Side-Channel Power Analysis of a GPU AES Implementation

***Abstract*—Graphics Processing Units (GPUs) have been used to run a range of cryptographic algorithms. The main reason to choose a GPU is to accelerate the encryption/decryption speed. Since GPUs are mainly used for graphics rendering, and only recently have they become a fully-programmable parallel computing device, there has been little attention paid to their vulnerability to side-channel attacks.**

In this paper we present a study of side-channel vulnerability on a state-of-the-art graphics processor. To the best of our knowledge, this is the first work that attempts to extract the secret key of a block cipher implemented to run on a GPU. We present a side-channel power analysis methodology to extract all of the last round key bytes of a CUDA AES (Advanced Encryption Standard) implementation run on an NVIDIA TESLA GPU. We describe how we capture power traces and evaluate the power consumption of a GPU. We then construct an appropriate power model for the GPU. We propose effective methods to sample and process the GPU power traces so that we can recover the secret key of AES. Our results show that parallel computing hardware systems such as a GPU are highly vulnerable targets to power-based side-channel attacks, and need to be hardened against side-channel threats.

I. Introduction

Graphics Processing Units (GPU), originally designed for 3-D graphics rendering, have evolved into high performance general purpose processors, called GPGPUs. A GPGPU can provide significant performance advantages over traditional multi-core CPUs by executing workloads in parallel on hundreds to thousands of cores. What has spurred on this development is the delivery of programmable shader cores, and high-level programming languages [1]. GPUs have been used to accelerate a wide range of applications [2], including: signal processing, circuit simulation, and molecular modeling. Motivated by the demand for efficient cryptographic computation, GPUs are now being leveraged to accelerate a number of cryptographic algorithms [3], [4], [5].

While cryptographic algorithms have been implemented to run on GPUs for higher performance, the security of GPU-based cryptographic systems remains an open question. Previous work has analyzed the security of GPU systems [6], [7], [8], [9]. The prior work focused more on using software methods to exploit the vulnerabilities of the GPU programming model. Side-channel vulnerabilities of GPUs have received limited attention in the research community. Meanwhile, cryptographic systems based on CPU, application-specific integrated circuits (ASICs), and FPGA platforms have been shown to be highly vulnerable to side-channel attacks. For example, Moradi *et. al.* showed that side-channel power leakage can be utilized by attackers to compromise cryptographic systems that use microcontrollers [10], smart cards [11], ASICs [12] and FPGAs [13], [14].

Different attack methods can be used for analyzing side-channel power leakage, e.g., differential power analysis (DPA) [15], correlation power analysis (CPA) [16] and mutual information analysis (MIA) [17]. These attack methods pose a large threat to both hardware-based and software-based cryptographic implementations. Given all of this previous side-channel power analysis activity, it is surprising that GPU-based cryptographic resilience has not been considered. In this paper, for the first time, we apply CPA on an AES implementation running on a GPU, and succeed in extracting the secret key through analyzing the power consumption of the GPU.

Note that the inherent Single Instruction Multiple Thread computing architecture of a GPU introduces a lot of noise into the power side-channel, as each thread can be in a different phase of execution, generating a degree of randomness. We certainly see that GPU execution scheduling introduces some timing uncertainties in the power traces. In addition, the complexity of the GPU hardware system makes it rather difficult to obtain clean and synchronized power traces. In this paper, we propose an effective method to obtain clean power traces, and build a suitable side-channel leakage model for a GPU. We analyze AES on an NVIDIA TESLA C2070 GPU [18] and evaluate power traces obtained on a Keysight oscilloscope. CPA analysis using the acquired traces shows that AES-128 developed in CUDA on an NVIDIA C2070 GPU is susceptible to power analysis attacks.

The rest of the paper is organized as follows. In Section II, we provide a brief overview of the CUDA GPU architecture, including both software and hardware models. We also describe our CUDA-based AES implementation. In Section III, we describe the experimental setup used to collect power traces, and discuss difficulties we faced during designing our power analysis attack on GPUs compared to on other platforms. In Section IV, we discuss the construction of our power model, and present our attack results. Finally, we conclude the paper in Section V.

II. Targeted GPU Architecture and AES Implementation

A. GPU Hardware/Software Model

A GPU is designed to support execution of hundreds to thousands of concurrent threads run on hardware cores. Compared to CPU execution, GPU threads are designed to be lightweight, and can execute a program in parallel with low

context switching overhead. The massive number of threads are used to hide long latency operations.

CUDA [19] is a parallel computing platform and programming model developed by NVIDIA, which provides extensions to C/C++. In this work we are utilizing CUDA 6.5. From a hardware standpoint, the GPU is equipped with its own memory hierarchy that includes global memory, local memory, L1 and L2 caches, and registers. From the software standpoint, the same code, referred to as kernel code, is executed in parallel using a large number of threads. These threads are grouped into *blocks*, which are then clustered into a *grid*, and which are scheduled to run on the hardware cores. Each thread is indexed by its *thread id* and *block id*. A 2-D thread structure is shown in Fig. 1, where one block consists of 12 threads, and there are 6 blocks in the grid. The programmer specifies how many threads are needed to run the kernel and partitions the workload for those threads. To initiate execution, the data is first copied from the CPU memory to the GPU memory. Next, the kernels start execution on the GPU. After the kernels complete execution, the results are copied back to the CPU.

