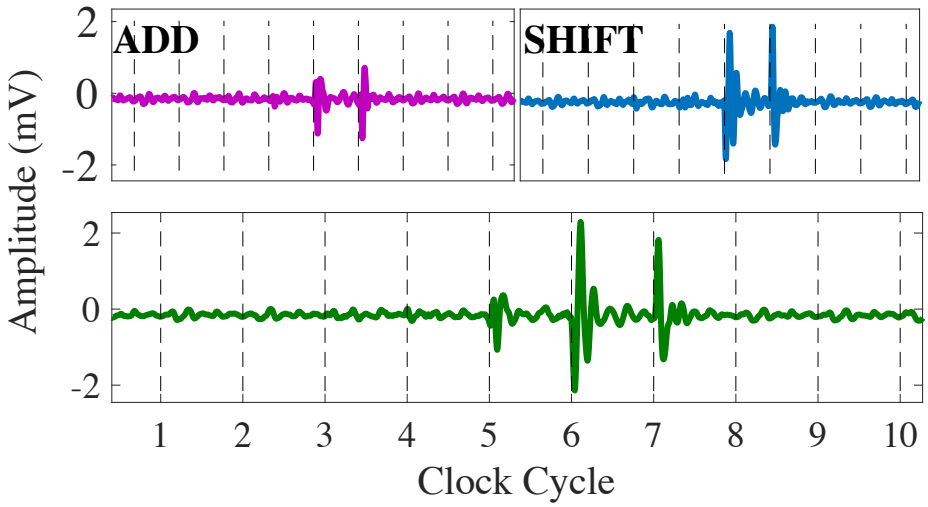


Figure 7.4: An example of how individual sources (pipeline stages) are combined to form the final signal. ***Top***: how the actual EM signal looks like when the instructions are executed in isolation (`NOP, inst, NOP`). ***Bottom:*** The actual EM signal when the instruction sequence is `NOP, ADD, SHIFT, NOP` (i.e., a combination of multiple instruction in the pipeline).

we use:

$$X = \delta_s + (\alpha A) \times M + \epsilon_s, \tag{7.9}$$

where αA is the vector of individual sources amplitudes (α is the activity factor and A is the baseline amplitude), δ and ϵ are the intercept and error vectors, M is the predicted coefficients, and X is the final amplitude which will be used in (7.6) to simulate the signal.

Figure 7.4 shows an example of how two individual sources are combined in each cycle to form the final signal. Figure 7.4 (top) shows `ADD` and `SHIFT` instructions when they are executed in isolation (i.e., `NOP, inst, NOP`), and Figure 7.4 (bottom) shows how the final signal looks like when the executed sequence is `NOP, ADD, SHIFT, NOP`. Specifically, cycle 6 is when the `ADD` instruction is in `WB` stage and `SHIFT` is in `MEM`, and the resulting signal is a linear combination of these two sources. Note that to find M, we need to measure all the possible combinations of the entire

instructions in the ISA, however, as we will show in Section 5, the number of required measurements can be significantly reduced using standard *clustering* algorithms.

7.5 Evaluations

We divide our evaluations into two main parts. First to show the correctness, accuracy, and robustness of our simulator, we present our experimental evaluations on how well the simulated signal *matched* with the original side-channel signal generated by the target hardware for *ALL* possible combinations of the instructions. We then explore the impact of variations such as manufacturing, environmental, etc. on the accuracy of *EMSim*.

The second part of our evaluations (presented in Section 7.6.2) is focused on the *EMSim* use-cases and its application in different domains such as security, debugging, etc.

7.5.1 Evaluating Model Accuracy

Setup. We implemented a RISC-V based processor on a Terrasic DE0-CV board with an Altera Cyclone-V FPGA [107] with 50 MHz clock-rate. To record side-channel signals, we used a Keysight digital oscilloscope (DSOS804A), with 1 GHz bandwidth and 10 GSa/s rate. We further studied the effect of changing the sampling-rate on the accuracy and found that similar accuracy can be achieved with much lower sampling-rate (about 200 MSa/s in our measurements). As a result, similar results can be achieved using a less expensive device (e.g., TBS1032B Tektronix Digital Oscilloscope [108] costs around $300) and/or a high sampling-rate device can be used for modeling devices with faster clock-rates. To receive EM

Cluster	Type	Inst.	No. Inst.
1	ALU	ADD,XOR,JAL, ...	13
2	Shift	SLLI,SRT, SRA, ...	10
3	MUL/DIV	MUL, DIV, REM, ...	8
4	Load	LB, LW, LH, ...	5
5	Store	SB, SH, SW	3
6	Cache	LB, LW, LH, ...	5
7	Branch	BEQ, BLT, BGE, ...	6

Table 7.1: RISC-V (R32IM) instruction-set and their cluster used in this chapter.

signals, we used a magnetic probe [98], placed 5 cm above the FPGA.

Model Building. In Section 7.3, we discussed that in order to fit a model, *ALL* possible combinations of instructions should be measured (i.e., about three hundred million combinations in RISC-V ISA). Clearly such a requirement makes the model building extremely time-consuming in practice. However, intuitively, we expect instructions with similar behaviors (e.g., ALU-type, memory-type, etc.) have similar side-channel signals since they share identical hardware activities. Using this intuition, we used the *hierarchical agglomerative algorithm* [109] with the *cross-correlation* as the distance metric to cluster instructions with similar EM pattern into a same cluster. We found that RISC-V ISA can be *clustered* into 7 categories (when the operands are similar) where a single instruction in each category can be a representative of all instructions in that category.

These categories are shown in Table 7.1. Using this table, we then used only a *representative* instruction of the cluster for model building which, in turn, reduce the model building complexity significantly. In our setup, the number of measurements was reduced from 300 million to only 16 thousands. Note that while the clustering algorithm did not use the micro-architecture model as a prior knowledge, the clusters confirmed that instructions with similar micro-architecture activities should be clus-

tered in a same group.

Metric. To measure how well the simulated signals "match" with the real signals, we leverage *normalized cross-correlation* as our metric. To compute that, we first normalize both signals, real and simulated, to have similar average. We then divide each signal to individual clock cycles, and then compare each cycle (between the simulated and the real signals) using cross-correlation as the distance metric. We then define *accuracy* as the average of this cross-correlation across all cycles for all measurements (i.e., we were able to match the waveform in this degree across all possible instruction sequences). Note that we specifically used this approach to show how well the time-domain signal *matches* with the original signal instead of relying on a specific *leakage metric* such as Hamming weight. However, to show the usefulness and versatility of our tool, those results will be shown in the next section.

Benchmark. To prove that our approach provides accurate simulated signals for *ALL* possible instruction combinations thus can be applicable to *ANY* complex program that uses the mixture of the implemented ISA (R32IM), we created a microbenchmark using all possible combinations of the representative instructions shown in Table 7.1. Particularly, for a 5-stage pipeline and 7 distinct clusters, there are $7^5 = 16807$ possible combinations that can appear together (in the pipeline) in a cycle. We created a program to generate all these combinations with random operands. We then manually modified branch instructions and assigned the target address and branch condition to create loops with random instruction and iteration sizes. To limit the execution time, we then randomly put these instructions into groups of 1024 combinations (i.e., 5120 instructions in

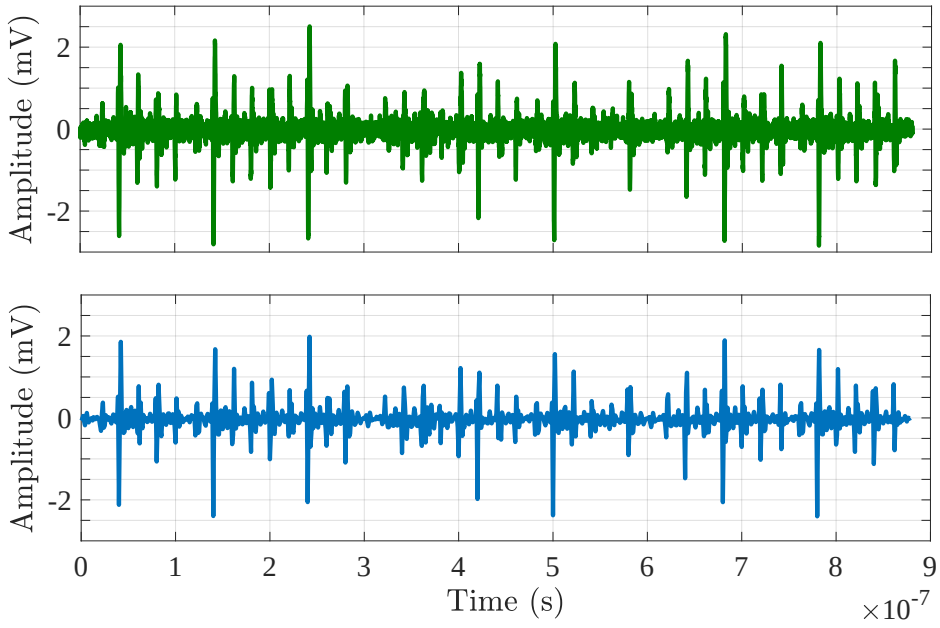


Figure 7.5: A comparison between the signal generated by a real hardware (top) and the simulated signal (bottom) in EMSim.

each group which were executed one after another similar to a real program). To cover all the combinations, 17 of such groups were needed (no two groups were similar). We then executed these randomly-generated groups on the processor normally, and recorded the real and simulated signals. To further prove the validity and correctness of our simulator, we also randomly created another 17 groups, this time from all instructions in the ISA and not just the representatives.

Results. Using these 34 groups/applications described above, we then compared the simulated signals with the actual ones using the our metric defined earlier. Each group/application takes about 9000 cycles to finish on average. The execution-time varied depending on the instructions used and microarchitectural events.

Figure 7.5 shows the simulated and actual EM-side-channel signals

for one of the groups tested in our evaluation (for clarity, only the first 50 cycles are shown in the figure). As can be seen from the figure, the simulated signal matches the real signal with high accuracy. We found that, on average, ***EMSim has about 94.1% accuracy in simulating side-channel signals across all possible instruction combinations***.

7.5.2 Effects of Distance

Transferring the ideas from communication theory literature, to find the effect of the distance (i.e., the position of the probe and its distance to the center of borad) on each source, a parameter, called ***loss-coefficient*** or β, can be considered as the channel coefficient of a flat-fading channel. Here, we need to note that, regardless of the position of the probe, the baseline amplitude, A, can not be measured solely because we do not have any control on the power distribution of the board for each instruction at each pipeline stage. Hence, *the resulting signal power is always a combination of the actual signal amplitude and the corresponding loss coefficient (i.e., $A\beta$)*. However, to deal with this problem, we choose the probe's location at the center of the processor as the *base point*, and define β_0 as the loss coefficient at this point. Further denoting A_0 is the actual emanated signal amplitude, we assume the amplitude of the signal can be written as $A = A_0\beta_0$ with respect to the base point. Therefore, with these assumptions, β is assumed to be one for all the measurements done in this section.

To further investigate the effect of β on the amplitude, we measured the signal (with the same input trace) at a different location, and compared the results with the base case. Figure 7.6 illustrates the effect of the antenna location on the loss coefficient factor β. Here, the training signals for the reconstruction are obtained from the base measurement, and the figure

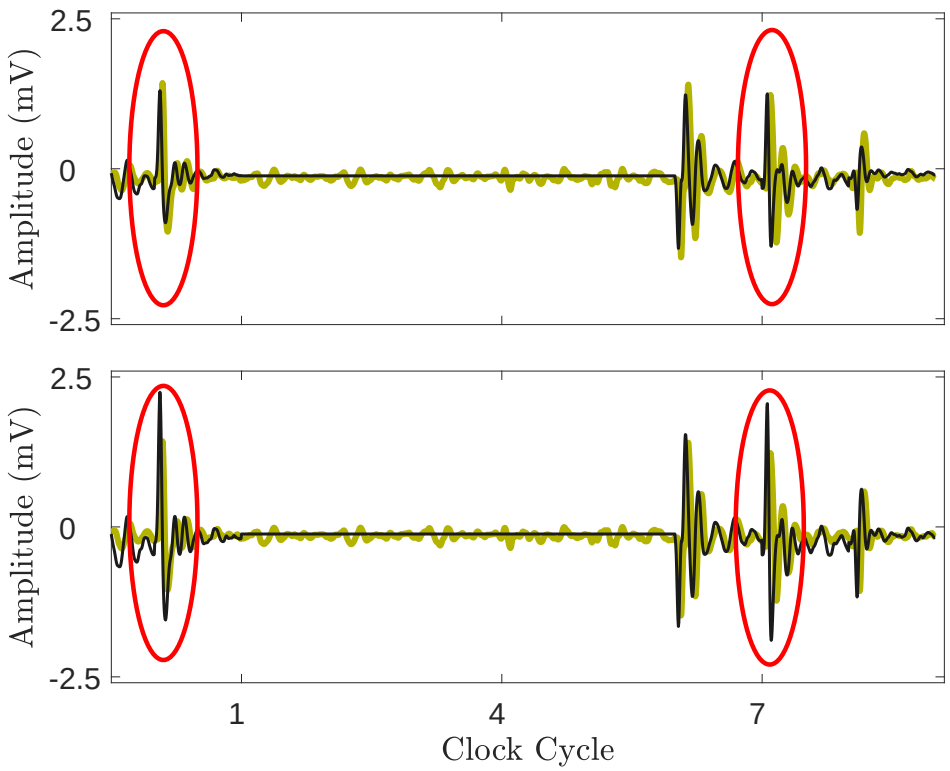


Figure 7.6: Effect of distance on the signal amplitude. For both figures, the plots with darker color correspond to reconstructed signal, and the other ones correspond to the original signal.

at the bottom is obtained by neglecting the effect of β (i.e., $\beta = 1$) during the simulation. The figure at the top is generated by solving the same linear regression model given in Equ. 7.9, this time by substituting A by $A\beta$, where β is not constrained to one any more (while A is the signal obtained in the base case). We can conclude that considering the effect of β is crucial to explain the changes due to antenna location since better correlation and root mean square results are obtained with the adjusted β. Note that, adjusting the β is only required during the model building (i.e., during measurements if the position of the probe changes), however, the user of the tool does not require to change/adjust β for his/her leakage estimation and can use the base case or numerous cases (depending on the availability) to obtain an "average" leakage estimation, or "worst" case

for $\beta = 1$.

7.6 Practical Use-cases for EMSim

This section describes several example use cases for *EMSim*'s ability to accurately simulate side-channel signals.

7.6.1 Side-Channel Leakage Estimation

An important step for defending against side-channel attacks (SCA) is estimating how much (sensitive) information can be possibly leaked (through a specific or set of side-channels) during the execution of an application. To estimate this leakage, different metrics can be used. Particularly, for EM side-channels one of the state-of-the-art methods is Signal Available to Attacker (SAVAT) [51] methodology.

Due to the lack of simulation tools, to properly calculate these metrics, several actual measurements should be performed. Unfortunately, these measurements often require sophisticated equipment and experts with various skills which, in turn, makes them expensive and difficult in practice. Using our approach, however, we show that *EMSim* is capable of generating highly-accurate simulated signals which can be used to calculate these metrics precisely which eliminates the need for an actual measurement infrastructure.

The following describes how *EMSim* can be used to obtain SAVAT values. It is important to mention that unlike prior work [105, 110, 111, 112], *EMSim is NOT limited to a specific metric or analysis, and it can be used for ANY analysis based on the EM signal.*

Signal Available to Attacker (SAVAT). This metric measures the side-

	LDM		LDC		NOP		ADD		MUL		DIV	
	R	S	R	S	R	S	R	S	R	S	R	S
LDM	0.02	0	3.71	3.91	5.34	5.32	5.24	5.20	5	5.02	4.98	4.98
LDC	3.72	3.91	0.04	0	0.81	0.85	0.74	0.74	0.21	0.24	0.21	0.23
NOP	5.35	5.32	0.8	0.86	0.01	0	0.08	0.1	0.67	0.69	0.66	0.69
ADD	5.24	5.20	0.74	0.75	0.07	0.1	0.03	0	0.98	1.05	1.03	1.1
MUL	4.98	5.01	0.22	0.21	0.66	0.68	0.94	1	0.03	0	0.04	0.01
DIV	4.97	4.99	0.21	0.21	0.65	0.68	1.05	1.13	0.03	0.01	0.02	0

Table 7.2: Signal Available to Attacker metric [51] for Real measurements (R) and Simulations (S).

channel signal created by a specific single-instruction difference in program execution, i.e., the amount of signal made available to an attacker who wishes to decide whether the program has executed instruction/event *A* or instruction/event *B*.

To measure this metric, Callan *et al.* [51] developed a microbenchmark which creates a controlled alternation between A and B instructions many times. Such alternation creates a periodic signal with period $t_p = t_A + t_B$, where for the half of the period *A* is executing and for the other half *B*. Such a periodic activity can then be observed in the frequency domain as a spike at $f_p = 1/t_p$. The key insight is that the corresponding energy of the spike (i.e., area under the curve) indicates how different *A* and *B* are from each other (in terms of side-channel signals) hence reveals how much signal would be available to an attacker when the difference between two samples is whether *A* was executed or *B*.

To compute SAVAT in both real measurements and simulated signals, we implemented the microbenchmark proposed by Callan *et al.* [51], and used the setup explained in §7.5.1 to collect the signals. Table 7.2 shows

SAVAT values in our processor for 6 pairs of instructions. As can be seen, the values retrieved from simulations are highly matched with the values computed using the real measurements. SAVAT can then be utilized to reveal the information leakage capacity of the system [19].

7.6.2 Application to Debugging/Profiling

While so far we have shown how EMSim can be utilized to accurately model EM side-channel signals and thus can be used for leakage estimation during developing secure software, in this section, we present another potentially useful use-case of EMSim and show how hardware designers and computer architects can also leverage this framework during hardware development.

Given that EMSim can accurately model the system for each pipeline stage and each micro-architecture event, it can potentially be used as a debugging tool in the chip-design flow such as a debugging tool for finding design bugs in post-place-and-route stage and/or for finding manufacturing bugs/defects in post-fabrication. In contrary with signal modeling, in this scenario, the signals simulated by the simulator can be assumed as the "ground-truth" or "expected" signal where the signals emanated by the hardware have to be matched to these *reference* models. A deviation from the *reference* model obtained by the simulations indicates that there is an unwanted change/error in the hardware.

The main advantage of this approach compared to existing standard testing methods is that the proposed approach is ***zero-overhead*** and does not require any testing infrastructure on the system which, in turn, saves a significant amount of area and reduces complexity.

To further demonstrate the feasibility of this approach, Figure 7.7 shows

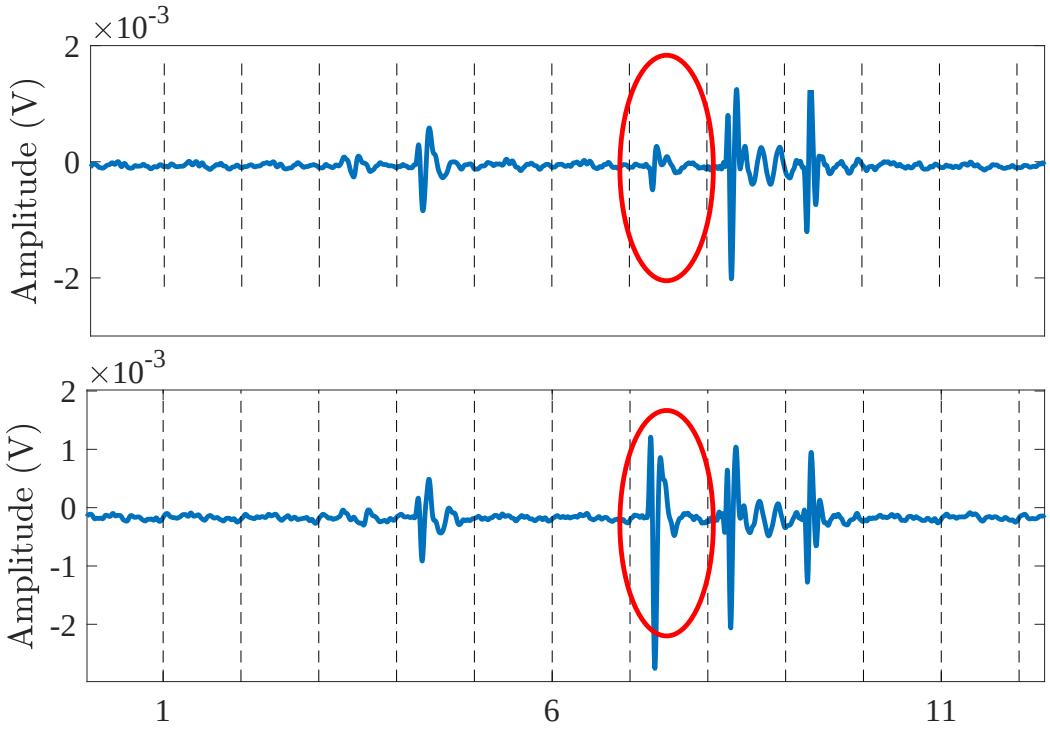


Figure 7.7: A case-study to show how EMSim can be used for debugging. The measured signal (top) does not match with the reference model obtained by the simulation model (bottom) which indicates that there is a potential error/issue in the hardware.

a scenario where there is a bug in designing a multiplier in the Execution stage. The multiplier is designed such that it calculates the result of multiplying two 16-bits operands in three cycles where the majority of the activity (i.e., writing the output register, etc.) takes place in the last (third) cycle. However, as seen from the figure, the amplitude of the measured signal (top) in the third cycle of the execution (shown in a red circle) is significantly lower than that of in the simulation (bottom). Further investigation reveals that instead of properly multiplying two 16-bits data, the designed multiplier only uses the lower half (i.e., 8-bit data) of each operand and ignores the upper half of those inputs hence results in a significantly lower activity factor and thus much smaller signal strength.

