A Power Modeling Approach to Protect GPUs from Side-Channel Attacks

A Dissertation Presented
by

Saoni Mukherjee

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Engineering

Northeastern University
Boston, Massachusetts

April 2020

ProQuest Number: 27837707

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent on the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 27837707

Published by ProQuest LLC (2020). Copyright of the Dissertation is held by the Author.

All Rights Reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

NORTHEASTERN UNIVERSITY

Graduate School of Engineering

Dissertation Signature Page

Dissertation Title: A Power Modeling Approach to Protect GPUs from Side-Channel Attacks

Author: Saoni Mukherjee **NUID:** 001167197

Department: Electrical and Computer Engineering

Approved for Dissertation Requirements of the Doctor of Philosophy Degree

Dissertation Advisor
Prof. David Kaeli

_____ Signature _____ Date

Dissertation Committee Member
Prof. Yunsi Fei

_____ Signature _____ Date

Dissertation Committee Member
Prof. Rafael Ubal

_____ Signature _____ Date

Department Chair
Dr. Srinivas Tadigadapa

_____ Signature _____ Date

Associate Dean of Graduate School:
Dr. Waleed Meleis

_____ Signature _____ Date

”It always seems impossible, until it is done.” - Nelson Mandela

Contents

List of Figures

List of Tables

List of Acronyms

GIPSim GPU Instruction-level Power SIMulator

NIST National Institute of Standards and Technology

UAV unmanned aerial vehicles

FFT Fast-Fourier transform

API Application Programming Interface

ISA Industry Standard Architecture

IC Integrated Circuit

CMOS Complementary Metal-Oxide Semiconductor

SNR signal-to-noise ratio

LSTM Long Short-Term Memory

GPU Graphic Processor Unit

GPGPU General Purpose computing on Graphic Processor Units

SIMT Single Instruction Multiple Thread

SIMD Single Instruction Multiple Data

SM Streaming Multiprocessor

CKE Concurrent Kernel Execution

DVFS Dynamic voltage and frequency scaling

CUDA NVIDIA's Compute Unified Device Architecture Framework

OpenCL Open Compute Language

PTX Parallel Thread Execution

NVML NVIDIA Management Library

SCA Side-Channel Attack

SPA Simple Power Analysis

DPA Differential Power Analysis

CPA Correlation Power Analysis

DOM Distance of Mean

HW Hamming Weight

HD Hamming Distance

DRP Dual-Rail Precharge

RCoal Randomized Coalescing

pSCA Profiling-based Side-Channel Attack

PCC Pearson Correlation Coefficient

QMS Quantitative Masking Strength

Acknowledgments

As I finish the Ph.D. program, I have many people to be grateful to during my journey as a graduate student. My time at Northeastern had been full of learning and growing as a person, and there are a lot of people who helped me throughout the process.

I can't thank my parents (Mr. Kesablal and Mrs. Swapna Mukherjee) enough for their unrelenting faith, support and love. My father had an unfulfilled dream of finishing a Ph.D., which he wanted to fulfill through his two children. Today, I hope I have been able to fulfill his dream and make him proud! Ph.D. is often tiring, demanding, and full of moments of frustration. On those days of distress, my mother has always been understanding and patient. Her unwavering belief in me inspires me to reach higher. I am also fortunate to have my brother, Amarnath, and sister-in-law, Devdutta, at all my beck and calls. I thank them for providing me with the comforts of home in this faraway land. I owe it to my family for whatever I am today.

I thank my advisor, Prof. David Kaeli, for providing me with the opportunity to work with him. His constant support and encouragement has brought out the best in me. I admire his attention to detail and personal interest in grooming each of his students. He has always been there when I needed him. Apart from research, I hope I have inculcated some of his virtues: how to be a good person, always grounded, respect, and help others. These will definitely pave my way to lead a better professional and personal life.

I also thank my co-advisor, Prof. Yunsi Fei, for her valuable suggestions throughout this thesis. There were many times I felt lost, and she helped me get back to the track with her expertise in the field of hardware security. I also thank her for letting me use the equipment in her lab to conduct the power measurement experiments. I appreciate the members of NUEESS lab for helping me solve problems with the equipment. I would also like to thank Prof. Rafael Ubal for agreeing to be on my committee.

I have had the great opportunity to work with the talented members of the NUCAR lab-Charu, Fritz, Elmira, Leiming, Fanny, Nico, Yifan, Yash, Xiangyu, Amir, Kavi, Xiang, Chen, Xun, Shi, Julian, Trinayan, Kaustabh, Yuihui. A big thanks to Dana Schaa for introducing me to Prof. Kaeli. Charu was always my one-stop solution for everything from research to life and more.

I am also thankful to my master's advisor, Prof. Anthony Skjellum, for inspiring me to do a Ph.D. Had he not encouraged the idea of pursuing a Ph.D., I would not have ended up traversing this long path, every inch of which I thoroughly enjoyed. I would also thank all my teachers, mentors, lab-mates, especially Prof. Miriam Leeser and Dr. Nicholas Moore, for helping me learn the basics of Computer Architecture and research in general.

During my stay in Boston, I looked forward to the weekends to spend time with some fantastic friends. They all brought different flavors to my life! Ayushmati was always a phone call away. Nupur, Debajyoti and Lipi ensured that I have a life outside my Ph.D., and Ayan and

Samragnee made sure that I never felt starved. During the last phase of my Ph.D., Sougata made sure that I could keep my calm perform to the best of my abilities. I thank all my friends near and far for the laughs we had that kept me refreshed.

Finally, I would like to thank the National Science Foundation and Semiconductor Research Consortium for funding my research.

Abstract of the Dissertation

A Power Modeling Approach to Protect GPUs from Side-Channel Attacks

by

Saoni Mukherjee

Doctor of Philosophy in Computer Engineering

Northeastern University, April 2020

Prof. David Kaeli, Advisor

Graphic Processing Units (GPUs) have become of accelerator of choice to speed up the execution of a wide range of applications. Given the massive number of compute cores on these devices, GPUs have become an attractive platform to accelerate security and cryptography applications. While performance is a critical quality to prevent online attacks, current accelerators are ill-equipped to protect against side-channel attacks. When launching a side channel, the attacker exploits the physical implementation of a cryptographic algorithm, rather than the inherent theoretical weaknesses of the algorithm. Side channel leakage can come in the form of power dissipation, performance degradation or electromagnetic waves. This thesis considers the first of these sources.

Power modeling of GPU devices has been well studied by the computer architecture community. The goal of power modeling is to be able to tradeoff power for performance in the design of the GPU. Prior work has shown that by recording the amount of energy consumed by a GPU during encryption or decryption, an attacker can capture secret information. If we can understand how the underlying microarchitecture leaks side-channel information to an attacker, we can build much more robust obfuscation approaches.

In this thesis, our aim is not to build yet another GPU power model. Instead, we deliver GIPSim, a framework to enable security researchers to reason about side-channel leakage present in the context of a GPU execution-driven simulator. We show how researchers can capture detailed power estimates while running CUDA programs on a Kepler-family GPU and use the information to obfuscate power by obscuring data-dependent power leakage. We show how traditional hiding and masking techniques can be applied in the context of a GPU. These, in turn, reduce the vulnerability present in this context. We also detail how we leverage machine learning techniques using a Long Short-Term Memory neural networks to further improve the obfuscation. Our goal is to design a system that can thwart power-based side-channel attacks. We demonstrate that we can model

data-dependent power dissipation, capturing the hamming distance of data values used during the execution of AES encryption. This same approach is used in power-based side-channel attacks. GIPSim is one of the first simulation environments that can be used for evaluating power side-channel resiliency and help build a more secure accelerator.

Chapter 1

Introduction

Computers, and the networks that interconnect them, have changed society and improved our way of life. While technology has benefitted all of us, these advances have increased our dependence on computer systems and the software that runs on them. Security threats and vulnerabilities in architectures, networks, and software have become commonplace. Each day we read about a new security breach, leaking sensitive information to attackers. As technology becomes increasingly ubiquitous, the rampant exploitation of system vulnerabilities represent a threat to our ability to use information technology safely. Problems range from the annoyances of spam emails, identity theft, and loss of productivity, to catastrophic damage to critical information or theft of financial assets. Given the number and variety of security threats, computing technology is changing from an enabling technology to a disabling technology.

A primary requirement of today's software and information security platforms is to protect the confidentiality and integrity of data that is computed and transmitted among multiple parties. Failure to protect this information can not only incur great financial loss, but can give rise to life-threatening situations, such as when patient medical records became inaccessible. In 2016, all MedStar Health hospital medical records were taken offline after their computer networks were attacked by Ransomware [1]. In western Ukraine, cyber-attacks left nearly a quarter-million people without power or heat for several hours in the middle of winter [2]. To reduce the exposure to these attacks, most sensitive data (e.g., personal health records, emails, income tax returns and financial reports) are typically encrypted using well-known cryptographic techniques.

1.1 Cryptography

With the ever-increasing proliferation of e-business practices, a massive volume of business transactions and data transmissions are regularly carried out in devices ranging from smartphones, workstations, and data centers. Cryptographic algorithms are employed to secure these transactions. Cryptography is the science of protecting information and communications so that only the authorized parties for whom the data is intended can read and process it. Exclusive access to the unprotected data (i.e., the *plaintext*) is achieved by encrypting the data using a *key* that is known to only authorized parties.

Since the time of the original benchmark paper by Claude Shannon that detailed the theory behind secure communications, numerous scientific advancements have been made on developing new cryptographic algorithms [3]. The algorithms used today are supported by extensive mathematical evaluations that provide guaranteed levels of secrecy. This ensures that an attacker would find it extremely difficult to guess the encryption key from the encrypted text, even if the plaintext and the encryption algorithm are known.

The importance of cryptographic solutions has continuously been growing in the last decade as a result of using the Internet in critical areas, including finance, government, and health-care.

1.1.1 Need for accelerated cryptography

The emerging era of ubiquitous computing or pervasive computing provides an array of smart products, such as RFID, smart meters, smart thermostats, smart refrigerators that can communicate with each other unobtrusively and are always available. This enables users to access information and services anywhere and anytime. Projects such as MIT Media labs *things that think* and Darpa's *smart space* are showing a path to help us realize the dream of seamless integration of digital infrastructure in our daily lives. These applications often require real-time implementations of cryptographic algorithms. Both unmanned aerial vehicles (UAV) and smart grids are such examples. UAVs continuously exchange dynamic information regarding the urban environment with a gateway that provides constant feedback regarding different optimization parameters. These parameters are used to include in the UAV's path planning algorithm to find the path that the UAV travels in to reach the destination safely. On the other hand, smart grids consist of a network of electric meters for reporting the usage or power outage to the power utility company in real-time. Here sensitive data, such as power consumption, price, outage awareness, are exchanged between the meters and the

utility company in real-time. Both UAV and smart grids are equipped with sophisticated technologies, such as GPUs, to provide high throughput to run in real-time [4, 5, 6].

Many hardware-based SSL acceleration solutions have been studied and proposed in both research and industrial fields. In the initial days of general-purpose GPUs (GPGPU), efficient GPU based cryptographic solutions were not possible to be implemented, mainly because:

- the GPUs lacked native support for integer representations of the data and lacked support for bitwise operations,
- the GPU programming models utilized graphics languages and lacked fundamental operations, such as *scatter*,
- the GPU's memory model was very restrictive, and
- there was significant overhead when using a fixed graphics pipeline.

Over the past decade, hardware vendors devised new solutions that significantly improved a GPU's programming and memory models, and better support general-purpose computations on the graphics hardware.

1.2 Graphics Processing Units

There has been a substantial paradigm shift in the computing landscape in terms of the introduction of accelerators, in particular, GPUs. Over the short timeframe that GPUs have been used for computing, they have evolved from serving as configurable graphics processors rendering high-quality 3-D graphics for games, to become the prevalent platform for general-purpose GPU computing. Today, GPUs are ubiquitous - they are present in PC, laptops, desktops, smartphones, workstations, and data centers. They are designed as multithreaded multiprocessor architectures that excel at both graphics and compute applications. Given that GPUs provide a massively parallel processing platform, GPUs power many of the world's supercomputers on the TOP500 list (the fastest 500 supercomputers in the world, published each year) [7].

The demand for faster and higher definition graphics continues to drive the development of increasingly parallel GPUs. Initially, programmers were forced to use graphics-centric programming languages to implement compute applications on these devices. However, in the past decade, new programming languages were introduced that significantly increased programmer productive when

developing code on GPUs [8, 9]. GPUs are the accelerator of choice for high-performance computing, deep learning and many other data-intensive applications. Along with advances in parallel languages, GPU vendors have help deliver a rich set of libraries and tools that increased developer productivity. These GPU codes are more effective than traditional CPUs in applications, where the workload is mainly compute-bound and data-parallel, supporting concurrent execution. In addition to high computational throughput, these devices are also able to exploit high bandwidth memory, providing a high degree of memory parallelism. GPUs are able to hide memory latency using fast context switching between threads.

1.2.1 Emerging GPU applications

GPU computing is at a tipping point now, becoming widely used in demanding consumer applications and high-performance computing. The range of applications accelerated by a GPU is continually increasing, and often, they are used in the areas that involve significant security constraints, such as financial markets and medical diagnostics [10, 11]. GPUs are being used to perform different types of safety-critical tasks such as pedestrian detection and avoidance and vehicle control [12, 13]. While GPUs are crucial in accelerating these applications, it is becoming increasingly important that GPU execution is safe and secure. Given the increasing demands of compute-intensive applications, which include security/cryptography algorithms, coupled with the availability of many-core GPU architectures and efficient high-level programming languages, security researchers/professionals are using these devices to accelerate cryptographic computations [14, 15, 16, 17].

1.3 Cryptography on GPUs

A block cipher is a cryptographic algorithm that encrypts/decrypts one fixed-size block of data at a time. This class of algorithms is very well-suited for parallel computing because of the independent data blocks and common operations performed on blocks of data. Many cryptographic libraries include implementations for encryption/decryption on GPUs [18, 19, 20]. Although they provide high throughput and multi-digit speedups over the fastest CPU implementations, research on GPU security is still in its infancy.

The strict standards of security guarantee the *mathematical strength* of cryptographic implementations, and are often secure from standard *cryptanalysis*. Cryptanalysis is the counterpart to cryptography and deals with retrieving plaintexts from encrypted text without the knowledge

of the key. For example, the best known cryptanalytic attack against the popular AES block cipher implementation [21] would require around 2^{126} operations before any information about the encryption key would be obtained from the encrypted text. This may take a long time for even the fast computing resources to recover the key.

When developing these software-based security applications, the underlying hardware systems that deliver computation, and support communicate, are typically assumed to be secure and reliable [22, 23]. However, such security assumptions about hardware cannot be easily guaranteed because hardware systems can also have vulnerabilities and are prone to attacks.

1.4 Attack on Cryptography

Irrespective of the device where a cryptographic algorithm is run, the execution of the algorithm always leaves some form of a *trace* on the system and its surrounding medium. This trace may reveal information about the internal state of the encrypted text that classical cryptanalytic attackers do not have. The mathematical proofs that guarantee security of the encryption fails to consider these vulnerabilities, resulting in attacks that are considerably more practical.

There has been a constant battle between the security researchers and attackers, with new classes of attacks discovered every day. Cryptographic algorithms developed in software and running on hardware need to be protected against attacks that attempt to reveal secret information. Hardware attacks can be characterized as follows:

- Active or Passive Attacks- Active attacks require the attacker to tamper or perturbate the hardware internals, such as, electrically probing them or through laser impingement, using such approaches to gather data from the device. On the other hand, passive attacks require the attacker only to observe and infer the secret by exploiting one or more physical characteristic of the system while it is in operation. The physical characteristics include power consumption, electromagnetic emanation, processing time, etc.

- Invasive, semi-invasive or non-invasive attacks - Invasive attacks require the attacker to open the system and fiddle with the circuitry within. Although semi-invasive attacks require the attacker to open the system, the attacker does not need to modify the internal circuits. Lastly, in the non-invasive attacks, the attacker does not open the system package.

1.5 Motivation for this thesis

Due to the demands of efficient cryptographic computation over large amounts of data, GPUs are being leveraged to accelerate several cryptographic algorithms. When the cryptographic algorithms are executed on a device, it leaves a trace in the form of information about the device. For example, one kind of trace can be the time-varying power consumed by the GPU during encryption. This trace, which is a result of the modulation of the dynamic power consumption, includes information about the data being processed and can be used by an attacker to retrieve the secret key in a small amount of time.

While several researchers have studied how to accelerate cryptographic algorithms on GPUs, there has been limited work done to evaluate or harden the security of such accelerators. Although side-channel analysis (SCA) has received a lot of attention in the research community, including timing analysis, differential fault analysis and differential power analysis, there has not been a lot of work done on evaluating these attacks on GPUs. Most of the work on SCA has focused on non-invasive attacks and passive attacks, where the physical characteristics of the system opens a *side-channel* or backdoor through which secret information used in the cryptography is leaked. The class of attacks that use these side-channels to retrieve the secret information is called *side-channel attacks*. Apart from power consumption, researchers have worked on many other side-channels over the years, such as electromagnetic emanation, execution time, high-pitched acoustic vibrations of the electronic components, etc. To design a defense, it may seem trivial to overcome side-channel leakage by simply adding noise, randomizing, or fuzzing the side-channels in the system. While these additions may successfully stymie certain classes of attacks on selected platforms, it becomes very challenging to analyze the side-channel vulnerabilities and later block them on GPUs due to their unique Single Instruction Multiple Threads (SIMT) execution model characteristics that differ from CPUs and FPGAs. We discuss this in more detail later in Chapter 4.

1.6 Scope and Contributions of this thesis

Power analysis is a branch of side-channel analysis where power consumption data is used as the side-channel to attack the system. In this thesis, we show that equipped with the right set of software tools, we can design effective countermeasures to thwart such attacks. We present the design and development of a novel simulation framework for assessing GPU power leakage. The model tracks power at the instruction level for an NVIDIA GPU. Here, we did not aim to build yet another

GPU power model. Our model calculates a base cost and inter-instruction overhead for different instructions with different input data values. We address some challenges of measuring the power on a GPU. Once we can model power usage, we can design new software-based obfuscation techniques to hide the power profile by carefully adding noise to the power signature of the cryptographic algorithm in use.

1.6.1 Current contributions

This proposal makes the following contributions:

- **Power leakage for Kepler GPU.**

  We provide a detailed analysis of power leakage and power acquisition on Kepler GPUs. We show how researchers can capture power profiles while running CUDA programs on a Kepler-family GPU.