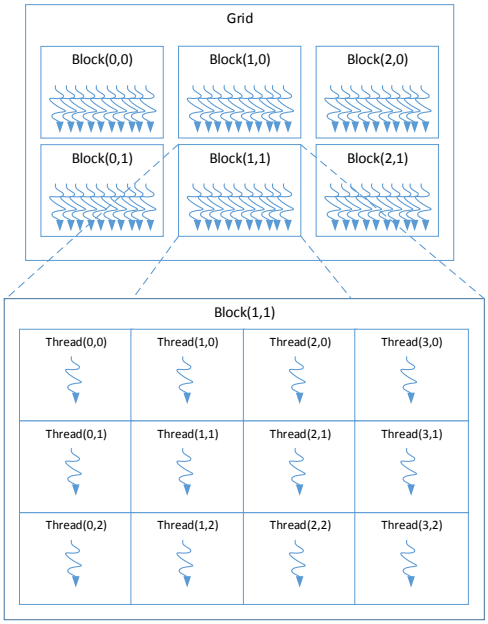

Fig. 1. A sample CUDA execution of threads and blocks in a single grid [20].

B. Target Device Model

In this study, we target a TESLA C2070 GPU, which has 14 steaming multiprocessors (SMs), where each SM has 32 CUDA cores, totaling 448 CUDA cores. These numbers will vary for different models of GPU. For memory, an off-chip global memory is shared across the SMs. An L2 cache is also shared, but it is on chip. An L1 cache, which serves as shared memory, is only shared by the CUDA cores within an SM.

Fig. 2 shows a block diagram of a single SM. It features 2 warp schedulers, a 3K 32-bit register file, 32 CUDA cores and other functional units. The CUDA cores are the basic computational unit on a GPU, and are used to execute the kernel. In one thread block, every 32 threads are grouped into a warp. If the number of threads in the block is not divisible by 32, the last warp will have less than 32 threads. Threads in the same warp are executed in a synchronized fashion on the CUDA cores of one SM, and share a common program counter. This provides us with some level of determinism during execution. It also eliminates the need to synchronize threads inside a single warp. Multiple blocks can reside on one SM concurrently. Typically, there are multiple warps in a single block. Whenever the warp in execution is stalled due to a data dependency or control hazard, the GPU's scheduler will dispatch another warp in the same block, or from another block, to the CUDA cores. Given the large size of the register file provided on the GPU, each thread has its own registers, we should not have to rename them or spill the other context to global memory. Therefore, the overhead of switching contexts on CUDA cores is minimized. As long as we have enough active warps, the CUDA core's pipeline can be kept full.

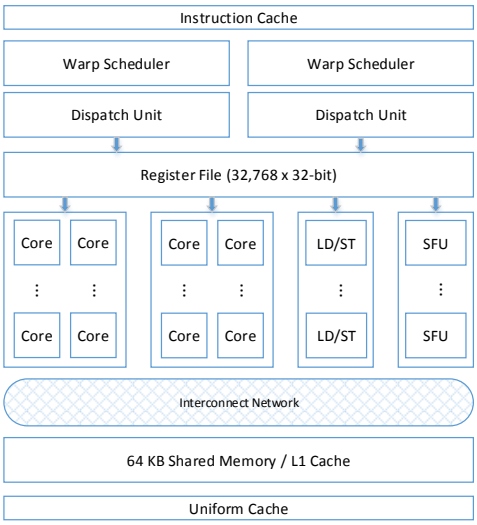

Fig. 2. Block diagram of one TESLA C2070 streaming multiprocessor [20].

To begin to launch a power attack on a GPU, we need to understand how work is dispatched to the CUDA cores. There are two schedulers in each SM, so two instructions can be executed at the same time. While prior generations of G20 and GT200 utilized a static round robin scheduler, the Fermi scheduling scheme has been reported to vary greatly from round robin. Further, NVIDIA has not publicly described the details of the scheme implemented in the Fermi. From a programmer's perspective, the scheduler's behavior needs to guarantee forward progress and avoid starvation, but it is difficult for us to assume more than this on Fermi, which makes our task of understanding the detailed behavior on the Fermi device challenging.

C. AES Implementation

In this paper, we implement a 128-bit ECB mode AES-128 encryption in CUDA based on the CUDA reference implementation by Margara [21]. Because the GPU's register width is 32-bit, the T-table version of the AES algorithm [22] is adopted. Each thread is responsible for computing one column of a 16-byte AES state, expressed as a 4×4 matrix, with each element a state byte. Four threads are needed to process a single block of data. Note here, the GPU thread block is different from AES data block, which is a 16-byte data block iteratively updated on each round, transforming the plaintext input to ciphertext output. Due to the ShiftRows operation, threads working on different columns share their computation results, and thus shared memory is used to hold the round state results, which facilitates communication between threads easily and efficiently. Since threads for one data block will be grouped into the same warp, there is no need to explicitly synchronize the threads.