It is important to mention that this method, fundamentally, relies on

the signal detection granularity (i.e., how fine-grained changes can be detected), thus it may not be useful in cases where the change in the signal is smaller than the algorithm's detection granularity.

7.7 Summary

This chapter presented *EMSim*, an approach that enables simulation of the EM side-channel signals cycle-by-cycle using a detailed micro-architectural model of the device. Our evaluation of *EMSim* finds that its simulation-derived signals closely matches signals measured from real hardware. To gain further insight, we also experimentally identified how the accuracy of the simulated signals degrades when key micro-architectural features and other hardware behaviors are omitted from the simulation model.

We envision a variety of uses for EMSim. For hardware, software, and compiler developers, it allows EM leakage to be quantified without having to build actual hardware and/or actually measure signals. More importantly, it allows simulated signals to be broken down and attributed to specific parts of the hardware and software. Furthermore, when hardware prototypes are available, significant discrepancies between the signal generated by EMSim and actual EM emanations can be used to identify where the actual design differs from the simulated microachitecture, which can be used to debug the hardware and/or to refine the simulation model to more closely match the hardware.

We believe that *EMSim* can be extended to more complex processors by using a similar multi-input-single-output methodology, where each pipeline stage acts as a single source. For out-of-order processors, we expect higher *baseline hardware amplitude* for each stage as the hardware of each stage becomes more complex. We also expect different values for activity fac-

tors and coefficients for individual stages. Further, since an OoO processor has more *shared* units, to accurately model the signal, these shared units should be carefully simulated and their signals added to each cycle. Nonetheless, since the root cause of creating side-channel signals are bit-flips at the gate-level, we do not expect any fundamental *modeling* difference between in-order and OoO designs.

CHAPTER 8

RESEARCH CONTRIBUTIONS AND FUTURE WORK

8.1 Research Contributions

This research investigated the severity of side/covert channels. We have demonstrated that covert/side channels can transmit thousands of bits per second. Considering the latest improvements that make computer systems more mobile, attacks based on these *unintended* channels become more threatening. The techniques proposed in this thesis provide a good solution for identifying vulnerable parts of devices and a tool for designers to test the resiliency of their systems. Hence, the assessment techniques can be utilized to circumvent information leakage to unauthorized parties. The research contributions of this work are:

1. We derived a mathematical relationship between electromagnetic side channel energy (ESE) of individual instructions and the measured pairwise side channel signal power [17, 18]. Assuming that only execution of an instruction is the source of emanated EM signals and instructions are ordered independently, we proposed a DMC channel model to quantify the leakage capacity. For the transition probabilities needed for estimating capacity, we utilized ESE measure. We modified Shannon's channel capacity theorem to address instead of deal with the variation in execution time of instruction. After proposing the model and the leakage capacity as the channel capacity of the model, we provided an assessment technique to analyze the vulnerability of a system. Finally, we illustrated how the proposed method

works by showing capacity of several practical systems.

2. A side channel information capacity created by execution of series of instructions (e.g. a function, a procedure, or a program) in a processor was proposed [19, 113]. Considering the structure of processor pipeline and that each program code is written systematically to perform a specific task, we proposed Markov Source model, which includes the dependencies that exist in instruction sequence. Emitted EM signals during instruction executions were considered as the sources for channel inputs and the states of the model were assumed to be instructions. We derived a mathematical relationship between the emanated instruction signal power (ESP) as it passes through processor pipeline and total emanated signal power while running a program to obtain the channel inputs for the proposed model. Finally, we provided experimental results to demonstrate that leakages could be severe and that a dedicated attacker could obtain important information.

3. An EM based covert channel was introduced as a communication channel [20, 21, 77]. These channels are not designed to transmit information. They are exposed errors created by 1) the transmission environment (the transmitted signal propagates through a channel hindered by metal and plastic), 2) varying execution time of computer activities, and 3) insertions from other computer activities such as interrupts. Considering these effects, we proposed to model the covert channel as an insertion channel where the transmitted sequence is a pulse amplitude modulated signal with random pulse position. We derived capacity bounds of this covert channel with random inser-

tion and substitution due to noise and jitter errors by utilizing the proposed channel model. We also proposed the receiver design that can correctly detect and demodulate computer-activity created signals. Finally, the theoretical derivations were compared to empirical results and show agreement.

4. A generalized channel model for various covert channels was proposed, which considers insertions, deletions, and asynchronous nature of covert channels [22]. We defined effective channel noise after deriving that the jitter error (error due to variation in signaling time) can be combined with additive channel noise. The derivation requires that the signaling time distribution has a normal behavior. In that respect, we experimentally demonstrated that the signaling time is normally distributed with mean μ and standard deviation σ. Based on the communication model and combined effective channel noise, we calculated the worst-case leakage through various covert channels. With this channel model, exact leakage capacity was obtained instead of providing loose capacity bounds. We also proposed an assessment technique for various covert channel attacks. Finally, we provided experimental results which demonstrated the severity of these channels and importance of including covert channel leakages into design considerations.

5. *EMSim*, a simulation tool that enables estimation of the electromagnetic side channel signals cycle-by-cycle was introduced considering micro-architectural model of the device [23, 67]. To obtain such a tool, the shape of the signals during execution of an instruction at any pipeline stage was experimentally obtained, and a mathematical

signal model that explains the mean behavior of emanated signals was proposed. Since the collected signals were a mixture of signals that were emitted from all pipeline stages, we proposed the channel between the processor and the antenna as a MISO channel. We compared the simulated signals against actual EM signals emanated from real hardware, which matched very closely. Finally, we provided some experiments illustrating how the simulation tool can be used to analyze system at *design-stage*.

8.2 Future Research Directions

Although this thesis provides leakage capacity bounds, it does not provide any method or communication scheme that can achieve these bounds. The main problem with side channel analysis arises especially when systems get more complex and faster as the clock frequencies of the sophisticated systems increase. For a proper analysis of these channels, measuring devices must have lower noise levels and higher sampling rates. However, the measuring devices with these capabilities either do not exist or are very expensive. Therefore, new techniques have to be developed such that the side channel analysis that can achieve capacity bounds will be possible even with much simpler measuring devices as software-defined radios (SDR). For example, tracking executed instruction order could be an application. As the new methods can achieve leakage capacities with less expensive measuring devices, these channels can be the main candidate for profiling system against malware. However, from the perspective of an attacker, this method means a perfect opportunity to steal secret information. Therefore, a method leads to be introduced for real-time monitoring of instruction-level signals to analyze whether having such a leakage jeop-

ardizes the sensitive information within the system.

Another direction related to side channels is the source separation problem. When the measurement is done from a distance, the collected signal contains components that are leaked from not only processor but also other components, i.e. memory, power unit, etc. Because of destructive/constructive effect of these signals on each other, developing a source separation algorithm can help improving insights about the system since each source will be processed individually. This approach can be also leveraged in profiling systems with no-overhead. Having such an informative tool, a system can be monitored to control whether it is working as it is supposed to.

Finally, a methodology that generalizes the information obtained from one device to estimate the behavior of others is required. The current approaches are generally ad-hoc and specific to devices that are studied. Hence, the same measurements have to be performed for each device to analyze their resilience to side channels, to train a system for monitoring the program activities, etc. This is because computer systems generally have different clock frequencies and operating systems, therefore, emanated signals can show differences in terms of spectral components. Therefore, developing a systematic mapping algorithm to estimate the behavior of side channel signals of a device from the signals that belong to another type of a device can reduce training time or data collection time extensively. Moreover, with this method, a *design-stage* analysis can be done since the signal mapping will provide information about side channel signal characteristic of a design.

APPENDIX A

THE RELATIONSHIP BETWEEN ESE AND MEASURED SPECTRAL POWER OF A MICROBENCHMARK

In this section, we show that $\mathrm{ESE}\,[\mathrm{X}_1, \mathrm{X}_2]$ defined in (3.1) is related to the alternation power $P(f_{\mathrm{alt}})$ as in (3.5) where $s_m^{(X_m)}[n]$ denotes the sampled signal when instruction X_m is inserted into the m^{th} loop given in Fig. 2.1. The assumptions given in Section 3.2 will be considered through derivations.

We start by noting that the signals generated by the ESE benchmarks can be represented as a specific mixture of two *periodic* signals with period N. For $n = 0, ..., N-1$, the first signal obtained from the first iteration of the first inner loop is denoted as

$$\hat{s}_1^{(X_1)} = [o_1[0], o_1[1], \cdots, o_1[n_O - 1], x_1[0], x_1[1], \cdots, x_1[n_X - 1]]$$

such that $N = n_O + n_X$. Note that $\mathbb{E}\left[s_1^{(X_1)}[n+N]\right] = \mathbb{E}\left[s_1^{(X_1)}[n]\right]$ because $s_1^{(X_1)}[n]$ is periodic which leads that we also assume the additive noise are also periodic. Following the same procedure for the second inner loop, we have

$$\hat{s}_2^{(X_2)} = \big[o_2[0], o_2[1], \cdots, o_2[n_O - 1], x_2[0], x_2[1], \cdots, x_2[n_X - 1]\big].$$

We denote the sampled voltage at the time points where instruction X_1 is active as $x_1[i]$ and the sampled voltage at the time points where instruction X_2 is active as $x_2[i]$ where $i \in \{0, 1, \cdots, n_X - 1\}$. Similarly $o_m[n]$ represents the other instructions in the benchmark necessary to make the benchmark practical (e.g. to create a loop around instruction X_1 or instruction X_2) for the m^{th} loop.

First, we derive the ESE between two instructions from the measurement given in (3.2) as follows:

$$\begin{aligned}
\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)}, \mathrm{s}_2^{(\mathrm{X}_2)}\right] &\equiv T_{\mathrm{I}} \sum_{l=0}^{N_s-1} \frac{\left(s_1^{(X_1)}[l] - s_2^{(X_2)}[l]\right)^2}{R} \\
&= \frac{T_{\mathrm{I}}}{R} \sum_{k=0}^{Nn_{\mathrm{inst}}-1} \left(s_1^{(X_1)}[k] - s_2^{(X_2)}[k]\right)^2 \\
&\approx \frac{\kappa}{2}\left[\sum_{k=0}^{N-1} \left(\hat{s}_1^{(X_1)}[k] - \hat{s}_2^{(X_2)}[k]\right)^2\right] \qquad \text{(A.1)}
\end{aligned}$$

where $\kappa = (2T_{\mathrm{I}}n_{\mathrm{inst}})/(R)$ and (A.1) follows the periodicity of $s_1^{(X_1)}[n]$ and $s_2^{(X_2)}[n]$. Based on the assumption in Section 3.2, we can write $x_1[i] = X_1^v + w_1^{(X_1)}[i]$, $x_2[i] = X_2^v + w_2^{(X_2)}[i]$ and $o_m[i] = O^v + w_m^{(O)}[i]$ where $w_1^{(X_1)}[i] \sim \mathcal{N}(0, \sigma_{X_1}^2)$, $w_2^{(X_2)}[i] \sim \mathcal{N}(0, \sigma_{X_2}^2)$, $w_m^{(O)}[i] \sim \mathcal{N}(0, \sigma_O^2)$ and $m \in \{1, 2\}$. Therefore,

$$\begin{aligned}
\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)}, \mathrm{s}_2^{(\mathrm{X}_2)}\right] &\approx \frac{\kappa}{2} \sum_{k=0}^{n_X-1} \left(X_1^v - X_2^v + w_1^{(X_1)}[k] - w_2^{(X_2)}[k]\right)^2 \\
&\quad -\frac{\kappa}{2} \sum_{k=0}^{n_O-1} \left(w_1^{(O)}[k] - w_2^{(O)}[k]\right)^2 \\
&= \frac{\kappa n_X}{2} \frac{1}{n_X} \sum_{k=0}^{n_X-1} \left(X_1^v - X_2^v + w_1^{(X_1)}[k] - w_2^{(X_2)}[k]\right)^2 \\
&\quad -\frac{\kappa n_O}{2} \frac{1}{n_O} \sum_{k=0}^{n_O-1} \left(w_1^{(O)}[k] - w_2^{(O)}[k]\right)^2 \qquad \text{(A.2)}
\end{aligned}$$

Assuming the number of samples taken during executions of instructions are large enough, sum operations given in (A.2) can be considered as the

expectation operation. Therefore, we can claim

$$\begin{aligned}\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_2)}\right] &\approx \frac{\kappa n_X}{2}\mathbb{E}\left[\left(X_1^v - X_2^v + w_1^{(X_1)}(k) - w_2^{(X_2)}(k)\right)^2\right] \\ &\quad -\frac{\kappa n_O}{2}\mathbb{E}\left[\left(w_1^{(O)}[k] - w_2^{(O)}[k]\right)^2\right] \\ &= \frac{\kappa}{2}\Big(n_X(X_1^v - X_2^v)^2 + n_X\sigma_{X_1}^2 + n_X\sigma_{X_2}^2 + 2n_O\sigma_O^2\Big). \quad \text{(A.3)}\end{aligned}$$

Furthermore, observe that (A.3) can be modified in terms of ESE as follows:

$$\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_2)}\right] \equiv n_{\mathrm{inst}}\mathrm{ESE}[\mathrm{X}_1,\mathrm{X}_2] + \frac{1}{2}\Big(\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] + \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_2)},\mathrm{s}_2^{(\mathrm{X}_2)}\right]\Big). \quad \text{(A.4)}$$

If we define

$$\hat{C}(X_1,X_2) = \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] + \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_2)},\mathrm{s}_2^{(\mathrm{X}_2)}\right], \quad \text{(A.5)}$$

ESE of two instructions in time domain can be written as

$$\mathrm{ESE}[\mathrm{X}_1,\mathrm{X}_2] = \frac{2\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_2)}\right] - \hat{C}(X_1,X_2)}{n_{\mathrm{inst}}}. \quad \text{(A.6)}$$

Although, we derive ESE power of two instructions in time domain, measuring ESE of two sequences can be cumbersome because the number of samples required can be huge. In that perspective, we derive the equation given in 3.5 which clarifies the relation between the ESE power of two instructions and the power at f_{alt}.

To relate $s_1^{(X_1)}[n]$ and $s_2^{(X_2)}[n]$ to our benchmarks, we define a square wave $p[n]$ with a 50% duty cycle as

$$p[0 \le n < Nn_{\mathrm{inst}}] = 1 \quad \text{(A.7)}$$

$$p[Nn_{\mathrm{inst}} \le n < 2Nn_{\mathrm{inst}}] = 0, \quad \text{(A.8)}$$

where $p[n]$, $s_1^{(X_1)}[n]$ and $s_2^{(X_2)}[n]$ are periodic with period $2Nn_{\text{inst}}$, since we assume the additive noise is also periodic. Then, we can take the discrete Fourier series of these signals over $2Nn_{\text{inst}}$ samples. We refer to $S_1^{(X_1)}[k]$, $S_2^{(X_2)}[k]$, and $P[k]$ as the discrete Fourier series (DFS) of $s_1^{(X_1)}[n]$, $s_2^{(X_2)}[n]$ and $p[n]$ respectively, defined for $0 \leq k < 2Nn_{\text{inst}}$.

We next define

$$v[n] = p[n]s_1^{(X_1)}[n] + (1 - p[n])s_2^{(X_2)}[n], \tag{A.9}$$

which represents the signal created by the sequence of instructions executed by the microbenchmarks.

We start the derivation of relationship between ESE and measured spectral power by observing that the DFS of $v[n]$ defined in (A.9) can be written as

$$\begin{aligned} V[k] &= P[k] * S_1^{(X_1)}[k] + (1 - P[k]) * S_2^{(X_2)}[k] \\ &= S_2^{(X_2)}[k] + P[k] * \left(S_1^{(X_1)}[k] - S_2^{(X_2)}[k] \right), \end{aligned} \tag{A.10}$$

where $*$ denotes periodic convolution, defined as in Appendix C.

Now we consider $V[1]$, the 2^{nd} Fourier coefficient (the first harmonic) of the $v[n]$ sequence:

$$V[1] = S_2^{(X_2)}[1] + \frac{\sum_{m=0}^{2Nn_{\text{inst}}-1} P[1-m] \left(S_1^{(X_1)}[m] - S_2^{(X_2)}[m] \right)}{2Nn_{\text{inst}}}. \tag{A.11}$$

Using the equation (C.3) from Appendix C, we can observe that $S_1^{(X_1)}[k]$ and $S_2^{(X_2)}[k]$ are non-zero only for $k = 2n_{\text{inst}}l$ for $l = 0, 1, ..., N-1$. Then $V[1]$ simplifies to

$$V[1] = \frac{\sum_{l=0}^{N-1} P[1 - 2n_{\text{inst}}l] \left(S_1^{(X_1)}[2n_{\text{inst}}l] - S_2^{(X_2)}[2n_{\text{inst}}l] \right)}{2Nn_{\text{inst}}} \tag{A.12}$$

Then, $V[1]$ can be further expanded as follows

$$\begin{aligned} V[1] = & \frac{P[1]\left(S_1^{(X_1)}[0] - S_2^{(X_2)}[0]\right)}{2Nn_{\text{inst}}} \\ & + \frac{P[1-2n_{\text{inst}}]\left(S_1^{(X_1)}[2n_{\text{inst}}] - S_2^{(X_2)}[2n_{\text{inst}}]\right)}{2Nn_{\text{inst}}} \\ & + \dots \end{aligned} \tag{A.13}$$

Here we note that next few higher order odd harmonics can be similarly expanded while the even harmonics are zero. Additionally, we note that $P[k]$ is the k^{th} coefficient of the discrete Fourier series for a square wave with period $2Nn_{\text{inst}}$ and can be written as ([104], Example 8.3)

$$\begin{aligned} \frac{|P[k]|}{2Nn_{\text{inst}}} &= \frac{\sin(\pi k/2)}{2Nn_{\text{inst}} \cdot \sin(\frac{\pi k}{2Nn_{\text{inst}}})} \\ \Rightarrow \frac{|P[k]|}{2Nn_{\text{inst}}} &\approx \frac{\sin(\pi k/2)}{\pi k} \\ \Rightarrow \frac{|P[1]|}{2Nn_{\text{inst}}} &\approx \frac{1}{\pi}. \end{aligned} \tag{A.14}$$

The last two steps follow by recognizing that

$$2Nn_{\text{inst}} \cdot \sin\left(\frac{\pi k}{2Nn_{\text{inst}}}\right) = \pi k \cdot \frac{\sin\left(\frac{\pi k}{2Nn_{\text{inst}}}\right)}{\frac{\pi k}{2Nn_{\text{inst}}}} \tag{A.15}$$

and noting that $\sin(x)/x \to 1$ as $x \to 0$ (i.e. large n_{inst}). Since n_{inst} is typically > 100, this approximation is valid. For $n_{\text{inst}} > 100$, $|P[1]| > 100|P[1-n_{\text{inst}}]|$, so

the higher order terms in (A.13) can be neglected, giving

$$\begin{aligned} |V[1]| &\approx \frac{|P[1]|}{2Nn_{\text{inst}}} \cdot \left| S_1^{(X_1)}[0] - S_2^{(X_2)}[0] \right| \\ \Rightarrow \pi\,|V[1]| &\approx \left| S_1^{(X_1)}[0] - S_2^{(X_2)}[0] \right|. \end{aligned} \tag{A.16}$$

The next step is to calculate the difference between $S_1^{(X_1)}[0]$ and $S_2^{(X_2)}[0]$. Here we note that , we decompose $s_1^{(X_1)}[n] = \hat{o}_1[n] + i_1^{(X_1)}[n]$ where the first N samples of $\hat{o}_1[n] = [o_1[0], o_1[1], ..., o_1[n_O - 1], 0, \cdots, 0]$ and the first N samples of $i_1^{(X_1)}[n] = [0, ..., 0, x_1[0], x_1[1], \cdots, x_1[n_X - 1]]$. We can decompose $s_2^{(X_2)}[n]$ similarly. By the linearity of the Fourier transform

$$\begin{aligned} S_1^{(X_1)}[k] - S_2^{(X_2)}[k] &= I_1^{(X_1)}[k] + \hat{O}_1[k] - \left(I_2^{(X_2)}[k] + \hat{O}_2[k] \right) \\ &= I_1^{(X_1)}[k] - I_2^{(X_2)}[k] + \left(\hat{O}_1[k] - \hat{O}_2[k] \right). \end{aligned} \tag{A.17}$$