- **Instruction level power model framework.**

  We present our power model named GIPSim (GPU Instruction-level Power SIMulator), a framework to enable security researchers to reason about side-channel leakage present on GPUs. Researchers can use GIPSim to explore how to add impurity in a power profile of program execution. We validate GIPSim, showing there is a strong correlation between the measured power on the GPU and modeled power in GIPSim.

- **Obfuscation approach for power side-channels on GPUs**

  We demonstrate GIPSim's ability to incorporate instructions that can add noise to cryptographic execution. When running together, they draw more power, and introduce randomness into the power profile, thus decreasing the strength of the power side-channel. We show that this approach decreases the vulnerability of power SCA.

Note that this work is not restricted to only this class of GPUs. A similar approach can be taken for any class of GPU. To the best of our knowledge, GIPSim is the first attempt to provide a simulation environment that can be used for evaluating and thwarting power side-channels.

1.6.2 Proposed contributions

The above contributions allow us to reason about power leakage that exists in the GPU execution and develop new methods to thwart the side-channel attack. With this new understanding and proposed methodology, we propose to pursue the following work to complete this thesis:

- **Obfuscation with hiding and masking:** For hiding, running a random computation along with the cryptographic computation lowers the signal strength. On the other hand, masking tries to secure the system by minimizing the correlation between the intermediate variables and the secret information. Given that this introduces some performance overhead, we want to evaluate the obfuscation strength of different options inside GIPSim.

- **Performance:** We want to be able to identify an obfuscation approach that limits the impact on the speed of the cryptographic computations. GIPSim should include an option to be configurable to trade-off performance and obfuscation strength.

- **LSTM-guided obfuscation:** Based on our work with training using different keys, we will produce obfuscation approaches that leverage a neural network to predict obfuscation for the target cryptographic computation.

1.7 Organization of the Thesis

The rest of this thesis is organized as follows: In Chapter 2, we present background on GPU architecture and the associated CUDA programming model. We also provide an overview of the cryptographic algorithm used in this thesis. Then we discuss the various types of side-channel attacks, and the attack model assumed. We review the body of work and the state of the art research related to this thesis in Chapter 3. In Chapter 4, we present our instruction-level power model, GIPSim and show how we validate the model using correlation. Chapter 5 describes the obfuscation approaches we used to hide and mask the power profile of the cryptographic algorithm and how it impacts signal strength. We also show how LSTM neural network can be used in predicting the obfuscation for cryptographic applications. Finally, Chapter 6 summarizes the thesis and provides a brief description of future work.

Chapter 2

Background

In Chapter 2, we present background on GPU architecture and its associated programming model. We also provide an overview of AES, the cryptographic algorithm we have tested so far in this thesis. We discuss the CUDA implementation of AES. Finally, we discuss side-channel attacks in general, going deeper into correlation power analysis, the class of attacks considered in this thesis.

2.1 GPU basics

GPUs are designed to be a throughput-oriented compute device. As a complement to CPUs, GPUs excel at executing latency-sensitive applications. Earlier GPUs provided a fixed-function graphics pipeline, designed for efficient 3-D graphics rendering. With the introduction of unified shader and high-level programming languages, modern GPUs are used as programmable data-parallel processors for accelerating general-purpose applications. There has been significant research by a large body of researchers and developers to enable GPU performance.

2.1.1 Compute Unified Device Architecture (CUDA)

The CUDA programming model acts as a bridge between an application and its implementation on available hardware. It provides an Application Programming Interface (API) to generate code for NVIDIA GPUs. In the context of programming, CUDA offers extensions to the C/C++ language, including libraries and rich data types to support execution on a GPU [24].

CUDA was designed to preserve several elements present in sequential programming models and extends them to a parallel threaded execution model. A CUDA program is composed of

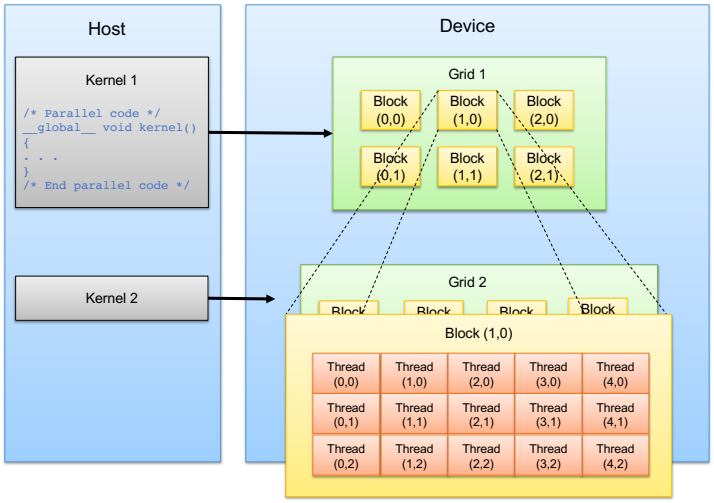


Figure 2.1: Thread Hierarchy in CUDA Programming.

two parts: 1) host code that runs on a CPU, and 2) device code that runs of one or multiple GPUs. A host is a system on which launches CUDA library calls, (i.e., an x86 CPU), while the device is an accelerator that executes the CUDA library code (i.e., a GPU). The portion of the program that is executed on the device is called a *kernel* and is identified using the `__global__` declaration specifier. When the kernel is called, it is executed in the form of many parallel instances that are mapped to a set of parallel threads. The user specifies the number of threads that will execute the kernel as part of the configuration syntax `<<<...>>>`. These threads are grouped into a *thread block*, and each thread in the thread block has a unique ID that can be obtained in the kernel through the built-in `threadIdx` variable. Similarly, a thread block is accessible through the `blockIdx` variable. The thread blocks form a *grid* that represents the whole computation domain, as shown in Figure 2.1.

CUDA has multiple memory spaces that differ in both performance and capacity. Each thread has its own private memory. While offering the highest bandwidth, this memory space is limited. For efficient inter-thread communication inside a thread block, the thread block offers an on-chip, high bandwidth, shared memory. Apart from the on-chip memory, the GPU also has an off-chip global memory that is accessible to all threads. Some special memory spaces are optimized for different memory usages and can be accessed by all threads:

– Read-only constant memory: This stores data that will not change throughout kernel execution. NVIDIA hardware provides 64KB of constant memory that is cached and in some situations,

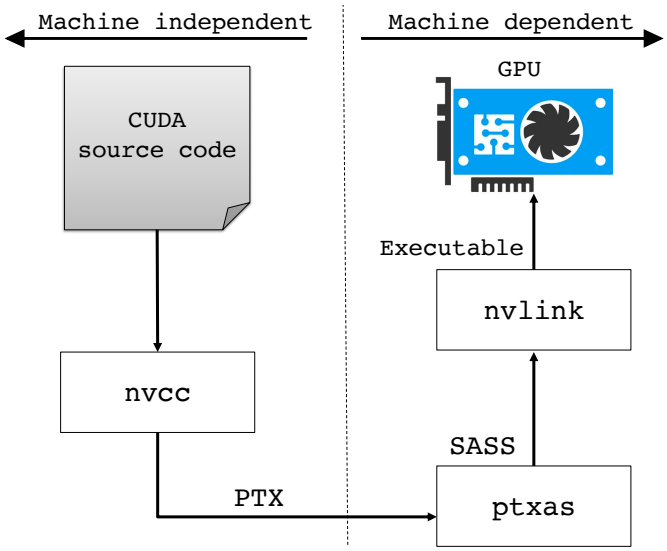


Figure 2.2: The workflow if NVIDIA `nvcc` compiler.

using constant memory rather than global memory will reduce the required memory bandwidth.

– Read-only texture memory: Although this was initially designed for classical OpenGL and DirectX rendering pipelines, this cached on-chip memory is very useful for applications where memory access patterns exhibit a great deal of spatial locality. This is specialized for 2D read-only coalesced memory.

2.1.1.1 Compilation Workflow

Programs developed in CUDA C/C++ are compiled into binary code using NVIDIA's LLVM-based CUDA Compiler, `nvcc`, before being executed on the GPU [25]. Starting with CUDA code as input, the front-end compiler generates an intermediate representation, LLVM-IR, and subsequently generates virtual instruction set (ISA) code, called PTX (Parallel Thread Execution), as shown in Figure 2.2. Next, the backend compiler, `ptxas`, translates that PTX to machine code called SASS. Finally, a final binary file is generated using NVIDIA's proprietary Optimized Code Generator. This binary code is packaged into a device code descriptor, that is included in the host code. The CUDA runtime system inspects this descriptor whenever the host code invokes the device code. To launch a kernel, the compiler looks for the `<<<...>>>` syntax in the host code and replaces it with the appropriate runtime library calls.

It can be very challenging to modify a SASS-level program with NVIDIA's framework. In this work, we use the Kepler assembler and disassembler, enabling us to work with SASS code, providing a framework to generate detailed instruction sequences [26].

2.1.2 GPU architecture

Next, we describe the NVIDIA Kepler architecture, given that we have selected this popular device to serve as our evaluation platform [27]. The centerpiece of the NVIDIA GPU architecture is the Streaming Multiprocessor (SM or SMX), the primary unit of computation on the GPU. A thread block is assigned to run on an SMX using a round-robin scheme until the resources of the GPU are occupied. After the block is scheduled, the remaining blocks are executed in the following rounds. Once a thread block is scheduled on an SMX, it remains there until execution is finished. Registers and shared memory are scant resources on an SMX since they have allocated in a shared space for all threads assigned to the SMX. NVIDIA's Kepler GK110 GPU architecture features up to 15 SMX units, each of which has 192 single-precision CUDA cores, 64 double-precision units, 32 load/store units, and 32 special function units. Each thread has access to 255 registers. Threads can only access its own private register file, but register values can be shared with other threads via special instructions. When a GPU kernel is executed, the SMX groups 32 software threads (identified by the programmer) into hardware threads called warps. Threads within a warp execute in a Single Instruction Multiple Data (SIMD) fashion. The SIMD execution model requires that all the threads in a warp always execute the same instruction. Each SMX has four warp schedulers and eight instruction dispatchers, which allows four warps and eight independent instructions (two per warp) to be issued and executed concurrently.

Modern GPUs have the capability to run multiple kernels, assigned to different streams, concurrently. A *stream* is a sequence of operations that are performed in order on the device. Multiple kernels can be run using different streams concurrently. Every CUDA kernel is invoked on an independent stream, which is queued to the GPU's hardware schedulers, determining the order of kernel execution. The Kepler, Maxwell and Pascal architectures support up to 32 concurrent streams. Each stream is assigned to a different hardware queue.

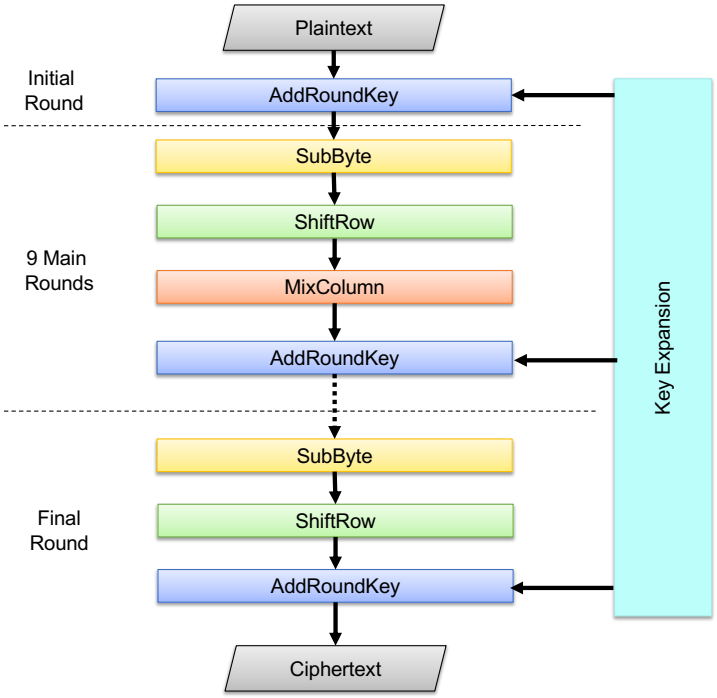


Figure 2.3: Overview of different rounds of AES.

2.2 The Advanced Encryption Standard

In 2001, the National Institute of Standards and Technology (NIST) announced AES as a new encryption standard [21]. Since then, AES has been a worldwide standard and adopted by the US government to encrypt data in applications ranging from personnel to highly confidential domains. AES is a symmetric-key block cipher that can use varying key sizes such as 128, 192 or 256 bits to encrypt and decrypt 128-bit blocks. Since we have used AES-128, we provide a description of the algorithm for that key size. The input to AES-128 is organized in a 4×4 matrix of bytes, called the *state*. The state undergoes a number of transformations in ten rounds during the encryption process, as shown in Figure 2.3

Figure 2.4 presents the operations that performed through one round of AES-128. The first operation performed on the input is called *AddRoundKey*, which serves to provide some initial randomness by mixing the input key. Then the state goes through nine rounds to further increase the

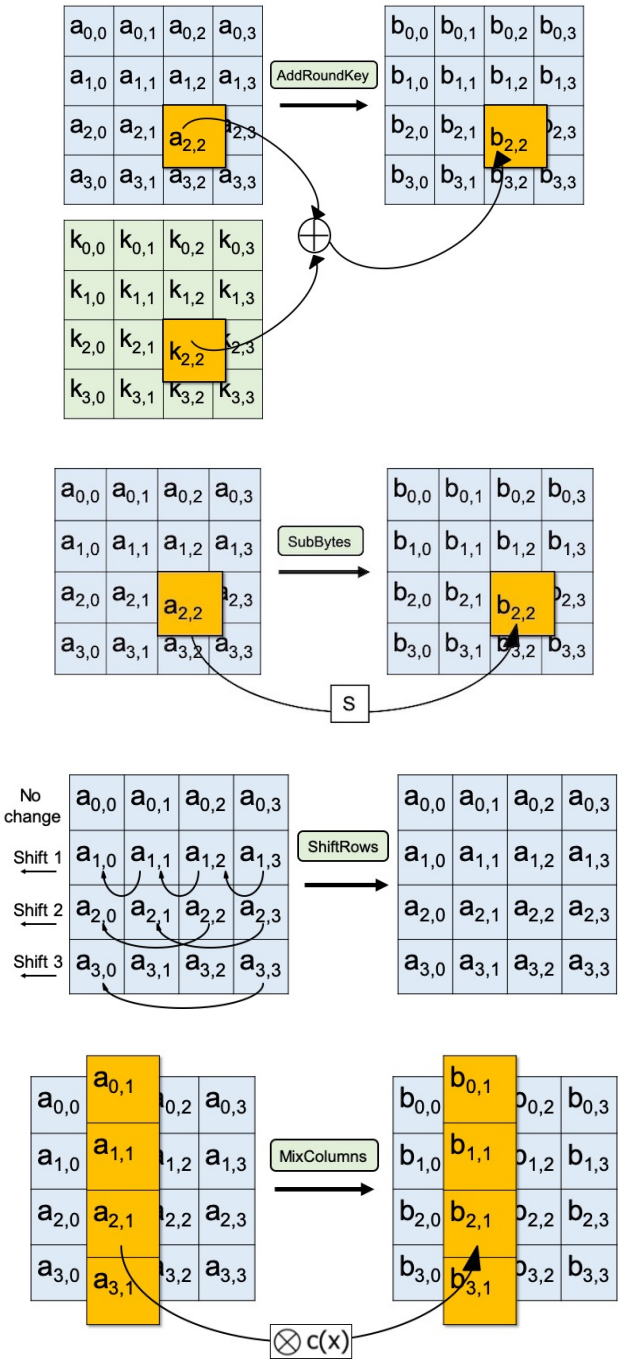


Figure 2.4: Overview of four steps in AES: AddRoundKey, SubBytes, ShiftRows, and MixColumns.

diffusion and confusion in the cipher. Each of these rounds comprise of four operations on the state: *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*. After that, the state goes through one final round, which includes all these operations except the *MixColumns* operation. These operations can be described as:

1. *AddRoundKey*: Each element in the state is bitwise XORed with a 128-bit round key. This round key is generated from the secret key using a key expansion algorithm described later.

2. *SubBytes*: Each element in the state is replaced by an affine transformation of its inverse in the field $GF(2^8)$. For a byte s_i in the state, this operation is denoted by $S(s_i)$.

3. *ShiftRows*: Here a cyclic shift of the ith row in the state occurs by i bytes toward the left (where $0 \leq i \leq 3$). In other words, each byte in the ith row is cyclically shifted to the left by i bytes.

4. *MixColumns*: This operation provides a column-wise linear transformation of the state matrix. Each column of the state matrix is considered as a polynomial of degree 3, with coefficients in $GF(2^8)$ and multiplied by the polynomial $\{03\}\alpha^3 + \{01\}\alpha^2 + \{01\}\alpha + \{02\}mod(\alpha^4 + 1)$. The combination of *ShiftRows* and *MixColumns* provide the necessary diffusion of the cipher.

The Key Expansion algorithm takes the secret key as input and generates round keys for 11 *AddRoundKey* operations performed in one complete run of AES-128. As described in Algorithm 1, there are two operations, `ROTWORD` and `SUBWORD` [28]. They cyclically shift and substitute the four bytes (B_0, B_1, B_2, B_3) as:

$$\texttt{ROTWORD}(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0) \tag{2.1}$$

$$\begin{aligned}\texttt{SUBWORD}(B_0, B_1, B_2, B_3) = (&\texttt{SubBytes}(B_0), \texttt{SubBytes}(B_1), \texttt{SubBytes}(B_2), \\ &\texttt{SubBytes}(B_3))\end{aligned} \tag{2.2}$$

Here, AES uses a round constant, defined as $RCon[1], \ldots RCon[10]$, as constants in hexadecimal before applying the key expansion Algorithm 1.