Fig. 3 shows the round operations for one column running in a single thread. The initial round is simply an XOR of the plaintext and the first round key. There are nine middle rounds for the 128-bit AES encryption. Each thread takes one diagonal of the state as its round input, and maps each byte into a 4-byte word through a T-table look-up-table. These four 4-byte words are XORed together with the corresponding 4-byte round key bytes, and the results are stored in a column of the output state. The last round has no MixColumns operation, and so only one out of four bytes is kept after the T-table lookup.

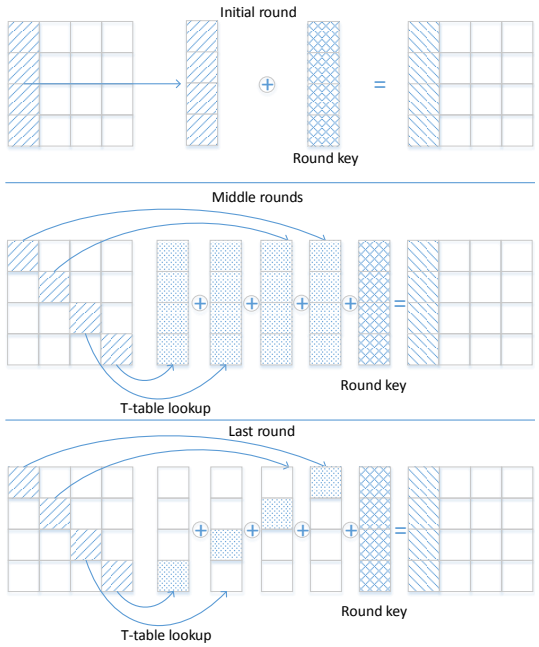

Fig. 3. The round operation running as one thread.

The T-tables are stored in the GPU's constant memory, which is cached to improve performance. The key scheduling is performed on the CPU, and then the expanded key is copied into GPU memory. According to the configuration of the TESLA C2070, there are at most 8 blocks and 48 warps residing together on one SM. The number of threads present in a single block is set to 192 $(48 \times 32/8)$, which fully utilizes the hardware. To perform an encryption, the grid size is determined by the number of plaintext blocks to be encrypted. In this work, 49,152 plaintext blocks are encrypted at one time, so the grid size is 1024 $(49152 \times 4/192)$ blocks, in order to achieve good occupancy on the GPU.

III. Power Leakage Acquisition

In our experiments, the TESLA C2070 GPU is hosted on the PCIE interface of a desktop server running Ubuntu. Fig.4 shows the experimental setup. In order to measure the power consumption of the GPU card, a 0.1 ohm resistor is inserted in series with ATX 12V power supply. The voltage drop on the resistor is measured by a Keysight MSOX4104A oscilloscope. The AES encryption runs on the server, providing an encryption service. The attacker sends plaintext over the network through TCP/IP. Upon receiving the data file, the server copies it to the GPU memory for encryption. The ciphertext is generated on the GPU, sent back to the server, and then returned to the attacker through the TCP connection.

During encryption, the oscilloscope records the power consumption and sends the measurement back to the attacker through the network. Since there is no GPIO (General-purpose input/output) or dedicated pins on GPU to provide a trigger signal to indicate the start or end of the encryption, the oscilloscope takes the rising edge of the power trace as the trigger signal. Because a power trace can be very noisy and its rising and falling edges are not distinct, it is challenging to always identify beginning of an encryption in a trace and the traces are not synchronized, like traces of an FPGA implementation.

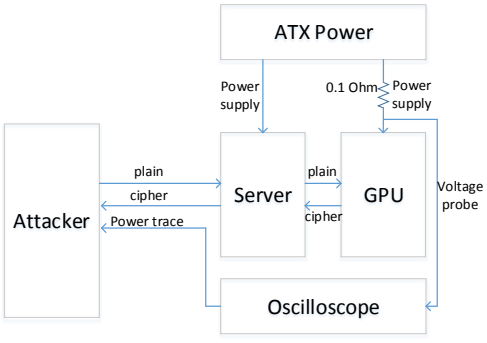

Fig. 4. The power measurement setup used in this work.

Power leakage acquisition on a GPU is performed very differently than the approaches used on MCUs, FPGAs and ASICs [10], [13], [12] for a number of reasons. *First*, our measurement point on the ATX power supply is *far away* from the GPU chip power supply. On the GPU card, there are many DC-DC units converting the 12V voltage into various other voltage values needed by the GPU. These power management

units contain many capacitors to supply a steady voltage, and can filter out fluctuations in the power consumption (exactly the kind of information side-channel attackers require). These capacitors act as a local energy reserve – when we need to provide a large amount of power in a short amount of time, current is drawn from the capacitors instead of the power supply, so the changes in terms of power consumption are not directly observable externally. What is worse, the DC-DC converter relies on switching circuits on and off at a high frequency, which generates a large amount of switching noise. This behavior can obscure any useful dynamic power signal.

