UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENADORIA DO CURSO DE ENGENHARIA DE SOFTWARE

MARCIANO DA ROCHA

IMPLEMENTAÇÃO DE CRIPTOGRAFIA DE DISCO COM AMBIENTE DE EXECUÇÃO CONFIÁVEL INTEL *SOFTWARE GUARD EXTENSIONS* (INTEL SGX)

TRABALHO DE CONCLUSÃO DE CURSO

DOIS VIZINHOS

2019

MARCIANO DA ROCHA

IMPLEMENTAÇÃO DE CRIPTOGRAFIA DE DISCO COM AMBIENTE DE EXECUÇÃO CONFIÁVEL INTEL *SOFTWARE GUARD EXTENSIONS* (INTEL SGX)

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Software, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Msc. Rodrigo Tomaz Pagno

Coorientador: Prof. Me. Newton Carlos Will

DOIS VIZINHOS

2019

TERMO DE APROVAÇÃO

Implementação de Criptografia de Disco com Ambiente de Execução Confiável Intel Software Guard Extensions (Intel SGX)

por

Marciano Da Rocha

Este Trabalho de Conclusão de Curso foi apresentado em 04 de Julho de 2019 como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software. O(a) candidato(a) foi arguido(a) pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

Rodrigo Tomaz Pagno

Presidente da Banca

_____

Anderson Chaves Carniel

Membro Titular

_____

Dalton Cézane Gomes Valadares

Membro Titular

* A Folha de Aprovação assinada encontra-se na Coordenação do Curso

Dedico este trabalho à minha mãe, por acreditar e investir em mim. Mãe, seu cuidado e dedicação foi quem me deram, em todos os momentos, esperança para seguir.

AGRADECIMENTOS

Agradeço a Deus, primeiramente, que me deu força para concluir esta etapa de minha vida. Agradeço a minha família, que sempre acreditou e me deu forças, em especial a minha mãe Abgair, por seu apoio e carinho incondicional em todos os momentos difíceis, de desânimo e cansaço. Agradeço ao Prof. Msc. Rodrigo Tomaz Pagno e ao Prof. Me. Newton Carlos Will por acreditarem em mim, e pela dedicação e apoio na elaboração deste trabalho. Agradeço a todos os professores que participaram da minha formação. Aos amigos e colegas, que não negaram força e ficaram na torcida, e a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

RESUMO

ROCHA, Marciano da. IMPLEMENTAÇÃO DE CRIPTOGRAFIA DE DISCO COM AMBIENTE DE EXECUÇÃO CONFIÁVEL INTEL *SOFTWARE GUARD EXTENSIONS* (INTEL SGX). 68 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2019.

Juntamente com a evolução dos sistemas computacionais utilizados por usuários e organizações, cresce também a quantidade de dados confidenciais a serem armazenados e o número de ameaças sobre eles. Nesse cenário, a Intel lançou no final de 2015, juntamente com sua linha de processadores de 6ª geração (*Skylake*), a tecnologia *Software Guard Extensions* (Intel SGX), a qual fornece mecanismos de segurança para a execução de códigos dentro de uma área protegida no software, chamada de *enclave*, permitindo aos desenvolvedores que realizem a integração da mesma com seus sistemas. Dentre os mecanismos fornecidos, a tecnologia provê o recurso para selagem dos dados que estão no enclave, permitindo que sejam armazenados de forma segura, utilizando-se de uma chave de criptografia única, gerada e mantida pelo processador a partir das informações deste e do enclave. No entanto, garantir a segurança sobre os dados de todos os sistemas computacionais é um processo complexo. O presente trabalho faz uso do recurso de selagem de dados provido pela tecnologia Intel SGX para a criptografia de arquivos, criando assim um sistema de arquivos virtual onde aplicações possam armazenar seus dados e os mesmos possuam as garantias de segurança fornecidas pela tecnologia Intel SGX, de modo que, se a mídia de armazenamento for comprometida, os dados estarão seguros. Para validação da proposta, é feita a integração do software Cryptomator com um enclave para a selagem de dados. Os resultados demonstram que a solução é factível, tanto no quesito de desempenho quanto em segurança, podendo ser expandida e refinada para uso prático.

Palavras-chave: Selagem de dados, Criptografia de arquivos, Confidencialidade, Integridade, Armazenamento seguro.

ABSTRACT

ROCHA, Marciano da. IMPLEMENTING DISK ENCRYPTION WITH TRUSTED EXECUTION ENVIRONMENT INTEL SOFTWARE GUARD EXTENSIONS (INTEL SGX). 68 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2019.

With the evolution of computer systems used by organizations and by users, the amount of confidential data to be stored and the number of threats on these data grow up too. In this scenario, Intel launched in late 2015, along with its line of 6th generation processors (Skylake), the Software Guard Extensions (Intel SGX) technology, which provides security mechanisms for code execution within a protected area in software, called *enclave*, allowing developers to integrate it with their systems. In the mechanisms provided, the technology has a resource for sealing the data that are in the enclave, allowing they to be stored in a secure way, using a unique encryption key, generated and maintained by the processor, with information from it and the enclave. However, ensuring data security across all computing systems is a complex process. The present work makes use of the data sealing feature, provided by the Intel SGX technology, for file encryption, creating a virtual file system where applications can store their data and have the security guarantees provided by the Intel SGX technology, so that, when the storage media is compromised, the data is safe. To validate the proposal, the Cryptomator software is integrated with an enclave for data sealing. The results demonstrate that the solution is feasible, in terms of performance and security, and can be expanded and refined for practical use.

