POSTER: A Hardware Fingerprint Using GPU Core Frequency Variations

Fengjun Li
Department of EECS
University of Kansas
fli@ku.edu

Xin Fu
Department of ECE
University of Houston
xfu8@central.uh.edu

Bo Luo
Department of EECS
University of Kansas
bluo@ku.edu

ABSTRACT

Hardware primitives provide significant promises to support cryptographic primitives and security mechanisms against various forms of compromises. In this work, we study the intrinsic hardware characteristics of modern graphics processing units (GPUs) due to random manufacturing variations, and exploits the inherent randomness to generate device-specific signatures. In particular, we present a novel GPU-based hardware fingerprint scheme to generate a unique, stable, physically unclonable, unpredictable, and random bit string from the inherent hardware features of a general purpose GPU (GPGPU). The generated fingerprint can be used to implement a physically unclonable function (PUF), and thus to create a trusted computing environment with GPUs as the trust anchor.

1. INTRODUCTION

Billions of electronic devices have been deployed at all scales throughout our everyday life. When the mobile and embedded devices become ubiquitous, the high inter-connectivity makes them more accessible to adversaries and thus more exposed to many attacks including invasive attacks, side-channel attacks, and physical attacks. Classic cryptographic primitives such as device identification and authentication are one of the fundamental enabling technologies for protection. A widely adopted approach is to associate secure operations with a device-specific secret and store the secret key in non-volatile memory such as EEPROM to enable cryptographic primitives such as encryption or digital signature. Obviously the security of this type of approaches relies heavily on the secrecy of the cryptographic key, which unfortunately is difficult to uphold when facing various attacks. For instance, the leakage of secret keys from the remote HTTPS web servers is the main cause of the Heartbleed incident which was reported in April 2014. This problem leads to the development of hardware primitives as a promising mechanism to support secure operations.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
ACM 978-1-4503-3832-5/15/10.
DOI: http://dx.doi.org/10.1145/2810103.2810105.

As an innovative hardware primitive, a physically unclonable function (PUFs) derives a secret from physical characteristics of a hardware and implements a function $\mathcal{F}$ to map any external stimuli or challenge C to a response R. The secret is *volatile* such that it is only available when the device is running and using the secret, and thus is impossible for an adversary to obtain or duplicate. Pappu et al. introduced the concept of PUF and presented an optical PUF design based on illumination key pairs to generate authentication tokens [5]. After that, various designs have been proposed in many applications including device identification and authentication, IP protection and anti-counterfeiting, secrete key generation, distribution and storage, etc.

In this project,we present a design based on a hardware fingerprint of commodity graphics processing units (GPUs). Such PUFs supporting only a small subset of challenges (or one in our case) are known as Physically Obfuscation Keys (POKs). We study the intrinsic hardware characteristics, i.e., the core frequency variations of modern GPUs, and generate a unique, GPU-specific signature from the inherent randomness introduced by manufacturing variations. The hardware fingerprint is directly derived from and implemented inseparably on the GPU of physical devices, which is unique, uncloneable, and difficult to read out by invasive attacks. Since GPUs are recently widely used in many devices such as workstations and personal computers, mobile phones, embedded systems, game consoles, etc., the GPU-enabled solution does not introduce additional hardware and implementation costs, comparing with other common examples such as SRAM-PUF, Butterfly PUF and Coating PUF.

2. BACKGROUND

GPU Architecture. The key component of a typical GPU is the in-order streaming multiprocessor (SMX). Each SMX includes 192 single-precision CUDA cores, 64 double-precision units (DP Unit), 32 special function units (SFU), and 32 load/store units (LD/ST). All these execution units will be evenly distributed into four execution clusters. In this project, we study the NVIDIA GPU Kepler Architecture which includes 15 SMX and applies CUDA programming model. In CUDA, the GPU executes highly-parallel kernel functions. The kernel is composed of a grid of light-weighted threads; a grid is divided into a set of blocks; each block is composed of hundreds of threads. Threads are distributed to SMXs at the granularity of blocks.

