Poster: GIPSim: Low-level Power Modeling for Resiliency from Side Channel Attacks on GPUs

Saoni Mukherjee, Yunsi Fei, David Kaeli
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA
{saoni, yfei, kaeli}@ece.neu.edu

I. Introduction

With the emergence of ubiquitous computing, smart devices are able to communicate with each other unobtrusively, enabling users to access information and services anywhere, and at any time. Often these applications require real-time performance when performing cryptographic operations. Since GPUs accelerate workloads using thousands of parallel threads running on thousands of hardware cores and providing high throughput, they can be used in these devices to accelerate cryptographic computations. Both unmanned aerial vehicles (UAV) and smart grids are equipped with GPUs to provide high throughput to encrypt/decrypt information in real-time [1], [2]. For example, in smart grid applications, highly sensitive data, including power consumption, pricing, and outage alerts are encrypted and exchanged between power meters and the utility company in real-time. An attack on such critical and dynamic information can significantly impact resource availability and costs. The cryptographic algorithms running on a GPU can be exploited to serve as a first line of defense against attacks, providing protection for the critical data stored on these devices.

Although GPUs are leveraged to speed up cryptographic algorithms and yield higher performance, often these accelerators fail to protect the computation against Side-Channel Attacks (SCA). SCA exploits the physical implementation of a cryptographic system, rather than the inherent theoretical weaknesses of the algorithm itself. Previous work has demonstrated successful side channel attacks on CPUs, field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs) and GPUs. While these devices are highly vulnerable to SCA, the cryptographic algorithms running on these devices are mathematically proven to be secure. To expose a potential vulnerability, an external manifestation (i.e., power consumption, processing time, electromagnetic emission) is measured by an attacker to identify patterns in program execution. Among the previously reported power side-channel attacks, simple power analysis (SPA), differential power analysis (DPA), and correlation power analysis (CPA) have been extensively explored. These attacks inadvertently leak information about the execution in the system [3], [4]. However, to understand how to defend against these attacks, we need to understand how the underlying microarchitecture leaks side-channel information to the attacker.

To begin to understand leakage better, instead of building yet another power model for the GPU, we have designed a data-dependent power simulation framework to characterize power leakage. Our model considers the hamming distance (HD) of data values used during program execution, a key element commonly exploited in many power side-channel attacks. We then use our model to design obfuscation approaches to thwart power SCAs.

Recent work has demonstrated different SCA on GPUs. Luo *et al.* recover the 16-byte secret key of AES with Correlation Power Analysis on a GPU [5], while Jiang *et al.* describe a timing attack using Correlation Timing Analysis [6]. More recently, Jian *et al.* presented a timing attack using Differential Timing Analysis [7].

This work was supported in part by Semiconductor Research Corporation (SRC and by the NSF STARSS Program).

II. Background

A. GPU basics and CUDA programming model

GPUs are basically co-processors that execute work offloaded by the CPU host. This work focuses on a CUDA implementation, the main API framework used to generate code for Nvidia GPUs. The portion of the program that is executed on the GPU, is called a *kernel*. The kernel is executed in the form of many parallel instances that are mapped to a set of parallel threads. A stream is a sequence of operations that are performed in order on the device. Multiple kernels can be run using *streams* that are executed concurrently.

SASS is the low-level assembly language which is generated during compilation of CUDA code. SASS executes natively on Nvidia GPU hardware.

B. AES

Our implementation of AES-128 is based on the Electronic Code Book mode [8]. AES-128 uses 10 rounds and each round performs four processing steps: AddRoundKey, SubBytes, ShiftRows, and MixColumns, except the initial and last rounds. At the beginning of the encryption, the plaintext is loaded into the GPU's global memory. Each thread is assigned a portion of the data and copies it to the local memory, based on its block and thread id. Once the encryption completes, the ciphertext that lies in local memory is copied to global memory, and then is transferred to the CPU memory. Running in ECB mode, data blocks can be processed independently, thus depending on the data size and resources available, the operation is highly parallelizable.

C. Power leakage acquisition

Our implementation of AES has been tested on an Nvidia Kepler K20c GPU. 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. The voltage drop across the resistor is recorded during encryption on the GPU.

III. GIPSim framework

The goal of this work is to deliver GIPSim (GPU Instruction-level Power SIMulator), a framework to enable security researchers to reason about side-channel leakage present in the context of a GPU execution-driven simulator. In this contribution, we show that GIPSim can provide accurate power estimation while running CUDA programs on an Nvidia GPU.