Keywords: Data sealing, File encryption, Confidentiality, Integrity, Secure storage.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACM	*Authenticated Code Module*
AES-CTR	*Advanced Encryption Standard-Counter*
AES-GCM	*Advanced Encryption Standard-Galois/Counter Mode*
AES-CMAC	*Advanced Encryption Standard-Cipher-based Message Authentication Code*
AES-NI	*Advanced Encryption Standard-New Instructions*
AEX	*Asynchronous Exit*
API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
BIOS	*Basic Input/Output System*
CPU	*Central Processing Unit*
DLL	*Dynamic-Link Library*
DMA	*Direct Memory Access*
ECALL	*Enclave Call*
EDL	*Enclave Definition Language*
EPC	*Enclave Page Cache*
EPCM	*Enclave Page Cache Map*
EPID	*Enhanced Privacy ID*
EXT	*Extended File System*
FBI	*Federal Bureau of Investigation*
FDE	*Full Disk Encryption*
FUSE	*Filesystem in Userspace*
GB	*Gigabyte*
HDD	*Hard Disk Drive*
ISV	*Independent Software Vendor*
JNI	*Java Native Interface*
KB	*Kilobyte*
LCP	*Launch Control Policy*
LKM	*Loadable Kernel Module*
LUKS	*Linux Unified Key Setup*
MAC	*Message Authentication Code*
MB	*Megabyte*
MBR	*Master Boot Record*
MEE	*Memory Encryption Engine*
MHz	*Megahertz*
NVMe	*Non-Volatile Memory Express*
OCALL	*Out Call*
ORAM	*Oblivious RAM*
PKCS	*Public Key Cryptography Standards*
PRM	*Processor Reserved Memory*
PSP	*Platform Security Processor*
RAM	*Random-Access Memory*

RPM	*Revolutions Per Minute*
RSA	*Rivest-Shamir-Adleman*
SED	*Self-Encrypting Drives*
SECS	*SGX Enclave Control Structure*
SDK	*Software Development Kit*
SGX	*Software Guard Extensions*
SHA	*Secure Hash Algorithm*
SO	Sistema Operacional
SoC	*System-on-Chip*
SSD	*Solid State Drive*
SVN	*Security Version Number*
TB	*Terabyte*
TCB	*Trusted Computing Base*
TPM	*Trusted Platform Module*
TXT	*Trusted Execution Technology*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem como objetivo contextualizar o tema e o motivo pelo qual ele foi escolhido. A Seção 1.1 descreve o cenário atual, no qual a busca por novas tecnologias que forneçam maior segurança nos dados aumentou, e a Seção 1.2 apresenta os motivos que norteiam a solução proposta, a qual é brevemente apresentada na Seção 1.3. Por fim, os objetivos são apresentados na Seção 1.4 e a Seção 1.5 descreve como o trabalho está organizado.

1.1 CONTEXTUALIZAÇÃO

Nos últimos anos o uso da tecnologia por pessoas e empresas vem crescendo, aumentando a quantidade de informações que são armazenadas nos mais diversos tipos de dispositivos. Diante desse crescimento é de suma importância aumentar a segurança dos dados, visto que a todo momento surgem inúmeras novas ameaças como vírus e ataques a servidores com o objetivo de roubo de informações. Relatórios de empresas especializadas e entidades governamentais de segurança indicam um número crescente de ameaças aos dados digitais, sejam de corporações ou de usuários. De acordo com a McAfee Labs, as empresas enfrentavam, em média, 25 novas ameaças por dia no ano de 2006, número que subiu para 500.000 em 2016. Além disso, estimativas do FBI, no ano de 2005, indicavam que os crimes cibernéticos geravam um montante de 67,2 bilhões de dólares de prejuízo anualmente (WILL; CONDÉ; MAZIERO, 2017).

Diante dos fatos, muitas empresas estão investindo em pesquisa e desenvolvimento de tecnologias baseadas em hardware a fim de garantir a segurança das informações, tanto para usuários corporativos quanto para usuários particulares. No final de 2015 a Intel lançou no mercado a tecnologia *Software Guard Extensions* (SGX) juntamente com os processadores Intel Core de 6ªgeração (*Skylake*), a qual possibilita o isolamento de uma aplicação ou uma parte dela em uma estrutura chamada de *Enclave*, criptografando as informações existentes na memória do computador, o que garante um nível maior de

segurança nas informações enquanto estas são processadas (HOEKSTRA et al., 2013).

No que tange ao uso de computadores pessoais, existem diversos sistemas de criptografia de disco, para os principais sistemas operacionais no mercado. Além disso, os *smartphones* com sistemas Android e iOS também fornecem recursos nativos para a criptografia dos dados. Isto só reforça a crescente preocupação das empresas em manter a confidencialidade dos dados dos usuários.

1.2 MOTIVAÇÃO

Devido à grande quantidade de aplicações existentes, garantir que toda informação será armazenada com segurança é um processo complexo. Uma das formas existentes são os sistemas de criptografia de disco, os quais rodam em segundo plano no sistema operacional e criptografam toda informação que é armazenada.

Esse tipo de sistema possui algumas vulnerabilidades, as quais podem gerar brechas para ataques e acesso indevido aos dados. Uma dessas vulnerabilidades é a necessidade de o usuário escolher a senha, a qual servirá como base na geração de uma chave que será usada no processo de criptografia. Levando em conta que usuários optam por senhas fracas, muitas vezes ingenuamente, e outras apenas por ser mais simples de memorizar, em muitos casos a criptografia pode ser quebrada apenas obtendo dados pessoais do indivíduo e realizando algumas combinações simples. Além disso, tal senha pode ser descoberta utilizando ataques por dicionário, uso de senhas padrões, *rainbow tables*[1] ou até mesmo força bruta.