The DFS coefficient $I_1^{(X_1)}[0]$ is

$$\begin{aligned} I_1^{(X_1)}[0] &= \sum_{n=0}^{2Nn_{\text{inst}}-1} i_1^{(X_1)}[n] \\ &= \sum_{r=0}^{2n_{\text{inst}}-1} \sum_{s=0}^{n_X-1} i_1^{(X_1)}[rn_{\text{inst}} + n_O + s] \\ &= \sum_{r=0}^{2n_{\text{inst}}-1} \sum_{s=0}^{n_X-1} \left(X_1^v + w_1^{(X_1)}[rn_{\text{inst}} + n_O + s] \right) \\ &= 2n_{\text{inst}} n_X X_1^v + 2n_{\text{inst}} \sum_{n=n_O}^{N} w_1^{(X_1)}[n]. \end{aligned} \tag{A.18}$$

where the last equality follows the assumption that noise is also circular. Similarly, $I_2^{(X_2)}[0] = 2n_{\text{inst}} n_X X_2^v + 2n_{\text{inst}} \sum_{n=n_O}^{N} w_2^{(X_2)}[n]$ and $O_i[0] = 2n_{\text{inst}} n_O O^v +$

$2n_{\text{inst}} \sum_{l=0}^{n_O-1} w_i^{(O)}$ where $i \in \{1, 2\}$. Therefore,

$$\begin{aligned} S_1^{(X_2)}[0] - S_2^{(X_2)}[0] \;=\; & 2n_{\text{inst}} \Bigg[n_X (X_1^v - X_2^v) \\ & + \sum_{l=0}^{n_O-1} \Big(w_1^{(O)}[l] - w_2^{(O)}[l] \Big) \\ & + \sum_{l=0}^{n_X-1} \Big(w_1^{(X_1)}[l] - w_2^{(X_2)}[l] \Big) \Bigg] \quad . \end{aligned} \tag{A.19}$$

Assuming n_O and n_X are large enough and additive noises are independent, we have

$$\begin{aligned} \Big| S_1^{(X_2)}[0] - S_2^{(X_2)}[0] \Big|^2 \;\approx\; & (2n_{\text{inst}})^2 \Big[(n_X (X_1^v - X_2^v))^2 \\ & + 2n_O \sigma_O^2 + n_X \sigma_{X_1}^2 \;\; + n_X \sigma_{X_2}^2 \Big] \\ =\; & 4 n_{\text{inst}}^2 D_{ab} \end{aligned} \tag{A.20}$$

where

$$D_{ab} = n_X \Big(n_X (X_1^v - X_2^v)^2 + \sigma_{X_1}^2 + \sigma_{X_2}^2 \Big) + 2n_O \sigma_o^2. \tag{A.21}$$

As the next step, we need to relate (A.20) to the power observed with the spectrum analyzer. The power observed with the spectrum analyzer is described by ([114, 115])

$$P(f_{\text{alt}}) = \frac{2}{R} \Big(\frac{|V[1]|}{2N n_{\text{inst}}} \Big)^2, \tag{A.22}$$

where $2N n_{\text{inst}}$ is the number of samples taken in one period T_{alt}. We also note that

$$n_{\text{inst}} f_{\text{alt}} = \frac{1}{2N T_I}. \tag{A.23}$$

So, by plugging (A.16) and (A.20) into (A.22), we have

$$\begin{aligned} P(f_{\text{alt}}) &= \frac{2}{R}\left(\frac{\left|S_1^{(X_1)}[0]-S_2^{(X_2)}[0]\right|}{2Nn_{\text{inst}}\cdot\pi}\right)^2 \\ &= \frac{2}{R}\left(\frac{4n_{\text{inst}}^2 D_{ab}}{4N^2 n_{\text{inst}}^2\cdot\pi^2}\right) \\ &= \frac{2}{R}\left(\frac{D_{ab}}{N^2\pi^2}\right). \end{aligned} \tag{A.24}$$

As mentioned before, since working on time domain is cumbersome, our main goal is to measure ESE in frequency domain. Therefore, using (A.23), (A.22) and (A.24), we obtain the relationship between ESE and $P(f_{\text{alt}})$ as follows:

$$P(f_{\text{alt}}) = \frac{2}{R}\left(\frac{D_{ab}}{N^2\cdot\pi^2}\right) \tag{A.25}$$

$$\Rightarrow \quad \frac{D_{ab}}{R} = \frac{P(f_{\text{alt}})\cdot N^2\cdot\pi^2}{2\cdot} \tag{A.26}$$

$$\Rightarrow \quad \frac{D_{ab}}{RNn_{\text{inst}}} = \frac{P(f_{\text{alt}})\cdot N\cdot\pi^2}{2\cdot n_{\text{inst}}} \tag{A.27}$$

$$\Rightarrow \quad \frac{2\cdot f_{\text{alt}}T_{\text{I}}D_{ab}}{R} = \frac{P(f_{\text{alt}})\cdot N\cdot\pi^2}{2\cdot n_{\text{inst}}} \tag{A.28}$$

$$\Rightarrow \quad \frac{2T_{\text{I}}n_{\text{inst}}}{R}D_{ab} = \frac{P(f_{\text{alt}})\cdot N\cdot\pi^2}{2\cdot f_{\text{alt}}} \tag{A.29}$$

$$\Rightarrow \quad \kappa D_{ab} = \pi^2\frac{P(f_{\text{alt}})\cdot N}{2\cdot f_{\text{alt}}} \tag{A.30}$$

where (A.28) follows the equality given in (A.23). Relating the equation in

(A.30) with (A.3) and (A.4), we have

$$\begin{aligned}\kappa D_{ab} &= \kappa \left(n_X \left(n_X (X_1^v - X_2^v)^2 + \sigma_{X_1}^2 + \sigma_{X_2}^2 \right) + 2 n_O \sigma_o^2 \right) \\ &= 2 n_X n_{\text{inst}} \, \text{ESE}[\text{X}_1, \text{X}_2] \\ &\quad + \mathcal{P}_\text{A} \left[\text{s}_1^{(\text{X}_1)}, \text{s}_2^{(\text{X}_1)} \right] + \mathcal{P}_\text{A} \left[\text{s}_1^{(\text{X}_2)}, \text{s}_2^{(\text{X}_2)} \right]. \end{aligned} \tag{A.31}$$

To simplify the notation, we define

$$C(X_1, X_2) = - \frac{\mathcal{P}_\text{A} \left[\text{s}_1^{(\text{X}_1)}, \text{s}_2^{(\text{X}_1)} \right] + \mathcal{P}_\text{A} \left[\text{s}_1^{(\text{X}_2)}, \text{s}_2^{(\text{X}_2)} \right]}{2 n_X n_{\text{inst}}} \tag{A.32}$$

which can be considered as the noise term added to ESE of (X_1, X_2) instruction pair because of the measurement done in frequency domain. Therefore, we have

$$\text{ESE}[\text{X}_1, \text{X}_2] = \frac{1}{2 n_X n_{\text{inst}}} \pi^2 \frac{P(f_{\text{alt}}) \cdot N}{2 \cdot f_{\text{alt}}} + C(X_1, X_2) \tag{A.33}$$

$$= \left(\frac{\pi}{2} \right)^2 \frac{P(f_{\text{alt}}) \cdot N}{n_X \cdot n_{\text{inst}} \cdot f_{\text{alt}}} + C(X_1, X_2) \tag{A.34}$$

which concludes the proof.

APPENDIX B

EXECUTION LOCATION BASED NOISE POWER ESTIMATION

In this section, we provide a method to estimate the additive noise power corresponding to the execution location of any instruction. Note that when the same instruction is inserted into both for-loops, we have

$$\begin{aligned}\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] = \; & \tfrac{\kappa}{2}\Big(n_X(X_1^v - X_1^v)^2 + n_X\sigma_{X_1}^2 \\ & + n_X\sigma_{X_1}^2 + 2n_O\sigma_O^2\Big) \\ = \; & \tfrac{\kappa}{2}\Big(2n_X\sigma_{X_1}^2 + 2n_O\sigma_O^2\Big). \end{aligned} \tag{B.1}$$

On the other hand, if we do not insert any instruction into these loops, we have

$$\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{NOI})},\mathrm{s}_2^{(\mathrm{NOI})}\right] = \frac{\kappa}{2}\Big(2n_O\sigma_O^2\Big). \tag{B.2}$$

Therefore, to obtain the noise power related to any instruction can be found by combining (B.1) and (B.2) as

$$\begin{aligned} & \kappa n_X\sigma_{X_1}^2 = \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] - \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{NOI})},\mathrm{s}_2^{(\mathrm{NOI})}\right] \\ \Rightarrow \quad & n_X\sigma_{X_1}^2 = \frac{\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] - \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{NOI})},\mathrm{s}_2^{(\mathrm{NOI})}\right]}{\kappa} \\ \Rightarrow \quad & \bar{\sigma}_{X_1}^2 = \frac{\mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{X}_1)},\mathrm{s}_2^{(\mathrm{X}_1)}\right] - \mathcal{P}_{\mathrm{A}}\left[\mathrm{s}_1^{(\mathrm{NOI})},\mathrm{s}_2^{(\mathrm{NOI})}\right]}{\kappa}. \end{aligned} \tag{B.3}$$

APPENDIX C

DISCRETE FOURIER SERIES

The discrete Fourier series pair of a periodic signal $x[n]$ with period M can be defined as [104]

$$\begin{aligned} X[k] &= \sum_{n=0}^{M-1} x[n] e^{-j(2\pi/M)kn} \qquad \text{(DFS)} \\ x[n] &= \frac{1}{M} \sum_{n=0}^{M-1} X[k] e^{-j(2\pi/M)kn} \quad \text{(IDFS)}. \end{aligned} \tag{C.1}$$

Multiplying two sequences $x_1[n]$ and $x_2[n]$ in time domain is equivalent to their periodic convolution in the frequency domain and can be written as [104]

$$x_1[n]x_2[n] \xrightarrow{\text{DFS}} X_1[k] * X_2[k] = \frac{1}{M} \sum_{l=0}^{M-1} X_1[l] X_2[k-l]. \tag{C.2}$$

The discrete Fourier series of a periodic signal $x[n]$ taken *over a period of* ML *samples* is equal to

$$\begin{aligned} X[k] &= \sum_{n=0}^{ML-1} x[n] e^{-j\frac{2\pi}{ML}kn} \\ &= \sum_{m=0}^{M-1} \left(x[m] \sum_{l=0}^{L-1} e^{-j\frac{2\pi}{ML}k(m+Ml)} \right) \\ &= \sum_{m=0}^{M-1} \left(x[m] e^{-j\frac{2\pi}{ML}km} \sum_{l=0}^{L-1} e^{-j2\pi k\frac{l}{L}} \right) \\ &= \sum_{m=0}^{M-1} \left(x[m] e^{-j\frac{2\pi}{ML}km} L \sum_{l=-\infty}^{\infty} \delta[k-lL] \right) \\ &= \begin{cases} L\sum_{m=0}^{M-1} x[m] e^{-j\frac{2\pi}{ML}km} & \text{for } k = Ll \\ 0 & \text{for } k \neq Ll \end{cases} \end{aligned} \tag{C.3}$$

The discrete time periodic impulse train with period M can be written as [104]

$$\sum_{l=-\infty}^{\infty} \delta[k - Ll] = \frac{1}{L} \sum_{l=0}^{L-1} e^{-j2\pi k \frac{l}{L}}. \tag{C.4}$$

APPENDIX D

GRADIENT DESCENT APPROACH FOR CAPACITY CALCULATION

In this section, we provide the gradient descent algorithm to solve the optimization problem in (3.7). We also derive a closed form solution to attain the probability of each instruction. First, we take the derivative of (3.7) with respect to an instruction probability P_m. Therefore,

$$\frac{\partial f(\mathbf{P})}{\partial P_m} = \frac{\partial(\frac{\Omega}{\Psi})}{\partial P_m} \tag{D.1}$$

$$\begin{aligned} \frac{\partial(\frac{\Omega}{\Psi})}{\partial P_m} &= \frac{\partial\left(\frac{\sum_{i,j} P_i p_{ij} \log p_{ij} - \sum_{i,j} P_i p_{ij} \log \sum_k P_k p_{kj}}{\sum_i P_i L_i}\right)}{\partial P_m} \\ &= \frac{\left(\sum_j p_{mj} \log p_{mj} - G_m\right)\Psi - \Omega L_m}{\Psi^2} \end{aligned} \tag{D.2}$$

where L_m is the length of m^{th} instruction, and

$$\mathbf{P} = [P_1 \quad P_2 \quad \cdots \quad P_K]^T \tag{D.3}$$

and

$$\begin{aligned} G_m &= \sum_j p_{mj} \log \sum_k P_k p_{kj} + \sum_{i,j} P_i p_{ij} \frac{p_{mj}}{\sum\limits_k P_k p_{kj}} \\ &= \sum_j p_{mj} \log \sum_k P_k p_{kj} + \sum_j p_{mj} \sum_i P_i p_{ij} \frac{1}{\sum\limits_k P_k p_{kj}} \\ &= \sum_j p_{mj} \log \sum_k P_k p_{kj} + \sum_j p_{mj} \\ &= \sum_j p_{mj} \log \sum_k P_k p_{kj} + 1. \end{aligned} \tag{D.4}$$

Since transition probabilities are calculated based on ESE measurements as explained in Appendix 3.3.2, we can calculate entropy for each instruction beforehand. Let $H_m = -\sum\limits_j p_{mj} \log p_{mj}$ be the entropy for the m^{th} instruction. So, the gradient vector for the optimization problem can be given as

$$\nabla_{\mathbf{P}} \left(\frac{\Omega}{\Psi} \right) = \nabla_{\mathbf{P}} f(\mathbf{P}) = \left[\frac{\partial(\frac{\Omega}{\Psi})}{\partial P_1} \quad \frac{\partial(\frac{\Omega}{\Psi})}{\partial P_2} \cdots \quad \frac{\partial(\frac{\Omega}{\Psi})}{\partial P_K} \right]^T \tag{D.5}$$

where $(\bullet)^T$ is the transpose operation and

$$\frac{\partial(\frac{\Omega}{\Psi})}{\partial P_m} = -\frac{(H_m + G_m)\,\Psi + \Omega L_m}{\Psi^2}. \tag{D.6}$$

Fig. D.1 delivers the insights about the algorithm to achieve the optimal solution. In the algorithm, $\mathbf{I}_K$ in line 12 stands for the identity matrix with size K and $sum(\bullet)$ in line 13 returns the sum of its vector argument.

Another goal of the section is to derive a closed-form solution to the defined problem in (3.7). We exploit the result in (D.2) and set the derivative equals to zero. Let us write the overall optimization problem in Lagrangian

```
1  //Initialization of the state probabilities
2  P(0) =  1_K\K;
3  //-- K is the number of instruction
4  //-- 1_K is the vector with full of zeros
5  //and whose length is K.
6  converge = false;
7  while(!converge){
8    ∇_P ← Calculate the gradient at time k
9    P(k+1) =  P(k+1) + μ∇_P f(P(k))
10   // μ is the step size
11   // Project into the feasible region
12   P(k+1) = (I_K − 1_K 1_K^T)P(k+1) + 1_K;
13   P(k+1) = P(k+1) / sum(P(k+1));
14   if [(|f(P(k+1)) − f(P(k))|) /|f(P(k))|] < ϵ{
15   converge = true;
16   }
17 }
```

Figure D.1: Gradient descent approach to achieve optimal solution

form such that

$$\mathcal{L}(\mathbf{P}) = f(\mathbf{P}) + \lambda(1 - \sum_i P_i) + \mu^T \mathbf{P} \tag{D.7}$$

where μ and λ are the dual variables. For the rest of the derivations, we will assume that each instruction occurs with nonzero probability so that, from KKT (Karush-Kuhn-Tucker) conditions, μ must equal to a zero vector. So,

$$\frac{\partial \mathcal{L}(\mathbf{P})}{\partial P_m} = 0 \tag{D.8}$$

$$\Rightarrow \quad \frac{(H_m + G_m)\,\Psi + \Omega L_m}{\Psi^2} - \lambda = 0 \tag{D.9}$$

$$\Rightarrow \quad \frac{(H_m + G_m) + R L_m}{\Psi} = \lambda \tag{D.10}$$

where $R = \max_{\mathbf{P}} f(\mathbf{P})$ which represents the value for the optimum solution. The reason for the insertion of R instead of the ratio Ψ/Ω is simply because calculations are done in derivative space and the optimization setting ensures convexity. Observe that left hand side of (D.10) simple equals to partial derivative of $f(\mathbf{P})$. Hence, for the optimal solution, the partial derivative of the objective function with respect to each instruction proba-

bility must equal to each other. By keeping that in mind and plugging the exact definition of G_m, we have

$$(H_m + G_m) + RL_m = \lambda\Psi \tag{D.11}$$

$$\Rightarrow \quad G_m = -H_m - R \cdot L_m + \lambda\Psi \tag{D.12}$$

$$\Rightarrow \quad \sum_j p_{mj} \log \sum_k P_k p_{kj} + 1 = -H_m - R \cdot L_m + \lambda\Psi \tag{D.13}$$

$$\Rightarrow \quad \sum_j p_{mj} \log \sum_k P_k p_{kj} = -(H_m + R \cdot L_m - \lambda\Psi + 1) \tag{D.14}$$

If we define the equality $\sum_m h_{mi} p_{mj} = \delta_{ij}$ where δ is the Kronocker delta function such that $\delta_{ij} = \delta[i-j]$, introduce the notation $\sigma_m = (H_m + R \cdot L_m - \lambda\Psi + 1)$, multiply both sides with h_{mi} and sum over m, we have

$$\sum_{m,j} h_{mi} p_{mj} \log \sum_k P_k p_{kj} = -\sum_m \sigma_m h_{mi} \tag{D.15}$$

$$\Rightarrow \quad \sum_j \delta_{ij} \log \sum_k P_k p_{kj} = -\sum_m \sigma_m h_{mi} \tag{D.16}$$

$$\Rightarrow \quad \log \sum_k P_k p_{ki} = -\sum_m \sigma_m h_{mi} \tag{D.17}$$

$$\Rightarrow \quad \sum_k P_k p_{ki} = exp(-\sum_m \sigma_m h_{mi}). \tag{D.18}$$

Finally, let us define $\sum_i z_{ti} p_{ki} = \delta_{tk}$ and perform the same operations as before, we have

$$\sum_{i,k} P_k z_{ti} p_{ki} = \sum_i z_{ti} exp(-\sum_m \sigma_m h_{mi}) \tag{D.19}$$

$$\Rightarrow \quad \sum_k P_k \delta_{kt} = \sum_i z_{ti} exp(-\sum_m \sigma_m h_{mi}) \tag{D.20}$$

$$\Rightarrow \quad P_t = \sum_i z_{ti} exp(-\sum_m \sigma_m h_{mi}) \tag{D.21}$$

$$\Rightarrow \quad P_t = \sum_i z_{ti} exp(-\sum_m (H_m + R \cdot L_m - \lambda\Psi + 1) h_{mi}) \tag{D.22}$$

Hence, we can extract the instruction probabilities if we have the optimal rate R. Since our purpose is to obtain the maximum rate and the

point which maximizes this rate, we can scan for a value of R in the range between 0 to $\log_2 K$ and λ such that it satisfies the conventional probability distribution constraints i.e. $\sum P_i = 1$. Our search is limited by $\log_2 K$ for the rate since the lengths of each instruction are at least unity, but unlimited for λ. Fortunately, proposed algorithm given in Fig. D.1 provides the rate, λ and the corresponding point, therefore, we do not need to perform such a scan. However, we can use the closed-form solution for inspection purposes.

APPENDIX E

ESTABLISHING THE DUALITY BETWEEN (4.3) AND (4.5)

Transforming the optimization problem given in (4.3) to the problem given in (4.5) helps to utilize the ExMa algorithm presented in [72]. However, the necessary step for that is to show that the duality holds between (4.3) and (4.5).