Algorithm 1: Key Expansion of AES-128

```
input  : 128 bit secret key
output : 11 round keys each of 4 words as w[0], ..., w[43]
begin
    Rcon[1] ← 01000000;
    Rcon[2] ← 02000000;
    Rcon[3] ← 04000000;
    Rcon[4] ← 08000000;
    Rcon[5] ← 10000000;
    Rcon[6] ← 20000000;
    Rcon[7] ← 40000000;
    Rcon[8] ← 80000000;
    Rcon[9] ← 1B000000;
    Rcon[10] ← 36000000;
    for i ≤ 0 to 3 do
        w[i] ← (k[4i], k[4i + 1], k[4i + 2], k[4i + 3])
    end
    for i ≤ 4 to 43 do
        temp ← w[i − 1];
        if i ≡ 0(mod4) then
            temp = SUBWORD(ROTWORD(temp)) ⊕ RCon[i/4];
        end
        w[i] ← w[i − 4] ⊕ temp;
    end
    return (w[0], ..., w[43])
end
```

2.2.1 CUDA implementation of AES

In this work, we use an Electronic Code Book mode AES-128 encryption algorithm as implemented in CUDA [29]. We leverage the T-table version since it is more efficient than the original byte-based SBox implementation of AES.

In our implementation, we integrate the three steps, SubBytes, ShiftRows and MixColumns, into T-table lookups and XOR operations. The initial round is an XOR operation of the plaintext and the first round key. After that, there are nine rounds, where one diagonal of the state is considered as its round input, and we map each byte into a 4-byte word through a T-table lookup. The four 4-byte words are XOR-ed with their respective 4-byte round key bytes. Finally, the result is stored in a column of the output state. Since the last round has no MixColumns operation, only one out of four bytes is kept after the T-table lookup. This is equivalent to an SBox lookup operation and ShiftRow. AddRoundKey is then performed on the four remaining bytes.

One column of a 16-byte AES state is computed by one thread, so a total of 4 threads are needed for a block of data. Note that earlier in this section we described a thread block, which is different from the 16-byte AES data block. The data block is iteratively updated after each round, transforming the plaintext to ciphertext.

At the beginning of the encryption, the plaintext is loaded into the GPU's global memory. Each thread takes a portion of the plaintext data and copies it to local memory. The plaintext is partitioned based on the thread's block and thread id. Once the encryption completes, the resulting ciphertext stored in the local memory is copied to global memory so that it can be transferred to CPU memory. In ECB mode, the data blocks are entirely independent; hence depending on the data size and resources available, the encryption process is easily parallelized.

2.3 Side-Channel Attack

The idea of using side-channels to steal information has been used for over a century. During World War II, the electromagnetic emissions from the devices were used for spying to reveal the plaintext that was transported in a secured manner from a distance. Later during the 1990s, in two seminal papers, Kocher *et al.* described how execution time and power consumption could be used to retrieve secret keys easily from naive implementations of cryptographic algorithms [30, 31]. Subsequently, substantial research activities have explored a range of architectural details, devising new ways to implement side-channels and motivating related defenses. We discuss them in detail in

Chapter 3.

There are mainly three requirements for a Side-Channel Attack (SCA) to be successful:

1. Perturbation: this happens when the attacker alters the behavior of the system or the state of the system.

2. Manifestation: when the system is not accessible to perform the perturbation, they are manifested through side-channels.

3. Observation: the attacker has to observe the side-channel to obtain the required information about the secret information.

Sometimes, even though all the three requirements of SCA are fulfilled, the attacker may not always be successful in retrieving the secret information. The amount of success of the attacker depends on additional factors. The first factor is how effectively perturbation conveys the secret. Another critical factor affecting the success is the noise in the channel.

When exploiting the unique Single Instruction Multiple Thread (SIMT) execution patterns of a GPU, multiple threads can be run concurrently with different data, introducing a significant amount of noise in the physical side-channel that the attacker observes. GPU execution is fairly non-deterministic, utilizing a hardware-based thread scheduler, which introduces temporal uncertainty into the resulting execution, presenting skew in the side-channel signal. These factors make SCA challenging on a GPU.

2.3.1 Correlation Power Analysis Attacks

SCA is a class of attacks where the attacker gains information from the physical implementation of the cryptosystem. The side-channel information can be in the form of power consumption, timing, electromagnetic emanation, or even sound [32]. The attacker uses correlation to retrieve the secret key from the leaked information. In the case of a Simple Power Analysis (SPA), the attacker inserts a temporal power variation and collects only one power trace to read the key bits directly. Correlation Power Analysis (CPA) uses the correlation between the power consumed by the hardware and the power estimates that the power model calculates. To calculate a correlation, the attacker runs the encryption with different plaintexts and collects the generated power traces. For a block cipher, such as AES, the attack is launched using a divide and conquer strategy, where each subkey byte can be retrieved one by one. The power model estimates the deterministic portion of the power consumption, using a Hamming distance model for CMOS technology. It computes the number of

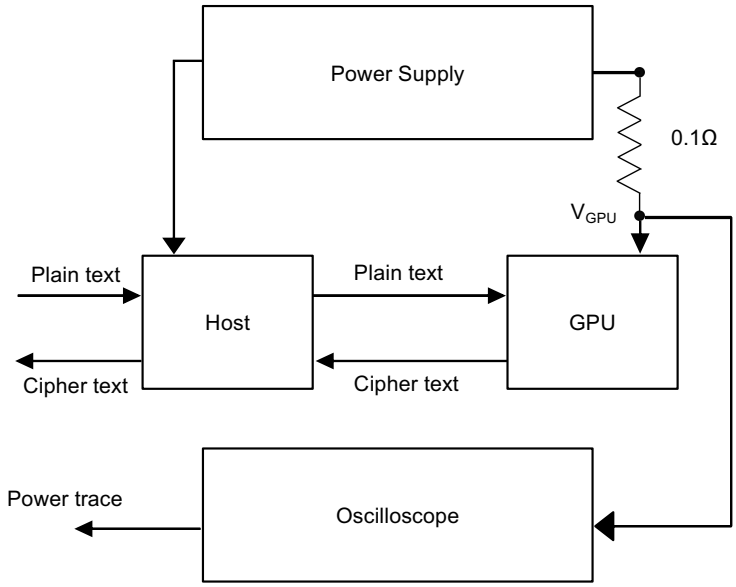


Figure 2.5: Overview of the experimental setup used to measure power.

logic changes (i.e., 0 to 1 or 1 to 0) based on the known plaintext and guessed subkey byte value. Once we have enough traces, we calculate the Pearson correlation coefficient for the collected traces and compute the cost for each guessed subkey value [33]. The correlation tends to be higher for the right subkey value. This requires significantly fewer iterations than a brute force attack (i.e., requiring orders of magnitude fewer traces).

2.3.2 Power Leakage Acquisition

Our AES kernel is developed in CUDA and tested on an NVIDIA Kepler family GPU Tesla K20c GPU, running Ubuntu 14.04.5 on an Intel Xeon CPU E5-1603 host. We use an Agilent MSOX4104A mixed signal oscilloscope to obtain power measurements. Power traces are captured by inserting a small resistor (0.1Ω) in series with the GPU card's external power supply line. Other parts of the GPU board are not touched to minimize invasiveness. The ATX power reads persistently $12V$ when we connected it to one end of the resistor. So we measure only at the other end, using the oscilloscope to find the voltage drop across the resistor. The voltage drop across the resistor is recorded when the code is executed on the GPU. The setup is shown in Figure 4.1.

As mentioned earlier in Section 2.2, the plaintext is sent to the GPU before encryption starts. Then it is used to generate the ciphertext on the GPU before the ciphertext is returned to the CPU. During encryption, the oscilloscope captures power traces at the maximum rate of $5GSa/s$

with $1GHz$ bandwidth, but the GPU has a maximum of the clock rate of $0.71GHz$. When the GPU is idle, it consumes a small amount of power. This power consumption is a function of time, but independent of the kernel. According to our measurements, this is similar to the supply voltage of $12V$. When the encryption starts, the GPU consumes more power, and the voltage drops but returns to an idle power level once the encryption finishes.

There are many challenges in capturing the traces from the GPU, including:

- Although the NVIDIA K20 has an onboard power sensor, the sensor provides much lower resolution than what is required for an SCA.
- The GPU has multiple converters that convert the $12V$ supply voltage into others as needed by the device. In addition to this onboard conversion, there are other things that contribute to the noise, including cooling fan, the PCIe interface, etc.
- The K20c GPU supports six clock speeds (from $324 - 758MHz$). However, only two memory clock speeds are supported (324 and $2600MHz$), one high speed and one power-saving speed. Although you can set any speed from these, there are hardware regulators that control the frequency.
- The GPU does not have a precise mechanism to provide a trigger signal to indicate the start and end of the encryption.

Chapter 3

Related Work

In this chapter, we provide an overview of the state-of-the-art of classical, as well as micro-architectural, side-channel attacks (SCAs) that have appeared in prior work. We focus on previous SCAs that explicitly exploit the programming or architectural features of a GPU. Since our work is based on power, finally, we wrap with by discussing the related work on power modeling. Publications are grouped and discussed to establish a comprehensive overview of these topics.

More than two decades ago, Kocher *et al.* generated a new branch of cryptographic research with the introduction of classical side-channel attacks. Before this time, it was assumed that the attackers only have *black box* access to the cryptographic devices [31, 30]. However, as these early papers concluded, this model was not strong enough to carry out a real-world attack, since the attacker needed to gain access to some internal state information from the side-channels. Typical side-channels leak information in the form of the timing of cryptographic devices, electromagnetic (EM) emanations and power consumption. In the following, we focus on how these attacks have been implemented to exploit timing and power side-channels.

3.1 Timing Side-Channel Attacks

Timing channels exploit the variations in the execution time of cryptographic processes to find information about the secret key. These variations occur for several reasons, including cache line collisions, non-constant execution flow, and memory bank conflicts. In his seminal paper, Kocher first introduced the term *timing attack*, where he analyzed the non-constant execution flow of RSA to expose the private key [30]. Later, Brumley and Boneh showed described a successful attack on a more realistic scenario on an OpenSSL based webserver[34]. Bernstein took advantage of the

timing variations due to a cache set access time differences to recover AES keys [35]. Crosby *et al.* worked on improving Kocher's work by reducing the jitter in the measurements and successfully recovered the key with a significantly lower number of traces [36]. Brumley further expanded upon his initial work to attack ECC implementations, showing that ECC can also suffer from similar timing leakage [37]. Timing attacks are also practical to recover plaintext sent over security protocols, such as TLS [38].

3.2 EM Side-Channel Attacks

EM emanation is a side-channel that has a long track record in the field of espionage and surveillance. In the public domain, in 1985, van Eck was the first one to demonstrate that EM emanations from computer monitors could be captured from a distance and analyzed to reconstruct the displayed text [39]. As a countermeasure to this, Kuhn invented unique fonts that reduced the EM leakage and made it difficult to reconstruct the text [40]. For integrated circuits (ICs) and CPUs, the first published works had limited scope - they required tiny antennas to be placed very close to the target IC [41, 42]. Some of the most successful attacks were also semi-invasive and required decapsulation of the chip packaging so that the signals of interest could be isolated. Removing these limitations, Agarwal *et al.* successfully attacked a CPU and cryptographic units from a distance using EM side-channel information [43].

3.3 Power Side-Channel Attacks

The premise of the power side-channel attack is to measure the power consumption of the circuit to find the correlation between the leaked data and the secret key. Since the original work on power side-channel attacks by Kocher *et al.*c [31], there has been vast interest in using power to recover secrets and encryption keys.

3.3.1 Simple Power Analysis

Simple Power Analysis (SPA) is a method for interpreting power traces collected during cryptographic operations. It is especially useful when the attacker has access to fewer traces. It is helpful during preliminary evaluation to determine when repetitive operations occur during the execution of an entire algorithm. However, it is difficult to use it to attack against AES. By visually inspecting the traces, it helps determine the structure of the algorithm (e.g., rounds in AES). Although

one is unable to deduce the secret key using this technique, it does present the capability to identify the cryptographic algorithm running on the device and enable more powerful attacks that specifically exploit any weakness of an algorithm to take place. Mangard presented a successful SPA attack against the AES key expansion [44], and Messerges *et al.* showed SPA attacks on Hamming weight and transition count leaks [45].

3.3.2 Differential Power Analysis

Differential Power Analysis (DPA) is a significantly more powerful attack. Using statistical analysis and error correction techniques, DPA extracts information correlated to the secret keys [31]. Although it requires no prior knowledge of the algorithm, unlike the SPA, the attacker needs to collect many traces to perform a successful attack. In a DPA attack, there are mainly two phases, data collection that involves collecting the device's power consumption during execution of cryptographic operations, and data analysis. Although the original DPA uses the distance of mean (DOM) test, later, another common DPA technique was shown that uses the correlation test [46], where a specific characteristic of the measurement, such as the mean or the maximum supply current in a clock cycle, is correlated with the predicted power. The correct key guess results in the highest correlation coefficient between the vector of the selected measurement and the vector of predicted power. This part is done in the data analysis phase.

AES encryption algorithm, as described in Section 2.2 converts plaintext into ciphertext using the round keys. At any point during the multiple rounds of AES, where the state (derived from plaintext) and the round key (derived from the key) enter a logic gate, the power consumption depends on both the key and the plaintext. By sampling this, a successful DPA can be mounted. In essence, Pramstaller *et al.* mentioned that in AES, the output of any operation is possible to be attacked [47]. The ShiftRows operation is a simple bit permutation, and as an operation, it is not suitable for DPA attacks. Any non-linear function increases the effectiveness of statistical attacks such as DPA [48, 49]. Hence, SubBytes is an option suitable for DPA. Similar to SubBytes, MixColumn is also a non-linear operation, but it is defined for 32 bits. So, to attack this operation, it would require 2^{32} key hypothesis, which is costly both in terms of time and memory compared to other AES operations. Finally, in AddRoundKey, the round key is XORed with the intermediate state. Although this operation is directly related to the round key, XOR is a low energy operation, since there is not even a carry ripple between bits. Hence, the power observability for this operation is low and can be drowned in the noise. Addresses to a memory (where SBox lookup tables are

kept) switch considerably more capacitance than the XOR operation of AddRoundKey, generating a more observable power signature. Hence, from the adversary's viewpoint, it is easier to observe the SubBytes operation than the AddRoundKey operation. Ambrose *et al.* performed DPA by exploiting table lookups for verifying their countermeasure against power analysis attack [50]. Han *et al.* showed DPA, multi-bit DPA and CPA attacks on AES both on a software and a hardware [51].

AES has eleven rounds comprising of a combination of these operations. If the first round is attacked, the key that is attacked is the secret key since the round keys are derived for the second to last round. In the last round, the final round key is attacked, and after disclosed, the secret key can be easily derived. In this attack, the variables are the ciphertext and the values that the registers contained in the previous round, which is dependent on the ciphertext and round key. Unlike first-round, here, only the ciphertext is needed. Authors have used the last round of AES in many literature [52, 53, 54].

3.3.3 Correlational Power Analysis

Brier et al. described a Correlational Power Analysis (CPA) that evolved from DPA, and provides a stronger correlational model that can exploit dependencies between the hypothetical intermediate values in power consumption traces [33]. They describe that CPA can be viewed as a multi-bit DPA method, capturing the linear relationship between the power consumption profile and the data.

Similar to using DPA, some knowledge of the structure of the underlying algorithm is required so the attacker can correlate the unknown key with some known information. The two most common CPA methods involve using either the Hamming Weight (HW) or the Hamming Distance (HD) [46, 55]. The HW is defined as the number of $1s$ in a binary number, and that value is the HW of that binary number. For example, for the binary number of 01010101, the HW is four. This model relies on the fundamental idea that the power drawn from a circuit changes while evaluating 0 as compared to 1. Given this hypothesis, we can tie power consumption to data values. Although Hamming Distance (HD) is even more effective, it requires prior knowledge of the data preceding or following the current data [56]. HW is a particular case of the HD model, where the first number from which the distance is to be calculated is all zeros.

The research community commonly uses the correlation coefficient as the statistical test for most power models and is used by an attacker to identify a key-value among all the candidates. For this purpose, the common choice is Pearson's correlation coefficient in conjunction with the Hamming weight or distance model [46]. The work presented in this thesis is also based on Pearson's

correlation coefficient and adopts a Hamming Weight (HW) model, as proposed by Mangard [56].

3.3.4 High-Order Differential Power Analysis

High-Order Differential Power Analysis is an advanced form of DPA. The most common way to thwart a DPA attack is by random masking, where intermediate operations are performed probabilistically to reduce statistical correlation [57]. However, masking cannot thwart the attack if the attacker correlates power consumption more than once during a computation. Messerges proposed a second-order, or more generally, a higher-order power analysis attack that combines multiple leakage signals [55]. It has been shown that second-order DPA attacks can overcome some countermeasures theoretically, but will sometimes break them in practice [58, 59, 60]. These attacks are usually more complex and delicate to perform because they usually require detailed knowledge of the platform architecture running the cryptographic algorithm. This kind of information is challenging to obtain for a GPU.

3.3.5 Countermeasures

Along with successful power analysis attacks on a variety of hardware platforms, many countermeasures have also been reported in the literature. Depending on the level of abstraction, countermeasures can be categorized into two broad categories: 1) algorithmic countermeasures and 2) circuit-level countermeasures [61]. In both categories, we can apply: 1) masking and 2) elimination. *Masking* tries to secure the system by minimizing the correlation between the intermediate variables and the secret information. *Elimination* tries to remove any power variations measurable by the attacker.

Algorithmic Countermeasures: At an algorithmic level, a countermeasure involves introducing randomness into the power measurements. To help hide the side-channel signal, random masking and operation shuffling are two conventional approaches [62, 63]. In random masking, a boolean or arithmetic operation is performed on every intermediate value using a random value, such that all the intermediate values become independent of the data. Eventually, the mask needs to be removed. Alternatively, operation shuffling can be used to distribute the signal containing the secret information into n time points. This reduces the corresponding correlation and SNR for power analysis attacks to $1/n$ [62]. Although these two approaches are practical for first-order DPA, they may still be ineffective for higher-order DPA. For elimination, an algorithmic approach can be implemented using one of the following approaches: 1) developing a program that is void

of secret-dependent branches, making it nonvulnerable to SPA, 2) balancing hardware logic and state transitions to protect against DPA and CPA [64]. Although they are effective, they incur poor performance and consume additional power, as well as require significant design effort. So it is crucial to strike a balance between designing the obfuscation approach, performance, and power.

Circuit-level Countermeasures: Common ways to inject randomness at a circuit level is inserting random delays, randomly changing the clock period, dynamically adjusting the threshold voltage and performing dynamic voltage and frequency scaling (DVFS) [65, 66, 67]. This breaks the trace alignment effectively, skewing the power leakage in time. More recent work manages voltage regulators present in the power delivery network to cut the power consumption monitored by the attacker, resulting in little power and area overhead [68, 69]. Power variations can be eliminated at the circuit-level. Dual-Rail Precharge (DRP) is the most popular technique for implementing constant power consumption in each clock cycle [61, 66, 64]. However, the hardware implementation of DRP circuits comes with two significant issues. First, DRP circuits are at least twice as large as their equivalent standard CMOS designs and consume more power. Second, when low-level electrical effects are considered, the constant-power argument, based on HW or HD does not hold [70].