To characterize SASS-code level data-dependent execution, GIPSim captures data-dependent power dissipation. We base our model off the power modeling work presented by Tiwari *et. al.* [9]. Our power model estimates the deterministic part of the power consumption, for example, the Hamming distance, 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 which predicts the power consumption of an 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 Hamming Distances (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 (or 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 low variance (0.039) for the average power consumption measured across a full run of the kernel. Given this low value, the power consumed if 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. 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 traces. 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.

For each instruction opcode, as we increase the HD increases, we see a linear increase in power consumption. So if 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 instruction i can be computed as:

$$P_i = P_{mincost} + slope_i \times HD(i) + p_{overhead} + p_{extra} \quad (1)$$

where P_i is the total power consumption for a particular instruction i, $P_{mincost}$ is the base cost 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.

IV. Evaluation of the Model

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 whether the power predicted by GIPSim has any correlation with the power measured from the device.

To achieve this, we pick ten random input text files $t_0, ..., t_9$ as inputs to AES-128 and run them through GIPSim to measure the predicted power $G_{t0}, \ldots, G_{t9}$. We also run the same input text files on the real device and measure power $R_{t0}, \ldots, R_{t9}$. Next, we take each R_i and compute the correlation with all G_is. The goal is to find the pairs $< G_i, R_i >$ with the same ordinal index value i (where i is the input file number $t_0, \ldots, t_9$) which exhibit the highest correlation. We expect the pair of traces (simulated and measured) that used the same input texts values to produce the highest correlations. The PCC values for each pair $< G_i, R_i >$ are greater than 0.8, suggesting that the results are highly correlated. In Figure 1, we show four such cases for $< G_i, R_i >$ and we observe that the top ten pairs $< G_i, R_i >$ produce a 40% higher correlation that the next most similar trace, demonstrating the strong predictability of GIPSim.

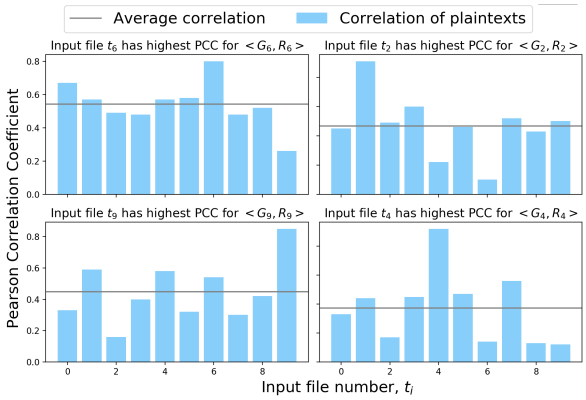

Fig. 1. Pearson Correlation between the modeled power and measured power.

V. Conclusion and Ongoing Work

In this work, we presented a novel model for GPU power leakage, GIPSim. The model tracks power at the SASS level for an Nvidia GPUs. Our model can predict a base cost and inter-instruction overhead for different instructions with different input data values, which can be used to design obfuscation approaches, hiding the power profile of cryptographic applications such AES.

We showed that GIPSim not only accurately characterizes power consumption on a GPU, but can also thwart power-based attacks. We already know how to launch power attacks using SPA or DPA. To thwart an attack, we can use GIPSim to add noise in the power profile for an encryption execution.

Our ongoing work investigates how to automatically select instructions that will run concurrently with the AES kernel, such that the strength of the power signal will be low, which in turn will make key recovery much more challenging. If the variance in the signal is high, the necessary number of samples may render the attack infeasible. These attacks are possible due to power fluctuations observed during byte substitution of an AES encryption in the last round. To prevent this leakage, we use GIPSim to automatically select instructions from the GIPSim corpus so that the power consumption is indistinguishable while running encryption.

References

[1] 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 and Devices, SSD 2017*, vol. 2017-January, pp. 145–150, 2017.

[2] 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*, 2013.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in cryptologyCRYPTO99*, pp. 789–789, Springer, 1999.

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

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

[6] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 394–405, IEEE, 2016.

[7] Z. H. Jiang, Y. Fei, and D. Kaeli, "A novel side-channel timing attack on gpus," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pp. 167–172, ACM, 2017.

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

[9] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Transactions on Very Large Scale Integrated Systems*, vol. 2, pp. 437–445, Dec. 1994.