Let $\mathbb{Y}_1^{\mathbf{n}_M}$ be the adjusted version of Y_1^n for the transformation of the proposed model in Section 4.2.3 to the model in Section 4.2.4 where $\mathbf{n}_M$ is the number of states after dividing each n state properly. We assume the leakage occurs at the exit state, and the rest of the states do not emit any signal for instructions which take more than one clock cycle. Actually, most accurate approach is to split the available leakage power to all sub-states. However, we note that for any intra/initial state, we can write $\mathbb{T}_{ij}$ as

$$\mathbb{T}_{ij} \;=\; g\left(\mathbf{T}_{ij}\right) \tag{E.1}$$

where $g\left(\bullet\right)$ is a function of $\mathbf{T}_{ij}$, and $\mathbf{T}_{ij}$ can be written as

$$\mathbf{T}_{ij} \;=\; \log \frac{P_t(i,j|\mathbb{Y}_1^{\mathbf{n}_M})^{\frac{P_t(i,j|\mathbb{Y}_1^{\mathbf{n}_M})}{\mathbf{u}_i\mathbb{P}_{ij}}}}{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})^{\frac{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})}{\mathbf{u}_i}}} \tag{E.2}$$

$$\;=\; \frac{P_t(i,j|\mathbb{Y}_1^{\mathbf{n}_M})}{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})\mathbb{P}_{ij}} \log \frac{P_t(i,j|\mathbb{Y}_1^{\mathbf{n}_M})}{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})}. \tag{E.3}$$

Applying Bayesian rule, we have

$$\begin{aligned}\mathbf{T}_{ij} &= \frac{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})P_t(j|i,\mathbb{Y}_1^{\mathbf{n}_M})}{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})\mathbb{P}_{ij}} \log \frac{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})P_t(j|i,\mathbb{Y}_1^{\mathbf{n}_M})}{P_t(i|\mathbb{Y}_1^{\mathbf{n}_M})} \\ &= \frac{P_t(j|i,\mathbb{Y}_1^{\mathbf{n}_M})}{\mathbb{P}_{ij}} \log P_t(j|i,\mathbb{Y}_1^{\mathbf{n}_M}). \end{aligned} \tag{E.4}$$

For the intra/initial states, we have

$$P_t(j|i,\mathbb{Y}_1^{\mathbf{n}_M}) \overset{(a)}{=} P_t(j|i) \overset{(b)}{=} \mathbb{P}_{ij}$$

where (a) follows that there exists only one path from state i, where i is an intra/initial state, to any other state independent of any given sequence, and (b) follows that the transition probability for the Markov chain provides sufficient information to describe any transition probability from one state to another at any given time. Therefore, assigning an arbitrary power values for these states do not affect the transition probability at time t given the output sequence. For the tractability of the mathematical derivations, we assume these states produce no signal at all. Moreover, for these states, we can simplify (E.4) further as

$$\mathbf{T}_{ij} = \frac{P_t(j|i)}{\mathbb{P}_{ij}} \log P_t(j|i) = \frac{\mathbb{P}_{ij}}{\mathbb{P}_{ij}} \log \mathbb{P}_{ij} = 0 = \mathbb{T}_{ij} \tag{E.5}$$

for any $j \in \mathcal{S}_M$, which means

$$\mathfrak{u}_i \sum_{j \in \mathcal{S}_M} \mathbb{T}_{ij} = 0.$$

Therefore, given an instruction with an execution time larger than one, the intra/initial states of this instruction do not contribute to the equation given in (4.5) in terms of $\mathbb{T}_{ij}$. As the second step, we have to check the con-

tribution of these intra/initial states to the definition of leakage capacity. In that respect, we have

$$-\mathfrak{u}_i \sum_{j \in \mathcal{S}_M} \mathbb{P}_{ij} \log \mathbb{P}_{ij} = 0$$

since $\mathbb{P}_{ij}$ is equal to zero or one. Therefore, for the intra/initial states, we have

$$\mathfrak{u}_i \sum_{j \in \mathcal{S}_M} \mathbb{P}_{ij} \left(\mathbb{T}_{ij} - \log \mathbb{P}_{ij} \right) = 0$$

which means total contribution is zero if the considered state is an intra/initial state of an instruction whose execution time takes more than one clock cycle.

Now, let's check the values obtained from the exit states. Here, our analysis is based on the assumption that the leakage occurs at exit states. To proceed further, we define $\mathbb{T}_{ij}$ as

$$\mathbb{T}_{ij} = \lim_{n \to \infty} \frac{1}{n} \sum_{t=1}^{n} \left[\log \frac{P_t(k, l|Y_1^n)^{\frac{P_t(k,l|Y_1^n)}{\mathfrak{u}_i \mathbb{P}_{ij}}}}{P_t(k|Y_1^n)^{\frac{P_t(k|Y_1^n)}{\mathfrak{u}_i}}} \right] \tag{E.6}$$

where i is the exit state of instruction k and j is the initial state of instruction l. The reason to redefine $\mathbb{T}_{ij}$ is to satisfy the duality between the problems because it is obvious that $T_{kl} = \mathbb{T}_{ij}$ after redefining $\mathbb{T}_{ij}$. Moreover, the transition probabilities for an exit state to an initial state are kept exactly the same with the corresponding instruction to instruction transition probabilities, i.e. $P_{DIV,SUB} = \mathbb{P}_{D_4,SUB}$, $P_{MUL,SUB} = \mathbb{P}_{M_3,SUB}$, etc. (Here, transitions are based on Fig. 4.2 and Fig. 4.3), to preserve the duality.

Therefore, for the exit state k of instruction i, we can write the following

equality:

$$\sum_{j\in\mathcal{S}} P_{ij}\left[\log\frac{1}{P_{ij}}+T_{ij}\right]=\sum_{j\in\mathcal{S}_M}\mathbb{P}_{kj}\left[\log\frac{1}{\mathbb{P}_{kj}}+\mathbb{T}_{kj}\right]. \tag{E.7}$$

Note that the number of states in the original Markov Model is the same as the number of exit states in the transformed Markov Model.

Since the transformed Markov Model is also an indecomposable model, it has a stationary distribution which can be written as

$$\mathbf{u}=\mathbf{u}\mathbf{P}$$

where $\mathbf{u}$ is the state probabilities, and $\mathbf{P}$ is the matrix containing the state transition probabilities. To derive mathematical results, we utilize the classical probability constraints. For that, let $\mathfrak{u}_i$ be the stationary distribution of k^{th} sub-state of instruction M, and $\mathfrak{u}_k^M$ be its mapped version. The constraints for the transformed model are

$$\sum_i \mathfrak{u}_i=\sum_{i,j}\mathfrak{u}_i^j=1$$

and

$$\mathfrak{u}_i^j=\mathfrak{u}_k^j,\ \forall i,k\in\{1,\cdots,L_j\}\text{ and }j\in\mathcal{S},$$

i.e., $\mathfrak{u}_{M_1}=\mathfrak{u}_{M_2}=\mathfrak{u}_{M_{L_{MUL}}}$. Therefore, we have

$$\sum_{i\in\mathcal{S}_M}\mathfrak{u}_i=\sum_{i\in\mathtt{E}(\mathcal{S}_M)}L_i\mathfrak{u}_{i_{L_i}}=1 \tag{E.8}$$

where $\mathtt{E}\left(\mathcal{S}_M\right)$ is the set containing exit states of instructions. Let us rewrite

the capacity definition for the transformed model as

$$C_T = \max_{\mathbb{P}_{ij}} \sum_{i,j:(i,j)\in\mathcal{T}_M} \mathfrak{u}_i \mathbb{P}_{ij} \left[\log \frac{1}{\mathbb{P}_{ij}} + \mathbb{T}_{ij}\right] \tag{E.9}$$

$$= \max_{P_{ij}} \sum_{i,j:(i,j)\in\mathtt{E}(\mathcal{T}_M)} \mathfrak{u}_i P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right] \tag{E.10}$$

where (E.10) follows the equality given in (E.7), and $\mathtt{E}(\mathcal{T}_M)$ represents the state transition set of all exit states. To proceed forward, we define

$$\mathfrak{u}^j_{L_j} = \frac{\mu_j}{\sum\limits_{k\in\mathcal{S}} L_k \mu_k}$$

which obeys the probability constraint that

$$1 = \sum_{i\in\mathcal{S}_M} \mathfrak{u}_i = \sum_{i\in\mathtt{E}(\mathcal{S}_M)} L_i \mathfrak{u}_i = \sum_{j\in\mathcal{S}} L_j \mathfrak{u}^j_{L_j} \tag{E.11}$$

$$= \sum_{j\in\mathcal{S}} \frac{L_j \mu_j}{\sum\limits_{k\in\mathcal{S}} L_k \mu_k} = \frac{\sum\limits_{i\in\mathcal{S}} L_i \mu_i}{\sum\limits_{k\in\mathcal{S}} L_k \mu_k} = 1 \tag{E.12}$$

where (E.11) follows the equality given in (E.8). Therefore, we have

$$C_T = \max_{\mathbb{P}_{ij}} \sum_{(i,j)\in\mathcal{T}_M} \mathfrak{u}_i \mathbb{P}_{ij} \left[\log \frac{1}{\mathbb{P}_{ij}} + \mathbb{T}_{ij}\right] \tag{E.13}$$

$$= \max_{\mathbb{P}_{ij}} \sum_{(i,j)\in\mathtt{E}(\mathcal{T}_M)} \mathfrak{u}_i \mathbb{P}_{ij} \left[\log \frac{1}{\mathbb{P}_{ij}} + \mathbb{T}_{ij}\right] \tag{E.14}$$

$$= \max_{P_{ij}} \sum_{(i,j)\in\mathcal{T}} \frac{\mu_i}{\sum\limits_{k\in\mathcal{S}} L_k \mu_k} P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right] \tag{E.15}$$

$$= \max_{P_{ij}} \frac{\sum\limits_{(i,j)\in\mathcal{T}} \mu_i P_{ij} \left[\log \frac{1}{P_{ij}} + T_{ij}\right]}{\sum\limits_{k\in\mathcal{S}} L_k \mu_k} \tag{E.16}$$

where (E.15) follows the equality given in (E.7). Since (E.16) is exactly

same with (4.3), the proposed transformation preserves the duality which ends the proof.

APPENDIX F

MATHEMATICAL DERIVATION OF ESP

In this section, we show how ESP is related to the alternation power at the corresponding frequency. For measurement and derivation purposes of ESP, the code given in Fig. 2.1 is used. Here, we assume that the sampled sequence of $s(t)$ is $s[m]$, and each sample can be written as $s[m] = i[m]+w[m]$ where $i[m]$ is the emanated signal sample and $w[m]$ is additive independent and identically distributed (i.i.d.) white noise with zero mean and variance σ_w^2. We assume that the noise term contains all disruptive signal powers and their variations.

Let $s_1^{L_1}[m]$ be the sequence corresponding to only one period of the first for-loop signal, and the length of $s_1^{L_1}[m]$ is N_L. We can decompose $s_1^{L_1}[m]$ into three different sequences. Assuming the depth of the pipeline is P_S, these sequences are:

* The samples of the considered instruction including all pipeline stages:

$$a_{\mathcal{A}_1}[m] = \left[0,\ \cdots,\ 0, a_{\mathcal{A}_1}^1, a_{\mathcal{A}_1}^2, \cdots, a_{\mathcal{A}_1}^p,\ a_{\mathcal{A}_1}[0],\ \cdots,\ a_{\mathcal{A}_1}[N_I-1], a_{\mathcal{A}_1}^{p+1}, \cdots, a_{\mathcal{A}_1}^{P_S}\right]$$

  where $a_{\mathcal{A}_1}^i$ is the i^{th} sample of the emitted signal when $\mathcal{A}_1$ is in a pipeline stage rather than execution.

* The samples of other activities rather than $\mathcal{A}_1$ to make the micro-benchmark practical including the pipeline effect:

$$o_{L_1}[m] = \left[o[0],\ o[1],\ \cdots,\ o[N_L-2],\ o[N_L-1]\right].$$

Here, we need to note that the samples taken for the first iteration of the inner for-loop will be different than the other iterations even for the ideal case due to pipeline depth. Although it looks like the periodicity is not valid for $o_{L_1}[m]$, we can able to ignore it thanks to the assumption that n_{inst} is large.

* Finally, the last sequence compromises all other components which are assumed to be Gaussian and given as

$$w_{L_1}[m] = [w[0],\ w[1],\ \cdots,\ w[N_L - 1]]\,.$$

Combining all these sequences, we have

$$s_1^{L_1}[m] = a_{\mathcal{A}_1}[m] + o_{L_1}[m] + w_{L_1}[m].$$

Following the same decomposition for the second for-loop signal, called $s_2^{L_2}[n]$, we have

* $o_{L_2}[m] = [o[0],\ o[1],\ \cdots,\ o[N_L - 2],\ o[N_L - 1]]$,

* $w_{L_2}[m] = [w[0],\ w[1],\ \cdots,\ w[N_L - 1]]$,

which leads to

$$s_2^{L_2}[m] = o_{L_2}[m] + w_{L_2}[m].$$

Here, we assume that NOP consumes very little energy as it passes through the stages of a pipeline, which means it produces a signal whose power is close to zero. Observe here that since both loops are almost identical except the part where $\mathcal{A}_1$ is inserted, we assume that $o_{L_1}[m]$ and $o_{L_2}[m]$ are identical to each other, therefore, we refer both sequences as $\mathrm{o}[m]$. Let $p[m]$

be a square wave with 50% duty cycle and period of $2N_L n_{inst}$ samples, and $\mathbf{s}[m]$ be the one period signal of the outer for-loop. Let also $\mathbf{a}_{\mathcal{A}_1}[m]$ and $\mathbf{o}[m]$ be generated by concatenating $a_{\mathcal{A}_1}[m]$ and $o_{L_1}[m]$ by $2 \cdot n_{inst}$ times, respectively. Furthermore, we can simply assume that the noise components are i.i.d. for both for-loops. Therefore, we have

$$\mathbf{s}[m] = p[m]\mathbf{a}_{\mathcal{A}_1}[m] + \mathbf{o}[m] + \mathbf{w}[m].$$

The first harmonic of $\mathbf{s}[m]$ can be written as

$$\mathbf{S}[1] = \frac{\sum_{\gamma=0}^{2N_L n_{inst}-1} P[1-\gamma]\mathbf{A}_{\mathcal{A}_1}[\gamma]}{2N_L n_{inst}} + \mathbf{O}[1] + \mathbf{W}[1]. \tag{F.1}$$

We know that $\mathbf{O}[k]$ and $\mathbf{A}_{\mathcal{A}_1}[k]$ have nonzero frequency components only if $k = 2 \cdot n_{inst} \cdot l,\ \forall l \in \{0, \cdots, N_L - 1\}$, and $|P[1]| \gg |P[1 - 2n_{inst}]|$. Therefore, (F.1) can be approximately written as

$$\mathbf{S}[1] \approx \frac{P[1]}{2N_L n_{inst}} \mathbf{A}_{\mathcal{A}_1}[0] + \mathbf{W}[1]. \tag{F.2}$$

If we take the magnitude square of both sides, we have

$$\begin{aligned} |\mathbf{S}[1]|^2 &= \left| \frac{P[1]}{2N_L n_{inst}} \mathbf{A}_{\mathcal{A}_1}[0] + \mathbf{W}[1] \right|^2 \\ &= \left| \frac{P[1]}{2N_L n_{inst}} \mathbf{A}_{\mathcal{A}_1}[0] \right|^2 + |\mathbf{W}[1]|^2 - \frac{\Re e\left\{P[1]\mathbf{A}_{\mathcal{A}_1}[0]\mathbf{W}^*[1]\right\}}{N_L \cdot n_{inst}} \end{aligned} \tag{F.3}$$

where $(\cdot)^*$ is conjugation and $\Re e\{\cdot\}$ takes the real part of its argument. Assuming

$$\left| \frac{P[1]}{2N_L n_{inst}} \mathbf{A}_{\mathcal{A}_1}[0] \right| \gg \Re e\left\{P[1]\mathbf{A}_{\mathcal{A}_1}[0]\mathbf{W}^*[1]\right\},$$

the first harmonic of $\mathbf{s}[m]$ can be simplified further as

$$|\mathbf{S}[1]|^2 \approx \left|\frac{P[1]}{2N_L n_{inst}}\mathbf{A}_{\mathcal{A}_1}[0]\right|^2 + |\mathbf{W}[1]|^2 . \tag{F.4}$$

To proceed further, we need to have the expression for $\mathbf{A}_{\mathcal{A}_1}[0]$. Utilizing the DFS, we have

$$\mathbf{A}_{\mathcal{A}_1}[0] = \sum_{\gamma=0}^{2N_L n_{inst}} \mathbf{a}_{\mathcal{A}_1}[\gamma] \overset{(a)}{=} 2n_{inst}\sum_{\gamma=0}^{N_L} \mathbf{a}_{\mathcal{A}_1}[\gamma] \tag{F.5}$$

where (a) follows the fact that $\mathbf{a}_{\mathcal{A}_1}[m]$ is periodic with N_L samples. Since, at each period, only $N_I + P_S$ of $\mathbf{a}_{\mathcal{A}_1}[m]$ have nonzero values, and assuming $N_I + P_S$ is large enough, (F.5) can be written as

$$\begin{aligned}\mathbf{A}_{\mathcal{A}_1}[0] &= 2(N_I + P_S)n_{inst}\mathbb{E}\left[\mathbf{a}_{\mathcal{A}_1}[m]\right] \\ &= 2(N_I + P_S)n_{inst}\mu_{\mathcal{A}_1}\end{aligned} \tag{F.6}$$

Note that exploiting (4.7), $\mathrm{ESP}[\mathcal{A}_1]$ can also be written as

$$\begin{aligned}\mathrm{ESP}[\mathcal{A}_1] &= \frac{T_s(N_I + P_S)}{R}\mathbb{E}\left[|\mathbf{a}_{\mathcal{A}_1}[m]|^2\right] \\ &= \frac{T_s(N_I + P_S)}{R}\left(\mu^2_{\mathcal{A}_1} + \sigma^2_{\mathcal{A}_1}\right) \\ &\approx \frac{T_s(N_I + P_S)}{R}\mu^2_{\mathcal{A}_1}\end{aligned} \tag{F.7}$$

where $\sigma_{\mathcal{A}_1}$ is the standard deviation of the samples while an instruction signal is executed, and (F.7) follows the assumption that the variation in measured signal during the execution of an instruction is much smaller than its mean value. Combining (F.6) with (F.7), we have

$$\mathrm{ESP}[\mathcal{A}_1] \approx \frac{T_s}{4R(N_I + P_S)n^2_{inst}}|\mathbf{A}_{\mathcal{A}_1}[0]|^2. \tag{F.8}$$

The final step is to show how $\mathrm{ESP}[\mathcal{A}_1]$ and the alternation power $\mathcal{P}(f_{\mathrm{alt}})$ are related to each other. The relation between the first harmonic of the signal and the power measure through the spectrum analyzer is given as [115], [114]

$$\mathcal{P}(f_{\mathrm{alt}}) = \frac{2}{R}\left(\frac{|\mathbf{S}[1]|}{2 \cdot N_L \cdot n_{inst}}\right)^2. \tag{F.9}$$

Let $\mathcal{P}_{\mathcal{A}_1}(f_{\mathrm{alt}})$ be the measured alternation power when $\mathcal{A}_1$ is inserted into first for-loop, and the second loop is kept empty. On the other hand, let $\mathcal{P}_0(f_{\mathrm{alt}})$ be the measured power when both for-loops are kept empty (Here, we need to remark that keeping the loops empty means inserting NOP as many as the total number of clock cycles required to execute $\mathcal{A}_1$). Finally, let $\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\mathrm{alt}})$ be the normalized alternation power for the instruction $\mathcal{A}_1$ which is defined as

$$\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\mathrm{alt}}) = \mathcal{P}_{\mathcal{A}_1}(f_{\mathrm{alt}}) - \mathcal{P}_0(f_{\mathrm{alt}}).$$

The critical observation is that the term related to $\mathcal{A}_1$ in (F.4) is zero when both for loops are kept empty. Assume $\mathbf{S}_{\mathcal{A}_1}[1]$ and $\mathbf{S}_0[1]$ denote the first harmonics of the signal when 1) $\mathcal{A}_1$ is inserted, and 2) both loops are kept empty, respectively. Considering this setup, we can write

$$|\mathbf{S}_{\mathcal{A}_1}[1]|^2 - |\mathbf{S}_0[1]|^2 \;\approx\; \frac{1}{\pi^2}\,|\mathbf{A}_{\mathcal{A}_1}[0]|^2 \tag{F.10}$$

where we utilize the approximation that $\pi|P[1]| \approx 2N_L n_{inst}$. Exploiting the definition of normalized alternation power, and employing the equations

given in (F.8), (F.9), and (F.10), we can write

$$\begin{aligned}
\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\text{alt}}) &= \mathcal{P}_{\mathcal{A}_1}(f_{\text{alt}}) - \mathcal{P}_0(f_{\text{alt}}) \\
&= \frac{2/R}{\left(2N_L n_{inst}\right)^2}\left(|\mathbf{S}_{\mathcal{A}_1}[1]|^2 - |\mathbf{S}_0[1]|^2\right) \\
&= \frac{2/R}{\left(2N_L n_{inst}\right)^2}\frac{1}{\pi^2}\left|\mathbf{A}_{\mathcal{A}_1}[0]\right|^2 \\
&= \frac{\text{ESP}[\mathcal{A}_1]}{\left(\pi N_L\right)^2}\ \frac{2(N_I + P_S)}{T_s}. \qquad \text{(F.11)}
\end{aligned}$$

To emphasize the relation between the power at the alternation frequency and ESP, we can write

$$f_{\text{alt}} \cdot n_{inst} = \frac{1}{2 \cdot N_L \cdot T_s}. \qquad \text{(F.12)}$$

Plugging the equation (F.12) into (F.11), we have

$$\begin{aligned}
\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\text{alt}}) &= \left(\frac{2}{\pi}\right)^2 \frac{\text{ESP}[\mathcal{A}_1]}{N_L/(N_I + P_S)}\frac{1}{2N_L T_s} \\
&= \left(\frac{2}{\pi}\right)^2 \frac{\text{ESP}[\mathcal{A}_1]}{N_L/(N_I + P_S)} f_{\text{alt}} n_{inst}. \qquad \text{(F.13)}
\end{aligned}$$

To finalize our proof, we need to keep $\text{ESP}[\mathcal{A}_1]$ alone on the one side. Therefore, we have

$$\text{ESP}[\mathcal{A}_1] = \left(\frac{\pi}{2}\right)^2 \frac{\boldsymbol{\mathcal{P}}_{\mathcal{A}_1}(f_{\text{alt}}) \cdot N_L}{(N_I + P_S) \cdot f_{\text{alt}} \cdot n_{inst}} \qquad \text{(F.14)}$$

which concludes the proof.