Vale destacar também que grande parte dos sistemas de criptografia de disco existentes no mercado mantém a chave de criptografia na memória principal enquanto está em execução, possibilitando que outras aplicações maliciosos realizem ataques a fim de obter a mesma.

Outro ponto analisado é que em alguns sistemas de criptografia de disco existe a possibilidade de transferir a responsabilidade pela criptografia dos dados para a unidade de disco quando esta possui o padrão *TCG Opal*[2] implementado, a fim de aumentar o desempenho do sistema. De acordo com Meijer e Gastel (2018) alguns modelos de unidades de estado sólido (*Solid State Drive* - SSD) possuem falhas no processo de

[1] *Rainbow Table* é uma tabela para consulta de *hashes* pré calculados, os quais são gerados a partir de combinações textuais aplicadas a determinado algoritmo de criptografia. É utilizada para localizar uma senha que corresponda à chave criptográfica encontrada.

[2] *TCG* significa *Trusted Computing Group*. *Opal* é uma especificação de segurança para dispositivos de armazenamento, definidos pelo TCG.

criptografia, permitindo que os dados armazenados sejam extraídos da unidade sem precisar conhecer a chave usada para realizar a mesma. O software BitLocker da Microsoft habilita a opção de criptografia através do hardware por padrão, fazendo com que usuários sem conhecimento dessa vulnerabilidade estejam suscetíveis a ela.

Dentre os diversos recursos disponibilizados pela tecnologia SGX, um deles é a selagem dos dados de um enclave em memória secundária utilizando como base para a criptografia uma chave única gerada para aquela plataforma específica. Esta chave é gerada a partir da combinação de uma chave de desenvolvedor e a chave do processador, a qual não fica armazenada em memória, sendo gerada novamente a cada requisição, garantindo que os dados somente serão abertos pelo enclave que os selou, e na plataforma em que ele foi selado (ANATI et al., 2013).

Outro recurso provido pela tecnologia Intel SGX é a criptografia dos dados em uma região da memória primária com uma chave aleatória que é gerada a cada ciclo de energia. Esta chave é conhecida apenas pelo processador e nunca sai dos seus limites (INTEL, 2016).

Nesse cenário pode-se eliminar a necessidade de utilizar uma senha do usuário para gerar a chave de criptografia dos dados e optar por utilizar unicamente a chave gerada pela tecnologia Intel SGX, a *Sealing Key*, no processo de selagem de dados. Desta forma, elimina-se também as questões inerentes à segurança da chave em memória, visto que ela é gerada sempre que é utilizada e não sai dos limites do processador.

Outra alternativa é combinar uma chave de criptografia gerada a partir de uma senha escolhida pelo usuário com a selagem de dados, adicionando, assim, uma segunda camada de proteção aos dados.

1.3 PROPOSTA

O presente trabalho visa aplicar os recursos e garantias de segurança e confidencialidade providos pela selagem de dados da tecnologia Intel SGX em um software de criptografia de disco já disponível no mercado, com a finalidade de prover uma camada de segurança extra no armazenamento destes dados.

Para isso, foram realizadas alterações no software *Cryptomator*, fazendo com que ele realize o processo de criptografia dos dados dentro de um enclave através do recurso de selagem de dados provido pela tecnologia, utilizando, dessa forma, uma chave única gerada para aquela plataforma específica. A alteração proposta não prevê a remoção

dos métodos e algoritmos já existentes no software, a fim de manter compatibilidade do mesmo com os processadores que não possuem a tecnologia Intel SGX.

A solução desenvolvida e apresentada neste trabalho é chamada de *Cryptomator-SGX*.

1.4 OBJETIVOS

Esta seção apresenta os objetivos a serem atingidos com a conclusão deste trabalho.

1.4.1 OBJETIVO GERAL

Esse trabalho tem como objetivo geral alterar um software de criptografia de disco de modo que o mesmo utilize a tecnologia Intel SGX para a selagem dos dados, podendo, desta forma, eliminar a necessidade da escolha de uma senha por parte do usuário para gerar a chave de criptografia, além de eliminar as questões inerentes ao armazenamento desta chave, visto que a chave criptográfica utilizada pelo recurso de selagem da arquitetura Intel SGX é fornecida pelo processador, e nunca sai dos seus limites.

1.4.2 OBJETIVOS ESPECÍFICOS

A partir do objetivo geral, são definidos os objetivos específicos do presente trabalho, sendo estes:

- Implementar um modelo de comunicação entre o software Cryptomator e um enclave SGX;
- Verificar quais são as implicações e ameaças à confidencialidade dos dados selados;
- Verificar o impacto de desempenho que tal solução acarreta ao software alterado e ao sistema como um todo;
- Analisar as garantias de segurança providas pela solução implementada.

1.5 ORGANIZAÇÃO DO TRABALHO

O presente documento está organizado em seis capítulos, onde este capítulo traz a contextualização, a motivação e, os objetivos do trabalho. No Capítulo 2, é apresentado uma visão geral sobre criptografia completa de disco (*Full Disk Encryption* - FDE), mecanismos de segurança baseados em hardware e o funcionamento da tecnologia Intel SGX, e, no Capítulo 3 são apresentados alguns trabalhos relacionados ao tema, os quais possuem o mesmo objetivo, a segurança das informações que são armazenadas. O Capítulo 4 descreve o contexto de aplicação, a justificativa, a metodologia utilizada, a arquitetura do software escolhido para a alteração e a arquitetura do mesmo após a implementação da solução proposta.