Second, the measured total power consumption of the GPU card also contains power consumption of the cooling fan, off-chip memory, PCIE interface and many other auxiliary circuits. These unrelated power consumptions further add to the noise level.

Third, the ATX 12V output is not the only power supply for the GPU card. The PCIE interface can supply power (at most 75W) through its 12V and 3.3V power supply pins.

The last and most important issue is the parallel computing behavior of the GPU, which as discussed in Section II, may cause timing uncertainty in the power traces. The GPU scheduler may switch one warp to another at any time, and this behavior is not under programmer's control. Moreover, there are multiple streaming multiprocessors, each performing encryption concurrently and independently. As a result, these facts all pose great challenges for GPU side-channel power analysis.

In Section IV-B, we propose strategies to deal with these problems. The first and last problems are addressed by averaging each power trace to produce a single value to represent the power/energy consumption. The second problem is mitigated by eliminating the erroneous part of traces affected by computation of irrelevant factors. For the third problem, we have to collect more traces to increase the side-channel signal to noise ratio.

Fig. 5 shows the difference between an FPGA power trace [23] and a GPU power trace of an AES encryption. The FPGA power trace is much cleaner, with little noise. The ten rounds of AES can be clearly observed, identified by a steep voltage drop. For the GPU power trace, the power data is rather fuzzy because of the large amount of noise, and the voltage drop at the beginning is not nearly as pronounced. There is no sign of round operations from the GPU power trace. The spikes are caused by the DC-DC converter's switching activities. The trace is almost flat during encryption due to the GPU scheduler's attempts to keep the hardware busy all the time.

IV. Our CPA Attack

To launch a correlation power analysis attack on the GPU, we need to design a power model to predict power consumption. The power model should capture the dependency of the GPU's power consumption on the secret key information. Second, we need to sensitize the model to varying the input and generate corresponding power traces during encryption.

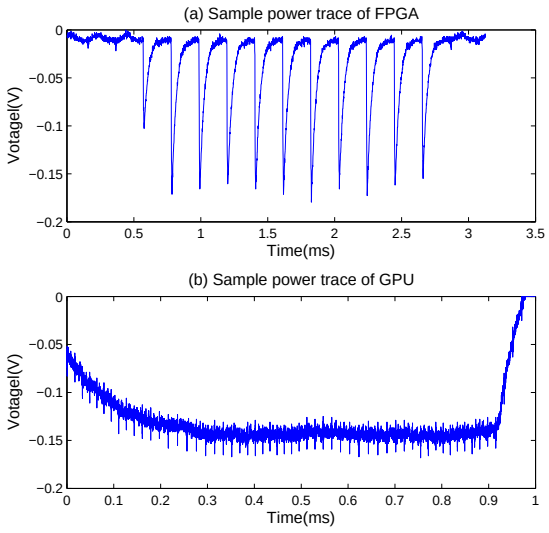

Fig. 5. Comparison of typical power traces between a FPGA and GPU.

Given the challenges presented in Section III, the power traces need to be collected and processed carefully in order to launch a successful side-channel power analysis attack.

A. Power Model Construction

In CMOS integrated circuits, the largest contributor to power consumption comes from power dissipation during switching, when an output transitions from a 0 to a 1 or from a 1 to a 0. During these transitions, capacitors are charged and discharged, which generates current flow, especially when driving nodes with large parasitic capacitances. Since register outputs need to be routed to many units, including ALUs, memory load/store units, and inputs of the register file, they can consume a large portion of the total dynamic power. Registers are also the most frequently switched component (besides the clock tree), making them a good target to build our power model on. Based on these facts, the Hamming distance between a register's values before and after a transition is chosen to drive our power model. The power model of our GPU becomes:

$$W = aH + b \tag{1}$$

where W is the instantaneous power consumption at any time point during an encryption, H is the Hamming distance of a selected register at that point in time, a is the unit power consumption of one state switch, and b is the power consumption due to other elements in the GPU (logic noises) and other physical noises. b is normally treated as a Gaussian distributed random variable, as has prior work [16].

For the attack, we choose the registers used in the last round operation of AES to build the power model. Since there is no MixColumns operation, every byte of the last round of the secret key determines part of the power consumption, and so is independent of other bytes. Since focus on the last round,

we only need the ciphertext and the key guess to predict the power consumption.

The CUDA PTX assembly code of the AES kernel is used to analyze the operations applied in the last round. The required steps for one state byte operation are shown in Fig. 6. First, a register is loaded with a byte of the input state. Based on the state value, the register is then loaded with a 4-byte T-table value, which are known constants. The T-table value is then ANDed with a mask to leave one byte untouched and other bytes zero, as shown in Fig. 3. Finally, the remaining byte is XORed with one corresponding byte of the last round key to produce one byte of the ciphertext. One thread processes all four state bytes, each with a different mask value and key byte position.