APPENDIX G
A MICROARCHITECTURE-LEVEL ELECTROMAGNETIC SIDE-CHANNEL SIGNAL MODELING

G.1 Overview

Monitoring computer system activities on the instruction level provides more resilience to malware attacks because these attacks can be analyzed better by observing the changes on the instruction level. Assuming the source code is available, many training signals can be collected to track the instruction sequence to detect whether a malware is injected or the system works properly. Most of the previous work on side channel instruction tracking has been concentrated on building side-channel-based disassembler [116, 117, 118]. As much as their work is a specific reverse engineering application and they take a different approach, our works are similar in the sense that we both try to extract the executed instruction of a device through side-channel analysis. Eisenbath et. al. [116] use a microcontroller with the simplest form of pipelining (two stages: prefetch and execute) and the clock frequency of the target device is as low as 1 MHZ. Park et. al. [117] works on a device whose operating clock frequency is 16 MHz, but the pipeline has again only two stages. Unlike the former papers which utilize the power side channel, Strobel et al. [118] uses EM side-channel; the operating frequency of the microcontroller used in their study is 4 MHz and the pipeline has only two stages. They also modify the target device by decapculating the IC to locate the probe closer to the die of the microcontroller [118]. Another similar application of instruction

tracking is Msnga et al.[62]'s instruction-level side channel profiling approach, where they use a microcontroller with 4 MHz clock frequency and two pipeline stages. The previous work demonstrates the feasibility of instruction tracking either through power or EM side channels for the specific target devices. However, the target devices that are used in previous work do not represent the modern computers due to the following reasons:

- The operating clock frequency of the target devices are very low.
- Target devices employ the simplest form of pipelining architecture whereas the modern computers have much more advanced pipelining architectures with more than several pipeline stages.

Therefore, training signals have to be collected with high sampling rate to ensure that the significant features of these signals do not vanish. Since the clock frequencies of the current computer systems are extremely high, we need to have a commercial device with high sampling rate, i.e. 10 GHz, which either costs remarkably high, or does not exist. To eliminate the deficiencies regarding the insufficient sampling rate, we propose a method to increase the sampling rate with the moderate commercial devices for training symbols. In that respect, we first generate some random instruction sequences which exist in the inspected source code. Then, these sequences are executed in a for-loop, and emanated electromagnetic (EM) signals from the processor are collected by a commercially available device with moderate sampling rate, i.e. sampling rate is much smaller than the clock frequency. Lastly, we apply a mapping of the gathered samples by utilizing modulo of their timings with respect to execution time of overall instruction sequence. As the final step, we provide some experimental results to illustrate that we successfully track the instruction sequence by

applying the proposed approach.

G.2 A Method for Generating Training Sequences for Single Instruction Tracking

In order to track instructions through emanated EM signals, it is necessary to have EM signatures for each instruction that can be used for comparison. Earlier work demonstrates several ways of generating signatures such as applying dimensionality reduction methods to the recorded signal and using machine learning classification techniques [116, 62, 118], and recording the signature of the instruction for several times and simply averaging them [119]. In this chapter, *modulo operation* technique that is explained in detail in Section G.3 is adapted for generating the EM signatures.

As mentioned earlier, two of the main challenges of our work can be expressed as the inadequate sampling rate and not having a model that describes the pipeline effect on the emanated EM signal. In order to address the latter issue, it's been established to use *sequences of instructions* instead of *single instructions*. When an instruction is executed, pipeline is filled with neighboring instructions that are just before or after that instruction, and those neighboring instructions are included in the instruction sequence. As much as the individual contributions from the neighboring instructions that appear in different stages of pipeline are not known, using the whole sequence as signature allows us to include their aggregate effect.

Regarding inadequate sampling rate, *modulo operation* becomes a very useful technique. Assuming we have a signal that is composed of samples from a periodic signal, *modulo operation* can be used to upsample that

signal (reader can refer to Section G.3 for detailed analysis of *modulo operation*). One should note that, when the instruction sequence is executed, the emanated EM signal occurs only once and hence it is not a periodic signal. However, since we have access to the device during training phase, we can artificially mimic the required periodicity condition by executing the same sequence consecutively. It should also be noted that the repetition of the instruction sequence is only used in the training phase and not in the testing phase, where instruction sequence is executed only once.

Considering the requirements above, we can summarize the setups used for training and testing phases as follows.

- **Training Setup:** This setup is used to generate EM signatures. Therefore, the instruction sequence is executed for N consecutive times by utilizing a for loop. The pseudo code for this setup can be seen in Figure G.1. The empty for-loops before and after the execution of the instruction sequence are used as markers to detect the beginning and ending of the instruction sequence.

- **Testing Setup:** This setup is used for testing the generated EM signatures. Unlike **Training Setup**, the instruction sequence is executed only once. Therefore, it is not possible to apply *modulo operation* in this case. This means that the sampling frequency of the captured EM signal in this setup is not as high as the resulting EM signatures. The pseudo code for this setup can be seen in Figure G.2. The empty for-loops before and after the execution of the instruction sequence are similarly used as markers to detect the beginning and ending of the instruction sequence.

Implementing *modulo operation* while generating the EM signatures

```
for
#empty for loop
end

for N times
#Instruction sequence
end

for
#empty for loop
end
```

Figure G.1: Pseudocode for **Training Setup**.

```
for
#empty for loop
end

#Instruction sequence

for
#empty for loop
end
```

Figure G.2: Pseudocode for **Testing Setup**.

Figure G.3: Pseudocodes for **Training** and **Testing** setups used for generating EM signatures and testing the generated signatures

does not only increase the sampling rate, but also the resulting EM signature is, in a way, an averaged version of N executions of the same instruction sequence. This characteristic of the operation is especially useful when the noise level is very high. The averaging nature of the *modulo operation* can be used to filter out the noise, and the underlying EM emanation that has been formerly overshadowed by the noise can be recognized and extracted as signature. One should also note that, repeating the instruction sequence consecutively results in a different pipeline effect. As opposed to the single execution case, **Training Setup** helps to decrease the contribution coming from the previous and subsequent instructions.

By using sequences of instructions instead of single instructions, we are inevitably increasing the total possible number of EM signatures that needs to be generated. On the other hand, the longer the sequence, the better the pipeline effect is included in the generated EM signature. This situation indicates the trade off between the total number of EM signatures and the ability to address the pipeline effect.

G.3 Modulo Operation to Increase Effective Signal Sampling Rate

The technique that we propose in this section, *modulo operation*, is a very simple but effective method that enables generating well-conditioned EM signatures. In this section, first, we explain the theory behind the technique, then illustrate the result of the technique through simple examples and finally discuss the required conditions for this operation to be applicable.

Theoretical Perspective: Let's assume that $y(t)$ is a long periodic signal with period T, and T_s is the sampling period of the measuring instrument, which (without loss of generality) can be formulated as

$$T_s = kT + \Delta_m. \tag{G.1}$$

Here, k is a fixed non-negative integer, T is the period of the target signal $y(t)$ and Δ_m (that will be referred as *modular offset*) is defined as:

$$\Delta_m = mod(T_s, T). \tag{G.2}$$

It should be noted any T_s can be expressed with the formulation given in (G.1). Now, let $y_s[n]$ be a discrete time sequence which is constructed by sampling $y(t)$ with the sampling period T_s as

$$y_s[n] = y(nT_s), \text{ for } \forall n \text{ where } n \in \{0, 1, 2, ..., N-1\}. \tag{G.3}$$

Since T_s is known, sampling times, t_n's, for each sample n are also known:

$$t_n = nT_s, \text{ for } \forall n \text{ where } n \in \{0, 1, 2, ..., N-1\}. \tag{G.4}$$

Let t_n^{mod} be the time indicating where the n^{th} sample corresponds in the fundamental period of $y(t)$ and it can be found as

$$\begin{aligned} t_n^{mod} &= mod\left(t_n, T\right) = mod\left(n\left(kT + \Delta_m\right), T\right) \\ &= mod\left(n\Delta_m, T\right), \text{ for } \forall n \text{ where } n \in \{0, 1, 2, ..., N-1\}. \end{aligned} \tag{G.5}$$

It should be noted that $0 \leq t_n^{mod} < T$. In a way, the value of the n^{th} sample $(y_s[n])$ can be wrapped around and used to fill in the value for t_n^{mod}. After repeating this process for all samples and sorting t_n^{mod}'s accordingly, we can obtain several samples in $y(t)$'s fundamental period $[0, T]$. Determining the exact number of the new samples in $y(t)$'s fundamental period $[0, T]$ is more subtle.

Number of Samples: As we repeat calculating t_n^{mod} for increasing n's, every sample fills in a distinct point within $[0, T]$ until t_n^{mod} becomes zero again (other than $n = 0$ for which t_n^{mod} is always zero). Let M denote the smallest nonzero sample index n for which $t_n^{mod} = 0$. For any sample with index number M or greater than M, it is not possible to fill in a distinct point within $[0, T]$ because the value at t_n^{mod} is already filled by a previous sample. In other words, any subsequent sample whose sample index $n \geq M$ does not increase the total number of sample points within the fundamental period. Then, determining the exact number of new samples transforms into finding M, the nonzero sample index for which $t_n^{mod} = 0$. In order to find that, we should focus on the definition that is given in (G.5): $t_n^{mod} = mod\left(n\Delta_m, T\right)$. One should realize that $t_n^{mod} = 0$ for $n = 0$, but we are looking for the sampling index $n > 0$. Then, it is not very difficult to see that $mod\left(n\Delta_m, T\right)$ becomes zero when $n\Delta_m$ is the same as the least common multiple of Δ_m and T.

Let $lcm(x, y)$ correspond to an operator whose result is the least common multiple of its operands x and y. It should be noted that we are using a looser least common multiple operator for which the result and operands x and y do not necessarily have to be integers, i.e., $lcm(0.01, 0.02) = 0.02$.

Hence, the total number of samples in one period of $y(t)$ is equal to M which can be found as:

$$M = \frac{lcm(\Delta_m, T)}{\Delta_m}. \tag{G.6}$$

New samples are spaced by $T_s^n = T/M$, therefore, the new sampling frequency f_s^n is:

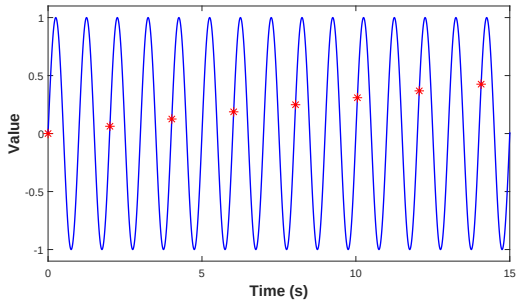
$$f_s^n = \frac{1}{T_s^n} = \frac{M}{T} = \frac{lcm(\Delta_m, T)}{T\Delta_m}. \tag{G.7}$$

If the total number of samples, N, is at least as much as M, we can make sure that we have at least one sample that represents the value of the signal at $t = 0, T_s^n, 2T_s^n, ..., (T - T_s^n)$. Therefore, when this condition is satisfied, new sampling time of the signal becomes T_s^n. Consequently, it can be concluded that the effective sampling rate has been increased from $\frac{1}{T_s}$ to $\frac{1}{T_s^n} = \frac{lcm(\Delta_m, T)}{T\Delta_m}$.

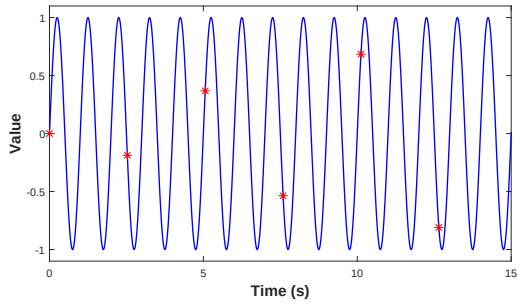
Detailed Explanation: In order to illustrate what *modulo operation* does, consider the following signal:

$$y(t) = \sin(2\pi t). \tag{G.8}$$

We consider two different cases of T_s: $T_{s_1} = 2.01$ and $T_{s_2} = 2.53$. Since the period of $y(t)$ is 1, the *modular offsets* are $\Delta_{m_1} = 0.01$ and $\Delta_{m_2} = 0.53$, respectively. After sampling $y(t)$ with sampling periods T_{s_1} and T_{s_2}, we obtain the following discrete time signals:



(a) $y(t)$ (solid curve) and $y_s[n]$ (markers) for $T_{s_1} = 2.01$.



(b) $y(t)$ (solid curve) and $y_s[n]$ (markers) for $T_{s_1} = 2.01$.

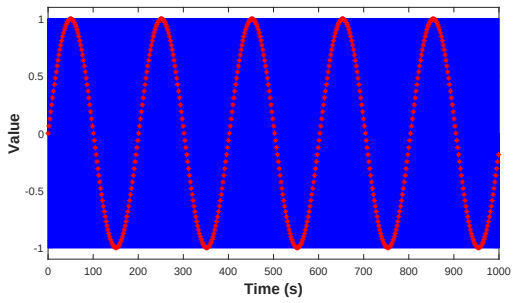
Figure G.4: The solid curves represent the continuous target signal $y(t) = \sin(2\pi t)$, whereas the markers represent $y_s[n]$ (samples obtained from $y(t)$ with sampling rates $T_{s_1} = 2.01$ (a) and $T_{s_2} = 2.53$ (b)). One should note that this is a case where the signal is heavily undersampled that causes aliasing.

$$y_{s_1}[n] = y(nT_{s_1}) = \sin(2\pi n T_{s_1}) \quad \text{and} \quad y_{s_2}[n] = y(nT_{s_2}) = \sin(2\pi n T_{s_2}), \text{ for } \forall n \tag{G.9}$$

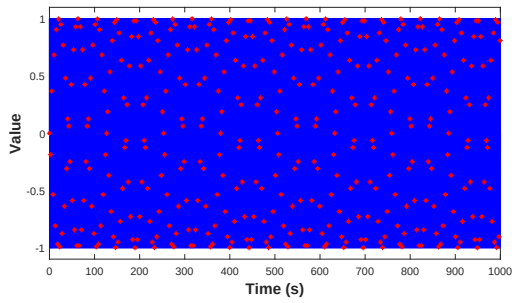
where $n \in \{0, 1, 2, ..., N-1\}$.

The solid curves in Figure G.4 illustrate $y(t)$, whereas the markers in Figure G.4(a) and G.4(b) represent $y_{s_1}[n]$ and $y_{s_2}[n]$, respectively. One should note that for both cases $T_s > T$ and we have less than 1 sample for each period of $y(t)$. This situation presents a scenario where the sampling rate is very low with respect to the frequency of the signal under investigation. In classical signal processing terms, sampling frequency is much less than the Nyquist rate and consequently, aliasing occurs. Due to aliasing, $y_{s_1}[n]$ and $y_{s_2}[n]$ are completely different from each other even though they are obtained from the same signal. Without any further assumption, $y_{s_1}[n]$ and $y_{s_2}[n]$ cannot be used to recover the target signal $y(t)$.

In Figure G.5, time axis is extended to $[0, 1000]$ s, and one can note that



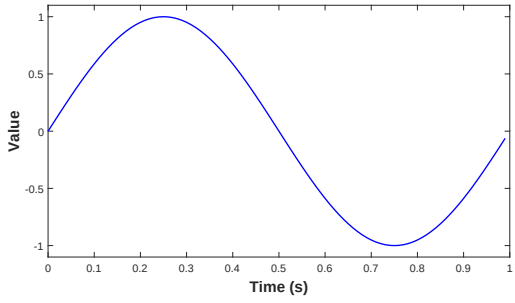
(a) $y_s[n]$ for $T_{s_1} = 2.01$.



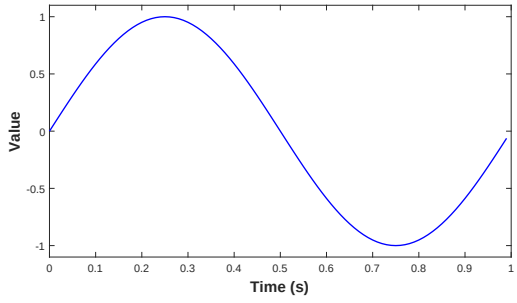
(b) $y_s[n]$ for $T_{s_2} = 2.53$.

Figure G.5: Time axis is extended to $[0, 1000]$ s for Figure G.4, and the markers start to resemble $y(t)$ in (a) but not in (b)

the waveform that is generated by the markers in Figure G.5(a) resembles the sinusoidal target signal $y(t)$ on a different timing scale. However, Figure G.5(b) does not resemble $y(t)$ at all. Aliasing in Figure G.5(b) is, therefore, very obvious, but one should also realize that aliasing occurs in Figure G.5(a) as well. Although the shape of the markers resemble the target signal, the period of the signal generated by markers is completely different than the period of the target signal.



(a) Samples sorted (t_n^{mod}) for $T_{s_1} = 2.01$.



(b) Samples sorted by (t_n^{mod}) for $T_{s_2} = 2.53$.

Figure G.6: When samples are sorted by their modular sampling timing (t_n^{mod}), the reconstructed signal fully resembles y(t) in its fundamental period for both (a) and (b).

The results obtained after applying the *modulo operation* are presented in Figure G.6 and it can be easily noted that the reconstructed signals from

both $y_{s_1}[n]$ and $y_{s_1}[n]$ are identical to $y(t)$ for $t \in [0, T]$. These figures expose the strength of the proposed approach to increase the sampling rate. We have first showed that $y_{s_1}[n]$ and $y_{s_2}[n]$ are samples obtained from the same target signal $y(t)$, but they are completely different signals due to aliasing. However, applying the *modulo operation* helped us to recover the underlying target signal in both of the cases. Therefore, beyond increasing the sampling rate, *modulo operation* is especially useful to deal with aliasing.

This example represents how *modulo operation* can be used to reconstruct a periodic signal in its fundamental period with higher effective sampling rate. One should note that we assume a few conditions that are required for *modulo operation* to be useful:

- The target signal is a periodic signal and its period is known or easily estimated.
- Modular offset ($\Delta_m = mod(T, T_s)$) is nonzero, in other words, the sampling period T_s is not an integer multiple of the period of the target signal.

For any signal that satisfies the forementioned conditions, *modulo operation* can be applied. We now show that the signal obtained by using **Training Setup** in Section G.2 indeed satisfied those conditions:

- Same instruction sequence is executed several times consecutively by using a for loop, and therefore, the signal becomes periodic during the execution of the for loop. Also, because the total number of executions of the instruction sequence is determined by us (thus, known to us), the period of the instruction sequence (the time it takes for the instruction sequence to be executed only once) can be estimated through dividing the total execution time by the number of iterations.

With this specific setup, the periodicity condition can be satisfied and the period can be easily estimated.

- Sampling period, T_s, is determined by the sampling equipment. Usually, it is very unlikely to have T_s be an integer multiple of T, but even if this occurs, sampling time of the equipment can be slightly adjusted in a way to avoid T_s from being an integer multiple of T.

The discussion above shows that **Training Setup** used in our experiments satisfies the required conditions for applying *modulo operation*. It is worth re-emphasizing that this setup is used for generating the EM signatures (where we have much more degree of freedom in modifying the source code), but it is not repeated for the actual instruction tracking stage **Testing Setup**, because, the instructions are executed only once in the actual tracking stage.

Finally, it may be worthwhile to note that *modulo operation* is not increasing the real-time sampling rate. Instead, *modulo operation* uses the samples that are recorded from several repetitions of a periodic signal and creates a new signal whose time scope is limited to one period of the signal, but the new signal contains samples that are spaced much closer to each other in time.

G.4 Experimental Results and Discussion

G.4.1 Experimental Results on Target Device 1 (FPGA)

The first target device that is used in our experiments is Altera DE1 Cyclone II, a high-density, low cost FPGA (Field Programmable Gate Array). For implementing our code on the device, we use Altera's embedded processor Nios II that employs a relatively advanced pipeline architecture with

6 pipeline stages (Fetch, Decode, Execute, Memory, Align and Writeback) and operates at 50 MHz clock frequency. The considered instructions are given in Table 2.1.