No Capítulo 5 são apresentados os resultados obtidos, por meio dos dados coletados nos testes de desempenho e segurança que foram executados na aplicação, e as vulnerabilidades às quais o software alterado está suscetível. Por último, o Capítulo 6 apresenta as considerações finais sobre o trabalho desenvolvido, contribuições obtidas, e trabalhos futuros que podem ser desenvolvidos a partir deste.

2 REFERENCIAL TEÓRICO

Este capítulo tem como objetivo apresentar a criptografia completa de disco (FDE, *Full Disk Encryption*) e o funcionamento da tecnologia Intel SGX, além de um breve histórico das tecnologias que precedem-na e que contribuíram de alguma forma para a criação da mesma.

2.1 CRIPTOGRAFIA COMPLETA DE DISCO

De acordo com o que foi descrito no Capítulo 1, muitas pessoas e empresas possuem informações sigilosas armazenadas nos mais diversos tipos de dispositivos. A criptografia completa de disco surgiu como uma necessidade para garantir a segurança dessas informações em casos de roubo ou perda dos dispositivos que as contém, devido que mecanismos como senhas e bloqueios por biometria não impedem que a mídia de armazenamento seja acessada de forma direta ao ser instalada em outro dispositivo. A criptografia completa de disco, abreviada do inglês como FDE (*Full Disk Encryption*), é um método eficaz na proteção de dados contra acesso não autorizado pois consiste na criptografia de volumes inteiros, ou seja, criptografar um disco inteiro ou uma partição do mesmo, garantindo que as informações não possam ser acessadas sem o mecanismo e chave usados para a criptografia. Os mecanismos existentes podem ser classificados em criptografia a nível de *firmware* (soluções baseadas em hardware) e em criptografia a nível de *kernel* (soluções baseadas em software).

Mecanismos FDEs baseados em hardware são chamados de SEDs (*Self-Encrypting Drives*, ou unidades de autocriptografia), onde a criptografia é realizada pelo controlador do disco e as chaves de criptografia não estão presentes na CPU ou na memória principal do computador, além de criptografarem o registro mestre de inicialização (MBR - Master Boot Record), impedindo ataques de manipulação ao mesmo. Este tipo de FDE está presente em alguns modelos específicos de mídias de armazenamento, como as unidades SSD 320 e 520 da Intel, e necessita de um ambiente pré-inicializado, como uma tela da

BIOS, para que a senha de acesso seja inserida.

Mecanismos FDEs baseados em software são aplicações executadas a nível de *kernel* pela CPU e seus dados e chaves usadas ficam armazenados na memória principal do computador. Esse tipo de FDE consiste em interceptar todas as requisições do sistema operacional (SO) e realizar a criptografia dos dados antes de serem armazenados ou buscar dados criptografados na mídia de armazenamento e retornar uma informação legível para o SO, além de não criptografar o MBR, o que permite ataques de manipulação ao mesmo.

Soluções FDEs baseadas em software estão presentes no mercado desde os anos 2000, começando com o lançamento do *Cryptoloop* para Linux, que foi antecessor do *Dm-Crypt* lançado em 2003. Neste mesmo ano a Apple lançou o *FileVault*, o qual permitia aplicar a criptografia apenas à partição inicial do usuário. Em 2004 a ferramenta *open source* multiplataforma *TrueCrypt* foi lançada. Em 2006 a Microsoft lançou o *BitLocker* para o Windows e somente em 2011 a Apple lançou uma atualização para o *FileVault*, a qual permitia aplicar a criptografia também na partição do sistema (MÜLLER; FREILING, 2015).

Em maio de 2014, após a Microsoft encerrar o suporte ao Windows XP, o projeto do *TrueCrypt* foi encerrado (TRUECRYPT, 2014). Logo após, em 3 de junho de 2014 foi lançada a primeira versão da ferramenta *VeraCrypt*. Sendo baseado no *TrueCrypt*, o *VeraCrypt* corrige muitas vulnerabilidades e questões de segurança que o seu predecessor possuía (VERACRYPT, 2018). Neste mesmo ano foi iniciado o projeto do *Cryptomator*, um software voltado para a criptografia de contêineres que é desenvolvido usando a linguagem Java e continua em constante manutenção (CRYPTOMATOR, 2019).

Além das citadas, outras ferramentas de criptografia completa de disco também surgiram no decorrer dos últimos anos, visando a proteção dos dados contidos em computadores pessoais e corporativos. Outro ponto que vale destacar é que, atualmente, *smartphones* Android e iOS já suportam a criptografia de dados nativamente, demonstrando a preocupação das empresas e dos usuários na proteção dos dados pessoais.

2.2 MECANISMOS DE SEGURANÇA BASEADOS EM HARDWARE

A fim de acelerar o processo de criptografia e garantir uma segurança maior aos dados, mecanismos baseados em hardware que precedem a tecnologia Intel SGX surgiram ao longo dos anos. Alguns desses mecanismos são listados abaixo, juntamente com uma breve descrição de cada um.

- **Primeiros Hardwares Criptográficos:** De acordo com Anderson et al. (2006), um cripto-processador típico é um processador embarcado que possui um conjunto pré-definido de operações e chaves internas com finalidade exclusiva para operações criptográficas, a fim de acelerar os cálculos criptográficos. Esse tipo de processador pode ser uma parte interna e exclusiva do hardware ou pode ser um hardware separado, o qual necessita ser acoplado ao processador principal. Outra característica é que algumas implementações não permitem que a mesma seja programável, permitindo apenas parametrizar o processo de criptografia.