3.4 Other Types of Side-Channel Attacks

Other types of side-channel leakage can also comprise encrypted systems. In 2004, Agrawal *et al.* reported a successful attack based on acoustic emanations from a computer keyboard, telephone, and ATM keypad [71, 72]. They showed that the key being pressed could be identified by differentiating the sound produced by pressing different keys. Acoustic emanations generated by computers and additive manufacturing systems (e.g., 3D printers) also allow attackers to learn about the encryption key or the objects being printed without requiring access to the original design [73, 74, 75]. Kuhn found that it is possible to retrieve the information on a CRT or LCD by using the indirectly reflected optical emanations [76, 77].

3.5 Side-Channel Attack on GPUs

There have been several research studies of side-channel vulnerabilities present on GPUs. Initial studies were focused on leveraging software exploits in the GPU programming model to purloin the confidential information. Pietro *et al.* demonstrated that, due to memory isolation issues, GPUs are susceptible to information leakage [78]. Miele *et al.* identified that an attacker could

overwrite a function pointer, resulting in a buffer overflow that can corrupt classified data or forcefully change the execution flow [79]. Other studies [80, 81, 82] found that information leakage is present due to memory contents that remained in GPU memory after a recently terminated GPU kernel process completes. The premise in this class of attacks is that GPU drivers do not automatically clear or delete data residing in different types of GPU memories, including shared memory, global memory, local memory, and registers after deallocation. This leads to the risk of leaking confidential data through reallocation of the same memory space to an attacker's kernel. However, this class of vulnerability can be easily resolved by clearing the memory before it gets reallocated to the new kernel. Although the prior work did not exploit GPU-based cryptographic implementations, they definitely pose a potential threat. Side-channel attacks such as timing, power, and EM analysis attacks are possible threats to GPU-based AES implementations. Although the security of GPU-based AES is starting to be explored, the prior research has already identified side-channel leakage present in the context of a GPU. Luo *et al.* designed a power attack by exploiting the behavior of the CUDA scheduler. They recovered all of the key bytes of the last round of AES using a Correlation Power Analysis on an NVIDIA GPU [83]. Later, Jiang *et al.* developed a timing attack to recover the encryption keys using a Correlation Timing Analysis [84]. Later, they mounted another timing attack in a similar setting using Differential Timing Analysis [85].

Equipped with this attack model, Kadam *et al.* found that the determinism in the coalescing mechanism led to security vulnerabilities. They proposed a solution to this attack by Randomized Coalescing (i.e., RCoal), which randomizes both the granularity of the warps across groups and the requests in each group. This provides an improved level of obfuscation, albeit with some performance overhead introduced [86]. Recently, Wang *et al.* described a Profiling-based Side-Channel Attack (pSCA) to overcome RCoal and recover the AES key in a reasonable time [87].

Karimi *et al.* demonstrated a cache timing attack and successfully recovered the full encryption key on a Qualcomm Snapdragon mobile GPU [88]. Along with power and timing, Gao *et al.* was the first one to show a potential weakness of both T-table based and bitsliced AES implementation to EM analysis [89, 90].

3.6 Power Modeling

Power is a pivotal design constraint for any computing device, including CPUs, GPUs, and FPGAs. The power consumption of these components can be obtained either through a direct or indirect measurement method. A direct measurement can be made using an internal or external

voltage/current sensor that periodically collects samples to estimate the average power consumed during program execution. The total energy consumed is calculated as the integral of the power consumed over the execution time. External power meters are usually connected between the power supply unit and the component under investigation. Although they are considered the most accurate source to collect power consumption data, they are not suitable for comprehensive power profiling and often produce inaccurate readings, especially in an HPC environment [91]. Alternatively, internal power meters can be built from built-in sensors, providing for convenient access to power data through a software interface. The NVIDIA Kepler K20 GPUs are equipped with such sensors. Burtscher *et al.* described an energy measurement methodology for GPU kernels running on a K20, using the on-board built-in power sensor [92]. However, similar sensor interfaces are not available on all GPUs, and there are limits on which the power sources can be measured.

Indirect measurements include modeling and simulation techniques for power consumption. Over the last couple of decades, researchers have delivered a number of power models for CPUs, including Wattch [93] and McPAT [94]. These tools enabled power modeling research to thrive in the computer architecture community. Later, when GPUs became increasingly popular, researchers started building power models for GPUs. The main challenge in building a power model for a GPU is the lack of micro-architectural and implementation details of the hardware available to the research community.

Initial power models for a GPU considered modeling power at a fixed frequency/voltage configuration, but did not consider the effects of DVFS. While Nagasaka *et al.* proposed a model for the Tesla GPU (GTX285) and achieved a 4.7% average prediction error, their approach cannot be used for more recent GPUs, such as GPUs from the Fermi generation [95]. Hong *et al.* proposed a model for the Tesla GPU (GTX280) that used offline PTX analysis to obtain highly accurate power predictions but at the cost of being specific for that class of GPUs [96]. So clearly, while these approaches are highly accurate, they cannot be generic to make reliable predictions across GPU architectures, or even for the same GPU family with a different microarchitecture.

With the help of artificial neural networks, Song *et al.* showed how we could predict GPU power consumption. They achieved higher prediction accuracy than other regression-based models [97]. While highly accurate, the downside of this approach was that the model lacks any semblance to the underlying hardware, making it hard to extract the physical meaning from a prediction. Leng *et al.* integrated Hong's power model inside the GPGPU-Sim [98] simulator and delivered GPUWattch [99]. Removing the limitation of the generality of Hong's model, GPUWattch supports both NVIDIA's Tesla and Fermi GPU architectures and estimates the cycle-level GPU

power consumption. It assumes that the power consumption of a GPU scales linearly with its frequency, but this assumption does not always hold. As a further improvement, Nath *et al.* created a performance model for DVFS in GPGPUSim [100]. However, extending this power model requires adding logic to the GPU scoreboard, which is challenging to replicate on real hardware. Lucas *et al.* released GPUSimPow, a framework for modeling GPU power consumption [101]. However, although these architectural simulators consider activity factors for different GPU components, such as integer or floating-point instructions, register reads/writes, memory accesses to calculate the power consumption, they ignore the data values processed by the datapath. One exception to this is the Wattch power simulator, which contains a `DYNAMIC_AF` option that includes the activity factor for some internal buses and memories, but ALU power is modeled as a simple constant [93].

3.7 GPU-based AES

Harrison *et al.* were the first to present a CPU competitive implementation of AES on GPU [102]. Later Manasvaki implemented AES in CUDA using the table lookup approach and stored pre-computed T-boxes in GPU constant memory [103]. Utilizing the highly efficient cached memory present in the context of a GPU, three of the four operations of AES (SubBytes, ShiftRows, and MixColumns) are merged into the table lookups [104]. Later Harrison *et al.* experimented with all available types of on-chip memory (shared memory, constant memory, and texture memory) to store lookup tables and concluded that using shared memory lead to the best performance [105]. Di Biagio *et al.*, Chonglei Mei *et al.* similarly implemented AES and validated this finding [106, 107]. AES has inherent parallelism in each round. To exploit this parallelism, Biagio *et al.* proposed a counter mode AES (AES-CTR) on NVIDIA GPUs that distributes the workload across Streaming Multiprocessors (four 32 word blocks are mapped to four scalar processors) [106]. Here, four T-tables are loaded into shared memory for faster access time for lookups. While efficient, implementations based on lookup tables are a target for SCA, and several practical attacks against the T-table based AES implementation of OpenSSL have been published [35, 108, 109]. The root of the problem stems from the address-dependent table accesses [110]. While this can be resolved with computing the AES S-Box and executing in constant execution time, the performance penalties of straightforward implementations would be considerable. To avoid that, a bitsliced method has been proposed that eliminates all tables and is based only on logic operations [111]. In bitsliced AES, with n identical input blocks, all the bits in a register will contain the same value, either all 0s or all 1s. This is relatively easy to find using a power or EM analysis attack. With the existence of vector-style

registers in GPUs, this vulnerability is, even more, concerning on GPUs compared to CPUs, because higher register width makes the leakage even more distinguishable between the states of all 0s and all 1s. Apart from the performance, researchers are also interested in optimizing the portability of parallel implementations across GPUs [112]. Guo *et al.* implemented a fast AES algorithm using the AES-NI extended instruction set based on CUDA [113]. Also, Fei *et al.* pointed out that the actual workload characteristics of the block cipher are an essential element to consider in a server environment [114]. Her work suggested six methods to implement AES in heterogeneous servers containing both CPUs and GPUs. However, the implementation does not exploit the fine-grained parallelism in block ciphers for faster response time in a server environment [115]. To overcome these vulnerabilities and retain high throughput, we implement OpenSSL-based table-based AES on contemporary NVIDIA GPUs and concentrate on attacking the last round to recover the secret key. Since the last round has no MixColumns operation, only one out of four bytes is kept after the T-table lookup. This is equivalent to an SBox lookup operation and ShiftRow. AddRoundKey is then performed on the four remaining bytes. This non-linear operation leaks power, which we correlate with our hypothetical HW-based power model, GIPSim, as described in the next chapter.

3.8 Summary

Now that GPUs are an integral device in many computing platforms, ensuring the security of these devices is of great concern. Given the different classes of side-channels present on these devices and the scarcity of GPU power models that are data-dependent, we need better analysis tools. To begin to characterize power leakage, we have designed a data-dependent power simulation framework to characterize power leakage on a GPU. Our model considers the HD of data values used during program execution. For obfuscation, manually checking for higher-order masked implementations is a more complicated and error-prone task. As a consequence, many schemes are later shown to be insecure [116, 117] and can be broken [118, 119]. So we address this issue by using our model to design obfuscation approaches with higher-order masking to thwart power-based SCAs.

Chapter 4

GIPSIM framework

Current cryptographic algorithms are designed to run on an integrated circuit (IC) chip that is manufactured of many logic gates that are made up of Complementary Metal-Oxide Semiconductor (CMOS) transistors. Transistors consume electric current from the IC's power supply to switch on and off. The logic function of a gate is represented based on the state of the transistors. The premise of a power analysis attack is that the power consumed on the chip varies with the aggregated switching of each gate. So the first step in any power analysis attack is to acquire one or more traces from the targeted device. A power trace is a digital documentation of the sampling sequence of instantaneous power consumption of the device over time, especially when the device is running cryptographic operations.

Sometimes, to reduce noise and increase time consistency, signal processing is performed on the collected power traces. Noise present in the power traces can be reduced with appropriate filters. Commonly, a trigger signal [48] is used, which indicates the beginning of the cryptographic operations we are trying to measure. This trigger signal is used to prompt the measuring device to start collecting power samples.

Usually, cryptographic devices require a constant voltage supply, denoted by V_{dd}. In our case, the ATX power supply is $12V$. From V_{dd}, the current that the chip draws is a time-varying value, $I_{dd}(t)A$. Hence the instantaneous power of the whole chip is $P(t) = V_{dd} \times I_{dd}(t)$, measured in watts.

An oscilloscope is commonly used as a sampling device to measure voltages. As shown in Figure 4.1, after a small resistor R is inserted in series with the supply power, the voltage drop at V_{GPU} is measured across the probes of the oscilloscope. Since we only measure the voltage at V_{GPU} to get the voltage drop across the GPU, this voltage drop is inversely proportional to the power

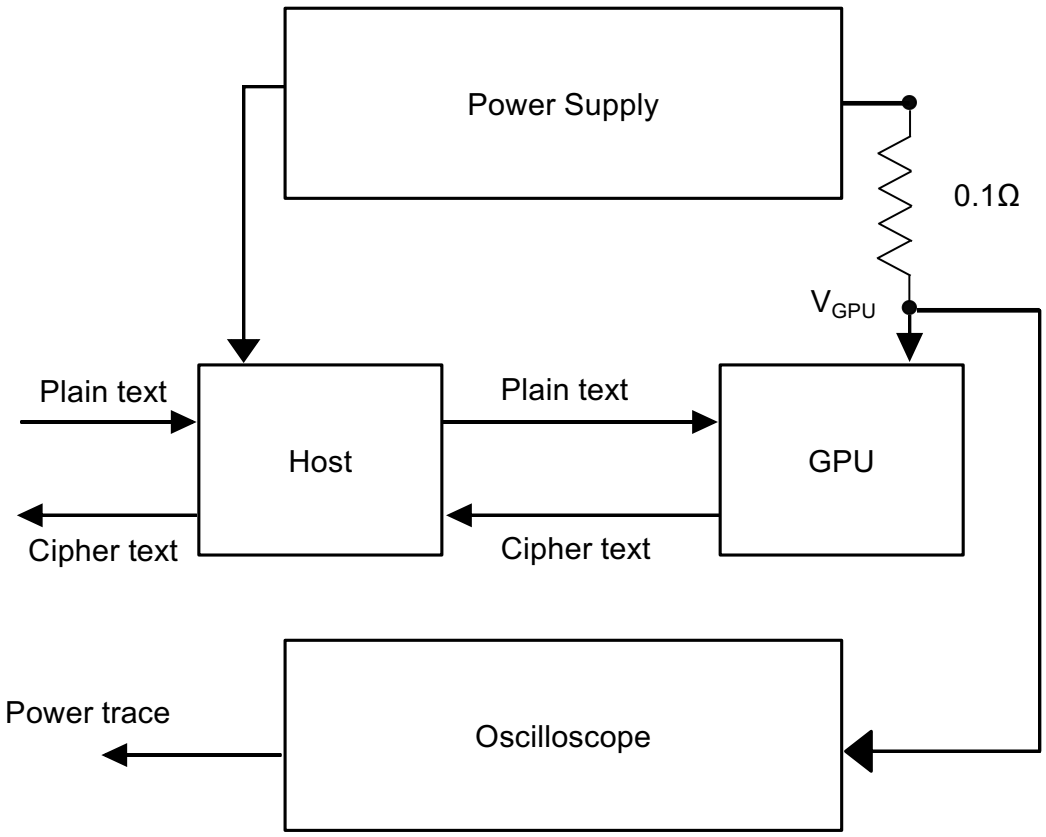


Figure 4.1: Experimental setup for power acquisition.

consumption. Figure 4.2 shows a sample power trace.

The power consumption of a CMOS circuit has two components, static and dynamic power. The static power depends on voltage source and device characteristics that lead to leakage and temperature variations. Static power typically does not vary much, and in our work, we have considered it as noise in our power measurements.

The dynamic power has two components, as well. The first includes the power needed to charge and discharge the circuit capacitance. The second component of the power is the short circuit formed by the PMOS and NMOS transistors to change the output voltage. The dynamic power consumption is linearly related to changes in bit values of the intermediate state changes captured by modeling the HD transitions between the former state and the new state of the circuit.

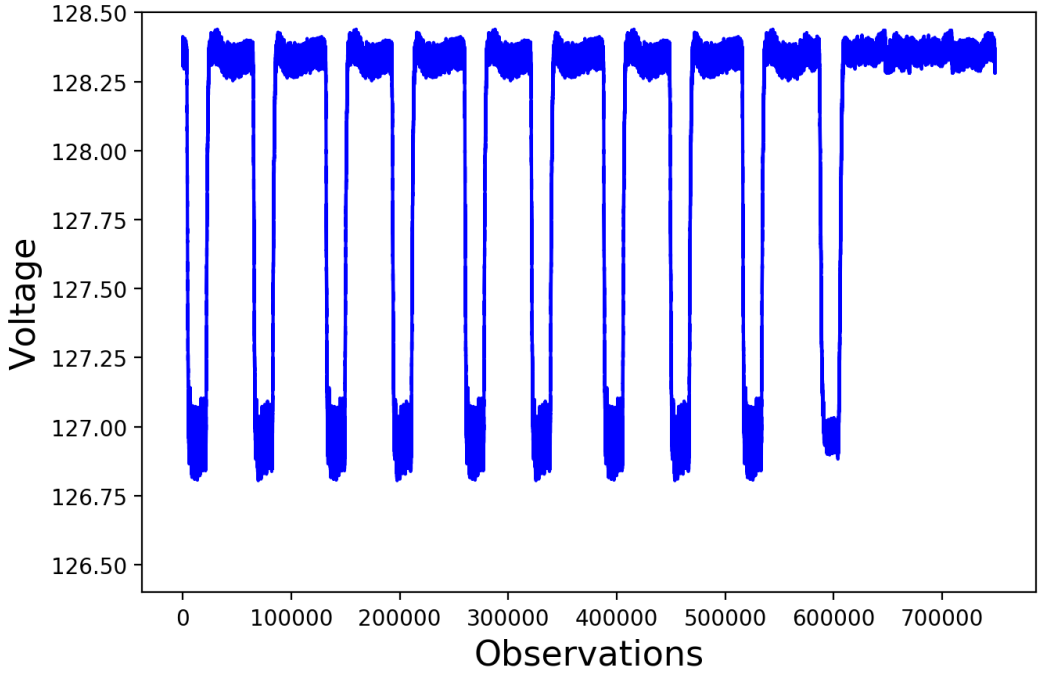


Figure 4.2: A sample trace of an AES execution.

4.1 Motivation

As discussed in the related work section, the existing GPU architectural simulators consider activity factors for various GPU components and use them to find the total power consumption of the GPU. While they include activities such as integer and floating-point instruction execution, register reads and writes, memory accesses, none of the simulators consider how often the datapath data lines switch between 0 and 1. Although Wattch contains a `DYNAMIC_AF` option in its source code that includes the activity factor for some internal buses and memories, ALU power is modeled as a simple constant [93]. The source code for Wattch includes the comment: ”FIXME: ALU power is a simple constant, it would be better to include bit AFs and have different numbers for different types of operations.”

For a CPU power simulator where control logic dominates the datapath, such a simple model may still be acceptable, but for a GPU, a large part of a GPU’s power and the area is dedicated to execution units and register files. In the context of an SCA, a data-dependent framework is missing to enable security researchers to reason about side-channel leakage present in the context of

a GPU execution-driven simulator. So we deliver the *GIPSim framework* to help researchers capture detailed power estimates and use the information to obfuscate power by adding noise to the runtime power profile. This, in turn, can reduce the vulnerability present during data encryption/decryption. We demonstrate that we can model data-dependent power dissipation, showing how the hamming distance of the data values (i.e., encryption keys) used during the execution of AES encryption. To the best of our knowledge, GIPSim is the first simulation environment that can be used for evaluating power side-channel resiliency and help secure future systems.