In this section, we consider 20 instruction sequences that are listed in Appendix G.6. As it can be noted, the instructions include the basic arithmetic operations such as `ADD`, `SUB`, `MUL`, `DIV`, as well as load `(LDM)` and store `(STM)` instructions. We would like to point out that the number of sequences that are used in the experiment is not enough to conclude instruction tracking at single instruction level. However, recognition of the presented sequences with reasonably high success rate supports the applicability of our proposed method. After having high success rate with the provided instruction sequences, instruction level tracking can be achieved by a rigorous sequence selection and experimentation.

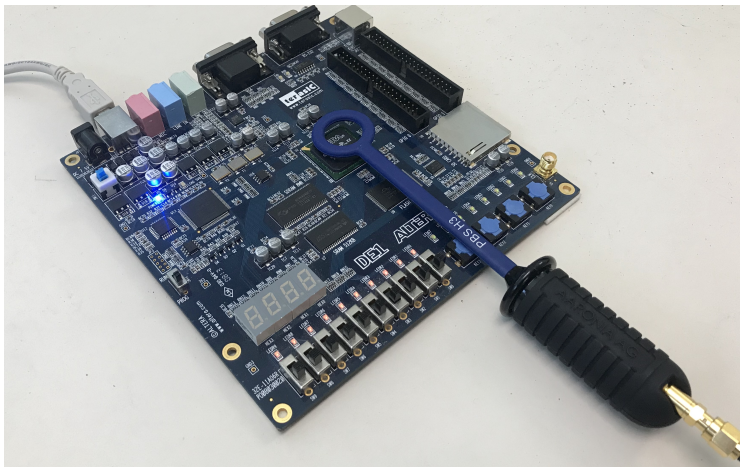


Figure G.7: Measurement Setup: Magnetic near field probe is located on top of the processor of the FPGA

Earlier work has shown that EM emanations occur at different frequencies [3]. In our study, we are particularly interested in the EM emanations around the clock frequency of our target device. We measure the emanated EM signals with a near field magnetic probe (AAronia PBS H3) that is lo-

cated above the processor of the target device as shown in Figure G.7. For recording the measured data, we use a spectrum analyzer (Keysight N9020A MXA). Main reason behind using this device is the built-in features of the device to downconvert the recorded signal and the visualize the signal around the clock frequency. However, any other less expensive device with similar sampling rate features can be used to repeat the same experiments.

To give the reader more insight on how the emanated EM signals and the result of *modulo operation* look like, Figure G.8 displays the recorded signal for a specific instruction sequence (`DIV-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD-DIV`). The top and bottom plots in Figure G.9 represent the *modulo operation* result and the single execution of the same instruction sequence, respectively. One should note that the plot on the top is a much more smooth plot due to higher resulting sampling rate and the averaging nature of *modulo operation*. In fact, for this specific instruction sequence, the sampling rate is increased by a factor of 125. In order to compare those two signals, we use Pearson correlation coefficient [120]. Clearly, these two discrete time signals have different number of samples. Therefore, the *modulo operation* result is resampled at the same time instances of the single execution signal. Resampled *modulo operation* result and the original single execution result are plotted on top of each other in Figure G.10 and the resulting correlation coefficient is 0.96. Promisingly, when the same *modulo operation* result is correlated with single execution of other instruction sequences, the correlation coefficient is less than 0.9 for all different sequences.

Table G.1 presents the correlation matrix between the EM signatures that are obtained with *modulo operation* and the single execution of the

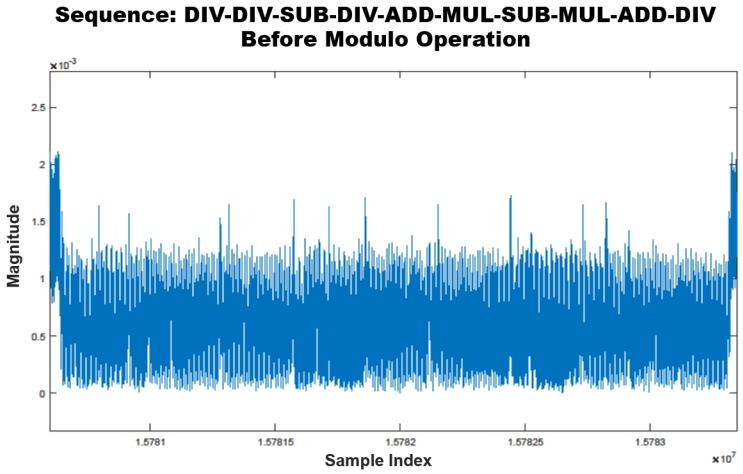


Figure G.8: Recorded signal for the given sequence in **Training Setup** before *modulo operation*.

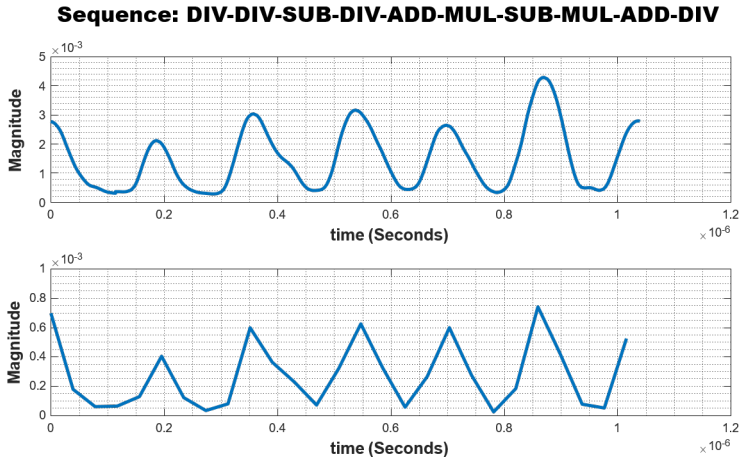


Figure G.9: The plot on the top presents the result of the *modulo operation* obtained by using **Training Setup**, plot on the bottom presents the single execution of the same instruction sequence obtained by **Testing Setup**.

instruction sequences. It can be easily noted that the dominant terms in the matrix are the diagonal terms, which supports the claim that the generated EM signatures by *modulo operation* can be used to detect the corresponding instruction sequences. One should also note that, similar instruction sequences (such as instruction sequence 10 and 11, which differ only in one instruction) have higher cross correlation coefficients,

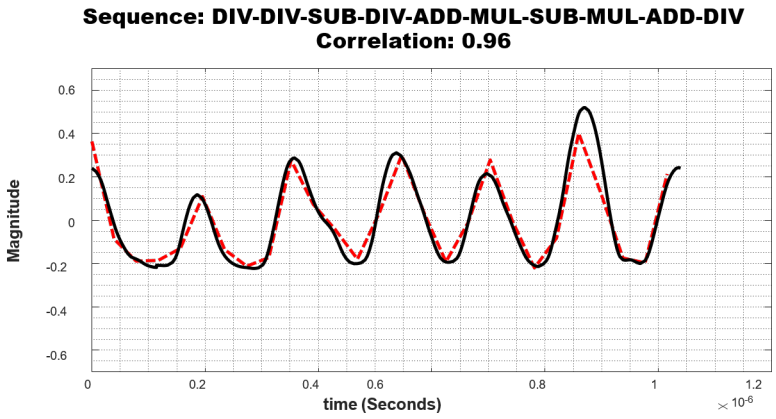


Figure G.10: Resampled *modulo operation* result (solid curve) vs. single execution of the same instruction sequence (dashed curve).

Table G.1: Correlation between the EM signatures and their one-time-run versions for Altera DE1 Cyclone II. The columns denote the EM signatures and the rows denote the one-time-run versions. The diagonal entries dominate the other terms, therefore, the generated EM signatures can identify the executed sequences and corresponding instructions (The values given in the table is correlation coefficient $\times$ 100).

		Generated EM signatures																			
		1A	**2A**	**3A**	**4A**	**5A**	**6A**	**7A**	**8A**	**9A**	**10A**	**11A**	**12A**	**13A**	**14A**	**15A**	**16A**	**17A**	**18A**	**19A**	**20A**
One-time-run versions	**1B**	**91**	54	73	15	23	11	21	19	28	43	21	23	7	54	18	37	27	24	14	2
	2B	12	**95**	50	16	57	43	50	21	39	56	58	6	65	58	50	57	10	46	9	18
	3B	61	67	**93**	6	55	42	50	2	18	62	66	28	52	68	24	55	8	46	5	16
	4B	19	6	20	**94**	46	61	42	73	62	10	29	73	50	14	44	4	67	27	78	66
	5B	9	53	18	52	**97**	79	80	75	72	30	49	39	55	32	53	50	55	62	61	67
	6B	11	31	2	66	80	**91**	87	85	75	16	50	67	61	16	59	43	70	72	68	79
	7B	2	43	20	32	70	80	**92**	64	73	24	54	48	50	38	61	55	55	72	54	76
	8B	27	8	34	41	30	47	38	**97**	66	19	26	83	42	21	56	2	76	24	77	66
	9B	41	3	31	57	44	62	57	80	**98**	19	37	82	62	19	76	0	90	34	68	69
	10B	22	57	53	20	55	40	44	11	20	**96**	88	22	70	76	33	83	3	47	4	11
	11B	20	49	33	44	57	53	50	50	62	65	**93**	26	82	58	58	70	46	51	47	47
	12B	12	41	48	42	10	18	7	80	52	47	7	**94**	10	50	47	33	68	14	75	66
	13B	20	41	20	59	61	60	57	69	80	36	80	45	**98**	40	62	39	75	58	61	57
	14B	33	65	72	11	65	44	46	8	31	76	83	30	81	**95**	29	62	7	49	7	10
	15B	39	20	25	34	33	46	47	64	80	3	45	66	54	4	**96**	17	70	25	56	55
	16B	13	53	41	26	53	57	66	25	36	82	78	1	60	60	48	**96**	12	66	17	36
	17B	32	25	41	57	28	51	36	78	85	38	15	82	42	32	64	21	**95**	20	74	56
	18B	12	42	27	49	61	81	91	49	58	38	48	32	54	38	45	63	45	**97**	31	53
	19B	23	11	20	46	40	42	25	77	59	23	19	66	32	16	51	9	70	4	**92**	70
	20B	8	6	17	40	46	55	59	72	66	13	23	71	34	8	57	14	58	25	77	**94**

but none of the cross correlation coefficients are above 0.9. Therefore, the corresponding threshold for these instruction sequences can be set as 0.9.

As mentioned earlier, in order to reach single instruction level tracking, we should include all possible instruction sequences. In this chapter, we have included only 20 instruction sequences as a preliminary result in order to demonstrate the applicability of our proposed *modulo operation* technique.

G.4.2 Experimental Results on Target Device 2 (ARM Board)

In order to demonstrate that *modulo operation* technique can be applied to devices with very high clock frequencies, we include another target device that has higher clock frequency. The second target device that is used in our experiments is A13-OLinuXino board, a single-board computer with Cortex A8 ARM processor. The operating frequency of the device is 1 GHz and the pipeline architecture is very complex with 13 pipeline stages.

For this device, we consider 7 instruction sequences that are listed in Appendix G.7. Similar to Section G.4.1, the instructions include the basic arithmetic operations such as `ADD`,`SUB`,`MUL`, as well as load `(LDR)` and store `(STR)` instructions. The instructions that considered are for this device are given in Table 2.1.

We measure the emanated EM signals with the same near field magnetic probe (AAronia PBS H3) that is located above the processor of the target device. For recording the measured data, we use a different spectrum analyzer (Keysight N9030B PXA), which enables us to sample the signal with 400 MHz span. Note that this is still not providing enough samples per clock cycle, because the operating clock frequency is much higher, i.e. 1 GHz.

Here, we would like to point out a modification in the **Testing Setup**. When we use the **Testing Setup** that is explained in Section G.2, we realize

that the the training signals do not resemble the testing signals due to the different pipeline effects that training and testing signals go through. One should note that, this is not the case in Section G.4.1, because the pipeline architecture of FPGA is not as complex as this board. In order to circumvent this problem, we modify the **Testing Setup** by repeating the instruction sequence within the for loop, but only crop one period of it. This enables us to make sure that the testing signal has the same pipeline effect as the training signal.

Figure G.11 displays the signal recorded by using **Training Setup** before *modulo operation* for a specific instruction sequence (Sequence 1 in Table G.4). This signal is used as the input to the *modulo operation*. .

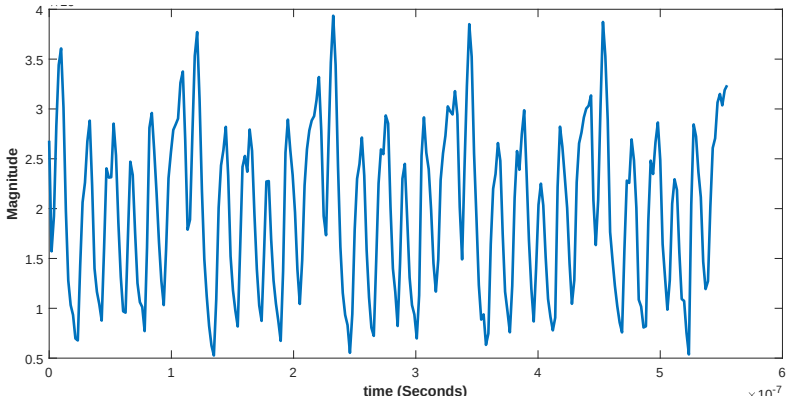


Figure G.11: Recorded **Training Setup** of Sequence 1 from Table G.4.

The top and bottom plots in Figure G.12 represent the *modulo operation* result and the single execution of the same instruction sequence, respectively. Similar to previous section, when we correlate the *modulo operation* result for this instruction sequence with other testing signals obtained from different instruction sequences, the correlation is much less than 0.9. Figure G.13 illustrates one example of this.

Table G.2 illustrates the correlation matrix of training and testing sig-

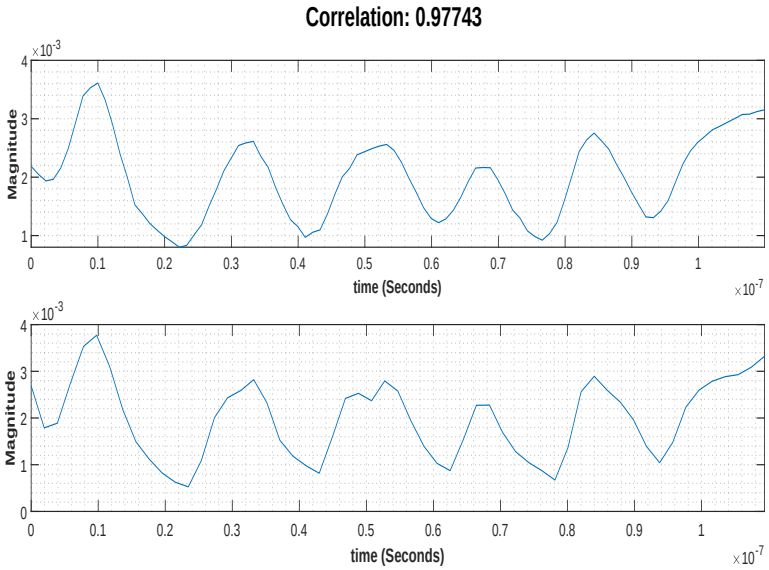


Figure G.12: The plot on the top presents the result of the *modulo operation*, plot on the bottom presents the single execution of the same instruction sequence obtained by modified **Testing Setup**, where the signal is preceded and followed by the same instruction sequence. Instruction Sequence used for this figure is Sequence 1 from Table G.4.

nals obtained from experiments on A13-OLinuXino board. It is not very difficult to realize that the diagonal terms are dominant, and the other terms are much less than 0.8. This indicates that we can detect the instruction sequences that are used in this experiment by using the training signals obtained by *modulo operation*. Therefore, we can conclude that *modulo operation* is applicable for devices with high clock frequencies as much as 1 GHz.

G.5 Summary

Earlier work have illustrated the usage of EM emanations for instruction tracking on processors with simple pipeline architectures and low operating frequencies. In this chapter, we show how EM emanations can be used to track instructions on a more advanced device (an FPGA) with a proces-

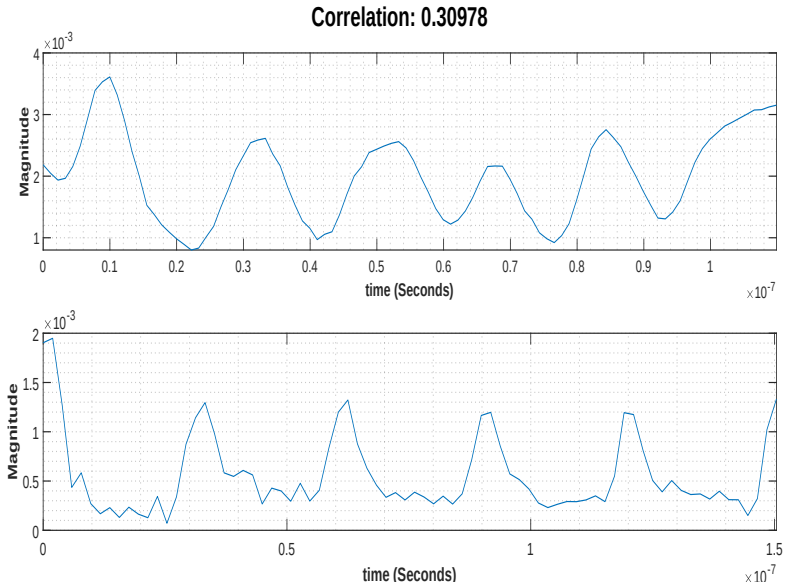


Figure G.13: The plot on the top presents the result of the *modulo operation* obtained by using modified **Training Setup** (concatenating different instances of **Testing Setup**) for Sequence 1 from Table G.4, plot on the bottom presents the single execution of the a different instruction sequence (Sequence 3 from Table G.4) obtained by **Testing Setup**. The correlation of those signals are very low as expected.

sor that has a more complex pipeline structure and higher operating clock frequency.

In order to track the instructions, we generate reference signals (called as EM signatures) to be compared with the emanated signals. The complex structure of the target device has motivated us to develop a new technique and modified experiment setups to generate EM signatures. Firstly, in order to incorporate the pipeline effect, EM signatures are generated for sequences of instructions rather than single instructions. Furthermore, we propose and implement a technique called *modulo operation*, which addresses the issues introduced by low sampling rate and the pipeline effect. Results show that *modulo operation* is successful not only to obtain an EM signature with higher sampling rate, but also to eliminate possible

Table G.2: Correlation between the EM signatures and their one-time-run versions for A13-OLinuXino board. The columns denote the EM signatures and the rows denote the one-time-run versions. The diagonal entries dominate the other terms, therefore, the generated EM signatures can identify the executed sequences and corresponding instructions (The values given in the table is correlation coefficient $\times$ 100).

		Generated EM signatures						
		1A	**2A**	**3A**	**4A**	**5A**	**6A**	**7A**
One-time-run versions	**1B**	**98**	43	48	29	60	35	40
	2B	34	**98**	72	47	42	59	44
	3B	37	72	**98**	33	37	62	46
	4B	31	55	32	**95**	30	62	25
	5B	58	41	47	31	**96**	43	71
	6B	37	61	59	67	47	**89**	51
	7B	45	66	66	32	59	53	**92**

measurement noise. Results demonstrate that a specific EM signature is highly correlated with the single execution of the same sequence and much less correlated with all other sequences. This confirms the applicability of EM signals that are generated by using *modulo operation*.

As much as the EM signatures for instruction sequences used in our experiment can be used to detect the corresponding sequences, achieving instruction tracking at instruction level necessitates generating EM signatures for all possible sequences. To this end, our experimental results show that using *modulo operation* for generating EM signatures for all possible sequences is a feasible approach to obtain high success rate.

