COMPUTERS & SECURITY 62 (2016) 33–48

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

On-demand bootstrapping mechanism for isolated cryptographic operations on commodity accelerators

Yonggon Kim [a], *Ohmin Kwon* [a], *Jinsoo Jang* [b], *Seongwook Jin* [a], *Hyeongboo Baek* [a], *Brent Byunghoon Kang* [b,*], *Hyunsoo Yoon* [a]

[a] *Division of Computer Science, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, South Korea*

[b] *Graduate School of Information Security, School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, South Korea*

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form 30 May 2016
Accepted 27 June 2016
Available online 1 July 2016

Keywords:
Secure systems
Trusted computing technology
Trustworthy execution
GPU security
GPGPU
Cryptographic key protection
SMM

ABSTRACT

General-Purpose computing on a Graphics Processing Unit (GPGPU) involves leveraging commodity GPUs as massively parallel processing units. GPGPU is an emerging computing paradigm for high-performance and data-intensive computations such as cryptographic operations. Although GPGPU is an attractive solution for accelerating modern cryptographic operations, the security challenges that stem from utilizing commodity GPUs remain an unresolved problem. In this paper, we present an On-demand Bootstrapping Mechanism for Isolated cryptographic operations (OBMI). OBMI transforms commodity GPUs into a securely isolated processing core for various cryptographic operations while maintaining cost-effective computations. By leveraging System Management Mode (SMM), a privileged execution mode provided by x86 architectures, OBMI implements a program and a secret key into the GPU such that they are securely isolated during the acceleration of cryptographic operations, even in the presence of compromised kernels. Our approach does not require an additional hardware-abstraction layer such as a hypervisor or micro-kernel, and it does not entail modifying the GPU driver. An evaluation of the proposed OBMI demonstrated that even adversaries with kernel privileges cannot gain access to the secret key, and it also showed that the proposed mechanism incurs negligible performance degradation for both the CPU and GPU.

1. Introduction

Cryptography has become an essential component in modern computer systems. As the amount of data requiring protection has increased due to the growing importance of security and privacy, the heavy computational workload of cryptographic operations has also become a challenging problem. To alleviate the performance bottleneck affecting modern cryptography, several previous works have suggested harnessing many-core accelerators such as Graphics Processing Units (GPUs), which can be used as massively parallel architectures. Recent GPUs offer significant improvements in throughput and performance-per-watt compared to commodity CPUs (Abe et al., 2012). Leveraging such benefits, several researchers have

* *Corresponding author.*
E-mail address: brentkang@kaist.ac.kr (B.B. Kang).
http://dx.doi.org/10.1016/j.cose.2016.06.006

shown that versatile modern cryptographic operations can be accelerated efficiently using commodity GPU devices (Harrison and Waldron, 2009; Lee et al., 2015; Manavski, 2007; Wang et al., 2014; Zheng et al., 2014). Furthermore, research shows that GPU-accelerated cryptography can be a cost-effective solution to real-world cryptographic implementations such as the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) (Jang et al., 2011).

Unfortunately, GPUs have several security problems. A recent study shows that GPU data stored in GPU memory can be retrieved from different processes, because the GPU does not flush its memory after its termination (Lee et al., 2014; Pietro et al., 2016). Even when implementing an appropriate flushing mechanism for the GPU, malicious users with kernel privileges can easily access the GPU device memory through MMIO (Maurice et al., 2014). Various memory-disclosure attacks (Heartbleed; Blass and Robertson, 2012) and kernel-compromising attacks show that the kernel-enforced security mechanisms can be bypassed. If an attacker can manipulate the kernel or the driver, the secret key can be stolen whenever it is exposed in the GPU device memory.

To protect the secret key under a broad range of GPU vulnerabilities and threats to the underlying host system, we propose a low-cost key-protection mechanism for general-purpose computing on a GPU (GPGPU). We leverage System Management Mode (SMM) and an SMM-based handler function to implement secure key management, along with a bootstrapping mechanism that enables the key-protected execution of GPU-accelerated cryptography. SMM is a privileged execution mode for x86 architectures offered by commodity CPUs. In SMM, only the authorized handler – the so-called System Management Interrupt (SMI) handler – can be executed. Other processes, including malicious processes, cannot be executed in SMM. Because the OS kernel and any malicious processes are unaware of the SMM execution, we can implement an SMI handler that transparently manages the GPU device.