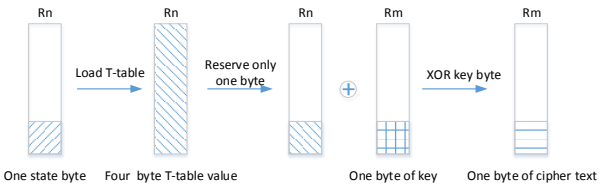

Fig. 6. Last round operation on registers for one state byte.

The pseudocode of these instructions for one AES state byte is shown below. The "LOAD" instruction looks up in the T-table and loads four bytes into register R_n. R_n is then ANDed with a mask 0xFF (or 0xFF00, 0xFF0000, 0xFF000000, depending the byte ordering). Finally, R_n is XORed with the round key R_m, resulting in one byte of ciphertext in R_m.

$$\begin{array}{llll} LOAD & R_n & [R_n] & \\ AND & R_n & R_n & 0X000000FF \\ XOR & R_m & R_m & R_n \end{array}$$

The corresponding Hamming distances of the three registers in these instructions can be calculated as below, where HW denotes the Hamming weight of a number, r_{n0} denotes the value of the selected state register before the "LOAD" instruction, and r_n is the register value after the T-table lookup. $r_n[0]$ holds the last byte of the 4-byte register value, and $r_n[3:1]$ holds the first three bytes.

$$\begin{aligned} For\ Load:\ & HW(r_n \oplus r_{n0}) \\ For\ AND:\ & HW(r_n[3:1]) \\ For\ XOR:\ & HW(r_n[0]) \end{aligned}$$

When ciphertext C is known (i.e., if we know the value of register R_m after the XOR is executed), then the state value before the XOR instruction (r_n) can be calculated using the ciphertext and a guessed key value. For one byte, this can be written as $r_n[0] = c \oplus k_g$, where the Hamming distance for the XOR instruction can be calculated. If we have ciphertext C and the guessed key byte, then the corresponding input state byte can be calculated directly from the AES algorithm (using an inverse of the substitution box operation, i.e., $r_{n0} = SBox^{-1}(c \oplus k_g)$. $r_n = T(r_{n0})$). Therefore the first and second Hamming distances above can also be calculated. All three Hamming distance expressions contain a key guess, explicitly.

If the correct key guess is used to calculate the Hamming distances, we should find significant correlation with the associated power consumption measurement [16]. The correlation power analysis (CPA) attack finds the correct key based on the guess that yields the maximum Pearson correlation between the power consumption trace and the predicted Hamming distance, i.e., $k_c = argmax(\rho_g)$, where:

$$\rho_g = \frac{cov(W, H_g)}{\sigma_W \sigma_{H_g}} = \frac{a\sigma_{H_g}}{\sigma_W} = \frac{\sigma_{H_g}}{\sqrt{a^2\sigma_{H_g}^2 + \sigma_b^2}} \qquad (2)$$

where σ_{H_g} is standard deviation of the selected register's Hamming distance at a time point for a key guess value k_g, and σ_W is the standard deviation of the corresponding measured power consumption.

With the parallel programming model of GPU and the uncertainty of warp scheduling, we choose to sum the three Hamming distances for one byte encryption as H_g in Equation (1). Accordingly, we take an average across all the sampling points for each power trace as W. We will discuss more about the power trace processing in the next section. This approach is a key difference between our CPA on a GPU and the previous CPA attacks on sequential computing platforms such as a CPU or MCU, or even a parallel computing platform with a fixed computing order such as a FPGA.

The secret round key length is 16-byte. We assume the Hamming distance of different key bytes are independent and have the same standard deviation. On the NVIDIA TESLA GPU used in this study, each thread processes a 4 byte column, independently, and therefore, the key bytes will be attacked one by one.

B. Power Trace Processing

Another challenge of power analysis on the GPU is the collection of clean power traces. During the development of our strategy, our first observation was that the power traces varied significantly across time at the beginning of our experiments. We needed to acquire a large number of traces using different plaintext inputs. The first few traces always had much lower power consumption. As the experiments progressed, the power consumption gradually grew, finally stabilizing at some point for the rest of the experiment. We found that this trend was caused by the GPU's cooling fan. A secondary effect can be that the rise in temperature increases leakage, but is less of a factor. At the beginning of our measurements, the temperature of the GPU is low, making the fan run slowly, which draws less power. Since the workload of the GPU is high during encryption, the GPU temperature rises, causing the fan to draw more power to cool the GPU. The fan power consumption is a substantial part of the measured power and contributes to

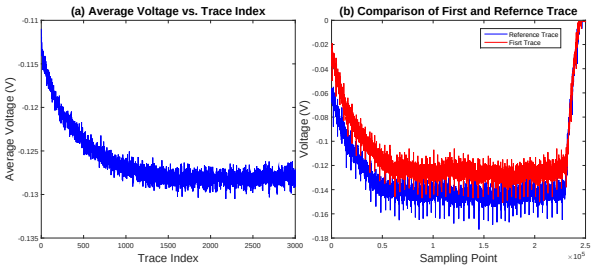

Fig. 7. Average power trends and comparison of the first and a reference trace.

variance (noise) in the power measurements. We need a way to address this.