G.6 Sequences Used for Target Device 1

In this section, we provide the instruction sequences investigated in Table G.1 (instruction sequences used on FPGA):

Table G.3: Instruction sequences that are used in the FPGA experiments

Seq. No.	Sequence
1	SUB-DIV-STM-DIV-STM-MUL-LDM-MUL-SUB-MUL-ADD-DIV-LDM-DIV-ADD-MUL
2	LDM-MUL-ADD-MUL-LDM-DIV-SUB-MUL-ADD-DIV-SUB-DIV-STM-DIV-STM-MUL
3	STM-MUL-STM-DIV-SUB-MUL-LDM-MUL-SUB-DIV-ADD-MUL-LDM-DIV-ADD-DIV
4	MUL-ADD-ADD-SUB-DIV-SUB-DIV-ADD-MUL-SUB
5	ADD-SUB-ADD-SUB-DIV-SUB-DIV-ADD-MUL-MUL
6	ADD-ADD-MUL-SUB-ADD-DIV-DIV-MUL-SUB-SUB
7	SUB-DIV-SUB-DIV-MUL-ADD-MUL-SUB-ADD-ADD
8	SUB-DIV-ADD-DIV-ADD-MUL-SUB-MUL-ADD-SUB
9	ADD-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD-DIV-SUB
10	DIV-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD-DIV-SUB
11	MUL-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD-DIV-SUB
12	ADD-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD
13	DIV-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD
14	MUL-DIV-SUB-DIV-ADD-MUL-SUB-MUL-ADD
15	DIV-SUB-DIV-MUL-SUB-MUL-ADD-DIV-SUB-MUL-ADD-DIV-SUB-SUB
16	DIV-SUB-DIV-MUL-ADD-DIV-SUB-MUL-ADD-DIV-SUB-SUB
17	DIV-SUB-DIV-DIV-SUB-MUL-ADD-DIV-SUB-SUB
18	DIV-SUB-DIV-MUL-ADD-DIV-SUB-SUB
19	DIV-SUB-DIV-DIV-SUB-SUB
20	DIV-SUB-DIV-SUB

G.7 Sequences Used for Target Device 2

In this section, we provide the instruction sequences investigated in Table G.2 (instruction sequences used on ARM Board):

Table G.4: Instruction sequences that are used in the ARM Board experiments

Seq. No.	Sequence
1	SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR-SUB-STR-LDR-ADD-STR
2	ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-MUL-MUL-ADD-AND
3	SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-MUL-MUL-SUB-AND
4	SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-SUB-SUB-MUL-MUL-SUB-AND
5	SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND-SUB-ADD-SUB-MUL-AND
6	SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-MUL-MUL-STR-LDR-AND-SUB-ADD
7	ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND-ADD-MUL-ADD-MUL-STR-LDR-ADD-AND

APPENDIX H

PSD OF PAM SIGNAL WITH RANDOM PULSE POSITION

The PAM signal with random pulse position $y(t)$ is given by $y(t) = \sum_k x_k \delta(t - k\mathcal{T} - \mathbf{T}_k)$. $y(t)$ is an impulse train whose amplitude is modulated by the sequence x_k and the impulse positions are randomly shifted by $\mathbf{T}_k$. Furthermore, the autocorrelation function and the power spectral density of x_k are denoted as $R_x[k]$ and $S_x(f) = \sum_k R_x[k] e^{-j2\pi f k \mathcal{T}}$, respectively. Here, we note that the signals $x_p(t)$ and $y(t)$ in (5.1) and (5.5) are cyclostationary random processes if the amplitude modulating sequence x_k and the random pulse position variation $\mathbf{T}_k$ are stationary [121]. For a cyclostationary random processes of period $\mathcal{T}$, the average autocorrelation function between 0 and $\mathcal{T}$ can be computed as [122]

$$R_y(\tau) = \frac{1}{\mathcal{T}} \int_0^{\mathcal{T}} R_y(t, \tau) dt, \tag{H.1}$$

where $R_y(t, \tau) = \mathrm{E}\Big[y(t), y(t - \tau)\Big]$ and $\mathrm{E}[\cdot]$ denotes the expectation. Here, $R_y(t, \tau)$ can be written as

$$\mathrm{E}\Big[\sum_i \sum_j x_i x_j \delta(t - i\mathcal{T} - \mathbf{T}_i)\delta(t - \tau - j\mathcal{T} - \mathbf{T}_j)\Big]. \tag{H.2}$$

It can be shown that $R_y(t, \tau)$ is also periodic in time with a period $\mathcal{T}$. Therefore, $y(t)$ is a cyclostationary random process. Using (H.1), we can

rewrite the correlation function $R_y(\tau)$ as

$$\begin{aligned} &= \frac{\int_0^{\mathcal{T}} \mathrm{E}\Big[\sum_{i,j} x_i x_j \delta(t - i\mathcal{T} - \mathbf{T}_i)\delta(t - \tau - j\mathcal{T} - \mathbf{T}_j)\Big] dt}{\mathcal{T}} \\ &= \frac{\sum_{i,j} \int_0^{\mathcal{T}} \mathrm{E}\Big[x_i x_j \delta(t - i\mathcal{T} - \mathbf{T}_i)\delta(t - \tau - j\mathcal{T} - \mathbf{T}_j)\Big] dt}{\mathcal{T}} \\ &= \frac{\sum_{i,j} \int_0^{\mathcal{T}} \mathrm{E}\big[x_i x_j\big]\mathrm{E}\Big[\delta(t - i\mathcal{T} - \mathbf{T}_i)\delta(t - \tau - j\mathcal{T} - \mathbf{T}_j)\Big] dt}{\mathcal{T}} \end{aligned} \tag{H.3}$$

where (H.3) follows the assumption that x_k and $\mathbf{T}_k$ are independent. Let $\lambda = t - i\mathcal{T}$. So, (H.3) can be written as

$$\frac{\sum_{i,j} \int_{-i\mathcal{T}}^{-(i-1)\mathcal{T}} \mathrm{E}\big[x_i x_j\big]\mathrm{E}\Big[\delta(\lambda - \mathbf{T}_i)\delta(\lambda - \tau - (j-i)\mathcal{T} - \mathbf{T}_j)\Big] d\lambda}{\mathcal{T}}.$$

Letting $j - i = m$, we can rewrite the correlation function as follows:

$$\begin{aligned} R_y(\tau) = \frac{1}{\mathcal{T}} \sum_m \sum_i \int_{-i\mathcal{T}}^{-(i-1)\mathcal{T}} \Bigg(&\mathrm{E}\big[x_i x_{i+m}\big] \times \\ &\mathrm{E}\Big[\delta(\lambda - \mathbf{T}_i)\delta(\lambda - \tau - m\mathcal{T} - \mathbf{T}_{m+i})\Big]\Bigg) d\lambda. \end{aligned} \tag{H.4}$$

Since x_k is a stationary sequence, we can deduce $\mathrm{E}[x_i x_j] = R_x[i-j]$. Exploiting that $\{\mathbf{T}_k,\ \forall k \in (-\infty, \infty)\}$ are statistically identical and independent

of each other, we can rewrite (H.4) as

$$\begin{aligned}&\frac{1}{\mathcal{T}}\sum_{m,i}\int_{-i\mathcal{T}}^{-(i-1)\mathcal{T}} R_x[m]\mathrm{E}\Big[\delta(\lambda-\mathbf{T}_0)\delta(\lambda-\tau-m\mathcal{T}-\mathbf{T}_m)\Big]d\lambda\\ =&\frac{\sum_m R_x[m]\int_{-\infty}^{\infty}\mathrm{E}\Big[\delta(\lambda-\mathbf{T}_0)\delta(\lambda-\tau-m\mathcal{T}-\mathbf{T}_m)\Big]d\lambda}{\mathcal{T}}.\end{aligned}$$

Taking the integration inside the expectation operator, $R_y(\tau)$ simplifies to

$$\begin{aligned}&\frac{1}{\mathcal{T}}\sum_m R_x[m]\mathrm{E}\Big[\int_{-\infty}^{\infty}\delta(\lambda-\mathbf{T}_0)\delta(\lambda-\tau-m\mathcal{T}-\mathbf{T}_m)d\lambda\Big]\\ =&\frac{1}{\mathcal{T}}\sum_m R_x[m]\mathrm{E}\Big[\delta(-\tau-m\mathcal{T}+\mathbf{T}_0-\mathbf{T}_m)\Big].\end{aligned}\tag{H.5}$$

Considering that the pulse positions $\mathbf{T}_k$ are independent and identically distributed (i.i.d.), the autocorrelation function $R_y(\tau)$ can be calculated as

$$\begin{aligned}&\frac{\mathrm{E}\Big[R_x[0]\delta(\tau)\Big]+\sum_{m\neq 0}R_x(m)\mathrm{E}\Big[\delta(-\tau-m\mathcal{T}+\mathbf{T}_0-\mathbf{T}_m)\Big]}{\mathcal{T}}\\ =&\frac{R_x(0)\delta(\tau)+\sum_{m\neq 0}R_x(m)\mathrm{E}\big[\delta(-\tau-m\mathcal{T}+\mathbf{T}_0-\mathbf{T}_m)\big]}{\mathcal{T}}.\end{aligned}\tag{H.6}$$

To proceed further, let us introduce $z_m(\tau)=\mathrm{E}\big[\delta(-\tau-m\mathcal{T}+\mathbf{T}_0-\mathbf{T}_m)\big]$.

Therefore,

$$
\begin{aligned}
z_m(\tau) =& \int_{-\mathcal{T}/4+\mu}^{\mathcal{T}/4+\mu}\int \delta(-\tau - m\mathcal{T} + t_0 - t_m) f_{\mathbf{T}_0}(t_0) f_{\mathbf{T}_m}(t_m) dt_0 dt_m \\
&= \int_{-\mathcal{T}/4+\mu}^{\mathcal{T}/4+\mu} f_{\mathbf{T}_0}(\tau + m\mathcal{T} + t_m) f_{\mathbf{T}_m}(t_m) dt_m \\
&\overset{(a)}{=} \int_{-\mathcal{T}/4+\mu}^{\mathcal{T}/4+\mu} f_{\mathbf{T}}(\tau + m\mathcal{T} + t_m) f_{\mathbf{T}}(t_m) dt_m \\
&\overset{(b)}{\approx} \int_{-\infty}^{\infty} f_{\mathbf{T}}(\tau + m\mathcal{T} + t_m) f_{\mathbf{T}}(t_m) dt_m \\
&= f_{\mathbf{T}}(-\tau + m\mathcal{T}) * f_{\mathbf{T}}(\tau) \\
&= \delta(\tau - m\mathcal{T}) * f_{\mathbf{T}}(-\tau) * f_{\mathbf{T}}(\tau) \\
&= \delta(\tau - m\mathcal{T}) * \phi(\tau)
\end{aligned} \tag{H.7}
$$

where (a) follows all distributions $\{\mathbf{T}_i | \forall i \in \{-\infty, \infty\}\}$ are i.i.d., (b) is due to support set assumption of distribution functions and $*$ denotes convolution. Plugging (H.7) into (H.6), we can write $R_y(\tau)$ as

$$
\begin{aligned}
&\frac{R_x(0)\delta(\tau) + \sum_{m=0} R_x(m) \left(\delta(\tau - m\mathcal{T}) * \phi(\tau)\right) - R_x(0)\phi(\tau)}{\mathcal{T}} \\
&= \\
&\frac{\left(\sum_m R_x(m)\delta(\tau - m\mathcal{T})\right) * \phi(\tau) + R_x(0)\Big(\delta(\tau) - \phi(\tau)\Big)}{\mathcal{T}}.
\end{aligned} \tag{H.8}
$$

The PSD $S_y(f)$ of the signal $y_p(t)$ is obtained by taking the Fourier transform of the above result. Using these results, we can write the spectrum of PAM signal with random pulse position as

$$
S_y(f) = \frac{1}{\mathcal{T}} S_x(f)\Phi(f) + \frac{R_x(0)}{\mathcal{T}}(1 - \Phi(f)), \tag{H.9}
$$

where $\Phi(f)$ is the Fourier transform of $\phi(\tau)$.

H.1 PSD of "on-off" Keying (OOK) With Random Pulse Position

The power spectrum of the PAM with random pulse position has already been derived in (5.11). In this section, we specify the equation in (5.11) for OOK modulation case. As the first step, we need to calculate $S_x(f)$ to investigate the effect of random pulse position on the spectral power of the signal. We assume the amplitude of a symbol is $\mathcal{A}$ when the symbol is "on" and 0 otherwise. Therefore, autocorrelation of these symbols can be written as

$$R_x[m] = \begin{cases} \mathcal{A}^2/2 & \text{if } m = 0, \\ \mathcal{A}^2/4 & \text{otherwise.} \end{cases} = \begin{cases} R_x[0] & \text{if } m = 0, \\ R_x[0]/2 & \text{otherwise.} \end{cases}$$

If we convert this discrete signal into continuous signal with period $\mathcal{T}$, we have

$$R_x(\tau) = \sum_m R_x[m]\delta(\tau - m\mathcal{T}). \tag{H.10}$$

To obtain the power spectral density of the signal, Fourier transform of $R_x(\tau)$ can be calculated as follows:

$$\begin{aligned} S_x(f) &= \int_{-\infty}^{\infty} \sum_m R_x[m]\delta(\tau - m\mathcal{T})e^{-j2\pi f\tau}d\tau \\ &= \sum_m R_x[m]e^{-j2\pi fm\mathcal{T}} \\ &= \frac{R_x[0]}{2} + \frac{R_x[0]}{2}\sum_m e^{-j2\pi fm\mathcal{T}} \\ &= \frac{R_x[0]}{2}\left(1 + \frac{1}{\mathcal{T}}\sum_m \delta(f - m/\mathcal{T})\right) \end{aligned} \tag{H.11}$$

If we insert (H.11) into (5.5), the power spectrum $S_y(f)$ can be written

as

$$\frac{R_x[0]}{2\mathcal{T}}\left(1+\frac{\sum\limits_m \delta(f-n/\mathcal{T})}{\mathcal{T}}\right)\Phi(f)+\frac{R_x(0)}{\mathcal{T}}(1-\Phi(f))$$

$$=\frac{R_x[0]}{\mathcal{T}}\left(\underbrace{\left(\frac{1}{2}+\frac{\sum\limits_m \delta(f-m/\mathcal{T})}{2\mathcal{T}}\right)\Phi(f)}_{\bar{S}_{xt}(f)}+\underbrace{(1-\Phi(f)}_{\bar{S}_{nt}(f)}\right). \tag{H.12}$$

Here, we need to note that since we assume that the random shift position is in the interval $\left(-\frac{\mathcal{T}}{4},\frac{\mathcal{T}}{4}\right]$ that has a Gaussian distribution, we consider $\mathcal{T}\gtrsim 12\sigma$ to ensure our interval assumption holds with very high probability.

APPENDIX I

COVERT CHANNEL CAPACITY DERIVATIONS

In this section, we provide the derivations for channel capacity bounds of the covert channel communications. The capacity of a discrete memoryless synchronization channel is given in (2.5). Here, $\bar{N}$ is the average number of received symbols per transmitted symbol. The number of insertions between consecutive input symbols are geometrically distributed and the average number of insertions per input symbol is

$$\begin{aligned}(p_{i0}+p_{i1})(1-p_{i0}-p_{i1}) + 2(p_{i0}+p_{i1})^2(1-p_{i0}-p_{i1}) \\ +3(p_{i0}+p_{i1})^3(1-p_{i0}-p_{i1}) + \dots \\ = \quad \frac{p_{i0}+p_{i1}}{1-p_{i0}-p_{i1}}.\end{aligned} \tag{I.1}$$

Hence, the average number of output symbols per input symbols is

$$\bar{N} = \frac{1}{1-p_{i0}-p_{i1}}. \tag{I.2}$$

In [16], [15], it is shown that channels with insertions and substitutions can be decomposed into a cascade of two channels, channel with insertions and channel with substitutions as shown in Fig. 2.5. Since both inputs and outputs of the covert channel are assumed to be equiprobable, it follows that

$$H(W^n) = n, \text{ and } H\left(X^{n/(1-p_i)}\right) = \frac{n}{1-p_i}, \tag{I.3}$$

where n is the number of input bits, $p_i = p_{i0} + p_{i1}$, and $H(\cdot)$ denotes the

entropy. From (2.5), it follows that we need to calculate mutual information $I\left(W^n, Y^{\frac{n}{1-p_i}}\right)$ between the input sequence and output of the second cascaded channel. This mutual information can be written as

$$I\left(W^n, Y^{\frac{n}{1-p_i}}\right) = I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}}\right) - I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}} | W^n\right). \tag{I.4}$$

To find a lower bound for $I\left(W^n, Y^{\frac{n}{1-p_i}}\right)$, we are required to obtain an upper bound for $I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}} | W^n\right)$. Therefore,

$$\begin{aligned} 0 &\leq I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}} | W^n, \right) \\ &= H\left(X^{\frac{n}{1-p_i}} | W^n\right) - H\left(X^{\frac{n}{1-p_i}} | W^n, Y^{\frac{n}{1-p_i}}\right) \\ &= H\left(X^{\frac{n}{1-p_i}}\right) - I\left(W^n, X^{\frac{n}{1-p_i}}\right) - H\left(X^{\frac{n}{1-p_i}} | W^n, Y^{\frac{n}{1-p_i}}\right) \\ &\leq H\left(X^{\frac{n}{1-p_i}}\right) - I\left(W^n, X^{\frac{n}{1-p_i}}\right). \end{aligned} \tag{I.5}$$

Combining (2.5), (I.4) and (I.5), C can be written as

$$\begin{aligned} &\sup_{\Xi} \lim_{n\to\infty} \frac{1}{n} \cdot I(W^n; Y^{\bar{N}n}) \\ =&\sup_{\Xi} \lim_{n\to\infty} \frac{1}{n}\left(I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}}\right) - I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}} | W^n\right) \right) \\ \geq&\sup_{\Xi} \lim_{n\to\infty} \frac{1}{n}\left(I\left(X^{\frac{n}{1-p_i}}, Y^{\frac{n}{1-p_i}}\right) - H\left(X^{\frac{n}{1-p_i}}\right) + I\left(W^n, X^{\frac{n}{1-p_i}}\right) \right) \qquad \text{(I.6)} \\ =&\sup_{\Xi} \lim_{n\to\infty} \frac{1}{n}\left(n\frac{I(X,Y)}{1-p_i} - \frac{n}{1-p_i} + I\left(X^{\frac{n}{1-p_i}}, W^n\right)\right) \\ =&\frac{I(X,Y)}{1-p_i} - \frac{1}{1-p_i} + \sup_{\Xi} \lim_{n\to\infty} \frac{1}{n}\left(I\left(X^{\frac{n}{1-p_i}}, W^n\right)\right) \\ =&\frac{I(X,Y)}{1-p_i} - \frac{1}{1-p_i} + C^i(p_i) \qquad \text{(I.7)} \end{aligned}$$

where $C^i(p_i)$ is the channel capacity of insertion channel with insertion probability p_i and (I.6) follows the assumption that the noisy substitu-

tion channel is a discrete memoryless channel (DMC), and the output are statistically independent and identical because of the random insertions [123]. To obtain a lower bound for the insertion channel, we exploit the relation between deletion and insertion channels, and previous results for the capacity lower bound of deletion channels. In [16], the relation between deletion and insertion channels is given as

$$C^d(pi) = (1 - p_i)C^i(p_i) \tag{I.8}$$

where $C^d(p_i)$ is the information rate of a deletion channel with equiprobable iid inputs whose deletion probability equals to insertion probability of the insertion channel. Morover, in [124], the capacity lower bound for the deletion channel is given as

$$C^d(p_i) \geq 1 - H_b(p_i) \tag{I.9}$$

where $C^d(p_i)$ represents the actual channel capacity of the deletion channel with deletion probability p_i and $H_b(\bullet)$ denotes the binary entropy.

The equation given in (I.9) is valid for any deletion channel. Therefore,

$$\begin{aligned} & 1 - H_b(p_i) \;\leq\; C^d(p_i) \;=\; (1 - p_i)C^i(p_i) \\ \Rightarrow \quad & C^i(p_i) \geq \frac{1 - H_b(p_i)}{1 - p_i}. \end{aligned} \tag{I.10}$$

If we combine (I.7) and (I.10), we have

$$\begin{aligned} C &\geq \frac{I(X,Y)}{1-p_i} - \frac{1}{1-p_i} + C^i(p_i) \\ &\geq \frac{I(X,Y)}{1-p_i} - \frac{1}{1-p_i} + \frac{1-H_b(p_i)}{1-p_i} \\ &= \frac{1-H_b(p_i)-H_b(p_e)}{1-p_i} \end{aligned} \tag{I.11}$$

where the last equation follows the assumption that the noisy channel is binary symmetric channel with substitution probability p_e. By definition, mutual information could not be less than zero, therefore, the lower bound can be written as

$$C \geq \max\left(0, \frac{1-H_b(p_i)-H_b(p_e)}{1-p_i}\right). \tag{I.12}$$

To prove the upper bound for the covert-channel capacity, we consider a channel where the receiver is provided with the positions of all insertions caused by the covert channel and the sequence $Z^n = \{z_0, z_1 \cdots, z_n\}$ with

$$z_k = \begin{cases} 0 & \text{if the } k^{th} \text{ bit is inserted bit,} \\ 1 & \text{otherwise} \end{cases} \tag{I.13}$$

which provides further information whether a bit is an information bit or an inserted bit. Therefore,

$$\begin{aligned} I(W^n; Y^{\bar{N}n}) &\leq I(W^n; Y^{\bar{N}n}) + I(W^n; Z^{\bar{N}n} | Y^{\bar{N}n}) \\ &= I(W^n; Y^{\bar{N}n}, Z^{\bar{N}n}) \\ &= I(W^n; \hat{Y}^n) \qquad \text{(I.14)} \\ &= n(1-H_b(p_e)) \qquad \text{(I.15)} \end{aligned}$$

where $\hat{Y}$ is the sequence obtained by removing the inserted bits. The equation given in (I.14) can be explained as follows: Knowing where the synchronization errors are located, the receiver can discard the inserted symbols. The capacity of this channel, therefore, is as large as the capacity of the channel with no synchronization errors. Finally, combining again (2.5) and (I.15), we can obtain the upper bound as

$$\begin{aligned} C &= \sup_{\Xi} \lim_{n\to\infty} \frac{1}{n} \cdot I(W^n; Y^{\bar{N}n}) \\ &\leq \sup_{\Xi} \lim_{n\to\infty} \frac{1}{n} \cdot I(W^n; \hat{Y}^n) \\ &= 1 - H_b(p_e) \end{aligned} \tag{I.16}$$

which concludes the proof.