- ***Trusted Platform Module* (TPM):** é um *chipset* capaz de armazenar dados de autenticação de uma plataforma computacional, como senhas, certificados e chaves criptográficas, podendo também ser usado para armazenar medidas de uma plataforma a fim de garantir que a mesma não foi modificada e é confiável (TCG, 2008).

- **Intel *Trusted Execution Technology* (Intel TXT):** Tecnologia da Intel que utiliza componentes de hardware para criar ambientes confiáveis, impedindo ataques via software e hardware. Para que o ambiente seguro seja iniciado, o Intel TXT necessita que alguns requisitos sejam atendidos: extensões integradas ao Intel Xeon e *chipset* Intel, módulos de código autenticados (*Authenticated Code Modules* - ACMs), ferramentas de LCP (*Launch Control Policy*), um módulo TPM, BIOS e *hipervisor* ou sistema operacional habilitado para o Intel TXT (GREENE, 2012).

- **ARM *TrustZone*:** Essa arquitetura de hardware permite que qualquer parte de um sistema seja protegida, fornecendo uma infraestrutura básica que permite aos projetistas de SoCs (*System-on-Chip*) utilizar componentes que auxiliam em funções específicas dentro de um ambiente seguro. A arquitetura ARM *TrustZone* permite que o sistema seja isolado em dois estados, seguro e inseguro, os quais permitem que dispositivos periféricos tomem decisões de controle de acesso. A aplicação pode isolar partes da memória para seu uso enquanto está sendo executada em um estado seguro, impedindo que aplicações que estão sendo executadas em um estado inseguro acessem essa região (ARM, 2009; LESJAK; HEIN; WINTER, 2015).

- **AMD *Secure Processor*:** É um subsistema de segurança em hardware integrado ao processador, que é conhecido como *Platform Security Processor* (PSP). O PSP faz uso de um microcontrolador ARM *TrustZone* de 32 *bits*, possui um coprocessador criptográfico composto de números aleatórios, mecanismos que processam algoritmos de criptografia e um bloco para armazenamento de chaves,

provendo um ambiente isolado do sistema operacional para tratamento de dados sensíveis. Apesar de utilizar a arquitetura *TrustZone*, o PSP possui um processador físico ao invés de um virtual, e funciona de modo independente dos núcleos do processador (ARTHUR; CHALLENER, 2015; WILL; CONDÉ; MAZIERO, 2017).

2.3 INTEL *SOFTWARE GUARD EXTENSIONS*

A tecnologia Intel SGX compõe um conjunto de instruções e mecanismos para acesso à memória com o intuito de garantir a confidencialidade e integridade de dados sensíveis de aplicações. Para tornar isso possível, dois recursos significativos foram adicionados na arquitetura x86 da Intel, sendo eles a mudança no acesso à memória do enclave e a proteção dos mapeamentos de endereços do aplicativo.

A primeira versão do SGX, chamado de SGX1, foi adicionada aos processadores Intel a partir da 6ªgeração (*Skylake*) e permite que uma aplicação inicie um contêiner protegido, chamado de enclave, o qual será tratado na Seção 2.3.1 (INTEL, 2014).

Uma segunda versão do SGX, nomeada de SGX2, deverá fornecer flexibilidade no gerenciamento em tempo de execução do enclave, possibilitando a alocação dinâmica de memória, além de permitir a execução de *threads* dentro do mesmo (MCKEEN et al., 2016). Apesar do SGX2 já estar presente nas documentações da Intel, esta versão ainda não está disponível comercialmente e também não há previsão de quando será liberada. Desta forma, o presente trabalho se limita a descrever e utilizar os recursos providos pelo SGX1.

2.3.1 O CONCEITO DE ENCLAVE

Um enclave é uma área protegida e de tamanho fixo no espaço de endereçamento do aplicativo, a qual permite que uma parte do código da aplicação seja executada de forma confidencial e segura, garantindo que outras aplicações não consigam acessar essas informações, mesmo que estes possuam alto privilégio de execução ou estejam executando outros enclaves (INTEL, 2014).

As tentativas de acesso não autorizado ao conteúdo de um enclave são detectadas e impedidas, ou então a operação é abortada. Enquanto os dados do enclave estão tramitando entre os registradores e demais blocos internos do processador, este utiliza mecanismos de controle de acesso interno para evitar o acesso não autorizado a estes dados. Quando os dados são transferidos para a memória principal, eles são automaticamente

criptografados e armazenados em uma região reservada, chamada de PRM (*Processor Reserved Memory*), a qual será abordada na Seção 2.3.2.

A criptografia de memória é feita utilizando um algoritmo de criptografia AES-CTR de 128 *bits* de tempo invariável e contendo proteções contra ataques de repetição. A chave de criptografia é armazenada em registradores internos do processador, não sendo acessível a componentes externos, e é alterada de forma aleatória a cada evento de hibernação ou reinício do sistema. As sondagens de memória ou outras técnicas que visam modificar ou substituir estes dados também são evitadas, e o fato de conectar o módulo de memória a outro sistema apenas dará acesso aos dados em sua forma criptografada (AUMASSON; MERINO, 2016; INTEL, 2016).

A tecnologia SGX reduz a base de computação confiável (*Trusted Computing Base* - TCB) ao excluir elementos como o sistema operacional, *hypervisor*, *drivers* e BIOS da área confiável da aplicação, restringindo, desta forma, possibilidades de ataques e aumentando a segurança nos dados sensíveis da aplicação, conforme mostra a Figura 1 (CONDÉ, 2017).