To understand the effects of the cooling fan, we simply calculated the average power (presented in volts) of each power trace and present these in Fig. 7(a). The average voltage starts from -0.115V and drops to -0.13V after trace 1000, and stays at that level for the rest of the traces. Fig.7(b) shows the first measured voltage trace and a later reference trace. The voltages in the first trace are clearly higher than the reference trace. Note here, we measure the voltage after the resistor, so we get negative voltage values. The more negative the voltage value, the higher the power consumption. Since traces before trace 1000 are much more heavily influenced by the cooling fan (i.e., they have much wider variations), this data will be excluded when computing correlations in our attack approach.

In a serial AES encryption implementation, plaintext blocks are processed sequentially, and the timing of leakage points in the power trace (under a specific power model) can be precisely identified. However, the GPU accelerates the encryption by processing multiple plaintext blocks in parallel, and given our lack of control over warp scheduling, leakage points for each block occur at different times in the traces. As shown in Fig. 4, an FPGA power trace exhibits round operations clearly [23], and the leakage points can be easily determined. However, for the GPU power trace, nothing about the encryption details can be told at this point.

Our assumption is that when one thread on the GPU is executing the leaky instructions of the last round, the power consumption caused by the thread at that time correlates with the Hamming distances of the registers. However, the measured power consumption contains a lot of noise, both spatially and temporarily, due to the fact that a number of threads are executing other instructions, and that the threads are not necessarily synchronized. In our experiments, we measured N power traces, and each trace is sampled into T time points. Assuming for each trace that there are Q threads performing AES encryption. We model the power of one thread corresponding with its last round operations in one trace as:

$$P_{thread\_i}(t) = h(t - L_i) \times H_i + b_i(t)$$

where $1 \leq t \leq T$ is the sampling point, $1 \leq i \leq Q$ is the index of threads, H_i is the Hamming distance of registers in the last round for one thread, function $h(t)$ is the power consumption of a thread with unit Hamming distance starting at time "0". Since threads are scheduled to run at different time, L_i is the delay time relative with time "0". Then the power trace can be modeled as:

$$\begin{aligned} P_{trace}(t) &= \sum_{i=1}^{Q} P_{thread\_i}(t) \\ &= \sum_{i=1}^{Q} h(t - L_i) \times H_i + B(t) \end{aligned}$$

Because L_i is random and unknown to us, we compute the average power of each trace to represent the corresponding energy/average power consumption as:

$$W = \frac{\sum_{t=1}^{T} P_{trace}(t)}{T} = a \times \sum_{i=1}^{Q} H_i + b$$

where b is the average of $B(t)$, and a is the average of $h(t)$. This matches the power model we built in Sec IV-A, in which:

$$H = \sum_{i=1}^{Q} H_i. \tag{3}$$

Based on this analysis, we average each trace to represent its corresponding energy consumption. This processing method also solves the problems caused by DC-DC power management units, because the noise is also reduced significantly by averaging the trace.

C. Attack Results

First, to reduce the overhead of calculating multiple Hamming distances, we use a single value for the plaintext block data. Therefore, the Hamming distances for every thread are the same. Thus:

$$\begin{aligned} H &= Q \times H_i. \\ \sigma_H &= Q\sigma_{H_i} \end{aligned}$$

For the attack to succeed, a sufficient number of traces are needed so we can differentiate the right key byte from the wrong ones. For each byte of the key, we try all 256 possible candidates. With each value, we calculate the corresponding Hamming distances based on the power model described in Section IV-A for each trace, and then the correlation coefficient is computed for the average power and the Hamming distances. As a result, we have 256 correlation coefficients for all of the key byte guesses. The candidate with the highest correlation coefficient is our hypothetical key byte value. Fig. 8 shows the correlation coefficients of the 256 key byte candidates versus the number of traces used. Because a lower voltage value translates to higher power consumption, the correlation coefficient of the right key is negative. After analyzing about 10,000 traces, the right key stands out with the maximum correlation coefficient (the absolute value).

To extract the whole last round key, we use a brute-force approach, working byte by byte. We repeat the procedure

