

An Evaluation of Intel Software Guard Extensions Through Emulation

Avaliação do Intel Software Guard Extensions via Emulação

Marco Aurélio Spohn^{1*} and Mateus Trebien¹

Abstract: The Intel Software Guard Extensions (SGX) technology, recently introduced in the new generations of x86 processors, allows the execution of applications in a fully protected environment (i.e., within enclaves). Because it is a recent technology, machines that rely on this technology are still a minority. In order to evaluate the SGX, an emulator of this technology (called OpenSGX) implements and replicates the main functionalities and structures used in SGX. The focus is to evaluate the resulting overhead from running an application within an environment with emulated SGX. For the evaluation, benchmark applications from the MiBench platform were employed. As performance metrics, we gathered the total number of instructions and the total number of CPU cycles for the execution of each application with and without OpenSGX.

Keywords: trusted execution environment — software guard extensions — emulator

Resumo: Ao permitir a execução de aplicações em um contexto totalmente protegido (i.e., dentro de enclaves), amplia-se as possibilidades para as novas gerações de processadores Intel da família x86 com a extensão Software Guard Extensions (SGX). Por se tratar de uma tecnologia recente, as máquinas que contam com essa tecnologia ainda são minoria. Objetivando avaliar o SGX, utilizou-se um emulador dessa tecnologia denominado OpenSGX, o qual implementa e reproduz as principais funcionalidades e estruturas utilizadas no SGX. O enfoque consistiu em avaliar o overhead, em termos de processamento, resultante da execução de uma aplicação em um ambiente com o SGX emulado. Para a avaliação, empregou-se aplicações de benchmark da plataforma MiBench, modificando-as para compatibilizar a execução em enclaves no OpenSGX. Como métricas de desempenho, coletou-se o número total de instruções e o número total de ciclos de CPU para a execução completa de cada aplicação com e sem o OpenSGX.

Palavras-Chave: ambiente de execução confiável — software guard extensions — emulador

¹ Federal University of Fronteira Sul, Brazil

*Corresponding author: marco.spohn@uffs.edu.br

DOI: <http://dx.doi.org/10.22456/2175-2745.77654> • Received: 25/10/2017 • Accepted: 12/02/2018

CC BY-NC-ND 4.0 - This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

1. Introdução

As tecnologias de *hardware* voltadas à computação confiável disponibilizam recursos e mecanismos para a criação e manutenção de ambientes de execução confiável (*Trusted Execution Environment*, TEE). Considera-se, de modo geral, que a execução é confiável em ambientes onde o controle sobre o sistema é pleno [1]. Recursos de TEE não constam, habitualmente, em processadores populares (e.g., família *Intel* x86). No entanto, a *Intel* introduziu, em agosto de 2015, a extensão denominada *Software Guard Extensions* (SGX) a partir da sexta geração da família x86, propondo-se como uma plataforma padrão de TEE nos processadores *Intel*.

Com o SGX, possibilita-se o isolamento da aplicação no momento de sua execução, utilizando-se estruturas a nível de *hardware* denominadas de enclaves. Estes são implementa-

dos para garantir a integridade e privacidade das informações durante todo o processamento, até mesmo em eventuais comprometimentos do sistema operacional ou do hipervisor. A proposta traz implicações diretas ao paradigma de computação nas nuvens (i.e., *cloud computing*), possibilitando ofertar garantias de integridade e privacidade às aplicações em níveis não atingíveis até o momento [2].

Enquanto o SGX não estiver amplamente disponível, pode-se empregar um emulador como o OpenSGX [3] para reproduzir as diversas funcionalidades do SGX. Os recursos oferecidos pelo OpenSGX propiciam, além de uma prévia experiência com um sistema de enclaves, a possibilidade de uma avaliação *a priori* sobre o custo adicional em processamento imposto pelo SGX.

Este artigo apresenta uma análise, sobretudo em termos de desempenho, do SGX via emulação, tendo-se como principais

métricas o número de instruções e o número de ciclos de CPU adicionais quando as aplicações são executadas em enclaves. A avaliação é conduzida utilizando-se aplicações padrões de *benchmark*.

O restante deste artigo é organizado da seguinte forma: A Seção 2 apresenta, resumidamente, o SGX. A Seção 3 apresenta alguns trabalhos relacionados e a Seção 4 apresenta o contexto da avaliação. A Seção 5 apresenta os resultados obtidos e sua análise. Por fim, a Seção 6 apresenta as conclusões.

2. Intel Software Guard Extensions

Para apresentar as principais particularidades do *Software Guard Extensions*, parte-se de uma explanação fundada desde a criação de um enclave até a finalização de uma aplicação (processo ilustrado na Figura 1). Durante a inicialização, uma área contígua de memória, denominada *Processor Reserved Memory* (PRM), é reservada para prover todas as funcionalidades e restrições ao sistema de enclaves [4][5]. A PRM possui garantia de segurança contra qualquer tipo de acesso, seja este do Sistema Operacional (SO) ou do *Hypervisor*.



Figure 1. Etapas do ciclo de vida de um enclave (Adaptado de: [6])

A criação de um enclave parte da execução pelo SO da instrução *ECREATE* [4], que tem como primeira ação reservar uma página na *Enclave Page Cache* (EPC), estrutura utilizada para alocação de memória para o enclave e demais informações de controle. A EPC, interna à PRM, é dividida em páginas para viabilizar múltiplos enclaves. Apesar do SO ser o responsável pelo gerenciamento de toda a memória da máquina, a memória reservada aos enclaves é acessível apenas via instruções específicas do SGX, garantindo-se a confidencialidade dos conteúdos nos enclaves [6]. Apesar de armazenados na mesma EPC, a interação direta entre enclaves também é protegida por mecanismos de autenticação.