4.2 Building a Baseline Model

To characterize SASS-code level data-dependent execution, GIPSim models data-dependent power dissipation. We base our model off the power modeling work presented by Tiwari *et al.* [120]. Our power model estimates the deterministic part of the power consumption, for example, the Hamming distance (HD), which computes the number of logic changes (i.e., 0-to-1 or 1-to-0) in the datapath. We start with a baseline model that predicts the power consumption of instruction for a specific data value. We then collect a rich corpus of power traces for each opcode. Instructions are executed with different input data values, producing a range of HDs, which are computed relative to an input value of zero. For each opcode-HD pair, we capture $1,000$ instances of the same instruction opcode, seeded with the same input data (hence, same HD), and then collect the average power used over the $1,000$ traces. We label this power value the *base cost* for that opcode-HD pair. We observe a low variance (0.039) for the average power consumption measured across a full run of the kernel. Given this low value, the power consumed is very stable. We repeat this measurement for different HD values. To reduce any noise introduced by the frequency-voltage regulator of the GPU, we perform calibration to avoid excessive noise in our measurements. We find that as we increase HD in the plaintext data values, the power consumption linearly increases as well, which indicates that leakage exists in the power consumption. We also find that the power consumed for two instances of the same instruction to be highly dependent upon the instruction that follows them. This follows the same conclusion that Tiwari *et al.* reported in prior work [120]. So along with base cost, we also measure the power consumed due to switching between instructions. We add this to our baseline model, predicting power for the inter-instruction overhead. To capture this factor, we select a pair of instructions and run them through our model in the same way we did for the single opcode-HD pair. In this case, we choose two instruction pairs and collect the average power consumption for that pair across the power traces.

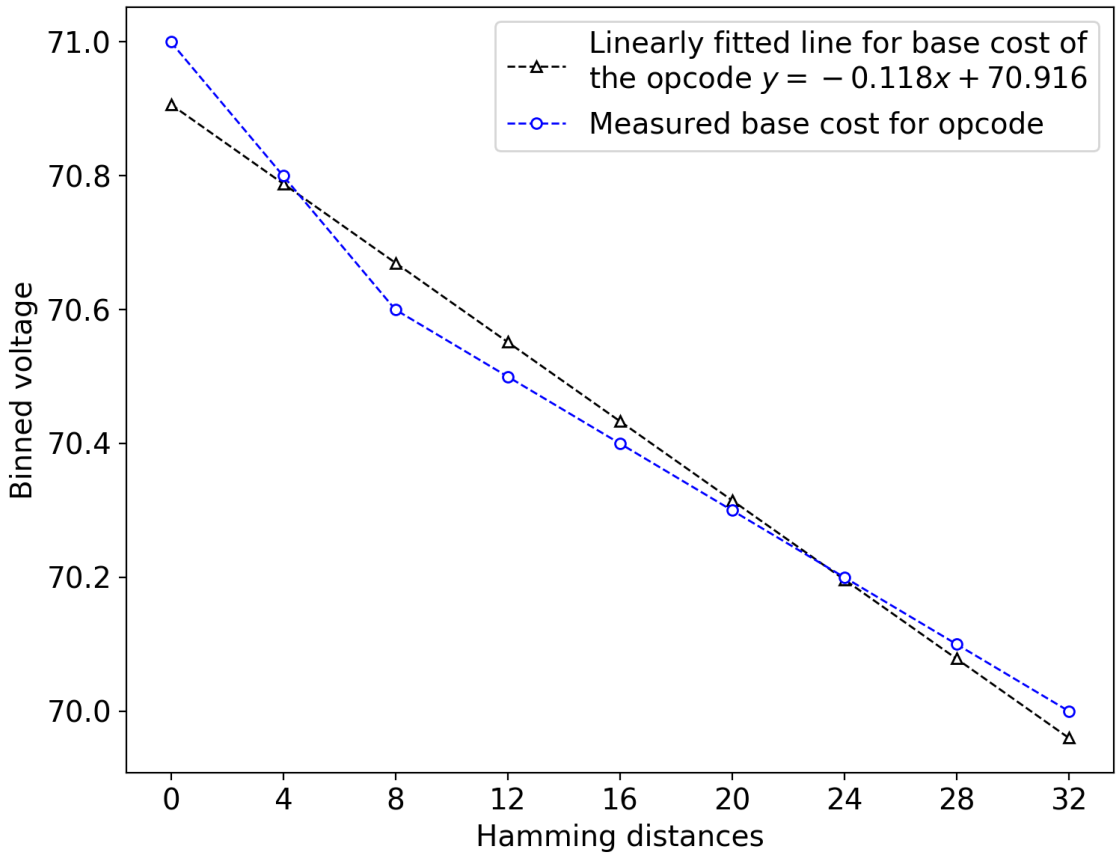


Figure 4.3: Baseline model for IADD instructions with different HDs.

For each instruction opcode, as the HD increases, we see a linear increase in power consumption. When we plot the power values with respect to the HD for an opcode, we obtain a straight line. Based on the slope of this line, the power consumption for any opcode i can be computed as:

$$P_i = P_{basecost} + slope_i \times HD(i) + p_{overhead} + p_{extra} \tag{4.1}$$

where P_i is the total power consumption for a particular instruction i, $P_{basecost}$ is the amount of power required for running that instruction when the data value is 0, and $slope_i$ is the slope of the line that tracks the power consumption for different HDs, for instruction i. $HD(i)$ is the HD of the data values for instruction i. While $p_{overhead}$ is the cost for switching from the previous instruction, p_{extra} includes all other unrelated power consumption. Figure 4.3 shows the base cost of $IADD$ instruction for HDs 0 to 32, which is shown on X-axis. The Y-axis shows the binned voltage, which is the value we receive from the oscilloscope. These values are normalized between 0-255 and the

Floating Point Instructions	
FFMA	FP32 Fused Multiply Add
FADD	FP32 Add
FMUL	FP32 Multiply
Integer Instructions	
IMAD	Integer Multiply Add
IMUL	Integer Multiply
IADD	Integer Add
SHR	Integer Shift Right
SHL	Integer Shift Left
ISETP	Integer Set Predicate
Movement Instructions	
MOV	Move
Compute Load/Store Instructions	
LDC	Load from Constant
LD	Load from Memory
LDL	Load from Local Memory
LDS	Load from Shared Memory
ST	Store to Memory
STL	Store to Local Memory
STS	Stored to Shared Memory
Control Instructions	
BRA	Branch to Relative Address
JMP	Jump to Absolute Address
RET	Return to Call
Miscellaneous Instructions	
S2R	Special Register to Register

Table 4.1: List of opcodes supported by the GIPSim framework.

relationship between this and the voltage is completely linear. The linearly fitted line in the figure represents the $slope$ in the Equation 4.1. Based on this behavior, we can interpolate the power values for the data values we have not measured yet for both base cost and inter-instruction overhead.

Table 4.1 shows the complete list of opcodes GIPSim currently supports. For opcodes that do not have a data value operand associated with their semantics *, e.g., a RET instruction*, we only consider the $P_{basecost}$.

Table 4.2: Relative power consumption for measured and modeled power, while change program inputs. Each benchmark has 3 different inputs.

Applications	Input	Relative measured power consumption	Relative modeled power consumption
MatrixMul	1 : 2	0.6274	0.6285
	2 : 3	0.6134	0.6147
	1 : 3	0.3843	0.3869
VecAdd	1 : 2	0.7286	0.7244
	2 : 3	0.9203	0.9193
	1 : 3	0.6705	0.6654
Histogram	1 : 2	0.6505	0.6487
	2 : 3	0.4548	0.4584
	1 : 3	0.2958	0.2963
BlackScholes	1 : 2	0.3724	0.3711
	2 : 3	0.7182	0.7228
	1 : 3	0.2657	0.2631

4.3 Model Accuracy

To calculate the prediction accuracy of GIPSim, we use this power model to predict power consumption for selected kernels from the NVIDIA CUDA SDK while executing the kernels with different input files. We compared this with the power measured from the GPU while running the same applications using specific input files, using the power measurement method described in Section IID. We selected these applications to give us a range of different behaviors on a GPU:

1. *MatrixMul* shows the features of NVIDIA GPU architecture, and it is memory bound.

2. As a simple function, *VecAdd* spends most of the time performing ALU operations and demonstrates the raw computing capabilities of the GPU.

3. *Histogram* includes several divergent branches and is a representative of how divergence impacts the power model.

4. *BlackScholes* is a massively parallel application that computes the price of European market options. This kernel calculates partial differentials. Memory is accessed frequently in BlackScholes.

Although the amount of power consumption predicted by GIPSim does not match the measured power from the device, the relative power consumption between any two runs of the same application, run with the same input, tracks very well. This demonstrates that we can reliably predict

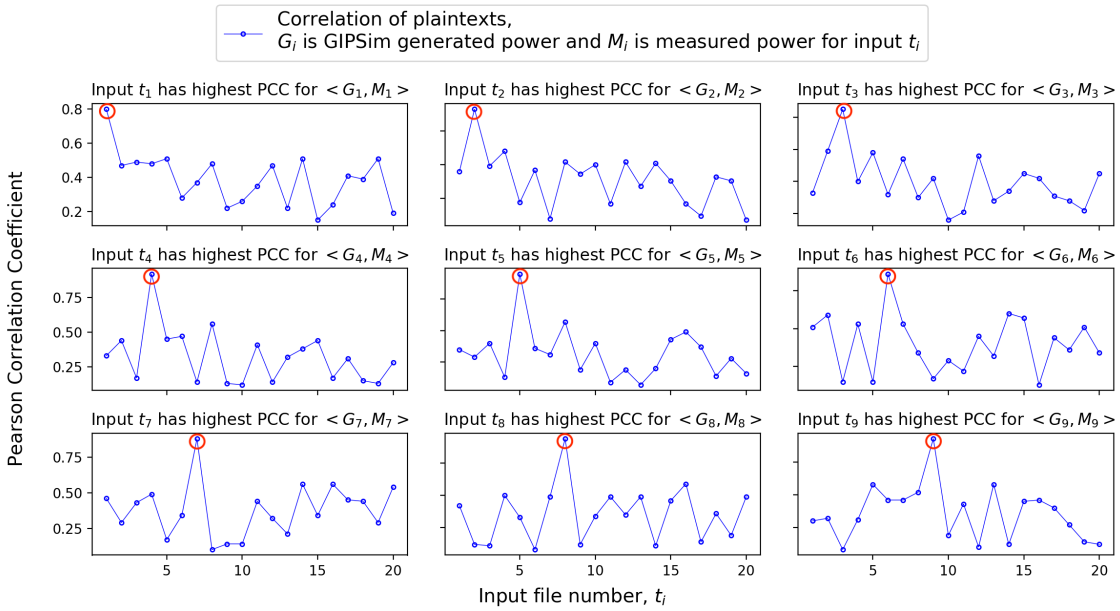


Figure 4.4: The PCC value comparing the modeled power and measured power.

power usage with GIPSim. When we compute the correlation between the modeled (i.e., GIPSim) and measured power on the K20c GPU across randomly selected inputs, we achieve a correlation of 0.99 (near-perfect correlation). We present three cases for each application in Table 4.2, representing three different input files. We compute and measure the relative changes in power when changing the program input. As we can see, the relative power tracks nicely. GIPSim predicts almost the same relative power consumption as measured across different program inputs.

4.4 Model Correlation to AES

The Pearson Correlation Coefficient (PCC) is a measure of the linear correlation between two variables. PCC values can range between +1 and -1, where 1 indicates a highly positive linear correlation, 0 indicates no correlation, and -1 indicates a highly negative linear correlation. We use PCC to determine the degree of whether the power predicted by GIPSim can be correlated with the power measured from the device. The PCC of variables X and Y, $\rho_{X,Y}$ is defined as:

$$\rho_{X,Y} = \frac{cov_{X,Y}}{\sigma_X \sigma_Y} \tag{4.2}$$

where cov is the covariance, and σ_X and σ_Y are the standard deviations of variables X and Y.

To evaluate PCC for GIPSim, we generate ten random input text files $t_1, ..., t_{20}$ as inputs to AES-128 and run them through GIPSim to measure the predicted power profile $G_1, \ldots, G_{20}$. We also run the same input text files on a real device and measure power $M_1, \ldots, M_{20}$. Next, we take each M_j and compute the correlation with all G_is. The goal is to find the pair $< G_i, M_j >$ (where i and j are the input file numbers $t_1, \ldots, t_{20}$) which exhibits the highest correlation. We expect the pair of traces (generated by GIPSim and measured) that used the same input text values to produce the highest correlation. The PCC values for each pair $< G_i, M_j >$ for i=j are always greater than 0.8, suggesting that GIPSim's predictions and the actual measurements are highly correlated. In Figure 4.4, we show nine such cases for $< G_i, M_j >$. We found that all top twenty pairs $< G_i, M_j >$ with i=j produces around 40% higher correlation that the average correlation for all, demonstrating the strong predictability of GIPSim.

One challenge here is that the measured power consumption on the real device includes many more data points than the model. To handle this issue with our measurements, we produce a histogram of the power numbers measured from the device. The histogram consists of several bins, where the number is equal to the number of instructions in the kernel (or the number of power data points, as predicted by the model). Each bin represents the same range of power values. We compute the average power for each bin, sort the values, and correlate the sorted profile with the modeled power, as mentioned above. We recognize that this approach assumes each instruction takes the same number of cycles, but as we will see, our assumption is not far off.

4.5 The Effect of DVFS on GIPSim

Dynamic voltage and frequency scaling (DVFS) is one of the power management strategies employed to save energy on CPUs and GPUs. Using this strategy, during program execution, the processor voltage/frequency can be modified to save energy or improve performance. For GIPSim to accurately predict the power consumption of applications running on a GPU, we need to understand how power consumption will be impacted when changing the GPU operating frequency/voltage. Note that DVFS generally specifies operating voltage and frequency pairs. It has already been shown that when DVFS is used, GPU resource utilization varies for applications [121, 122]. We have previously demonstrated that our GIPsim model can be used to predict the power consumption for a fixed frequency (and voltage) configuration. The question is whether this holds when DVFS is present. We need to understand how GIPSim predictions change with DVFS.

Memory Freq. (MHz)	GPU Core Freq. (MHz)	GPU Core Voltage State	GPU Core Voltage (mV)
2600	758	V5	987.5 - 1112.5
	705	V4	950 - 1062.5
	666	V3	925 - 1050
	640	V2	912.5 - 1025
	614	V1	900 - 1000
324	324	V0	875 - 875

Table 4.3: The K20c supported memory and core frequencies.

The total power consumption of a CMOS circuit consists of static and dynamic power dissipation. Dynamic power dissipated when transistors switch states.

$$P_{dynamic} = aCV^2f \tag{4.3}$$

Equation 4.3 shows the general formula for computing dynamic power. The power dissipation is dependent on the following factors: the average utilization ratio (a), the total capacitance (C), the chip supply voltage (V) and the operating frequency (f) [123]. DVFS can adjust the runtime supply voltage and the frequency, which results in a change in dynamic power, and as a result, change the total power consumption.

GPU devices typically have two pairs of voltage and frequency values: 1) the core voltage and frequency pair, and 2) the memory voltage and frequency pair. The core voltage is the supply voltage of the GPU SMs, and the memory voltage refers to the supply voltage of the DRAM. In terms of the frequency values, the core frequency governs the speed of the SM's executions, and the memory frequency governs the speed of the DRAM I/O throughput.

As mentioned earlier, our experimental platform includes an NVIDIA Tesla K20c Kepler GPU card. The GPU supports DVFS for both the core and memory voltages/frequencies. The support DVFS values are shown in Table 4.3. Note that the core and memory speeds are set as a pair, otherwise under-voltage or over-frequency conditions can occur.

To control the clock frequencies and voltages we used `nvidia-smi` and Nvidiux [124, 125]. The `nvidia-smi` utility uses the NVIDIA Management Library (NVML) for management and control. This enables the user to change the GPU core frequencies within the values allowed by the GPU, as specified in Table 4.3 [126]. For changing the voltage, we used Nvidiux, which is a graphical Python-based tool for Linux to provide finer control over the voltage and frequency of NVIDIA GPUs. Note that using these tools may void the warranty of the GPU. A step in the

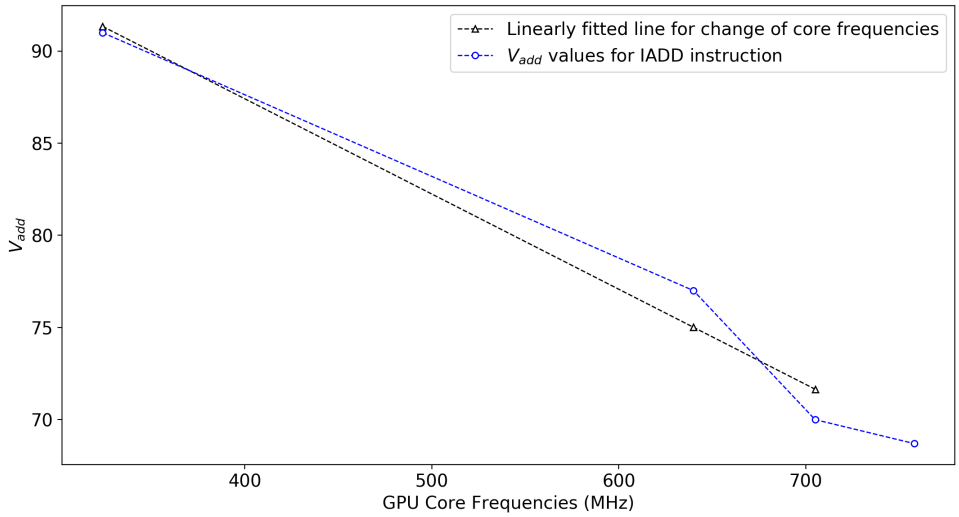


Figure 4.5: The binned voltage consumption, V_{add} in varied GPU clock frequency settings.

installation requires changing xorg.conf which may break booting to Linux and can potentially cause damage to the system.