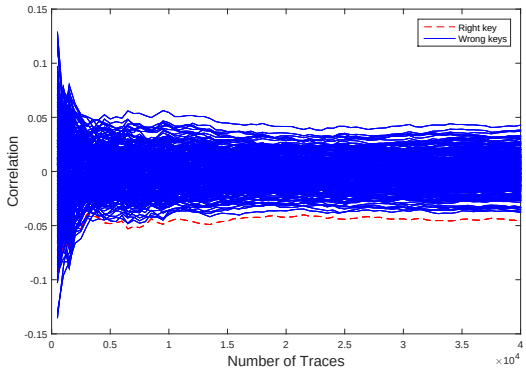

Fig. 8. Correlation between power traces and Hamming distances for all of the key candidates

above 16 times and obtain the complete last-round key. Fig. 9 shows the results from using 160,000 traces. For each trace, 49,152 blocks of plaintext are encrypted, and every plaintext block holds the same value to reduce the intermediate value computation complexity. For different traces, the plaintexts are set differently. In Fig. 9, the correlation coefficient of the right key byte value is marked with '*', and the largest correlation coefficient is marked with 'o'. For all the 16 bytes, the right key bytes have the highest correlation. However, we found that the highest correlation is still small, around -0.05, and the difference between the highest correlation coefficient value and the second highest coefficient value (for a wrong key guess) is not very large. The low correlation value is due to the fact that our power model only correlates with a small part of the average power consumption, and therefore the signal-to-noise ratio is really low. The small correlation difference is because the main component of the power model is linearly dependent on the key's Hamming weight. Those wrong key bytes that have similar Hamming weights as the right key also result in similar correlation coefficients. However, the right key can still be discerned with the given number of traces.

We next experiment with multiple different blocks of plaintext in one encryption. The results show that the last round key can still be retrieved in such cases. The power model will be scaled, accordingly. For example, for S different blocks of plaintext, 49,152 blocks are grouped into S equal-sized sets, and each set is designated a different plaintext block. To compute the correlation coefficients, the Hamming distances of the last round for the S plaintext blocks are computed independently, and then summed up as the final Hamming distance.

$$H = \frac{Q}{S}(\sum_{k=1}^{S} H_{setk})$$

$$\sigma_H = \frac{Q}{S}\sqrt{S\sigma_{H_{setk}}} = \frac{Q}{\sqrt{S}}\sqrt{\sigma_{H_{setk}}}$$

where H_{setk} is the last round registers' Hamming distances for one thread in the plaintext block set k, $1 \leq k \leq S$. We assume that each H_{setk} is independent, and has the same standard deviation. The standard deviation of the Hamming distances σ_H then becomes $\sqrt{S}$ times smaller. From Equation (2), since σ_b will remain the same, the correlation coefficient ρ should be lower, and more traces would need to be captured to recover the right key.

Our results show that all 16 bytes of the last round key can be extracted when there are 1-to-4 sets of plaintext blocks, when equipped with 160,000 traces. When the number of sets increases to 8, only 15 bytes are recovered; and 14 bytes for 16 sets. We conclude that in one encryption, when there is variation in the data values in different blocks of plaintext, many more traces are needed, and thus the attack is more challenging.

By processing the GPU power traces and constructing the appropriate power models, our attack succeeds in cracking the AES GPU implementation and obtains the last round key. With key scheduling, the entire AES key will be recovered.

In addition to power measurements, electromagnetic emission could be a more practical alternative which does not require physical contact with the targeted system. An attack analysis based on EM signals, however, will be very similar to the power analysis procedure presented in this work. Therefore, our method is readily applicable to EM side channels. Our work on side-channel attacks for GPUs can also be extended to other parallel computing platform such as multi-core CPUs and DSPs. The timing uncertainty and parallelism of instruction execution can be addressed by averaging the power trace and building the corresponding power model presented in this paper.

V. Conclusions and Future Work

In this paper, we present the first side-channel power analysis on a GPU AES implementation. The setup of the power consumption measurement of a GPU is described in details. The various challenges of power analysis on a GPU are highlighted. To overcome these difficulties, we have proposed different strategies to process the power traces for a successful correlation power analysis. The corresponding power model is built based on the CUDA PTX assembly code. The attack results show that a GPU, a representative complex parallel computing system, is vulnerable to a side-channel power analysis attack.

As the first side-channel power analysis of GPU cryptographic implementation, this work will pave the way for a rich set of future work. We plan to extend our work to measure a GPU's electromagnetic emission signals in a non-invasive fashion, and evaluate our current work equipped with EM signals. We will also experiment with different GPU devices and multi-core CPUs. We plan to further study the statistical characteristics of the attack, including the relationship between the attack success rate and the implementation details, like the number of blocks and the underlying GPU models. The attack method can also be further improved, and countermeasures on GPUs against side-channel power analysis attacks will be investigated.