Os mecanismos de verificação de segurança do SGX impedem que agentes externos possam burlar e acessar áreas da memória que são restritas. O *Enclave Page Cache Map* (EPCM), interna à PRM, armazena informações referentes a cada um dos enclaves na EPC. Outra estrutura de controle, o *SGX Enclave Control Store* (SECS), armazenada em uma página da EPC mantém os metadados de cada enclave. Há uma página SECS para cada enclave criado na EPC, com sua

alocação ocorrendo antes de qualquer alocação de páginas para o enclave.

Para carregar código e dados para o enclave, utiliza-se a instrução *EADD* que também valida as informações que advêm da memória externa aos enclaves. A inicialização do enclave ocorre via instrução *EINIT*, mas seguindo um procedimento diferenciado: empregando um enclave privilegiado, fornece-se um *token* à instrução *EINIT*, permitindo-a marcar a SECS do enclave como inicializado [6]. Esse enclave privilegiado, denominado *Launch Enclave* (LE), possui uma criptografia especial especificada pela Intel, atuando como um autenticador para a instrução *EINIT* [7].

As estruturas *Thread Control Structure* (TCS) e *State Save Area* (SSA) tem função relevante no controle da execução dos códigos dos enclaves, bem como no tratamento às interrupções [8]. Cada TCS, armazenada em uma página da EPC, é disponibilizada para apenas um processador lógico e contempla informações que permitem a transição entre a execução de código interno e externo aos enclaves. O SSA serve como um tipo de *backup* da computação do enclave até o momento de uma falha ou interrupção.

Enquanto executa código no enclave, o processador está no modo *enclave mode* [4]. Em caso de exceções (*i.e.*, falhas ou interrupções), a mudança do modo de execução ocorre através de uma saída assíncrona denominada *Asynchronous Enclave Exit* (AEX), garantindo-se que se mantenha consistente o estado do enclave antes de passar o controle ao SO. Após a execução do tratador de exceções do SO, a execução interrompida do enclave é retomada via instrução *ERESUME* [4][8].

A mudança do modo de execução também pode ser síncrona, através da execução da instrução *EEXIT*, resultando na finalização da execução do enclave com sucesso. A instrução induz o processador a sair do modo enclave, restaurando-se o conteúdo dos registradores ao estado quando da chamada da instrução *EENTER* [6].

O procedimento de remoção de um enclave é inicializado com a execução da instrução *EREMOVE* pelo SO. O enclave só é definitivamente removido após a eliminação de qualquer estrutura ou referência que ainda se tenha a ele.

2.1 Atestado entre enclaves

O processo de atestado entre dois enclaves surge da necessidade da troca de segredos entre dois enclaves, garantindo-se também que ambos executem em um ambiente SGX [9]. A geração de atestados é ramificada em dois modos: a local, onde os dois enclaves executam na mesma máquina/plataforma; e a remota, quando os dois enclaves executam em máquinas distintas.

2.1.1 Atestado local

Nessa modalidade, um dos enclaves realiza um “desafio” (*i.e.*, *challenge*) ao enclave que deseja atestar pertencer à mesma plataforma [8]. O enclave desafiado usa a instrução *EREPOR*T para criar e enviar uma espécie de relatório acompanhado de uma *MAC tag*, uma espécie de certificado de autenticação da

mensagem, gerada sobre o relatório e usando um algoritmo AES 128-bit a partir de uma *Report Key* (somente compartilhada entre o enclave destino e a implementação SGX) [6]. O enclave desafiante, utilizando-se da instrução EGETKEY, obtém a chave simétrica para verificar a autenticidade do MAC recebido. Uma ilustração desta troca de mensagens visando a autenticação é vista na Figura 2.

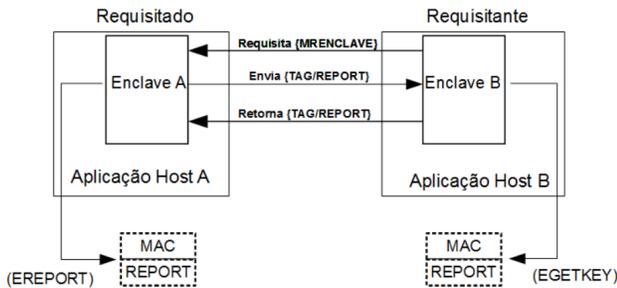


Figure 2. Processo de atestado local entre os enclaves A e B

2.1.2 Atestado remoto

O atestado remoto facilita a execução de enclaves remotamente, como no caso da computação em nuvem, com a garantia de que a execução ocorra em uma plataforma SGX [7]. Neste caso, ao contrário do atestado local, emprega-se mecanismos de certificação que incluem chaves assimétricas [6]. O processo (ilustrado na Figura 3) inicia com o desafio de um enclave em uma plataforma remota (enclave B). O enclave B solicita o atestado ao enclave A que, como na autenticação local, procede com a chamada da instrução EREPORT para a criação do *Report* a ser enviado a B. Contudo, o *Report* passará ainda por uma nova etapa, com auxílio de um enclave especial do SGX, denominado *Quoting Enclave* (QE) [10]. O QE certifica o *Report*, assinando-o com uma chave privada EPID, obtida pela entidade certificadora da Intel, denominada *Intel Provisioning Service* (IPS) [11]. O *Report* assinado recebe a denominação *Quote*. O enclave A encaminha o *Quote* ao enclave B que, utilizando-se da instrução EGETKEY e consultando a IPS, obtém as derivações de chave necessárias a verificar a autenticidade do enclave A e, conseqüentemente, atestando sua execução em uma plataforma SGX [7].