Threads in the SMX execute on the single instruction multiple data (SIMD) model. A number of individual threads (i.e. 32 threads in Nvidia GPU) from the same block are grouped together, called warp. In the pipeline, threads within a warp execute the same instruction but with different data values. Each SMX interleaves multiple warps at cycle-by-cycle basis. At every cycle, an instruction warp that is ready for execution is selected and issued by a warp scheduler, and all threads belonging to that warp start the execution in an execution cluster simultaneously. The execution of a branch instruction in the warp may cause warp divergence when some threads jump while others fall through at the branch. Threads in a diverged warp have to execute in serial fashion which causes multiple lanes to be idle in the SIMD pipeline. Each warp has its own stack in the branch unit recording the reconvergence PC (RPC) and active mask (used to describe the active threads in the warp) to handle the warp divergence. The load/store instruction may cause the off-chip memory access that can last hundreds of cycles, and a long latency memory transaction from one thread would stall all threads within a warp.

Process Variations. Process variations (PV) are a combination of random effects (e.g. due to random dopant fluctuations) and systematic effects (e.g. due to lithographic lens aberrations) that occur during transistor manufacturing. Random variations refer to random fluctuations in parameters from die to die and device to device. Systematic variations refer to layout-dependent variations which cause nearby devices to share similar parameters. Among the design parameters, effective channel length (L_{eff}) and threshold voltage (V_{th}) are two key parameters subject to large variations [9]. As process technology keeps scaling down, the increasing process variations (PV) have become a growing threat to processor design and fabrication [6]. PV is the divergence of device parameters from their nominal values, which is caused by the challenging manufacture process at very small feature technologies. PV induces delay variations among circuit paths, and this impact is further exacerbated in GPUs which contain tremendous amount of parallel critical paths [7, 4].

Generally, the random and systematic components have equal variances for both V_{th} and L_{eff} [1, 3]. GPU contains tremendous amount of parallel paths to deliver high computing throughput, and is quite sensitive to process variations. In order to afford a greater number of threads executing simultaneously in the SMX, the number of CUDA cores continuously increases in recent GPU product generations. For example, there are total *2880* CUDA cores in Nvidia's GPU Kepler architecture, therefore, they exhibit substantial frequency variations. In our preliminary experiments, we observe that the ratio of frequencies between the fastest and the slowest core in a GPU chip can reach as high as 2.2.

3. GPU FINGERPRINTING

We present a hardware *fingerprint* from the physical features of a GPU. This fingerprint is a binary string of N bits (in our initial design, $N = 256$), with the following features: (1) *unique* and *stable* for each GPU, (2) *physically unclonable* from hardware manufacturing perspective, (3) *unpredictable* so that it could be used as a secret, (4) *random* to provide maximum entropy. The entire fingerprint generation process, including core selection, frequency measurement and digitalization, and fingerprint assembling, is completely performed within GPU using auxiliary circuits. The area and power overhead of adding such auxiliary circuits is negligible to the entire GPU chip. In this section, we introduce the design of a workable GPU fingerprint: we first present the on-chip frequency measurement for each GPU CUDA core, and then further convert it to binary digits, to be used as the fingerprint.

GPU Core Frequency Measurement. As the first step of this project, we perform the *online GPU core frequency measurement*. Recently, time-to-digital converters (TDC) has been widely used to for precise measurement of time intervals or precise conversion of time interval to digital data [2, 9, 10]. It can be integrated into each GPU CUDA core during the chip fabrication to measure the core clock cycle time, thus, the frequency at run time. There are various methodologies to measure the time intervals in TDC depending on the requirements on the measurement resolution. For example, the coarse counter method [2] works well at the nanosecond resolution level, while the "fine" measurement [9, 10] provides much better resolution but smaller measuring range. Generally, the GPU CUDA core runs at GHz level, and the frequency measurement should have the strong capability to recognize the core frequency variations across the GPU chip which will largely affect the robustness of our key generation. We thus choose the "fine" measurement method whose resolution can achieve as high as 1 picosecond [9, 10]. Note that the TDC is usually composed of inverters, which is quite simple to be implemented in the integrated circuit technology, and has negligible area and power overhead to the GPU chip.