By implementing a simple bootstrapping mechanism within the SMI handler, we securely upload the secret key into the GPU cache. Unlike the GPU device memory, even privileged CPU processes cannot access the GPU cache. There are multiple types of GPU caches, and the GPU constant cache can be easily utilized as a key storage by exclusive use of the GPU constant memory for the secret key. When the SMI handler clears the remaining footprint of the secret key in the device memory, the secret key is isolated within the GPU constant cache. Similarly, we can also isolate the entire GPU code at the GPU instruction cache to prevent any control-flow modifications of the GPU program.

The main challenge to implementing the bootstrapping mechanism arises because, although the SMI handler can securely access the GPU device, it is unable to utilize the existing GPU driver and OS kernel. This is because the OS kernel and GPU driver are suspended in SMM. Modern GPUs are complex, and the GPU driver is responsible for controlling the underlying hardware. Without the help of the GPU driver, engineering efforts to implement the bootstrapping process increase dramatically. To overcome this challenge, we split all the GPU control logics needed for bootstrapping mechanism into two classes of tasks: security-sensitive tasks, and security-insensitive tasks. Consequently, security-insensitive tasks can be handled by the existing GPU driver and rudimentary OS functionality. We devised two consecutive steps for the bootstrapping process: one step in normal CPU mode (i.e., protected mode), and the other in SMM. By delegating most of GPU control logic (i.e., security-insensitive tasks) to the first step, we can significantly reduce the complexity of the control logic required for the SMI handler.

With our bootstrapping mechanism, the secret keys used by GPU-accelerated cryptography are not exposed to attackers. Before bootstrapping, the secret keys are stored in the protected memory space, which is only accessible by the SMI handler. In SMM, the secret key and GPU program are safely uploaded to the GPU caches. Although processor environments isolated in SMM are only momentarily utilized until the upload is complete, we can preserve the confidentiality of the uploaded key and program during the acceleration of cryptographic operations, since GPU caches are inaccessible from any host processes. If the uploaded GPU program terminates, all content in the GPU caches is invalidated. Thus, any subsequent GPU program cannot retrieve the secret keys.

To minimize the performance degradation incurred by the proposed mechanism, we devise several optimizations for efficient bootstrapping. Furthermore, to demonstrate the feasibility of our bootstrapping mechanism, we implemented a prototype using a commodity CPU and GPU. Our prototype includes GPU-accelerated RSA and AES cryptographic operations, and results in minimal performance loss. We carefully evaluated the proposed mechanism in terms of its security, in order to confirm that it is robust even to attackers with kernel privileges.

Our bootstrapping mechanism is advantageous in many ways: (i) transparency – we leverage existing GPU drivers and operating systems, without the need to modify them; (ii) a small TCB – our mechanism adds only a few hundred lines of code for the SMI handler, significantly minimizing the size of the TCB; (iii) compatibility – our mechanism is based on commodity hardware; and (iv) simplicity and speed – the additional code required for bootstrapping is relatively simple compared to other approaches (Sani et al., 2014; Yu et al., 2015).

In particular, our work makes the following contributions:

1. To our best knowledge, we are the first to suggest using the GPU cache to store cryptographic keys, such that the commodity GPU can safely accelerate cryptographic operations without revealing sensitive information.
2. We suggest an SMM-based bootstrapping mechanism, referred to hereafter as the On-demand Bootstrapping Mechanism for Isolated cryptographic operations (OBMI). OBMI securely uploads the secret key into the GPU cache in an on-demand fashion.
3. OBMI includes mechanisms for checking the integrity of the accelerated GPU code. We propose a code-verification mechanism to guarantee that only reliable code can utilize the GPU device for cryptographic key-related operations.
4. We implemented a prototype using a commodity Nvidia GPU. Our evaluation shows that OBMI incurs minimal performance overhead and is scalable to multiple secret keys.
5. By exploring several possible attacks, we show the robustness of OBMI. We demonstrate that our approach enables secure operations on commodity GPUs without increasing the TCB of the system or degrading its overall performance.