In our next experiments, we remeasured the power consumption values used to build GIPsim (as shown in Figure 4.3) while varying the frequency (as described in Figure 4.5). When we modified the GPU core frequency, the memory frequency also changes. For most cases, as we increased the clock frequency, the voltage drop decreases (or power consumption increases). During execution, the binned voltage consumption remains the same over time for all frequencies except 758 MHz, where the voltage values increase (i.e., current or power consumption drops), though after 60 seconds, the frequency changes to 705 MHz. When the GPU operates at its peak frequency for some time (e.g., a minute), the internal temperature management system automatically lowers the operating frequency, keeping the device from overheating. Since the K20c can not maintain the maximum rate for an extended period, we should consider this fact in the design of GIPsim.

Voltage states are labeled V1-V5 in Table 4.3. The default voltage for the GPU is in a range as marked with V_4 run with the default frequency of 705 MHz. To find the effect of changing the voltage using GIPSim, we use the Nvidiux tool and ran the same instruction add kernel (iadd) while changing the frequency. We use voltage values of 900 mV, 925 mV, and 987.5 mV, representing the lowest values for the V1, V3, and V5 states, though maintains the default clock frequency. When we decreased the frequency, the overall voltage consumption also decreases with higher core voltage. In other words, since the voltage drop is inversely proportional to current or power, more power is

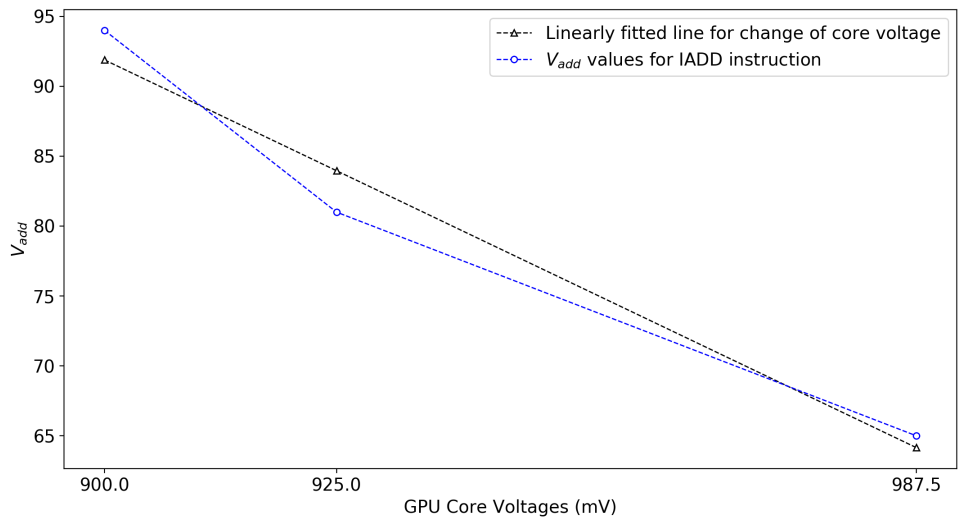


Figure 4.6: The binned voltage consumption, V_{add} in varied GPU core voltage settings.

drawn when running at higher core voltages, as shown in Figure 4.6.

While using the K20c at its default settings for fixed frequency and voltage, we can continue to use the GIPSim predictions, as mentioned earlier in the baseline model. If DVFS is active, as we have seen in Figures 4.6 and 4.5, the base cost will change. With our limited measurements, we see that there is a trend that can be observed with varying voltages and frequencies. Current data shows that with varying core voltage, we need to use this equation to find the base cost for running the opcode:

$$v_{add} = -0.315 \times v_{core} + 375.25 \tag{4.4}$$

where v is the voltage used. On the other hand, for variable core frequencies/memory frequencies pair, as shown in Figure 4.5, we need to use this equation to find the base cost for running the opcode:

$$f_{add} = -0.052 \times f_{core} + 107.15 \tag{4.5}$$

in the same way we did for varying core voltage. Since 758 MHz drops down to 705 MHz after around 60sec, for calculating V_{add} we can consider it as 705 MHz. To get a better idea of this, we need to perform more experiments and is detailed in Section 6.

Chapter 5

Obfuscation Approaches

In this chapter, we describe how GIPSim can be used in practice for designing countermeasures to thwart SCAs. Prior research on obfuscation can be categorized as either using hiding or masking. Hiding provides a strategy to hide the power consumption of key-dependent cryptographic computations by effectively decreasing the signal-to-noise ratio (SNR) of the attack surface.

5.1 Hiding

Hiding can be accomplished in two ways. One, equalization where the goal is to make the total power consumption the same all the time. Another, randomization where we attempt to randomize the power consumption by continually changing the execution order, or generating noise directly.

5.1.1 Randomization

Randomization tries to increase the amount of noise present in the power side-channel. It can be done by running some other computations in parallel during encryption. As mentioned earlier, one common way to provide obfuscation is to hide the actual signal. This can be done by running some other computations in parallel during encryption. Using GIPSim, we can select data values that reduce the SNR. In our initial studies, we used randomly generated keys in a concurrent kernel that ran with the AES kernel. One feature of a GPU is that is can run multiple kernels concurrently. Using Concurrent Kernel Execution (CKE), thread blocks from multiple kernels run simultaneously on each of the GPU's streaming multiprocessors. The NVIDIA K20c used in our work supports up to 32 concurrent streams. These streams are independent, so they do not require any communication

or have any inter-stream dependencies. Since different streams may have different execution times, to guarantee that the streams will finish at the same time, a CUDA synchronization barrier can be used. However, it is not trivial to ensure that the two concurrent kernels will start at the time such that the computations will completely overlap. To resolve this issue, we run two keys, one actual and one to introduce noise, inside one warp, where all threads execute the same instruction in lock-step. We run half warps or 16 threads with the actual key and rest with the data generated by GIPSim. We want to investigate different schemes, such as:

1. AND-ing the key with 1,
2. generating a key where the bits are flipped,
3. picking different data values for the AES instructions from GIPSim that reduce SNR.

and find if GIPSim can identify the best obfuscation technique, attaining the lowest SNR for all cases. We also investigate the performance penalty for each of this obfuscation.

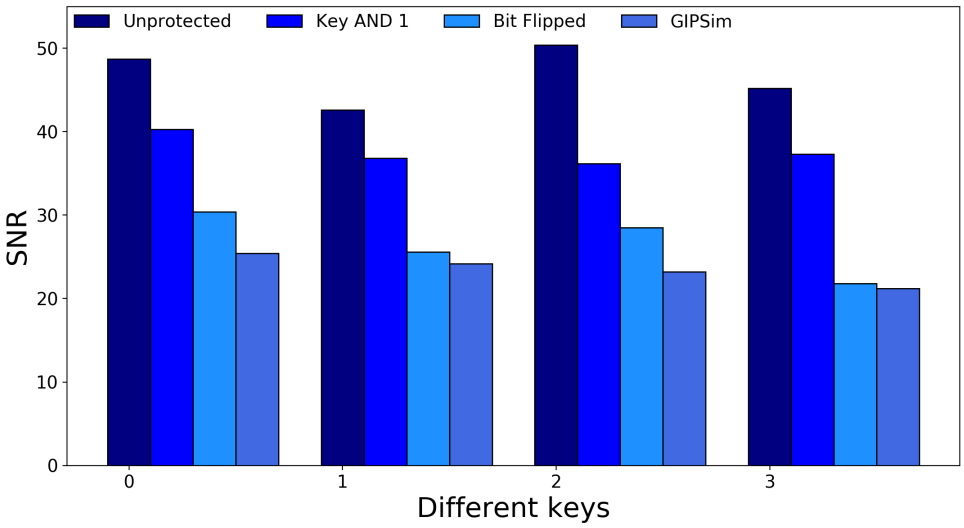


Figure 5.1: Different combination of keys contribute in reduced SNR

In Figure 5.1, we find that GIPSim is able to identify the best obfuscation technique, attaining the lowest SNR for all cases. This obfuscation affects performance, introducing a 60% slowdown since we are using only half of the warps to execute AES.

5.1.2 Equalization

As mentioned earlier, one common way to provide obfuscation is to hide the actual signal by making the power consumption constant at all times. This can be done by running the second computation in parallel during encryption. Using GIPSim, we can select data values so that the total power consumption remains fairly constant over the entire encryption, thus reducing the SNR. Just like last time, we run the two workloads (the encryption kernel and the compensation) inside a single warp, where all threads execute the same instruction in lock-step. We run half warps or 16 threads for the encryption, while the remaining 16 threads use the instruction-data selected by GIPSim to perform obfuscation. To help us select the best instruction-data pair for the compensation kernel, we use an LSTM. In the context of GIPSim, the sequence of instructions executed in the kernel is treated as the data series in the LSTM, where each instruction should have a corresponding instruction that should flatten the power signal as much as possible. We select a constant tp as the total constant power consumption, and the power each encryption instruction consumes constitutes the $\overrightarrow{P}$ vector. The selected obfuscation instructions for the output of the LSTM represented as $\overrightarrow{tp - P}$. The main challenge here is to create a feature vector from the instructions that provide an instruction's essential characteristics in terms of the degree of obfuscation.

5.1.3 Using an LSTM

LSTM (Long Short-Term Memory) Neural Networks are a particular class of recurrent neural networks (RNN) that has recently received increased attention within the machine learning community [127]. An LSTM network includes internal contextual state cells that can act as long-term or short-term memory cells. The state of these cells determines the output of the LSTM network. So when the prediction of a neural network depends on the historical context of a sequence of inputs versus the very last input, LSTM networks are commonly used. The LSTM unit includes a forget gate which enables us to learn long-distance dependencies [128]. For sequential labeling tasks such as obfuscation, an LSTM model can take into account an effectively infinite amount of context for an instruction execution with the selected data values, eliminating the problem of the limited context that applies to any feed-forward model [129].

Inspired by the recent successes of LSTMs for different sequence prediction tasks, including handwriting analysis [130] and speech generation [131], we will leverage an LSTM for guiding obfuscation prediction in the context of side-channel analysis. A recent line of work has investigated new attacks based on machine learning techniques to defeat both unprotected [132, 133]

and protected cryptographic implementations [134, 135]. In this approach, we try to find the pair of instructions that run together with the cryptographic instructions to provide the obfuscation. Now each of these obfuscation instructions in includes a base cost and an inter-instruction overhead introduced by the instruction that ran before this instruction. Now, all RNNs have feedback loops in the recurrent layer, which helps them maintain a *memory* over time. However, it becomes increasingly challenging to train standard RNNs for problems with long-term temporal dependencies. With encryption algorithms that are arbitrarily long with many instructions, the gradient of loss function decays exponentially with time (or instructions in this case). As already mentioned, LSTM maintains information in memory for long periods of time with the help of specialized memory cells and gates, which is useful in processing long lengths of cryptographic instructions. We want to investigate how ML can help in safeguarding cryptographic computations.

An LSTM is an artificial neural network that contains LSTM blocks. This class of network attempts to overcome the vanishing gradient problem by introducing multiplicative gates that implement constant error flow through the internal state. This network is well-suited for capturing the structure of time series based data and predicting at different time scales. For a time series $X = [x_1, x_2, ..., x_n]$, where x_t is an multi-dimensional vector $[x_t^1, x_t^2, ..., x_t^m]$ which represents the features of a sample, we can train a model with prior obfuscations from the former n samples of a power profile to predict the next obfuscation of m features. We use the key as the primary feature. In other words, we take $n \times m$ units in the input layer and m units in the output layer. The LSTM units consist of two hidden layers, and each unit in a lower LSTM hidden layer is fully connected to each unit in the LSTM hidden layer above it through feed-forward connections.

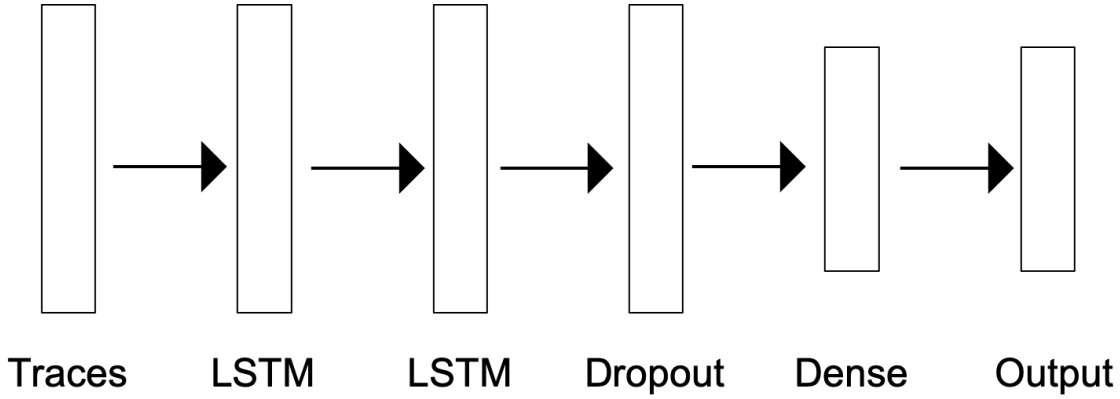


Figure 5.2: The layer structure of the deep neural network used for obfuscation using GIPSim.

We will use an LSTM model to capture the temporal context of a data series. In the

context of GIPSim, the data series is the instruction sequence, where each instruction should have a corresponding instruction that generates randomness in the power profile to provide obfuscation. So the power each instruction consumes constitutes the X vector, and the relevant obfuscation instructions are the output of the LSTM. The main challenge here is to create a feature vector from the instructions that provide an instruction's essential characteristics in terms of the degree of obfuscation. First, we build a corpus of obfuscation approaches using instructions that provide low overall SNR by equalizing the total pwoer consumption. These are used as input to train the LSTM model, performing inference on instructions that have yet to be seen. In our experiment, the network we are using contains two LSTM hidden layers followed by a dropout layer, and then a dense layer with a softmax activation function used for the output. The layered structure of the deep neural network is shown in Figure 5.2.

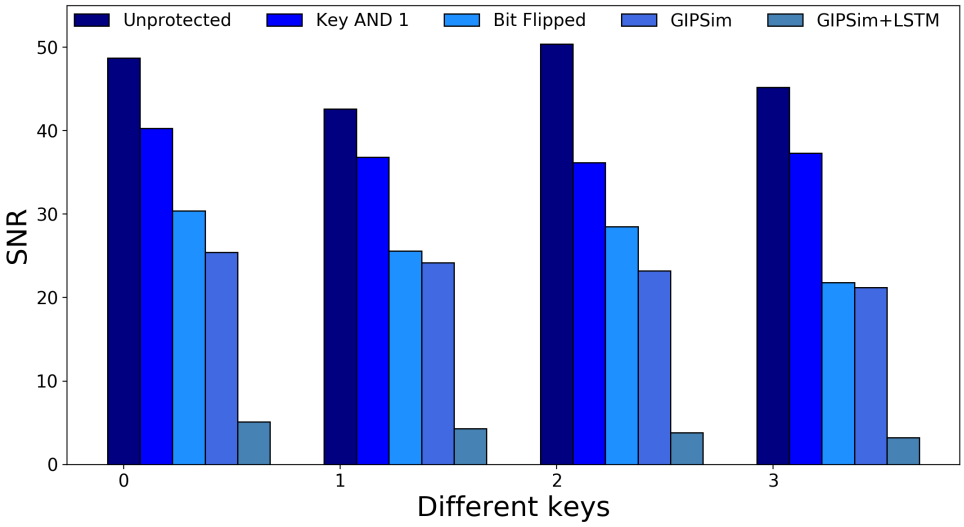


Figure 5.3: Different obfuscation approaches including LSTM

In the power profile of unprotected AES, each of the ten spikes indicates the ten rounds of AES are clearly distinguishable. The goal of the obfuscation technique mentioned above is to make the last round hidden, among other profiles. Here the divergent paths we follow to run the obfuscation technique within a warp introduces some randomness in power because of the finer level interleaved execution pattern within a warp. Also, in the obfuscation, to keep the performance the same, we run two similar warps (half for AES, half for equalization). When we look closely at the last round, in Figure 5.4b, the spikes have flattened now compared to the unprotected execution

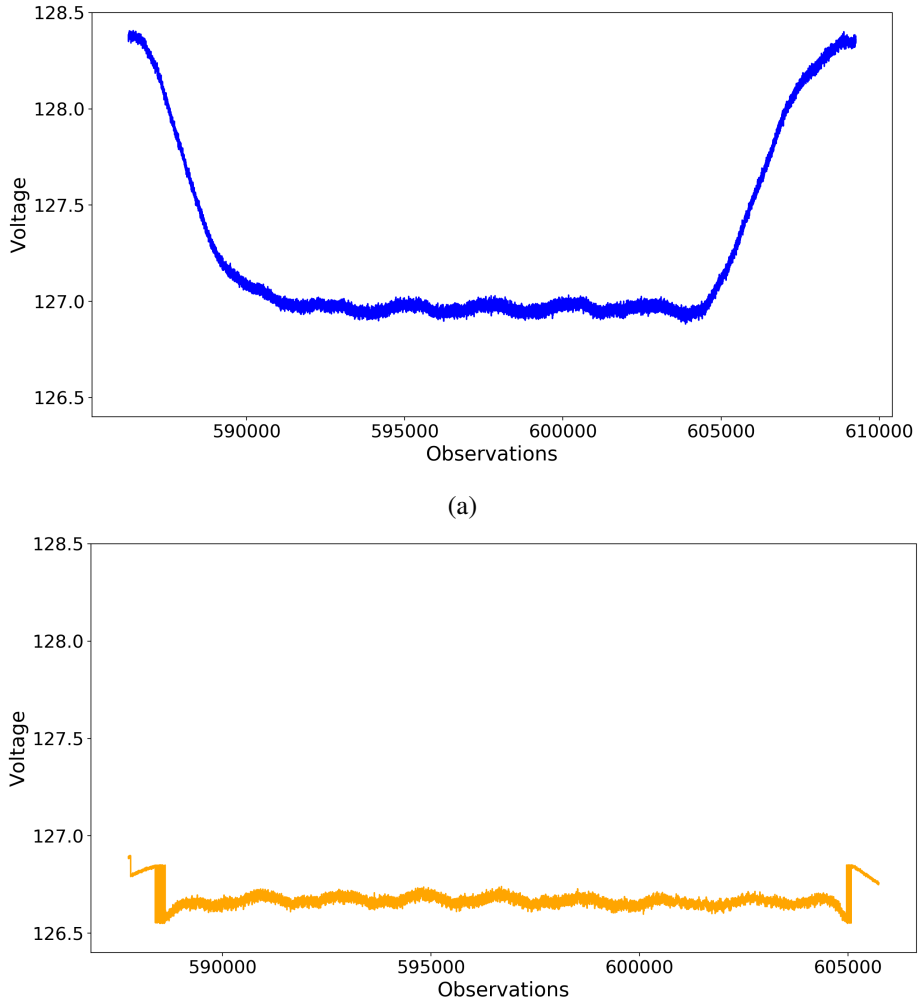
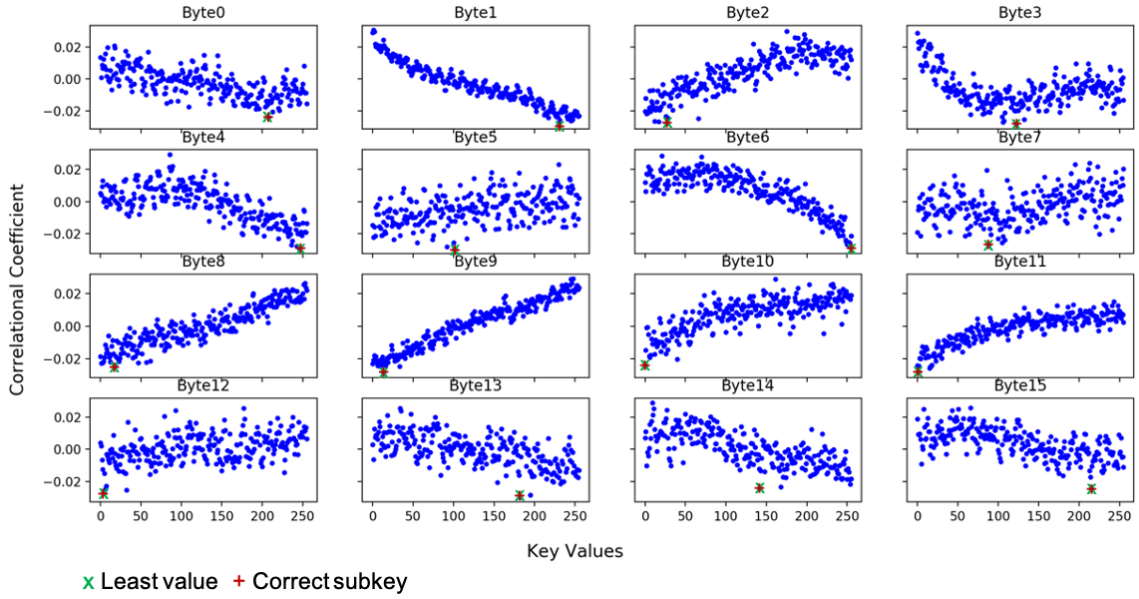