Fingerprint Generation. The next step is to convert the measured frequency of each GPU core into binary digits. The goal is to ensure that each bit (b_i) has an equal probability of 0 or 1: $P(b_i = 0) == P(b_i = 1) == 0.5$. Ultimately, this leads to a completely random binary fingerprint, which provides maximum entropy – providing lower collision probability with the same fingerprint length. Empirically, due to process variations, the actual frequency of the CUDA core is considered to follow normal distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

where μ is the designated operation frequency of the GPU Core, e.g. for a 3GHz GPU, $\mu = 3G$. The standard deviation σ is determined by the processing technology, i.e., whether the chip is fabricated under 32nm or 11nm technology. The cumulative distribution function (CDF) of a normal distributions is represented as:

$$F(x) = \frac{1}{2}(1 + \mathsf{erf}(\frac{x-\mu}{\sqrt{2}\sigma})) \tag{2}$$

where $\mathsf{erf}()$ is the Gauss error function. The inverse of the cumulative distribution function (CDF) is the *quantile function*, denoted as $x = F^{-1}(p) = \mu + \sqrt{2}\sigma \cdot \mathsf{erf}^{-1}(2p-1)$. That is, given a probability p, $F^{-1}(p)$ returns the corresponding threshold x so that $\Pr(X \leq x) = p$.

To generate an n-bit code from each GPU core, $2^n - 1$ thresholds is needed to separate the core frequency spectrum into 2^n brackets. The thresholds are denoted as $\{T_1, ..., T_{2n-1}\}$, so that a frequency bracket ($\mathbf{B}_i$) is the interval $[T_{i-1}, T_i]$. Note that we have two special thresholds $T_0 = -\infty$ and $T_{2n} = \infty$. To ensure a uniform distribution of the code (or, 0.5 probability of 1 and 0 at each bit), the integral of the probability distribution function over each interval should be identical, that is:

$$\int_{T_{i-1}}^{T_i} f(x)dx = F(T_i) - F(T_{i-1}) = \frac{1}{2^n} \tag{3}$$

With the quantile function, we have: $T_i = F^{-1}(i/2^n)$. Unfortunately, the quantile function of a normal distribution cannot be expressed in closed form in elementary functions. That says, we cannot directly compute $T_i = F^{-1}(p)$ for a given p. There are several different numerical approximations for the normal distribution CDF or the quantile function. Tools/tables are available to compute/check percentiles for *standard normal distribution* $(\Phi^{-1}(p))$. For a general normal distribution, $F^{-1}(p)$ could be calculated with: $F^{-1}(p) = \mu + \sigma\Phi^{-1}(p)$. For instance, we want to encode 2 bits from each CUDA core, four intervals are used: $[-\infty, \mu-0.674\sigma]$, $[\mu-0.674\sigma, \mu]$, $[\mu, \mu+0.674\sigma]$, $[\mu+0.674\sigma, \infty]$. With the 4 brackets defined as above, encoding is straight forward. For instance, when the measured frequency $Fq_{core_i} \in [-\infty, \mu - 0.674\sigma]$, the 2-bit fingerprint segment generated by Core i is "00". Similarly, $[\mu - 0.674\sigma, \mu]$ corresponds to "01", $[\mu, \mu + 0.674\sigma]$ corresponds to "10", and so on. We also XOR the n-bit string from each core with the last n bits of the core ID. This will not change the "good" properties of the fingerprint (e.g., randomness), however, it mitigates the possible derandomization effect caused by frequency degradation.

In a GPU chip, we extract n bits from each CUDA core. To obtain an N-bit fingerprint, we select $m = \frac{N}{n}$ CUDA cores for fingerprint generation. Initially, we set $N = 256$ (AES key length), $n = 2$, and $m = 128$. At each CUDA core, circuits sre designed to compare measured frequencies with pre-set thresholds, and emit n fingerprint bits. Output from the measurement and encoding mechanism for each core is assembled (concatenated) to generate the fingerprint. The design of the digital circuits for encoding and fingerprint assembly is straightforward, hence, we omit the technical details here. It is worth pointing out that, as we have claimed before, the area and power overhead of adding such auxiliary circuits is negligible to the entire GPU chip.