3. Trabalhos relacionados

Zhao *et al.* [12] conduziram uma avaliação do SGX no sistema operacional *Windows Server™2016 Technical Preview 5* executando em uma máquina equipada com um processador Intel™Core i5-6200U. Os resultados da avaliação apontam que o SGX precisa evoluir para superar alguns desafios, destacando-se os seguintes: (a) o SGX degrada excessivamente o desempenho das aplicações, devendo-se, sobretudo, ao fato de não se poder manipular chamadas de sistema dentro dos próprios enclaves, bem como ao custo muito alto no acesso à memória do enclave; (b) aplicações precisam ser refatoradas para tirar vantagem do SGX, pois se requer que o

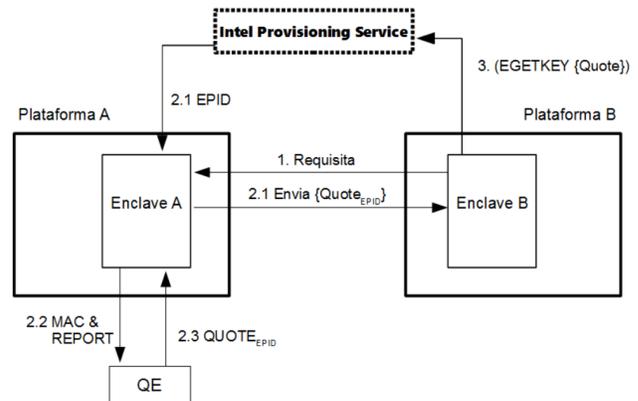


Figure 3. Atestado remoto entre dois enclaves em plataformas distintas

código de qualquer aplicação seja segmentado em um componente não confiável e outro componente confiável.

Gjerdrum *et al.* [13] realizaram uma avaliação do SGX em uma máquina equipada com processador Intel™Core i5-6500 executando o sistema operacional Linux Ubuntu 14.04 atualizado com um módulo SGX de código aberto. A avaliação esteve focada em identificar peculiaridades de desempenho do SGX quando da sua utilização em serviços na nuvem. Identificou-se que o custo é semelhante tanto na entrada como na saída de enclaves, estando esse diretamente relacionado ao tamanho do *buffer* passado como argumento no procedimento de transição (*i.e.*, entrada ou saída do enclave). Observou-se que o custo aumenta vertiginosamente quando o *buffer* ultrapassa a barreira dos 64 KBytes. Da avaliação resultaram cinco recomendações ao se empregar o SGX em serviços na nuvem: (a) tornar o código do enclave auto-suficiente, minimizando assim a cópia de dados para áreas externas ao enclave; (b) limitar o tamanho do enclave em 64 KBytes; (c) baseando-se em conhecimento prévio do padrão de consumo e acesso à memória das aplicações, otimizar o gerenciamento da memória dos enclaves no módulo SGX do *kernel* do sistema operacional; (d) quando possível prever o uso futuro de determinados enclaves, possibilitar o provisionamento antecipado dos recursos necessários à sua instanciação; e, por último, (e) considerar a granularidade de isolamento exigido pela aplicação, tendo-se como parâmetro o fato que uma granularidade menor resulta em custos adicionais na criação dos enclaves.

Costan *et al.* [14] apresentam um modelo de programação (vide Figura 4), denominado *Sanctum*, com as mesmas promessas do SGX, isolando-se módulos de *software* que executam concorrentemente e compartilhando recursos. O protótipo desenvolvido opera sobre um *Rocket RISC-V core*, com um conjunto de instruções para computador de propósito geral (*ISA*) e uma implementação de código aberto, deixando mais acessíveis os elementos adotados como mecanismo de segurança. O projeto também prevê que tal extensão possa ser adaptada a outros núcleos de processadores, devido ao fato de não ser modificado nenhum bloco de construção principal da CPU. A

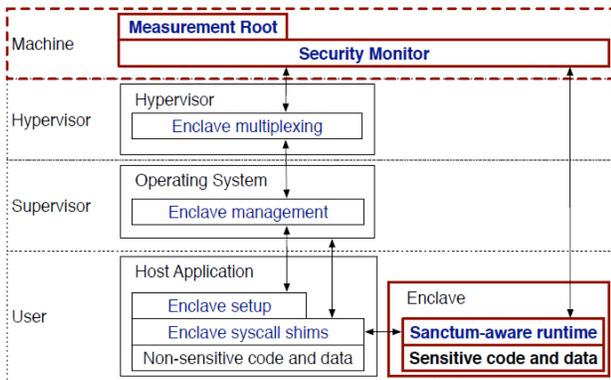


Figure 4. Modelo de programação *Sanctum* (Reproduzido de: [14])

grande mudança do *Sanctum* é a adoção de um componente de *software* confiável, denominado *Security Monitor*, que substitui o microcódigo SGX e é protegido por ser executado no nível de privilégio mais alto, o nível de máquina do *RISC-V*.

Evyushkin *et al.* [15] apresentam o *Iso-X*, um *framework* que possui por objetivo prover isolamento para a execução envolvendo partes críticas de segurança de aplicações, mesmo que sob presença de aplicações não confiáveis executando no mesmo ambiente da plataforma. O isolamento é implementado através da criação e gerenciamento de compartimentos alocados para hospedar trechos críticos de código e dados das aplicações, obtendo-se um isolamento a nível de páginas de memória, alocação flexível e uma TCB de baixa complexidade. Requer a adição mínima de *hardware* e um pequeno número de novas instruções ISA para gerenciar os compartimentos, com mudanças mínimas no Sistema Operacional. O *Iso-X* promete maior flexibilidade de memória que o SGX, permitindo tanto o particionamento do espaço de memória disponível como também o aumento dinâmico dos compartimentos.