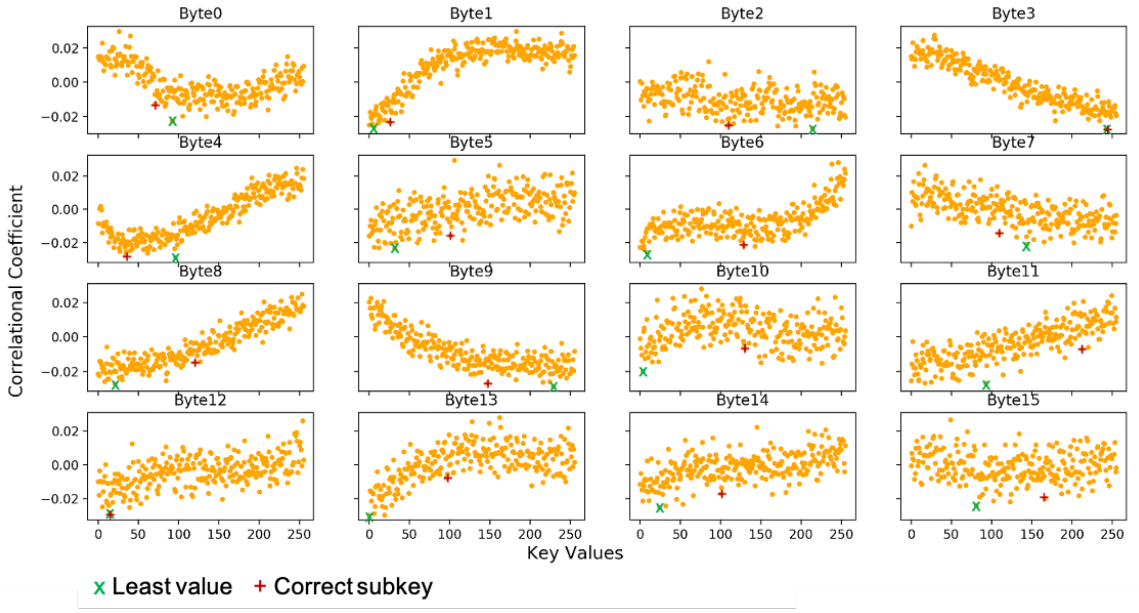
Figure 5.4: Power profile of the last round of (a) unprotected and (b) obfuscated AES.

in Figure 5.4a. After obfuscation, the last round is not so clearly distinguishable from the rest of the execution. We measure the variance on the power consumption for both the unprotected and obfuscated AES. While there is a high variance of around 0.342 for unobfuscated AES, when we use LSTM-guided obfuscation, the variance is as low as 0.017, which is a reduction of 95% in the variance.

We also evaluate the obfuscation in a real attack scenario. We run CPA on the power traces, intending to extract the last round key, byte by byte. Figure 5.5a shows the attack results for all bytes with 10^6 traces. The correct subkey bytes are marked with '$+$', and the ones with lowest value of



(a)



(b)

Figure 5.5: CPA attack results against (a) unprotected and (b) obfuscated AES.

correlation are marked with a '×'. As we can see in this figure, all correct subkey byte values have the lowest correlation coefficients. So the attacker can recover the exact last round key byte by byte

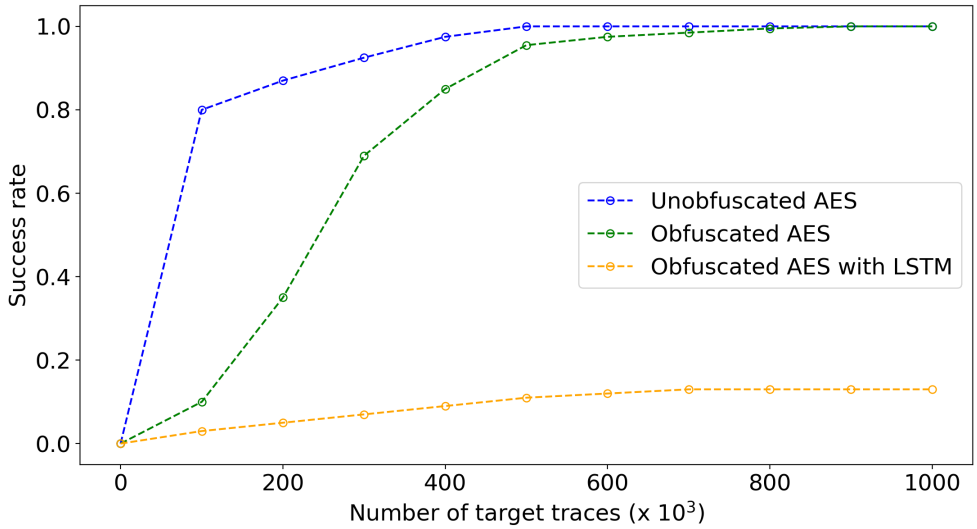


Figure 5.6: Success rate of attacks with and without deep learning techniques.

with control of the plaintext. However, in the next Figure 5.5b, when we run the experiment with equalized AES in the same setting, only two correct subkeys bytes result in the lowest correlation coefficient. For an attack scenario targeting a subkey (i.e., a key byte) and allowing for the sorting of different candidates, we define a success rate of order o as the probability that the correct subkey is guessed correctly among the first o candidates. The success rate (S_r) is generally computed as a function of the number of total attack traces. The attack against an unprotected AES implementation can achieve a high success rate (≥ 0.8) using 10^5 traces. A 100% success rate can be obtained using 5×10^5 traces. When running an obfuscated AES kernel, leveraging co-running obfuscation threads selected with our LSTM model, the key cannot be guessed with 10^6 traces, as shown in Figure 5.6. When we compared this ML-based obfuscation with the ones mentioned in Figure 5.1, we see that the SNR has been reduced even more, as shown in Figure 5.3. Clearly, the obfuscated version of AES that was guided by the LSTM model provides better protection, making it more challenging to launch a successful DPA attack versus an unprotected version of AES.

5.2 Masking

Masking randomizes the intermediate values of a cryptographic computation to avoid dependencies between the values and power consumption. This is usually applied at an algorithmic

level (unlike hiding) and does not rely on the power characteristics of the device. To achieve higher-order masking, each intermediate operation is transformed into multiple operations that can reduce the statistical correlation between operations and data values [57].

5.2.1 Higher-order masking

The main idea of higher-order masking is to split the key-dependent data into separate pieces that are randomly generated. In other words, masking is sharing a sensitive value, s, such that:

$$s = s_1 \odot s_2 \odot \cdots \odot s_d \tag{5.1}$$

For a group relation $\odot$, each s_i is randomly selected for $0 < i < d-1$. Since s is distributed between d shares, the last value s_d is calculated such that it satisfies Equation 5.1, and is called the masked variable. A dth-order masking is always theoretically vulnerable to a $(d+1)$th-order SCA that exploits the leakage related to $(d+1)$ intermediate values. Ishai *et al.* were the first to propose a dth-order masking scheme [136]. When implemented in hardware, this scheme is known to be sensitive to glitches [137]. This issue can be addressed through synchronization elements or by performing additional sharing, but these increase circuit area significantly [138]. The challenge of implementing this dth-order masking scheme in software is to process all logical operations of the SBox using a boolean representation in a masked way. As an example, key k is masked with two random variables, s_1 and s_2 in six masking schemes that result in $m_1, m_2, \ldots, m_6$ respectively.

$$m_1 = k \wedge (s_1 \wedge s_2) \tag{5.2}$$

$$m_2 = k \vee (s_1 \vee s_2) \tag{5.3}$$

$$m_3 = k \oplus (s_1 \wedge s_2) \tag{5.4}$$

$$m_4 = k \vee (s_1 \oplus s_2) \tag{5.5}$$

$$m_5 = k \wedge (s_1 \oplus s_2) \tag{5.6}$$

$$m_6 = k \oplus (s_1 \oplus s_2) \tag{5.7}$$

In Table 5.1, we present a truth table where all six ms are only dependent on s_1 and s_2. These values satisfy the following masking properties:

1. *Completeness*: at the end of the computation, these shares can be accumulated to yield the expected ciphertext.

k	s_1	s_2	m_1	m_2	m_3	m_4	m_5	m_6
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	0	1	1	1	1	0
1	1	0	0	1	1	1	1	0
1	1	1	1	1	0	0	1	1

Table 5.1: How masking variables leak data.

2. *d-th order security*: every tuple of the d elements is independent of s.

However, all except m_6 still leak secret information.

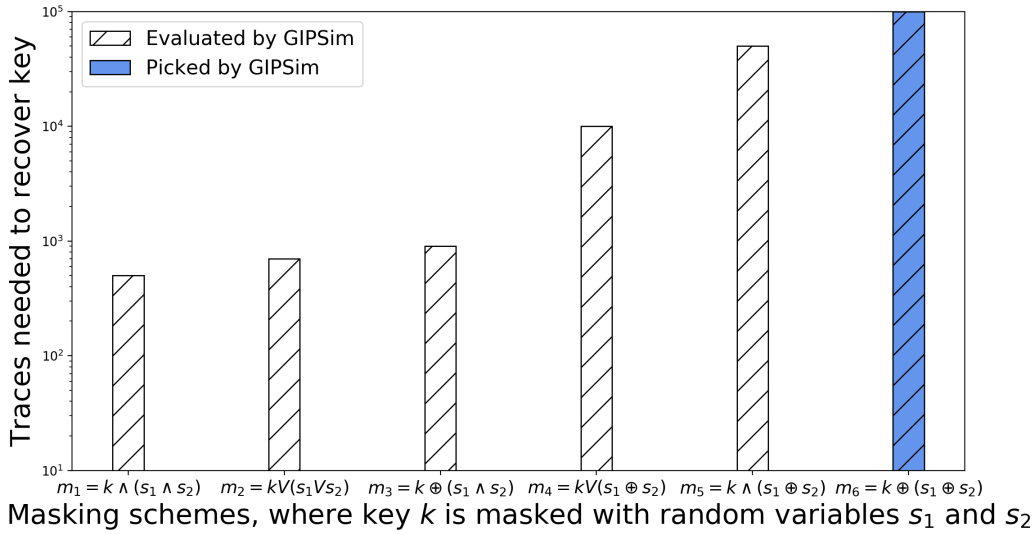


Figure 5.7: Six different masking variable combinations evaluated with GIPSim. We show the QMS achieved for each scheme, and the number of traces required to recover the key.

In our AES implementation, we generate all possible combinations of s_1 and s_2 using `cuRAND` [139]. When we run DPA on each of these combinations, we find that the number of traces required for each combination varies significantly [31]. So clearly, not all randomly generated combinations of masking variables are entirely masked. Higher-order masking schemes can be expensive since, for any change in the scheme, the implementation needs to be changed. To decide whether an implementation provides secure obfuscation, apart from performing an actual DPA with

the obfuscation in place, the Quantitative Masking Strength (QMS) can be calculated [140]. It is possible to evaluate the masking strength using an SAT-solver, but it is often time-consuming and has a limitation on the type of leakages that they can detect [141]. Instead, we can use GIPSim to judge whether the values used in the cryptographic computation satisfy a given obfuscation requirement.

Given that a dth-order masking should withstand any DPA exploit with an order less than $d + 1$, we investigate other combinations that failed to protect AES from a second-order DPA. Note that prior work has identified that the most effective bias is either 0 or 1, which are two extreme cases of the spectrum, and most of the generated numbers fall in between [142]. Often a random number generator includes some bias as a result of increasing the efficiency of the random number generation [143]. In general, generating one number over another impacts the strength of the obfuscation and violates the premise of perfect higher-order masking. GIPSim helps us identify the side-channel resistance from a large number of choices in between $0 < bias < 1$, as shown in Figure 5.7. Software implementations leveraging higher-order masking have been reported to be orders of magnitude slower than an unprotected implementation [144]. Owing to the massive parallelism available on a GPU, this obfuscation approach only incurs a 10% slowdown compared to an unprotected AES implementation. It is possible to utilize GIPSim to improve a performance penalty that can be thresholded.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the contributions of this dissertation and discuss the important lessons learned. Finally, we conclude with future directions for continuing this line of research work.

6.1 Dissertation Summary

Whenever a computer hardware platform performs computation, the power consumption of the hardware devices will depend on the values of the data being processed. Exploiting this relationship in the form of power consumption, power-based side-channel attacks attempt to correlate power consumption on the targeted device with data values used in an encryption algorithm to recover secret information. In the light of these attacks, we must attempt to implement the cryptographic primitives at an architectural level such that the effort (or cost) of the adversary for retrieving the key is too high. A critical direction for countermeasures is implementing cryptographic primitives such that the total power consumption of the hardware device leaks as little information as possible about the secret keys or data.

In this dissertation, we show that equipped with the right set of software tools, we can design effective countermeasures to thwart such attacks. In Chapter 4, we present the design and use of GIPSim, a novel simulation framework for assessing GPU power leakage. The model tracks power at the SASS level for an NVIDIA GPU. Note that the idea is generic and can be applied to other classes of GPU as well. Our model calculates a base cost and inter-instruction overhead for different instructions with different input data values, running on a GPU. We validated the predictions on a live GPU running applications taken from CUDA SDK. We have also correlated the predicted power

with measurements while running AES on a GPU with various input data. We also capture a trend of how the predicted power changes with DVFS in effect.

Equipped with GIPSim, we present obfuscation approaches to protect the cryptographic primitives in Chapter 5. All standard approaches to hide the relationship between power and secret key, such as randomization, equalization, masking, are implemented/evaluated with GIPSim. In randomization, we ran the cryptographic primitives in the half warp of the GPU, and in another half, we ran some computation to hide cryptographic power signal. Now to ensure these two overlap, we used warps because on GPU these are guaranteed to run together, overlapping the power signatures. For hiding the power signal or for running the computation on the other half, we evaluated the following techniques:

1. AND-ing the key with 1,
2. generating a key where the bits are flipped,
3. picking different data values for the AES instructions from GIPSim that reduce SNR

We found that GIPSim can identify the best obfuscation kernel, resulting in the lowest SNR for all cases with a maximum penalty of 60% in performance for encryption, because we are using half of the resources to do the encryption. To further improve obfuscation, we leveraged artificial recurrent neural networks. The goal is to identify an instruction stream using GIPSim so that the total power consumption obfuscates the power signal of the encryption algorithm. This equalization countermeasure reduces the variance in power consumption to 95%. This implementation is tested on a real attack scenario to show that this implementation is not vulnerable to a power attack. An attack against an unprotected AES implementation can achieve a high success rate ($\geq$ 0:8) using 10^5 traces and delivers a success rate of 1 or all key bytes are achieved using 5×10^5 traces. When running obfuscated AES optimized with our LSTM model, the right candidate keys could not be guessed with 10^6 traces.

The other prevailing countermeasure, masking, attempts to split the key-dependent data into separate pieces that they are randomly generated. However, the challenge of implementing this masking scheme in software is to process all logical operations of the SBox using a boolean representation in a masked way. We have shown that not all randomly generated combinations of masking variables are entirely masked. Higher-order masking schemes can be expensive since, for any change in the scheme, the implementation needs to be changed, and evaluating the strength of the masking with existing SAT solvers can be time-consuming. Instead, we can use GIPSim to judge

whether the values used in the cryptographic computation satisfy a given obfuscation requirement with a maximum of 10% slowdown in overall performance. It is possible to configure GIPSim further such that the performance penalty is within a threshold.

Our work on GIPSim paves the way toward building a more resilient future system that can thwart power side-channel attacks. The GIPSim framework enables security researchers to reason about side-channel leakage present in the context of a GPU execution-driven simulator. This is one of the first simulation environments that can be used for obfuscation design.

6.2 Future Work

There are several research questions yet to be answered in this area of research. As mentioned earlier, supporting DVFS completely to this model would be an excellent addition. This would require more experiments to perform with various frequency and voltage settings with various opcode and HD, similar to the aforementioned baseline experiments. Finally, once the corpus is ready, we can verify the values with similar experiments we did to validate the model. For all these experiments, it is essential to be vigilant about the actual operating frequency/voltage of the device because the device can change both the voltage and frequency. To avoid oveheating, the device lowers the voltage/frequency, although the user sets it to a higher value.