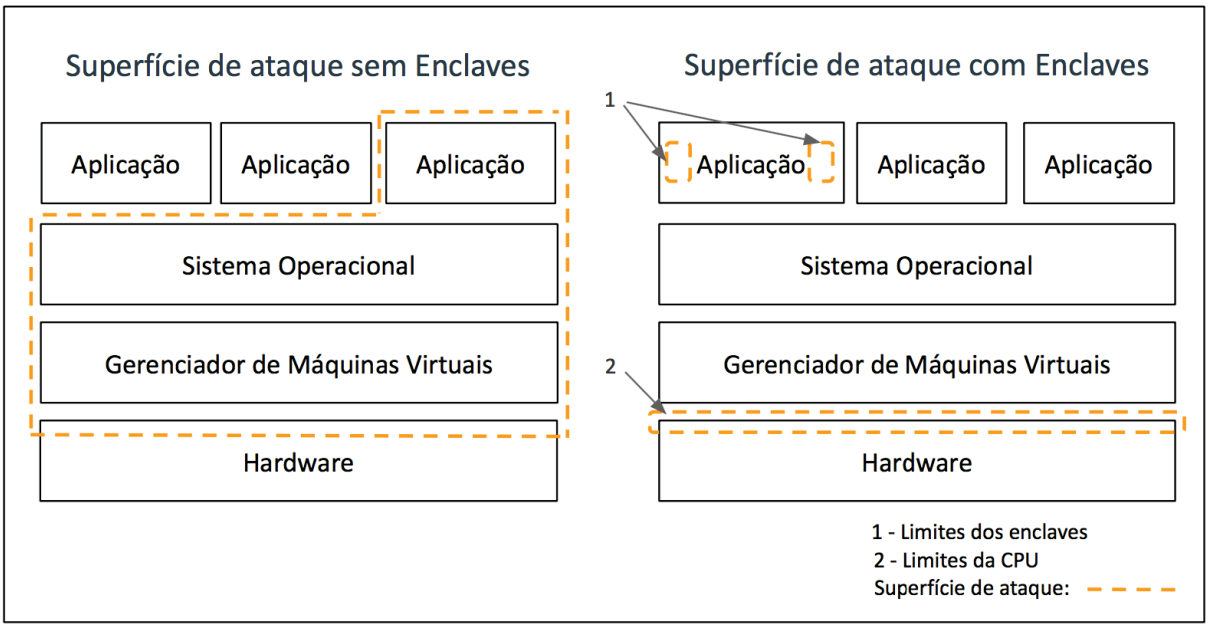

Figura 1: Superfície de ataque em aplicações com e sem enclaves SGX.
Fonte: Condé (2017)

2.3.1.1 CICLO DE VIDA DO ENCLAVE

O código de uma aplicação que faz uso da tecnologia SGX é dividido em duas partes: confiável (código do enclave) e não-confiável (restante do código).

Quando o enclave é solicitado pela primeira vez, a instrução *ECREATE* inicia sua criação e inicializa o SGX *Enclave Control Structure* (SECS), o qual irá conter informações globais sobre o mesmo. Devido à restrição do SGX1, na qual um enclave não possui flexibilidade no gerenciamento de memória em tempo de execução, todas as páginas de memória que serão utilizadas posteriormente são adicionadas no EPC (*Enclave Page Cache*) durante a criação do enclave. O EPC será abordado em mais detalhes na Seção 2.3.2.

Uma página de memória é adicionada utilizando a instrução *EADD* em conjunto com a instrução *EEXTEND*, a qual adiciona a medição criptográfica do seu conteúdo. Por fim, a instrução *EINIT* completa o processo de criação do enclave e cria a sua identidade. Durante a criação do enclave, o código e os dados iniciais são carregados para o EPC e, portanto, antes da criação do enclave, todo o código está livre para inspeção e análise.

Após a inicialização do enclave, para que a aplicação utilize-o, normalmente é necessário executar a instrução *EENTER*, transferindo o controle para o código que está no EPC. Posteriormente, o enclave pode retornar o controle para o método que o chamou através da instrução *EEXIT*. Após a entrada no enclave, o controle é transferido via hardware para o software dentro do mesmo. O software, por sua vez, alterna o ponteiro da pilha para um ponteiro dentro do enclave. Ao sair do enclave, o software troca o ponteiro para a pilha novamente e executa a instrução *EEXIT*. Para chamar um procedimento externo ao enclave, também pode ser utilizado a instrução *EEXIT*. Todo o processo de entrada e saída do enclave é descrito em mais alto nível na Seção 2.3.1.2.

Se a saída do enclave ocorrer devido a algum evento ou falha, o processador executará uma rotina chamada *Asynchronous Exit* (AEX), que irá salvar o estado do enclave, limpar os registradores e armazenar o endereço da instrução que gerou a falha, permitindo que a execução seja posteriormente retomada, invocando a instrução *ERESUME*.

Um enclave inicializado utiliza recursos disponíveis do *EPC*. Para gerenciamento de enclaves inicializados e que não estão em uso, a tecnologia Intel SGX fornece a instrução *EREMOVE*, a qual recupera os recursos utilizados pelo enclave atual, deixando-os disponíveis para uso por outro enclave (CONDÉ, 2017; WILL; CONDÉ; MAZIERO,

2017; INTEL, 2014). O ciclo de vida do enclave, bem como as instruções responsáveis por seu gerenciamento que foram descritas, é representado graficamente na Figura 2.