Arnautov *et al.* [16] apresentam o *Scone*, objetivando uma mescla entre o *Docker* e o SGX. O *Docker* oferece uma modalidade de virtualização a nível de sistema operacional baseado em *containers*, propiciando a execução de vários sistemas (*i.e.*, diversos *containers*) de forma isolada em apenas um único *host* (*i.e.*, sobre o mesmo *kernel* da máquina hospedeira). O SGX, por sua vez, é aplicado aos *containers*, garantindo-se sua integridade e confidencialidade sobre um *kernel* Linux.

4. Contexto de avaliação

4.1 QEMU

O QEMU é um emulador e virtualizador genérico de código aberto, possibilitando emular um processador via um mecanismo de tradução dinâmica [17]. Além da emulação do processador, o QEMU permite ainda a emulação completa de um sistema, oferecendo dois modos de operação [18]:

- **Emulação completa:** possibilita a emulação de um sistema completo, incluindo todos os processadores e

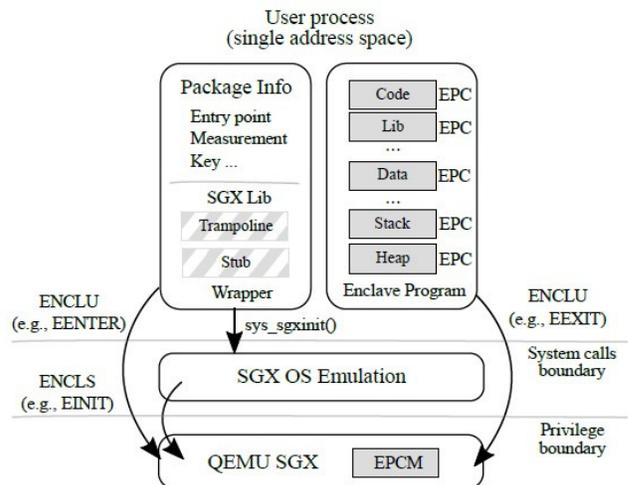


Figure 5. Estrutura de um programa carregado em enclave no OpenSGX (Reproduzido de: [3])

periféricos suportados.

- **Emulação modo usuário:** permite a execução de programas compilados para uma CPU específica em uma CPU distinta.

Quando o processador emulado é idêntico ao da máquina hospedeira, o QEMU oferece um mecanismo de aceleração, permitindo-se a execução do código emulado diretamente na CPU hospedeira [18]. A tradução dinâmica corresponde à conversão em tempo de execução de instruções de uma CPU distinta à CPU hospedeira.

4.2 OpenSGX

O OpenSGX [3] é uma extensão do QEMU, desenvolvida em conjunto pela *Georgia Institute of Technology* e a *Korea Advanced Institute of Science and Technology* (KAIST), que se propõe a oferecer as principais funcionalidades do SGX. A Figura 5 ilustra, resumidamente, a relação entre enclaves, OpenSGX e o QEMU. O *Wrapper* inclui as bibliotecas do SGX, comunicando-se com o *SGX OS Emulation* através de instruções no nível de usuário (*i.e.*, ENCLU). Chamadas às instruções privilegiadas do OpenSGX (*i.e.*, ENCLS) só podem ser realizadas pela camada *SGX OS Emulation*, concretizando-se assim a comunicação com a camada QEMU SGX onde, efetivamente, executa-se as operações sobre os enclaves.

A geração de atestados entre plataformas no OpenSGX difere do modelo adotado no SGX. Enquanto no SGX o atestado baseia-se na premissa do EPID (*i.e.*, um esquema de assinatura de grupo), o OpenSGX adota apenas um esquema de assinatura pública (*i.e.*, o RSA); sobretudo devido a ausência de uma entidade certificadora (CA).

A principal modificação inicial no processo de codificação de uma aplicação no OpenSGX, além da inclusão da biblioteca *sgxlib*, consiste em substituir a função *main()* (nativa à linguagem C) pela função *enclave_main()*. Esta declaração serve

para demarcar o ponto de entrada padrão do código que será carregado no enclave.

Em síntese, o OpenSGX é uma plataforma para teste e avaliação das funcionalidades do SGX via emulação, o que não só permite explorar as funcionalidades do SGX mas também explorar os níveis inferiores de implementação do próprio SGX (o que fica oculto em uma plataforma real deste). Além do recurso de depuração intrínseco à parceria entre QEMU e OpenSGX, há as funcionalidades de monitoramento de desempenho. Jain *et al.* [3] demonstraram a viabilidade do OpenSGX em avaliar os potenciais impactos do SGX no desempenho de uma aplicação baseada na rede de anonimato Tor [19].

4.3 MiBench

Objetivando avaliar as funcionalidades desenvolvidas pelo OpenSGX e, concomitantemente, obter o número de instruções e de ciclos de CPU necessários à execução de uma aplicação, adotou-se um conjunto de aplicações de *benchmark* disponibilizadas na plataforma *MiBench* [20]. Sua escolha também se deve ao fato dela ser orientada à avaliação de sistemas móveis e embarcados, apresentando similaridades às aplicações que se espera executar em enclaves, contemplando-se diversas cargas de trabalho com ampla variação de operações de controle e de acesso à memória.

O *MiBench* oferece um conjunto de *benchmark* dividido em seis categorias; no entanto, selecionaram-se quatro dessas: *Network*, *Office*, *Security* e *Telecom*. Cada uma dessas categorias visa, através da execução das aplicações, estressar o *hardware* focando em categorias distintas de áreas computacionais. De forma resumida, a Tabela 1 lista as categorias que tiveram um de seus algoritmos de *benchmark* selecionados, apresentando-se uma breve descrição dos algoritmos contemplados na categoria.

Nesse contexto, selecionou-se uma dentre cada categoria de aplicações do *MiBench*, levando-se em consideração a viabilidade à portabilidade da aplicação para execução no OpenSGX. A Tabela 2 apresenta as principais informações acerca das aplicações selecionadas.