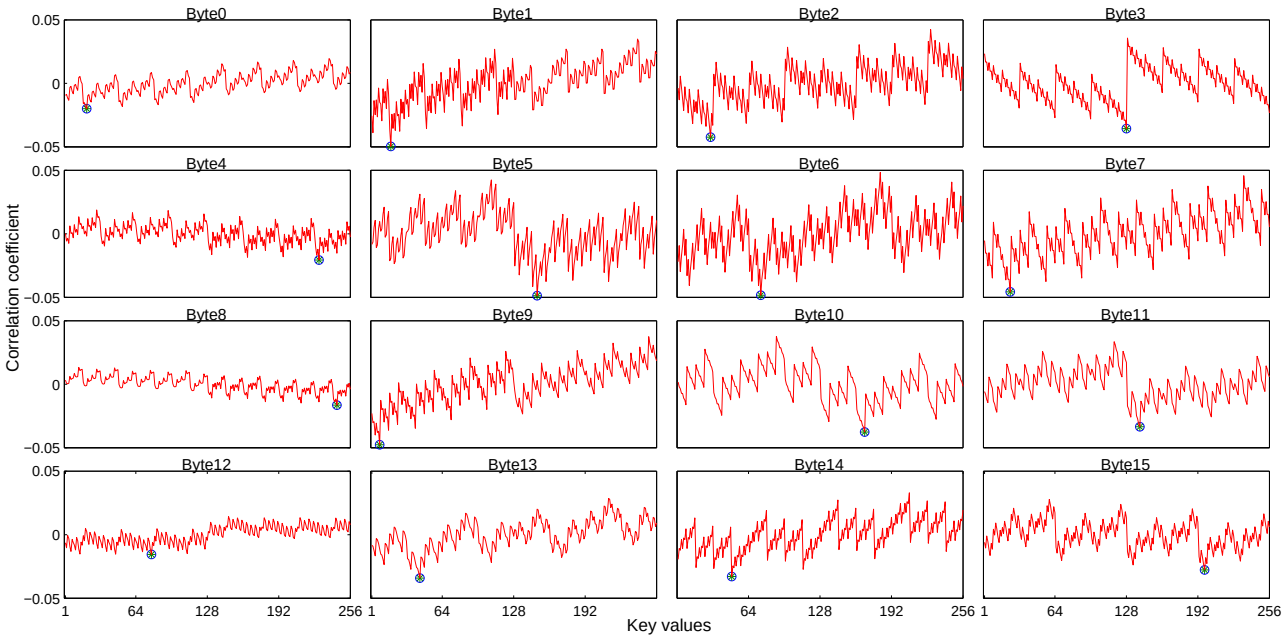

Fig. 9. Our CPA attack results.

REFERENCES

[1] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[2] W.-m. Hwu, *GPU Computing Gems Emerald Edition*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[3] K. Iwai, T. Kurokawa, and N. Nisikawa, "AES encryption implementation on CUDA GPU and its analysis," in *Int. Conf. on Networking & Computing*, Nov. 2010, pp. 209–214.

[4] S. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," in *IEEE Int. Conf. on Signal Processing & Communications*, Nov. 2007, pp. 65–68.

[5] A. Cohen and K. Parhi, "GPU accelerated elliptic curve cryptography in GF(2m)," in *IEEE Int. Midwest Symp. on Circuits & Systems*, Aug. 2010, pp. 57–60.

[6] R. Di Pietro, F. Lombardi, and A. Villani, "CUDA leaks: information leakage in GPU architectures," *preprint arXiv:1305.7383*, July 2013.

[7] M. J. Patterson, "Vulnerability analysis of GPU computing," Ph.D. dissertation, Iowa State University, 2013.

[8] J. Danisevskis, M. Piekarska, and J.-P. Seifert, "Dark side of the shader: Mobile GPU-Aided malware delivery," in *Information Security & Cryptology*, Oct. 2014, vol. 8565, pp. 483–495.

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

[10] A. Moradi and G. Hinterwälder, "Side-Channel security analysis of ultra-low-power FRAM-based MCUs," *Proc. Int. WkShp on Constructive Side-channel Analysis & Secure Design*, Mar. 2015.

[11] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *Cryptographic Hardware & Embedded Systems*, 1999, pp. 144–157.

[12] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-Analysis attack on an ASIC AES implementation," in *Int. Conf. on Info. Tech.: Coding & Computing*, vol. 2, Apr. 2004, pp. 546–552.

[13] P. Luo, Y. Fei, X. Fang, A. A. Ding, M. Leeser, and D. R. Kaeli, "Power analysis attack on hardware implementation of MAC-Keccak on FPGAs," in *Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*, Dec. 2014, pp. 1–7.

[14] S. B. Örs, E. Oswald, and B. Preneel, "Power-analysis attacks on an FPGA–first experimental results," in *Cryptographic Hardware & Embedded Systems*, 2003, pp. 35–50.

[15] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology*, Dec. 1999, pp. 388–397.

[16] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware & Embedded Systems*, 2004, vol. 3156, pp. 16–29.

[17] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *Cryptographic Hardware & Embedded Systems*, 2008, pp. 426–442.

[18] T. NVIDIA, "C2050/C2070 gpu computing processor," 2010.

[19] C. Cuda, "Programming guide," *NVIDIA Corporation, July*, 2012.

[20] N. Leischner, V. Osipov, and P. Sanders, "Nvidia fermi architecture white paper," 2009.

[21] P. Margara, "engine-cuda, a cryptographic engine for cuda supported devices," 2015. [Online]. Available: https://code.google.com/p/engine-cuda/

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

[23] T. Swamy, N. Shah, P. Luo, Y. Fei, and D. Kaeli, "Scalable and efficient implementation of correlation power analysis using (GPUs)," in *Workshop on Hard. & Arch. Support for Sec. and Priv.*, 2014, p. 10.