REFERENCES

[1] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973.

[2] J. Millen, "20 years of covert channel modeling and analysis," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, IEEE, 1999, pp. 113–114.

[3] A. Zajic and M. Prvulovic, "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 56, no. 4, pp. 885–893, 2014.

[4] M. Guri, A. Kachlon, O. Hasson, G. Kedma, Y. Mirsky, and Y. Elovici, "Gsmem: Data exfiltration from air-gapped computers over GSM frequencies," in *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, 2015, pp. 849–864, ISBN: 978-1-931971-232.

[5] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic, "A new side-channel vulnerability on modern computers by exploiting electromagnetic emanations from the power management unit," *IEEE International Symposium on High-Performance Computer Architecture (HPCA-26)*, 2020.

[6] B. A. Bash, D. Goeckel, and D. Towsley, "Limits of reliable communication with low probability of detection on awgn channels," *IEEE journal on selected areas in communications*, vol. 31, no. 9, pp. 1921–1930, 2013.

[7] M. R. Bloch, "Covert communication over noisy channels: A resolvability perspective," *IEEE Transactions on Information Theory*, vol. 62, no. 5, pp. 2334–2354, 2016.

[8] L. Wang, G. W. Wornell, and L. Zheng, "Limits of low-probability-of-detection communication over a discrete memoryless channel," in *2015 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2015, pp. 2525–2529.

[9] J. K. Millen, "Covert channel capacity," in *1987 IEEE Symposium on Security and Privacy*, IEEE, 1987, pp. 60–60.

[10] Z. Wang and R. Lee, "Capacity estimation of non-synchronous covert channels," in *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, 2005, pp. 170–176.

[11] M. C. Davey and D. J. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, 2001.

[12] R. Venkataramanan, S. Tatikonda, and K. Ramchandran, "Achievable rates for channels with deletions and insertions," *IEEE Transactions on Information Theory*, vol. 59, no. 11, pp. 6990–7013, 2013.

[13] A. Kirsch and E. Drinea, "Directly lower bounding the information capacity for channels with iid deletions and duplications," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 86–102, 2010.

[14] J. Hu, T. M. Duman, M. F. Erden, and A. Kavcic, "Achievable information rates for channels with insertions, deletions, and intersymbol interference with iid inputs," *IEEE Transactions on Communications*, vol. 58, no. 4, 2010.

[15] M. Rahmati and T. M. Duman, "Bounds on the capacity of random insertion and deletion-additive noise channels," *IEEE Transactions on Information Theory*, vol. 59, no. 9, pp. 5534–5546, 2013.

[16] H. Mercier, V. Tarokh, and F. Labeau, "Bounds on the capacity of discrete memoryless channels corrupted by synchronization and substitution errors," *IEEE Transactions on Information Theory*, vol. 58, no. 7, pp. 4306–4330, 2012.

[17] B. B. Yilmaz, R. Callan, M. Prvulovic, and A. Zajic, "Quantifying information leakage in a processor caused by the execution of instructions," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, IEEE, 2017, pp. 255–260.

[18] B. B. Yilmaz, R. Callan, A. Zajic, and M. Prvulovic, "Capacity of the em covert/side-channel created by the execution of instructions in a processor," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 605–620, 2018.

[19] B. B. Yilmaz, M. Prvulovic, and A. Zajic, "Electromagnetic side channel information leakage created by execution of series of instructions in a computer processor," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 776–789, 2019.

[20] B. B. Yilmaz, M. Prvulovic, and A. Zajic, "Capacity of deliberate side-channels created by software activities," in *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, 2018, pp. 237–242.

[21] B. B. Yilmaz, N. Sehatbakhsh, A. Zajić, and M. Prvulovic, "Communication model and capacity limits of covert channels created by software activities," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 605–620, 2018.

[22] B. B. Yilmaz, N. Sehatbakhsh, M. Dey, C. L. Cheng, M. Prvulovic, and A. Zajic, "Covert channel information leakage capacity: A generalized approach," *Submitted to IEEE Transactions on Information Forensics and Security*, vol. 1, pp. 1–13, 2020.

[23] N. Sehatbakhsh, B. B. Yilmaz, A. Zajic, and M. Prvulovic, " EMSim: A microarchitecture-level simulation tool for modeling electromagnetic side-channel signals," *IEEE International Symposium on High-Performance Computer Architecture (HPCA-26)*, 2020.

[24] H. J. Highland, "Electromagnetic radiation revisited," *Computers & Security*, vol. 5, no. 2, pp. 85–93, 1986.

[25] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2002, pp. 29–45.

[26] W. van Eck, "Electromagnetic radiation from video display units: An eavesdropping risk?" *Computers and Security*, vol. 4, no. 4, pp. 269 –286, 1985.

[27] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*, Springer, 1996, pp. 104–113.

[28] W Schindler, "A timing attack against RSA with Chinese remainder theorem," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 2000*, 2000, pp. 109–124.

[29] D. Boneh and D. Brumley, "Remote Timing Attacks are Practical," in *Proceedings of the USENIX Security Symposium*, 2003.

[30] P Kocher, J Jaffe, and B Jun, "Differential power analysis: leaking secrets," in *Proceedings of CRYPTO'99, Springer, Lecture notes in computer science*, 1999, pp. 388–397.

[31] L Goubin and J Patarin, "DES and Differential power analysis (the "duplication" method)," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 1999*, 1999, pp. 158–172.

[32] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smart cards," in *Proceedings of Cryptographic Hardware and Embedded Systems - CHES 1999*, 1999, pp. 144–157.

[33] S Chari, C. S. Jutla, J. R. Rao, and P Rohatgi, "Towards sound countermeasures to counteract power-analysis attacks," in *Proceedings of CRYPTO'99, Springer, Lecture Notes in computer science*, 1999, pp. 398–412.

[34] A. G. Bayrak, F Regazzoni, P Brisk, F.-X. Standaert, and P Ienne, "A first step towards automatic application of power analysis countermeasures," in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011.

[35] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 35, 2007, pp. 494–505.

[36] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi, "Cryptanalysis of block ciphers implemented on computers with cache," Jan. 2002.

[37] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.

[38] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 605–622.

[39] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, 2009, pp. 199–212.

[40] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 168–179.

[41] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *Security and Privacy (SP), 2011 IEEE Symposium on*, IEEE, 2011, pp. 490–505.

[42] O. Aciiçmez, c. K. Koç, and J.-P. Seifert, "On the power of simple branch prediction analysis," in *Proceedings of the 2Nd ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '07, Singapore: ACM, 2007, pp. 312–320, ISBN: 1-59593-574-6.

[43] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, Williamsburg, VA, USA: ACM, 2018, pp. 693–707, ISBN: 978-1-4503-4911-6.

[44] S. K. Khatamifard, L. Wang, S. Köse, and U. R. Karpuzcu, "A new class of covert channels exploiting power management vulnerabilities," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 201–204, 2018.

[45] D. Evtyushkin and D. Ponomarev, "Covert channels through random number generator: Mechanisms, capacity estimation and mitigations," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, ACM, 2016, pp. 843–857.

[46] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{drama}: Exploiting {dram} addressing for cross-cpu attacks," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 565–581.

[47] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, Bellevue, WA: USENIX, 2012, pp. 159–173, ISBN: 978-931971-95-9.

[48] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and*

Communications Security, ser. CCS '18, Toronto, Canada: ACM, 2018, pp. 2139–2153, ISBN: 978-1-4503-5693-0.

[49] M. G. Khun, "Compromising emanations: eavesdropping risks of computer displays," *The complete unofficial TEMPEST web page: http://www.eskimo.com/~joelm/tempest.html*, 2003.

[50] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers," in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, 2010, pp. 307–322.

[51] R. Callan, A. Zajic, and M. Prvulovic, "A Practical Methodology for Measuring the Side-Channel Signal Available to the Attacker for Instruction-Level Events," in *Proceedings of the 47th International Symposium on Microarchitecture (MICRO)*, 2014.

[52] B Coppens, I Verbauwhede, K. D. Bosschere, and B. D. Sutter, "Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors," in *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009, pp. 45–60.

[53] D. Genkin, I. Pipman, and E. Tromer, "Get your hands off my laptop: Physical side-channel key-extraction attacks on pcs," in *Cryptographic Hardware and Embedded Systems - CHES 2014*, ser. Lecture Notes in Computer Science, L. Batina and M. Robshaw, Eds., vol. 8731, Springer Berlin Heidelberg, 2014, pp. 242–260, ISBN: 978-3-662-44708-6.

[54] J. Brouchier, T. Kean, C. Marsh, and D. Naccache, "Temperature attacks," *Security Privacy, IEEE*, vol. 7, no. 2, pp. 79–82, 2009.

[55] M. Hutter and J.-M. Schmidt, "The temperature side channel and heating fault attacks," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, A. Francillon and P. Rohatgi, Eds., vol. 8419, Springer International Publishing, 2014, pp. 219–235, ISBN: 978-3-319-08301-8.

[56] M. Guri, M. Monitz, and Y. Elovici, "Usbee: Air-gap covert-channel via electromagnetic emission from USB," *CoRR*, vol. abs/1608.08397, 2016. arXiv: 1608.08397.

[57] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded Sys-*

tems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings, 2001, pp. 251–261.

[58] R. Callan, N. Popovic, A. Daruna, E. Pollmann, A. Zajic, and M. Prvulovic, "Comparison of electromagnetic side-channel energy available to the attacker from different computer systems," in *Electromagnetic Compatibility (EMC), 2015 IEEE International Symposium on*, 2015, pp. 219–223.

[59] M. Alam, H. A. Khan, M. Dey, N. Sinha, R. L. Callan, A. G. Zajic, and M. Prvulovic, "One&done: A single-decryption em-based attack on openssl's constant-time blinded RSA," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, 2018, pp. 585–602.

[60] A. Ketterlin and P. Clauss, "Profiling data-dependence to assist parallelization: Framework, scope, and optimization," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 2012, pp. 437–448.

[61] R. Joshi, M. D. Bond, and C. Zilles, "Targeted path profiling: Lower overhead path profiling for staged dynamic optimization systems," in *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, IEEE Computer Society, 2004, p. 239.

[62] M. Msgna, K. Markantonakis, and K. Mayes, "Precise instruction-level side channel profiling of embedded processors," in *International Conference on Information Security Practice and Experience*, Springer, 2014, pp. 129–143.

[63] J. Park, F. Rahman, A. Vassilev, D. Forte, and M. Tehranipoor, "Leveraging side-channel information for disassembly and security," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 1, pp. 1–21, 2019.

[64] G. T. Becker, D. Strobel, C. Paar, and W. Burleson, "Detecting software theft in embedded systems: A side-channel approach," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 4, pp. 1144–1154, 2012.

[65] R. Callan, F. Behrang, A. Zajic, M. Prvulovic, and A. Orso, "Zero-overhead profiling via em emanations," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ACM, 2016, pp. 401–412.

[66] R. Rutledge, S. Park, H. Khan, A. Orso, M. Prvulovic, and A. Zajic, "Zero-overhead path prediction with progressive symbolic execution," in *Proceedings of the 41st International Conference on Software Engineering*, IEEE Press, 2019, pp. 234–245.

[67] B. B. Yilmaz, E. M. Ugurlu, A. Zajic, and M. Prvulovic, "Instruction level program tracking using electromagnetic emanations," in *Cyber Sensing 2019*, International Society for Optics and Photonics, vol. 11011, 2019, 110110H.

[68] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic, "Spectral profiling: Observer-effect-free profiling by monitoring em emanations," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Press, 2016, p. 59.

[69] R. Callan, N. Popovic, A. Zajic, and M. Prvulovic, "A new approach for measuring electromagnetic side-channel energy available to the attacker in modern processor-memory systems," in *2015 9th European Conference on Antennas and Propagation (EuCAP)*, 2015, pp. 1–5.

[70] A. Nazari, N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic, "Eddie: Em-based detection of deviations in program execution," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2017, pp. 333–346.

[71] E. Cole, *Advanced persistent threat: understanding the danger and how to protect your organization*. Newnes, 2012.

[72] A Kavcic, "On the capacity of Markov sources over noisy channels," in *2009 IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 5, 2001, pp. 2997–3001.

[73] R. L. Dobrushin, "Translated from problemy peredachi informatsii," *Probl. Inf. Transmiss.*, vol. 3, no. 4, pp. 11–26, 1967.

[74] S. Verdú and S. Shamai, "Variable-rate channel capacity," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2651–2667, 2010.

[75] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[76] S. Verdú, "On channel capacity per unit cost," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 1019–1030, 1990.

[77] B. B. Yilmaz, A. Zajic, and M. Prvulovic, "Modelling jitter in wireless channel created by processor-memory activity," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 2037–2041.

[78] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design MIPS Edition: The Hardware/Software Interface*. Newnes, 2013.

[79] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2009, pp. 443–461.

[80] G. Becker, J Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G Kenworthy, T Kouzminov, A Leiserson, M Marson, P. Rohatgi, *et al.*, "Test vector leakage assessment (tvla) methodology in practice," in *International Cryptographic Module Conference*, vol. 1001, 2013, p. 13.

[81] T. Schneider and A. Moradi, "Leakage assessment methodology," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2015, pp. 495–513.

[82] F.-X. Standaert, "How (not) to use welch's t-test in side-channel security evaluations," in *International Conference on Smart Card Research and Advanced Applications*, Springer, 2018, pp. 65–79.

[83] M. Prvulovic and A. Zajic, *Rf emanations from a laptop*, `http://youtu.be/ldXHd3xJWw8`, 2012.

[84] J. Proakis, *Digital Communications*, ser. McGraw-Hill Series in Electrical and Computer Engineering. Computer Engineering. McGraw-Hill, 2001, ISBN: 9780072321111.

[85] DE1 FPGA on NIOS Processor, `https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=83&PartNo=2.`.

[86] Dual Polarized Panel Antenna, `http://www.l-com.com/wireless-antenna-24-ghz-16-dbi-dual-polarized-panel-antenna-n-female-connectors.`.

[87] OlinuXino A13, `https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino/open-source-hardware.`.

[88] AH-118, *Double Ridge Horn Antenna*, https://www.com-power.com/ah118_horn_antenna.html.

[89] P. Juyal, S. Adibelli, N. Sehatbakhsh, and A. Zajić, "A directive antenna based on conducting disks for detecting unintentional em emissions at large distances," *IEEE Transactions on Antennas and Propagation*, vol. 66, no. 12, pp. 6751–6761, 2018.

[90] U. R. Inc., *Aor la390 wideband loop antenna*, https://www.universal-radio.com/catalog/sw_ant/2320.html, 2014 (accessed Feb., 2019).

[91] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, 2015, pp. 865–880, ISBN: 978-1-931971-232.

[92] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," in *Information Security and Cryptology — ICISC 2002*, P. J. Lee and C. H. Lim, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 343–358, ISBN: 978-3-540-36552-5.

[93] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, 2015, pp. 207–228.

[94] L. N. Nguyen, C.-L. Cheng, M. Prvulovic, and A. Zajic, "Creating a backscattering side channel to enable detection of dormant hardware trojans," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[95] C.-L. Cheng, L. N. Nguyen, M. Prvulovic, and A. Zajic, "Exploiting switching of transistors in digital electronics for rfid tag design," *IEEE Journal of Radio Frequency Identification*, vol. 3, no. 2, pp. 67–76, 2019.

[96] M. Dey, A. Nazari, A. Zajić, and M. Prvulovic, "Emprof: Memory profiling via em-emanation in iot and hand-held devices," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2018, pp. 881–893.

[97] D. Precision, https://www.dell.com/en-us/work/shop/workstations-isv-certified/sc/workstations/precision-laptops..

[98] AARONIA PBS, https://www.tequipment.net/Aaronia/PBS1-5/Standard/Passive-Oscilloscope-Probes/?rrec=true..

[99] Power Rail Probe, https://www.keysight.com/en/pd-2471132-pn-N7020A/power-rail-probe?&cc=US&lc=eng..

[100] Keysight Signal Analyzer, https://www.keysight.com/en/pdx-x202266-pn-N9020A/mxa-signal-analyzer-10-hz-to-265-ghz?pm=spc&nid=-32508.1150426&cc=US&lc=eng..

[101] W. Mendenhall, R. J. Beaver, and B. M. Beaver, *Introduction to probability and statistics*. Cengage Learning, 2012.

[102] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers," in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, 2010, pp. 307–322.

[103] A. Waterman and K. Asanovic, *Risc-v instruction reference*, https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf, 2019 (accessed Nov. 6, 2019).

[104] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.

[105] D. McCann, E. Oswald, and C. Whitnall, "Towards practical tools for side channel aware software engineering: Grey box' modelling for instruction leakages," in *Proceedings of the 26th USENIX Conference on Security Symposium*, ser. SEC'17, Vancouver, BC, Canada: USENIX Association, 2017, pp. 199–216, ISBN: 978-1-931971-40-9.

[106] D. J. Hand, "Statistical concepts: A second course, fourth edition by richard g. lomax, debbie l. hahs-vaughn," *International Statistical Review*, vol. 80, no. 3, pp. 491–491, 2012.

[107] Terassic, *De0-cv*, http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=364, 2019 (accessed Nov. 6, 2019).

[108] Tektronix, *Tbs1000-digital-storage-oscilloscope*, https://www.tek.com/oscilloscope/tbs1000-digital-storage-oscilloscope, 2019 (accessed Nov. 6, 2019).

[109] W. B. Frakes and R. Baeza-Yates, *Information retrieval: Data structures & algorithms*. Prentice Hall Englewood Cliffs, NJ, 1992, vol. 331.

[110] A. Barenghi and G. Pelosi, "Side-channel security of superscalar cpus: Evaluating the impact of micro-architectural features," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18, San Francisco, California: ACM, 2018, 120:1–120:6, ISBN: 978-1-4503-5700-5.

[111] A. Heuser, W. Schindler, and M. Stöttinger, "Revealing side-channel issues of complex circuits by enhanced leakage models," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2012, pp. 1179–1184.

[112] T. Ming, W. Pengbo, M. Xiaoqi, C. Wenjie, Z. Huanguo, P. Guojun, and J. Danger, "An efficient sca leakage model construction method under predictable evaluation," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 12, pp. 3008–3018, 2018.

[113] B. B. Yilmaz, M. Prvulovic, and A. Zajic, "Capacity of em side channel created by instruction executions in a processor," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, 2019, pp. 0340–0345.

[114] G. Heinzel, A. Rüdiger, and R. Schilling, "Spectrum and spectral density estimation by the discrete fourier transform (dft), including a comprehensive list of window functions and some new at-top windows," 2002.

[115] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in c*, 1988.

[116] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler," in *Transactions on computational science X*, Springer, 2010, pp. 78–99.

[117] J. c. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, IEEE, 2018, pp. 1–6.

[118] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "Scandalee: A side-channel-based disassembler using local electromagnetic emanations," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, EDA Consortium, 2015, pp. 139–144.

[119] D. Vermoen, M. Witteman, and G. N. Gaydadjiev, "Reverse engineering java card applets using power analysis," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 138–149, ISBN: 978-3-540-72354-7.

[120] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*, Springer, 2009, pp. 1–4.

[121] A. Papoulis, *Probability, random variables, and stochastic processes*, ser. McGraw-Hill series in electrical engineering. New York: McGraw-Hill, 1991, ISBN: 0-07-100870-5.

[122] W. A. Gardner, "Two alternative philosophies for estimation of the parameters of time-series," *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 216–218, 1991.

[123] J. Massey, "Causality, feedback and directed information," in *Proc. Int. Symp. Inf. Theory Applic.(ISITA-90)*, Citeseer, 1990, pp. 303–305.

[124] S. Diggavi and M. Grossglauser, "On information transmission over a finite buffer channel," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1226–1237, 2006.

VITA

Baki Berkay Yılmaz grew up in Çameli, Turkey, received B.Sc. and M.Sc. degrees in Electrical and Electronics Engineering from Koc University, Turkey in 2013 and 2015, respectively. During his studies in Koc University, he worked as a Teaching Assistant and did research on channel equalization and sparse reconstruction.

He joined Georgia Institute of Technology in Fall 2016 where he also received a M.Sc degree in Electrical and Computer Engineering in 2018. He has pursued a Ph.D. degree in School of Electrical and Computer Engineering and has worked as a Graduate Research Assistant in the Electromagnetic Measurements in Communications and Computing Lab focusing on quantifying covert/side-channel information leakage. His research interests span areas of electromagnetic, machine learning, signal processing and information theory.