5. Resultados e análise

Para avaliar o *overhead* que a adoção do SGX agrega ao processamento das aplicações selecionadas, elas foram executadas no OpenSGX com entradas menores (*i.e.*, *small*) e maiores (*i.e.*, *large*), conforme descrito na Tabela 2. As aplicações foram executadas em uma máquina com processador *Intel Core i7-3630QM 2,4GHz*, memória RAM de 8GB a 1333MHz e o Linux *Ubuntu 15.04 64-bits*.

Para avaliação, coletou-se duas métricas: o total de instruções e o total de ciclos de CPU para a execução completa da aplicação, ambos obtidos com e sem a utilização do OpenSGX (*i.e.*, com e sem enclaves). Apesar do OpenSGX oferecer outras métricas úteis à avaliação de desempenho (*e.g.*, mudanças de contexto), optou-se por essas duas porque, de uma forma geral, capturam o custo total de execução. Todavia, tem-se

como desvantagem a dificuldade em se discriminar todas as fontes responsáveis pelo custo total.

Os resultados estatísticos apresentados correspondem a dez execuções para cada um dos algoritmos de *benchmark*. Para garantir aferições estatísticas aceitáveis, determinou-se o Coeficiente de Variação (CV) das amostras. Como disposto nas Figuras 6 e 7, englobando todas as amostras, os maiores coeficientes de variação encontrados foram de 0,034% para o total de instruções e 1,76% para o total de ciclos, demonstrando a pequena variação e, portanto, a retidão das amostras obtidas nos testes.

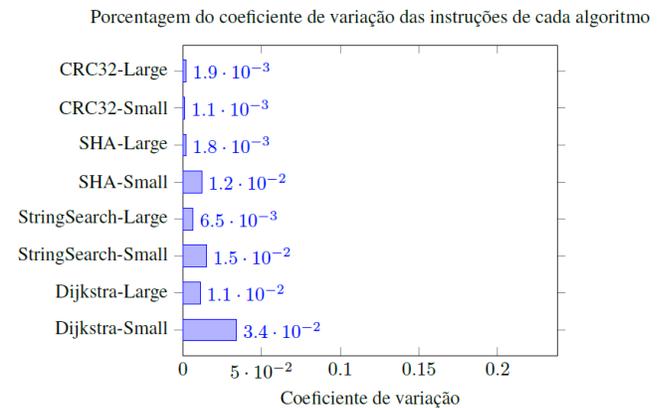


Figure 6. Coeficiente de variação das amostras de instruções das aplicações no OpenSGX

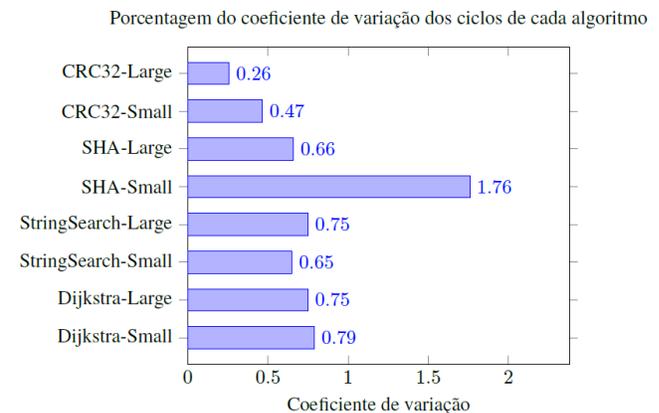


Figure 7. Coeficiente de variação das amostras de ciclos das aplicações no OpenSGX

Com relação ao número total de instruções (Figura 8), ressalta-se que cada uma das aplicações de *benchmark* explora características de processamento distintas; portanto, antecipa-se uma diferença de valores entre cada aplicação, não havendo qualquer correlação sobre as estatísticas entre as aplicações.

Como esperado, há um aumento significativo no número de instruções quando as aplicações são executadas no OpenSGX. A Figura 9 apresenta os resultados desse aumento em percentual (dado em milhares). Esse comparativo demonstra um aumento mínimo de 2.000% até um máximo próximo dos 100.000%, observado com a execução do algoritmo SHA

Table 1. Categorias de *benchmarks* no *MiBench*.

Categoria	Descrição
<i>Office</i>	Algoritmos para manipulação de textos visando representar funcionalidades de dispositivos de impressão, fax e processadores de texto.
<i>Network</i>	Algoritmos representando processadores embarcados em dispositivos de redes como <i>routers</i> e <i>switches</i> .
<i>Security</i>	Engloba algoritmos de criptografia, descriptografia e funções <i>hash</i> .
<i>Telecomm</i>	Algoritmos de codificação e decodificação de voz/som, análise de frequência e algoritmos de <i>checksum</i> .

Table 2. Algoritmos/aplicativos de *benchmark* selecionados

Algoritmo	Descrição	Entradas
Dijkstra	Menor caminho em grafos.	Small: 20 amostras de 100 vértices. Large: 100 amostras de 100 vértices.
<i>StringSearch</i>	Identifica a ocorrência de um subconjunto de palavras em um conjunto de <i>strings</i> .	Small: 57 palavras e 57 <i>strings</i> . Large: 1332 palavras e 1466 <i>strings</i> .
SHA	Cálculo da função <i>hash</i> SHA utilizando como entrada um arquivo de texto.	Small: 16 linhas, totalizando 312 mil caracteres. Large: 1408 linhas, totalizando mais de 3 milhões de caracteres.
CRC32	Computa um <i>checksum</i> sobre um arquivo de som no formato <i>Pulse Code Modulation</i> (PCM).	Small: 16-bit PCM de 1,4MB de tamanho em disco. Large: 16-bit PCM de 26,6MB de tamanho em disco.

com entrada *large*. Ressalta-se que, para a execução de cada instrução original, o SGX requer um número significativo de instruções extras à manutenção das propriedades do enclave (*e.g.*, criptografia e descriptografia interna à CPU para cada instrução executada e/ou dado transferido entre a memória e a CPU).