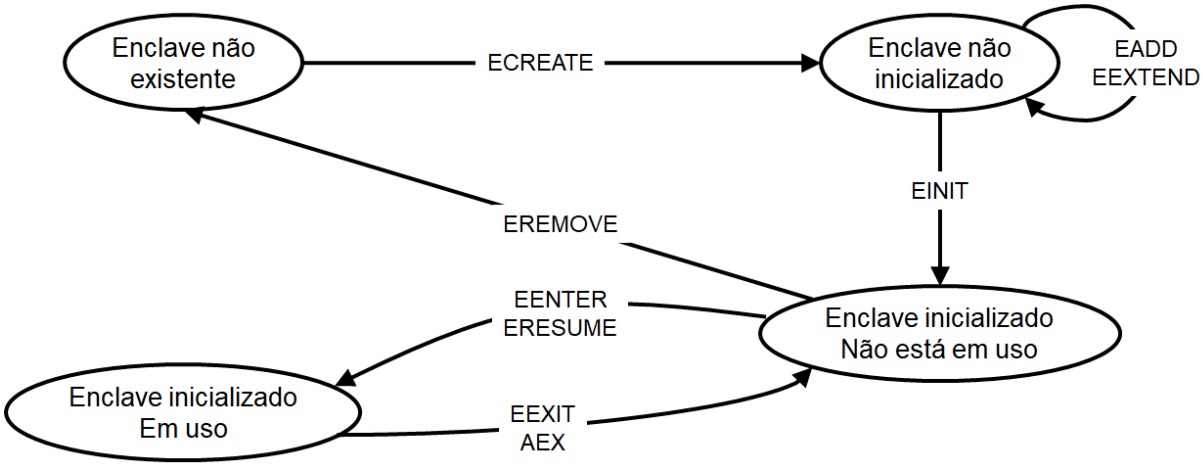

Figura 2: Instruções de gerenciamento do ciclo de vida do enclave e diagrama de transição de estados.

Fonte: Will, Condé e Maziero (2017)

2.3.1.2 INTERFACE DO ENCLAVE

De modo geral, para os ISVs (*Independent Software Vendor*, ou desenvolvedores independentes de software), o desenvolvimento de um enclave é semelhante ao desenvolvimento de uma biblioteca compartilhada em um projeto (DLL no Windows, Biblioteca Dinâmica no OSX e Objeto Compartilhado no Linux e Android), na qual uma parcela do código é separada do código da aplicação e sempre que é necessário executar o mesmo, a aplicação faz uma chamada para a biblioteca. Da mesma forma, um enclave contém uma pequena parte do código da aplicação, aquela que deve ser executada de forma segura.

De acordo com a Intel, embora seja possível mover a maioria do código do aplicativo para um enclave, isso não é recomendado. O primeiro passo para a implementação de um enclave é separar somente a parte de código da aplicação que realmente deve ser executada de forma segura. Isso se deve ao fato de que o SGX1 requer que toda funcionalidade de um enclave seja ligada de forma estática em tempo de compilação, criando um *trade-off* de desempenho/tamanho, o qual impacta no tamanho do TCB e deve ser uma preocupação constante durante o desenvolvimento. Apesar de que a tecnologia Intel SGX protege todo o conteúdo do enclave, quanto maior a TCB, maior será a superfície de ataque a ser exposta. Ao utilizar uma funcionalidade de uma biblioteca

estática, os ISVs têm duas opções: fornecer uma camada para chamar a funcionalidade fora do enclave na qual é adicionada uma sobrecarga de desempenho na aplicação, ou, incluir a implementação da biblioteca como parte do enclave, causando um aumento no tamanho da TCB.

O código de um enclave não é acessível de forma direta para a aplicação, para isso são criadas as rotinas de borda (métodos e/ou funções), as quais ficam entre a parte confiável (enclave) e a não-confiável (aplicação), podendo ser acessadas tanto de dentro quanto de fora do enclave. As rotinas de borda são divididas em confiáveis (ECALL - *Enclave Call*), as quais são chamadas pela aplicação e executam um processo dentro do enclave, e não-confiáveis (OCALL - *Out Call*), as quais são chamadas pelo enclave e executam um processo na aplicação. As rotinas de borda que o enclave disponibiliza são pré-definidas e declaradas no arquivo EDL (*Enclave Definition Language*) pelo desenvolvedor.

ECALLs expõem a interface do enclave para a aplicação, devido a isso, deve ser limitada a sua quantidade a fim de reduzir a superfície de ataque. A passagem de parâmetros para o enclave é realizada utilizando atributos especiais, os quais permitem à rotina de borda manipular a transição dos dados entre a aplicação e o enclave de forma segura através do uso de *buffers*. No entanto, quando um parâmetro é passado por referência, apenas o endereço do valor referenciado estará dentro do enclave, de modo que o valor não será seguro e pode ser examinado/alterado a qualquer momento, sendo necessário a verificação dos mesmos antes de serem usados pelo enclave. Nesse cenário, deve-se ter um cuidado maior ao trabalhar com ponteiros, devido que o aplicativo pode passar um ponteiro referenciando um endereço de memória dentro do limite do enclave, processo no qual o enclave pode sobrescrever seus dados ou código de forma indevida, podendo até mesmo ocasionar o vazamento de informações confidenciais.