For future directions to continue this work, it would be interesting to see how GIPSim works for additional cryptographic primitives, such as public-key cryptography. Further, it would be useful for the architecture/security community to make a relative power model similar to GIPSim for different discrete and mobile GPUs from vendors such as Qualcomm and Apple.

Since no existing power models consider data, a reasonable extension of this work would be to add the data dependency of the power model to existing power models for K20 or Kepler simulators such as Multi2Sim [145].

In the past decade, with the advent of practically realizable quantum technologies, we need to find ways to protect large scale quantum computers running public-key cryptographic algorithms. When researchers have found them to be vulnerable [146], a spur began in the cryptographic community to find countermeasures for designing safe cryptographic primitives in the context of large scale quantum computing systems. It would be interesting to investigate if we can build similar relative power models for such systems and use them to create safe ciphers.

Bibliography

[1] I. Duncan and A. K. McDaniels, "MedStar hack shows risks that come with electronic health records," 2016. [Online]. Available: https://www.baltimoresun.com/health/bs-md-medstar-healthcare-hack-20160402-story.html

[2] K. Zetter, "Inside the cunning, unprecedented hack of Ukraine's Power Grid," 2016. [Online]. Available: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/

[3] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[4] V. Roberge and M. Tarbouchi, "Fast path planning for unmanned aerial vehicle using embedded gpu system," in *2017 14th International Multi-Conference on Systems, Signals & Devices (SSD)*. IEEE, 2017, pp. 145–150.

[5] R. Hossain, S. Magierowski, and G. G. Messier, "Gpu enhanced path finding for an unmanned aerial vehicle," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 1285–1293.

[6] R. C. Green, L. Wang, and M. Alam, "Applications and trends of high performance computing for electric power systems: Focusing on smart grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 922–931, 2013.

[7] Top500, "TOP500 Supercomputing Sites," 2010.

[8] J. Nickolls, I. Buck, and M. Garland, "Scalable parallel programming," pp. 40–53, 2008.

[9] K. OpenCL, "OpenCL Specification," *ReVision*, 2009.

[10] J. Sirignano, A. Sadhwani, and K. Giesecke, "Deep learning for mortgage risk," 2016.

[11] M. Leeser, S. Mukherjee, and J. Brock, "Fast reconstruction of 3d volumes from 2d ct projection data with gpus," *BMC research notes*, vol. 7, no. 1, p. 582, 2014.

[12] V. Campmany, S. Silva, A. Espinosa, J. C. Moure, D. Vázquez, and A. M. López, "Gpu-based pedestrian detection for autonomous driving," vol. 80. Elsevier, 2016, pp. 2377–2381.

[13] Z. Shen, K. Wang, and F. Zhu, "Agent-based traffic simulation and traffic signal timing optimization with gpu," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2011, pp. 145–150.

[14] W. Fan, X. Chen, and X. Li, "Parallelization of rsa algorithm based on compute unified device architecture," in *2010 Ninth International Conference on Grid and Cloud Computing*. IEEE, 2010, pp. 174–178.

[15] J. Gilger, J. Barnickel, and U. Meyer, "Gpu-acceleration of block ciphers in the openssl cryptographic library," in *International Conference on Information Security*. Springer, 2012, pp. 338–353.

[16] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of aes encryption on gpu," in *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. IEEE, 2012, pp. 843–848.

[17] Y. Yang, Z. Guan, H. Sun, and Z. Chen, "Accelerating rsa with fine-grained parallelism using gpu," in *International Conference on Information Security Practice and Experience*. Springer, 2015, pp. 454–468.

[18] R. Fernando, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[19] K. Jang, S. Han, S. Han, S. B. Moon, and K. Park, "Sslshader: Cheap ssl acceleration with commodity processors." in *NSDI*, 2011, pp. 1–14.

[20] J. M. Aamir Majeed Usman Aziz Salman Ul Haq, "Bulk encryption on GPUs," Tech. Rep. [Online]. Available: https://developer.amd.com/article/bulk-encryption-on-gpus-salman-ul-haq-jawad-masood-aamir-majeed-usman-aziz/

[21] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1998.

[22] C. Kalra, F. Previlon, X. Li, N. Rubin, and D. Kaeli, "Prism: Predicting resilience of gpu applications using statistical methods," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 866–879.

[23] F. G. Previlon, C. Kalra, D. R. Kaeli, and P. Rech, "Evaluating the impact of execution parameters on program vulnerability in gpu applications," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 809–814.

[24] NVIDIA, "Cuda C Programming Guide," 2015.

[25] Nvidia, "The CUDA Compiler Driver NVCC," 2015.

[26] X. Zhang, G. Tan, S. Xue, J. Li, K. Zhou, and M. Chen, "Understanding the gpu microarchitecture to achieve bare-metal performance tuning," vol. 52, no. 8. ACM, 2017, pp. 31–43.

[27] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110/210," Tech. Rep. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[28] C. Rebeiro, D. Mukhopadhyay, and S. Bhattacharya, *Timing channels in cryptography: a micro-architectural perspective*. Springer, 2014.

[29] P. Margara, "Engine-CUDA, a cryptographic engine for CUDA supported devices," in *https://code.google.com/p/ engine-cuda/*, 2015.

[30] 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.

[31] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[32] D. Genkin, A. Shamir, and E. Tromer, "Rsa key extraction via low-bandwidth acoustic cryptanalysis," in *Annual Cryptology Conference*. Springer, 2014, pp. 444–461.

[33] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[34] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[35] D. J. Bernstein, "Cache-timing attacks on AES," Tech. Rep., 2005. [Online]. Available: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[36] S. A. Crosby, D. S. Wallach, and R. H. Riedi, "Opportunities and limits of remote timing attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 12, no. 3, p. 17, 2009.

[37] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 355–371.

[38] N. J. Al Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 526–540.

[39] W. Van Eck, "Electromagnetic radiation from video display units: an eavesdropping risk?" *Computers & Security*, vol. 4, no. 4, pp. 269–286, 1985.

[40] M. G. Kuhn and R. J. Anderson, "Soft tempest: Hidden data transmission using electromagnetic emanations," in *International Workshop on Information Hiding*. Springer, 1998, pp. 124–142.

[41] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2001, pp. 251–261.

[42] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and countermeasures for smart cards," in *International Conference on Research in Smart Cards*. Springer, 2001, pp. 200–210.

[43] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel (s)," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 29–45.

[44] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," pp. 343–358, 2002.

[45] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards." vol. 99, 1999, pp. 151–161.

[46] J.-S. Coron, P. Kocher, and D. Naccache, "Statistics and secret leakage," in *International Conference on Financial Cryptography*. Springer, 2000, pp. 157–173.

[47] N. Pramstaller, F. K. Gurkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner, "Towards an aes crypto-chip resistant to differential power analysis," pp. 307–310, 2004.

[48] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.

[49] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE transactions on computers*, vol. 51, no. 5, pp. 541–552, 2002.

[50] J. A. Ambrose, R. G. Ragel, S. Parameswaran, and A. Ignjatovic, "Multiprocessor information concealment architecture to prevent power analysis-based side channel attacks," *IET computers & digital techniques*, vol. 5, no. 1, pp. 1–15, 2011.

[51] H. Yu, X.-c. ZOU, Z.-l. LIU, and Y.-c. CHEN, "The research of dpa attacks against aes implementations," *The Journal of China Universities of Posts and Telecommunications*, vol. 15, no. 4, pp. 101–106, 2008.

[52] S. B. Örs and B. Preneel, "Power analysis of an fpga implementation of rijndael: Is pipelining a dpa countermeasure," in *in Cryptographic Hardware and Embedded Systems*. Citeseer, 2004.

[53] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an asic aes implementation," in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, 2004, vol. 2, pp. 546–552.

[54] K. Tiri, D. Hwang, A. Hodjat, B. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "A side-channel leakage free coprocessor ic in 0.18 μm cmos for embedded aes-based cryptographic and biometric processing," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 222–227.

[55] T. S. Messerges, "Using second-order power analysis to attack dpa resistant software," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 238–251.

[56] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[57] J.-S. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 231–237.

[58] D. Agrawal, J. R. Rao, P. Rohatgi, and K. Schramm, "Templates as master keys," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 15–29.

[59] E. Oswald and S. Mangard, "Template attacks on masking—resistance is futile," in *Cryptographers' Track at the RSA Conference*. Springer, 2007, pp. 243–256.

[60] E. Oswald, S. Mangard, C. Herbst, and S. Tillich, "Practical second-order dpa attacks for masked smart card implementations of block ciphers," in *Cryptographers' Track at the RSA Conference*. Springer, 2006, pp. 192–207.

[61] V. Sundaresan, S. Rammohan, and R. Vemuri, "Defense against side-channel power analysis attacks on microelectronic systems," in *2008 IEEE National Aerospace and Electronics Conference*. IEEE, 2008, pp. 144–150.

[62] P. Luo, L. Zhang, Y. Fei, and A. A. Ding, "Towards secure cryptographic software implementation against side-channel power analysis attacks," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2015, pp. 144–148.

[63] J.-S. Coron, "A new dpa countermeasure based on permutation tables," in *International Conference on Security and Cryptography for Networks*. Springer, 2008, pp. 278–292.

[64] A. W. Fritzke, "Obfuscating against side-channel power analysis using hiding techniques for aes," 2012.

[65] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 142–159.

[66] M. Zhang and N. K. Jha, "Finfet-based power management for improved dpa resistance with low overhead," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 7, no. 3, p. 10, 2011.

[67] S. Yang, W. Wolf, N. Vijaykrishnan, D. N. Serpanos, and Y. Xie, "Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 64–69.

[68] W. Yu, O. A. Uzun, and S. Köse, "Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 115.

[69] D. Das, S. Maity, S. B. Nasir, S. Ghosh, A. Raychowdhury, and S. Sen, "High efficiency power side-channel attack immunity using noise injection in attenuated signature domain," in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2017, pp. 62–67.

[70] Z. Chen and P. Schaumont, "Virtual secure circuit: porting dual-rail pre-charge technique into software on multicore," 2010.

[71] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE, 2004, pp. 3–11.

[72] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, p. 3, 2009.

[73] M. Backes, M. Dürmuth, S. Gerling, M. Pinkal, and C. Sporleder, "Acoustic side-channel attacks on printers." pp. 307–322, 2010.

[74] A. Faruque, M. Abdullah, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic side-channel attacks on additive manufacturing systems," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*. IEEE Press, 2016, p. 19.

[75] A. Shamir and E. Tromer, "Acoustic cryptanalysis," *presentation available from http://www. wisdom. weizmann. ac. il/ tromer*, 2004.

[76] M. G. Kuhn, "Compromising emanations: eavesdropping risks of computer displays," 2002.

[77] ——, "Optical time-domain eavesdropping risks of crt displays," in *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 3–18.

[78] R. D. Pietro, F. Lombardi, and A. Villani, "Cuda leaks: a detailed hack for cuda and a (partial) fix," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 15, 2016.

[79] A. Miele, "Buffer overflow vulnerabilities in cuda: a preliminary analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 2, pp. 113–120, 2016.

[80] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting gpu vulnerabilities," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 19–33.

[81] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "Confidentiality issues on a GPU in a virtualized environment," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 119–135.

[82] Z. Zhou, W. Diao, X. Liu, Z. Li, K. Zhang, and R. Liu, "Vulnerable gpu memory management: towards recovering raw data from gpu," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 57–73, 2017.

[83] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli, "Side-channel power analysis of a gpu aes implementation," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, 2015, pp. 281–288.

[84] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *2016 IEEE International symposium on high performance computer architecture (HPCA)*. IEEE, 2016, pp. 394–405.

[85] ——, "A novel side-channel timing attack on gpus," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 167–172.

[86] G. Kadam, D. Zhang, and A. Jog, "Rcoal: mitigating gpu timing attack via subwarp-based randomized coalescing techniques," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 156–167.

[87] X. Wang and W. Zhang, "Cracking randomized coalescing techniques with an efficient profiling-based side-channel attack to GPU," in *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2019, p. 2.

[88] E. Karimi, Z. H. Jiang, Y. Fei, and D. Kaeli, "A timing side-channel attack on a mobile gpu," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 67–74.

[89] Y. Gao, H. Zhang, W. Cheng, Y. Zhou, and Y. Cao, "Electro-magnetic analysis of gpu-based aes implementation," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 121.

[90] Y. Gao, Y. Zhou, and W. Cheng, "Efficient electro-magnetic analysis of a gpu bitsliced aes implementation," *Cybersecurity*, vol. 3, no. 1, pp. 1–17, 2020.

[91] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding gpu power: A survey of profiling, modeling, and simulation methods," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, p. 41, 2016.

[92] M. Burtscher, I. Zecena, and Z. Zong, "Measuring gpu power with the k20 built-in sensor," in *Proceedings of Workshop on General Purpose Processing Using GPUs*. ACM, 2014, p. 28.

[93] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," vol. 28, no. 2. ACM, 2000.

[94] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 469–480.

[95] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *International conference on green computing*. IEEE, 2010, pp. 115–122.

[96] S. Hong and H. Kim, "An integrated gpu power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 280–289.

[97] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 673–686.

[98] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2009, pp. 163–174.

[99] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: enabling energy optimizations in gpgpus," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 487–498.

[100] R. Nath and D. Tullsen, "The crisp performance model for dynamic voltage and frequency scaling in a gpgpu," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 281–293.

[101] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills gpusimpow: A gpgpu power simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 97–106.

[102] O. Harrison and J. Waldron, "Aes encryption implementation and analysis on commodity graphics processing units," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 209–226.

[103] S. A. Manavski, "Cuda compatible gpu as an efficient hardware accelerator for aes cryptography," in *2007 IEEE International Conference on Signal Processing and Communications*. IEEE, 2007, pp. 65–68.

[104] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[105] O. Harrison and J. Waldron, "Practical symmetric key cryptography on modern graphics hardware," *USENIX Security Symposium*, 2008.

[106] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a parallel aes for graphics hardware using the cuda framework," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. IEEE Computer Society, 2009, pp. 1–8.

[107] C. Mei, H. Jiang, and J. Jenness, "Cuda-based aes parallelization with fine-tuned gpu memory utilization," in *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*. IEEE, 2010, pp. 1–7.

[108] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 201–215.

[109] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," in *Cryptographers' track at the RSA conference*. Springer, 2006, pp. 1–20.

[110] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2012, pp. 159–176.

[111] R. K. Lim, L. R. Petzold, and Ç. K. Koç, "Bitsliced high-performance aes-ecb on gpus," in *The New Codebreakers*. Springer, 2016, pp. 125–133.

[112] G. Agosta, A. Barenghi, A. Di Federico, and G. Pelosi, "Opencl performance portability for general-purpose computation on graphics processor units: an exploration on cryptographic primitives," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 14, pp. 3633–3660, 2015.

[113] G.-l. Guo, Q. Qian, and R. Zhang, "Different implementations of aes cryptographic algorithm," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 2015, pp. 1848–1853.

[114] X. Fei, K. Li, W. Yang, and K. Li, "Practical parallel aes algorithms on cloud for massive users and their performance evaluation," vol. 28, no. 16. Wiley Online Library, 2016, pp. 4246–4263.

[115] W.-K. Lee, B.-M. Goi, and R. C.-W. Phan, "Terabit encryption in a second: Performance evaluation of block ciphers in gpu with kepler, maxwell, and pascal architectures," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 11, p. e5048, 2019.

[116] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 413–427.

[117] K. Schramm and C. Paar, "Higher order masking of the aes," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*. Springer-Verlag, 2006, pp. 208–225.

[118] J.-S. Coron, E. Prouff, and M. Rivain, "Side channel cryptanalysis of a higher order masking scheme," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 28–44.

[119] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche, "Higher-order side channel security and mask refreshing," in *International Workshop on Fast Software Encryption*. Springer, 2013, pp. 410–424.

[120] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," pp. 384–390, 1994.

[121] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," *ACM Sigplan Notices*, vol. 45, no. 3, pp. 129–142, 2010.

[122] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *IEEE International Symposium on Workload Characterization (IISWC'10)*. IEEE, 2010, pp. 1–11.

[123] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power cmos," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, 1997.

[124] Nvidia, "The NVIDIA System Management Interface (nvidia-smi)." [Online]. Available: https://developer.nvidia.com/nvidia-system-management-interface

[125] "Nvidiux." [Online]. Available: https://github.com/RunGp/Nvidiux

[126] Nvidia, "NVIDIA Management Library (NVML)." [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml

[127] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Advances in neural information processing systems*, 1997, pp. 473–479.

[128] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[129] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[130] R. Messina and J. Louradour, "Segmentation-free handwritten chinese text recognition with lstm-rnn," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 171–175.

[131] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International conference on machine learning*, 2014, pp. 1764–1772.

[132] L. Lerman, G. Bontempi, O. Markowitch *et al.*, "Power analysis attack: an approach based on machine learning." *IJACT*, vol. 3, no. 2, pp. 97–115, 2014.

[133] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis)," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 20–33.

[134] R. Gilmore, N. Hanley, and M. O'Neill, "Neural network based attack on a masked implementation of aes," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 106–111.

[135] L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked aes," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 123–139, 2015.

[136] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," pp. 463–481, 2003.

[137] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked cmos gates," in *Cryptographers' Track at the RSA Conference*. Springer, 2005, pp. 351–365.

[138] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *Journal of Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.

[139] Nvidia, "NVIDIA CUDA Random Number Generation library." [Online]. Available: https://developer.nvidia.com/curand

[140] H. Eldib, C. Wang, M. Taha, and P. Schaumont, "Quantitative masking strength: quantifying the power side-channel resistance of software code," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1558–1568, 2015.

[141] A. G. Bayrak, F. Regazzoni, D. Novo, and P. Ienne, "Sleuth: Automated verification of software power analysis countermeasures," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 293–310.

[142] H. Eldib, C. Wang, M. Taha, and P. Schaumont, "Qms: Evaluating the side-channel resistance of masked software from source code," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–6.

[143] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger, "Rsm: a small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 1173–1178.

[144] D. Goudarzi and M. Rivain, "How fast can higher-order masking be in software?" in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 567–597.

[145] X. Gong, R. Ubal, and D. Kaeli, "Multi2sim kepler: A detailed architectural gpu simulator," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2017, pp. 269–278.

[146] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.