Core Selection. As we have discussed, m (e.g., 128) out of 2880 CUDA cores are selected to each emit an n-bit fingerprint segment, so that an N-bit fingerprint is generated from each GPU. The selection of the m CUDA cores is a nontrivial problem. In particular, process variations also contain the systematic effects. It implies that the nearby cores have high possibility to exhibit similar frequency. When one core is used for the key generation, its nearby cores should not be further chosen to maintain the randomness of the selection. In the initial design, we evenly distribute the m cores into each GPU SMX. Inside the SMX, we further evenly distribute the number of cores into each execution cluster. In other words, two cores (when m is set as 128) are randomly selected from each execution cluster to produce the fingerprint in our investigated NVIDIA GPU Kepler Architecture. In order to keep a record of the cores used for the key generation, we attach an SRAM-based buffer to each GPU SMX, which contains 192 bit with each bit describing the selection of its corresponding CUDA core.

4. CONCLUSION AND FUTURE WORK

In this poster, we introduce a hardware signature based on GPU core frequency variation. We present a mechanism to extract bits from inherent hardware features of GPU cores. We further generate a unique, physically unclonable, unpredictable and random bitstring from the extracted bits. The bitstring is then used as a hardware fingerprint of the GPS.

The extracted fingerprint cannot be used directly for authentication – although fingerprint is inherently uncloneable, anyone who has access to the fingerprint can fabricate a device that contains a same fingerprint in memory to impersonate the original device. For authentication purpose, we will further extend the proposed POK to generate multiple challenge response pairs (CRPs) similar to [5, 8], where each response is unique to an external challenge. Meanwhile, GPU cores are suffering the wear-out effects during runtime execution, which could cause frequency degradation and affect the key stability. Another future work is to efficiently avoid the frequency degradation of the selected cores, and to develop a protocol to handle fingerprint update.

5. ACKNOWLEDGEMENTS

Fengjun Li was supported in part by NSF EPS-0903806, University of Kansas GRF-2301075, and KU RIC Strategic Initiative Grant INS-0073037. Xin Fu was supported in part by NSF CCF-1320730, NSF CCF-1351054. Bo Luo was supported in part by NSF CNS-1422206, NSF IIS-1513324, NSF OIA-1308762, and KU GRF-2301876.

6. REFERENCES

[1] A. Agrawal, A. Ansari, and J. Torrellas. Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip edram modules. In *HPCA*, 2014.

[2] J. Kalisz. Review of methods for time interval measurements with picosecond resolution. *Metrologia*, 41(1):17, 2004.

[3] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas. Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *DSN*, pages 1–11, 2012.

[4] J. Lee, P. Ajgaonkar, and N. S. Kim. Analyzing throughput of gpgpus exploiting within-die core-to-core frequency variation. In *IEEE ISPASS*, pages 237–246, April 2011.

[5] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297(5589), 2002.

[6] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. Eval: Utilizing processors with variation-induced timing errors. In *MICRO*, pages 423–434. IEEE, 2008.

[7] S. Seo, R. G. Dreslinski, M. Woh, Y. Park, C. Charkrabari, S. Mahlke, D. Blaauw, and T. Mudge. Process variation in near-threshold wide simd architectures. In *DAC*, pages 980–987, 2012.

[8] G. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *ACM/IEEE DAC*, June 2007.

[9] Z. Xu, M. Miyahara, and A. Matsuzawa. A 1 ps-resolution integrator-based time-to-digital converter using a sar-adc in 90nm cmos. In *NEWCAS*, 2013.

[10] Z. Xu, M. Miyahara, and A. Matsuzawa. Picosecond resolution time-to-digital converter using $g_m - c$ integrator and sar-adc. *Nuclear Science, IEEE Transactions on*, 61(2):852–859, 2014.