Devido ao fato do OpenSGX ser implementado como uma camada no *user-mode* do QEMU, há também um acréscimo nas instruções necessárias para a efetiva execução das instruções OpenSGX à custa do processo de tradução binária destas às nativas do QEMU e, posteriormente, à efetiva execução das mesmas no processador. No SGX nativo, essa camada necessária à emulação inexistente, não acarretando esse custo adicional. Segundo os desenvolvedores da plataforma OpenSGX, os principais influenciadores de tal aumento no número de instruções são as chamadas à biblioteca *sgxlib*, bem como o suporte às chamadas de sistema (*e.g.*, chamada para criação de enclaves).

Efetivamente, essas chamadas podem ser custosas pois englobam a alocação da área reservada à EPC, antes ainda da

alocação e criação do enclave, podendo ainda resultar em mais custos caso se aumente o espaço reservado à EPC durante a execução, operação que é suportada pela implementação do OpenSGX. De forma análoga, o custo total é maior com o aumento, durante a execução, do número de páginas reservadas a um enclave. Isso inclusive pode ser uma das justificativas para o aumento no número de instruções das aplicações SHA e CRC32, tendo em vista que os dados de entrada, contidos em arquivos locais, apresentam maior tamanho efetivo e, conseqüentemente, ocupando um maior número de páginas da EPC. Somando-se a isso o custo do carregamento e inicialização do enclave, gera-se um maior número de execuções da medição do enclave (definida pelo campo *measurement* no Intel SGX), que também é responsável por assegurar que todo o carregamento de código e dados, assim como a inicialização, seguiu os parâmetros esperados para a execução em uma plataforma de enclaves.

Embora as execuções tenham resultado em um grande acréscimo no número de instruções de cada aplicação no OpenSGX, essa informação isolada não revela todas as par-

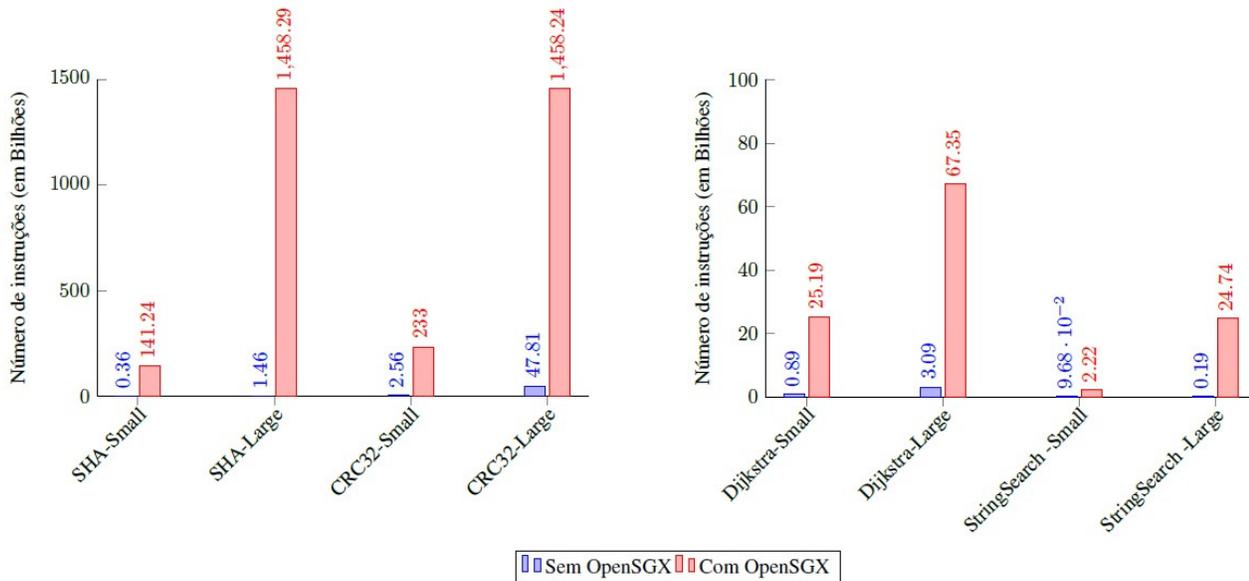


Figure 8. Número total de instruções com e sem o OpenSGX

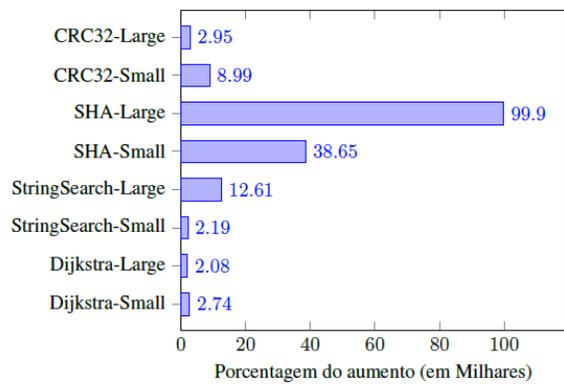


Figure 9. Porcentagem (em milhares) no aumento do número de instruções com o uso do OpenSGX

particularidades do custo de execução em enclaves, pois uma sequência maior de instruções pode não ser sinônimo de um aumento efetivo no número de ciclos ou mesmo no tempo total de execução. Nesse contexto, as Figuras 10 e 11 apresentam o número efetivo de ciclos de CPU consumidos por cada uma das aplicações avaliadas. A primeira constatação visível é a correlação entre o número de instruções e o número de ciclos consumidos entre as aplicações, demonstrando, a princípio, que o aumento no número de instruções com o uso do OpenSGX acarreta, de mesma forma, o aumento no número de ciclos necessários às execuções das aplicações em enclaves.

Como observado por Jain *et al.* [3], o aumento no número total de ciclos parece advir, principalmente, do uso da biblioteca *sgxlib*, necessária para a adaptação das aplicações ao formato de enclave e do suporte às chamadas de sistema implementadas na camada de emulação do OpenSGX, prin-

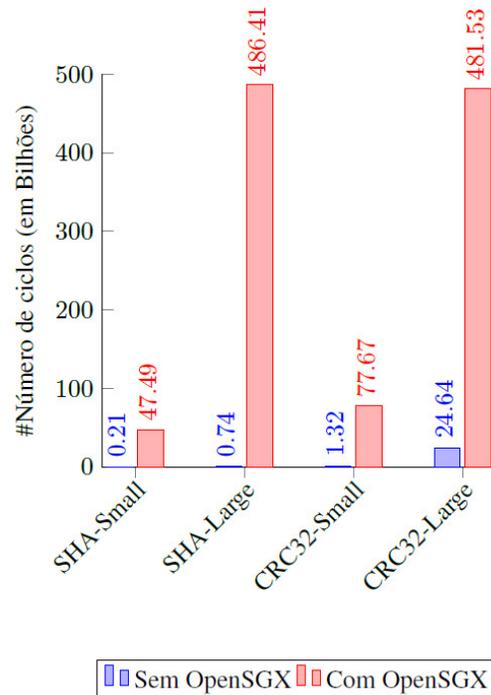


Figure 10. Número de ciclos de CPU dos algoritmos SHA e CRC32 com e sem o OpenSGX

cipalmente relacionadas ao processo de instanciamento da área reservada às páginas EPC dos enclaves e, posteriormente, toda a etapa de carregamento de código e dados, incluindo-se a geração de chaves criptográficas e todas as outras certificações necessárias para confirmar a inicialização segura do enclave.

Quando analisadas em conjunto, as informações apresentadas na Figura 8 e nas Figuras 10 e 11 evidenciam que o crescimento entre as aplicações com entradas *Small* e *Large*

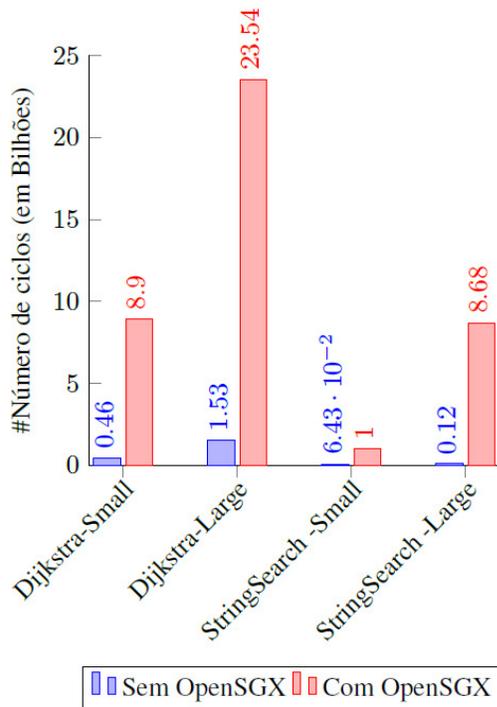


Figure 11. Número de ciclos de CPU dos algoritmos *Dijkstra* e *StringSearch* com e sem o OpenSGX

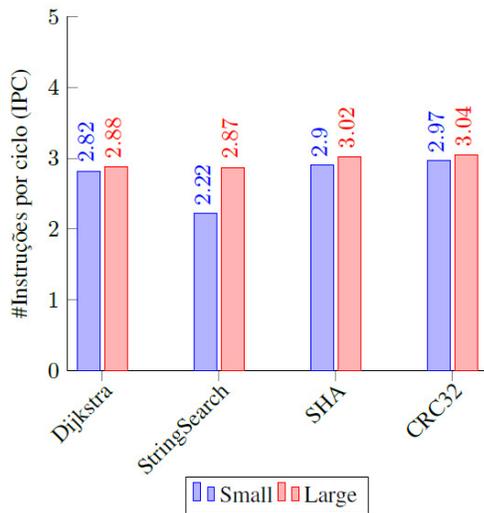


Figure 12. IPC dos algoritmos com e sem o uso de enclaves

não é uniforme. Isto é, as taxas de crescimento dos algoritmos com entradas maiores são menores que a diferença efetiva entre o tamanho das entradas. Isso permite induzir, em um primeiro momento, que todo o processo de instanciação e alocação do sistema OpenSGX consiste um fator importante de acréscimo no *overhead* em todo o conjunto.

Embora os resultados não permitam apontar os fatores exatos no aumento do número de ciclos decorrentes das instruções envolvendo toda a alocação e inicialização do sistema de enclaves, pode-se analisar em conjunto o número médio de

instruções por ciclo (*Instructions Per Cycles, IPC*), conforme apresentado na Figura 12. Nota-se um leve aumento no IPC na execução dos algoritmos com entradas maiores, indicando uma possível sobrecarga relativa às instruções de alocação e inicialização dos enclaves.

6. Conclusões

Ao permitir a execução de aplicações em um contexto totalmente protegido (*i.e.*, dentro de enclaves), amplia-se as possibilidades para as novas gerações de processadores Intel da família *x86* com a extensão SGX. A princípio, mesmo um comprometimento total do sistema operacional, ou hipervisor, não possibilitaria o acesso ao conteúdo legível da aplicação ou aos resultados de sua execução.

Por se tratar de uma tecnologia recente, as máquinas que contam com essa tecnologia ainda são minoria. Objetivando avaliar o SGX, utilizou-se um emulador dessa tecnologia denominado OpenSGX, o qual implementa e reproduz as principais funcionalidades e estruturas utilizadas na plataforma Intel SGX. O enfoque consistiu em avaliar o *overhead*, em termos de processamento, resultante da execução de uma aplicação em um ambiente com o SGX emulado.

Para a avaliação, empregou-se aplicações de *benchmark* da plataforma *MiBench*, modificando-as para compatibilizar a execução em enclaves no OpenSGX. Como métricas de desempenho, coletou-se o número total de instruções e o número total de ciclos de CPU para a execução completa de cada aplicação com e sem o OpenSGX.

O processo de instanciação das estruturas de controle em memória (*e.g.*, alocação do espaço reservado à EPC), aliado aos mecanismos necessários para a manutenção do enclave sob um esquema de criptografia aumentam o custo em volume de instruções e ciclos necessários para a execução de uma aplicação. Esse custo torna-se inerente a todas as aplicações, independente do tamanho destas.

Os resultados corroboram o fato que o SGX exige um acréscimo significativo ao número total de instruções necessárias à execução de uma aplicação com todo o aparato dos enclaves. Apesar do SGX ter sido avaliado em um ambiente emulado, pode-se esperar resultados semelhantes em uma plataforma nativa do SGX, tendo em vista que o OpenSGX implementa mecanismos e bibliotecas que são essenciais à execução do SGX. Ou seja, o OpenSGX reproduz de forma satisfatória os mecanismos do SGX adicionando apenas as camadas mínimas para possibilitar a emulação da extensão.

A proposta do Intel SGX é prover um ambiente de execução confiável. Nesse quesito, tudo indica que o mecanismo de enclaves possibilita atingir esse propósito. Os resultados obtidos antecipam o que se pode esperar em termos de desempenho. No entanto, tendo-se como perspectiva a possibilidade de se executar uma aplicação de forma totalmente protegida em, por exemplo, um provedor público na nuvem pode ser o principal fator decisivo, deixando a degradação no desempenho como um efeito colateral inevitável.

Contribuição dos Autores

Os autores contribuíram igualmente na realização deste trabalho.

Referências

- [1] PROUDLER, G.; CHEN, L.; DALTON, C. *Trusted Computing Platforms: Tpm2.0 in context*. 1. ed. Bristol(UK): Springer International Publishing, 2014.
- [2] SCHUSTER, F. et al. Vc3: Trustworthy data analytics in the cloud using sgx. In: *Proceedings of the 36th IEEE Symposium on Security and Privacy (SP)*. San Jose, CA: IEEE Press, 2015.
- [3] JAIN, P. et al. Opensgx: An open platform for sgx research. In: *Proceedings of the 2016 Network and Distributed System Security Symposium (NDSS 2016)*. San Diego, CA: Internet Society, 2016.
- [4] MCKEEN, F. et al. Innovative instructions and software model for isolated execution. In: *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*. New York, NY, USA: ACM, 2013. (HASP '13), p. 10:1–10:1.
- [5] INTEL. *Intel Software Guard Extensions Programming Reference*. 2014.
- [6] COSTAN, V.; DEVADAS, S. *Intel SGX Explained*. Cambridge, Massachusetts, EUA, 2016.
- [7] ANATI, I. et al. Innovative technology for cpu based attestation and sealing. In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy (HASP 2013)*. Tel-Aviv, Israel: ACM, 2013.
- [8] SINHA, R. et al. Moat: Verifying confidentiality of enclave programs. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Denver, Colorado, USA: ACM, 2015.
- [9] INTEL. *Intel Software Guard Extensions: Developer Guide*. 2016.
- [10] KIM, S. et al. A first step towards leveraging commodity trusted execution environments for network applications. In: *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*. Philadelphia, PA, USA: ACM, 2015.
- [11] JOHNSON, S. et al. *Intel Software Guard Extensions: EPID Provisioning and Attestation Services*. 2016.
- [12] ZHAO, C. et al. On the performance of intel sgx. In: *2016 13th Web Information Systems and Applications Conference (WISA)*. Wuhan, China: IEEE Press, 2016. p. 184–187.
- [13] GJERDRUM, A. T. et al. Performance of trusted computing in cloud infrastructures with intel sgx. In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science*. Porto, Portugal: SCITEPRESS, 2017. p. 696–703.
- [14] COSTAN, V. et al. Sanctum: Minimal hardware extensions for strong software isolation. In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016.
- [15] EVTYUSHKIN, D. et al. Iso-x: A flexible architecture for hardware-managed isolated execution. *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium*, 2014.
- [16] ARNAUTOV, S. et al. Scone: Secure linux containers with intel sgx. In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. Savannah, GA, USA: USENIX Association, 2016.
- [17] QEMU Emulator User Documentation. 2013. <<https://qemu.weilnetz.de/doc/qemu-doc.html>>. Acessado: Jun. 2017.
- [18] JONES, M. T. *Emulação do Sistema com o QEMU*. 2007. <<https://www.ibm.com/developerworks/br/library/l-qemu/index.html#authorN1001C>>. Acessado: Jun. 2017.
- [19] DINGLEDINE, R.; MATHEWSON, N.; SYVERSON, P. Tor: The second-generation onion router. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. Berkeley, CA, USA: USENIX Association, 2004. (SSYM'04), p. 21–21.
- [20] GUTHAUS, M. R. et al. Mibench: A free, commercially representative embedded benchmark suite. In: *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*. Austin, TX, USA: IEEE Press, 2001. p. 3–14.