OCALLs são chamadas de rotinas externas ao enclave e que não dependem de serem executadas de forma segura, as quais podem ocorrer quando o enclave precisa acessar um serviço do sistema operacional (*system calls*), sincronização de *threads*, operações de E/S e/ou execução de outras rotinas da aplicação. O enclave deve tratar situações em que uma OCALL não tenha sido executada por completa, evitando que um invasor intercepte o processo e emita uma resposta falsa. Nessa situação, devido às rotinas definidas no enclave não possuírem ordem para serem executadas, a aplicação pode efetuar uma chamada ECALL novamente, retornando a execução para o enclave, sendo necessário que o desenvolvedor limite quais ECALLs podem ser executadas durante

a execução de uma OCALL. Quando uma OCALL é executada, apenas os dados passados a ela por parâmetro são expostos. Se o valor de retorno de uma OCALL é um ponteiro, é necessário ter os mesmos cuidados que exitem na passagem de ponteiros para ECALLs (CONDÉ, 2017; WILL; CONDÉ; MAZIERO, 2017; INTEL, 2014). O processo no qual é realizado a troca entre os domínios confiável (enclave) e não-confiável (aplicação), descrito acima, é representado na Figura 3.

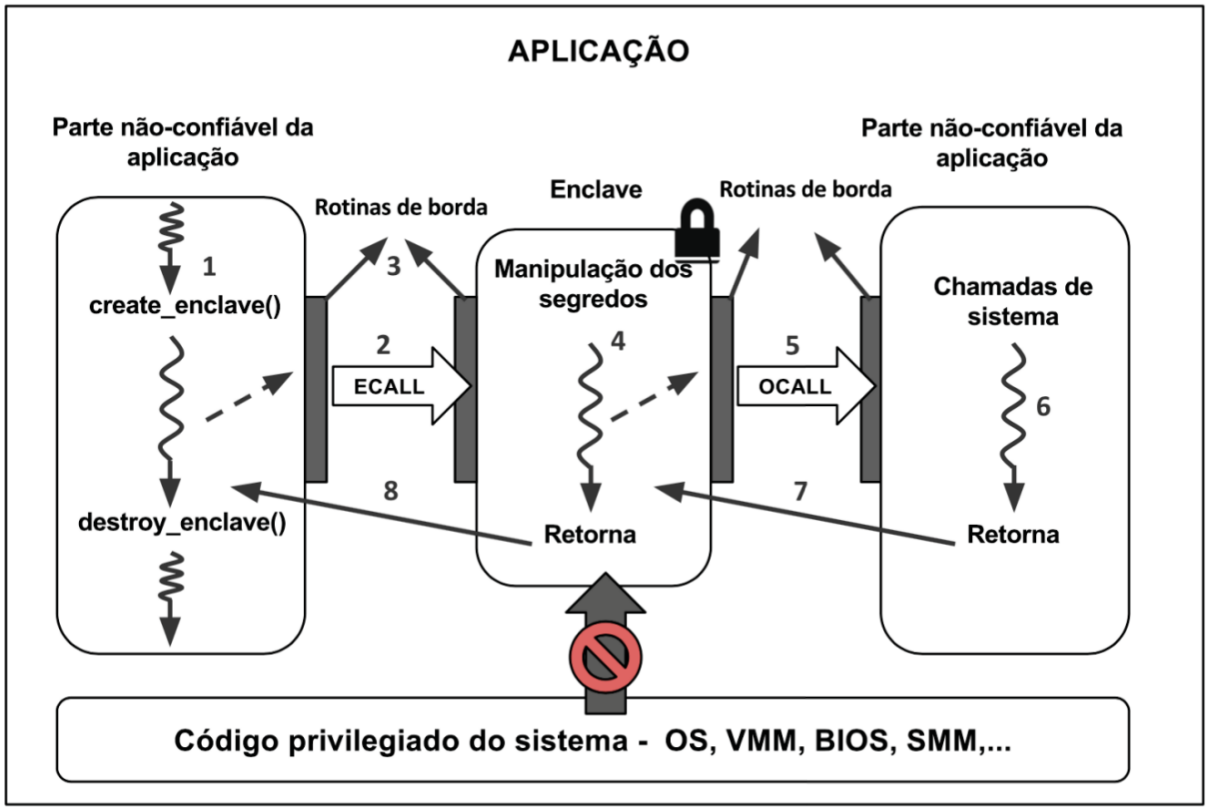

Figura 3: Interface de comunicação entre a aplicação e o enclave.
Fonte: Condé (2017)

2.3.2 A MEMÓRIA DO ENCLAVE

A tecnologia Intel SGX utiliza um subconjunto reservado da memória principal chamado de *Processor Reserved Memory* (PRM), a qual é uma área contígua de memória que possui tamanho e endereço de início como um valor inteiro e potência de dois, permitindo que seus endereços possam ser verificados por um hardware simples e barato. A PRM não pode ser acessada diretamente por outras aplicações. Quando uma transferência via DMA (*Direct Memory Access*, ou Acesso Direto à Memória) é realizada, o controlador da CPU rejeita-a, protegendo do acesso por meio de dispositivos periféricos.

Uma pequena parcela da PRM é reservada para uso do hardware e o subconjunto restante, chamado de *Enclave Page Cache* (EPC), armazena os códigos e dados dos enclaves. O EPC é subdividido em páginas de 4 *KBytes*, as quais podem ser atribuídas a diferentes enclaves e são gerenciadas pelo mesmo software que controla o restante da memória física (*hypervisor* ou *kernel* do sistema operacional), o qual utiliza as instruções do SGX para alocação e liberação das mesmas quando necessário. Essa divisão permite a execução de vários enclaves simultaneamente, o que é uma necessidade em ambientes multiprocessados. A organização da PRM descrita acima é representada na Figura 4 (WILL; CONDÉ; MAZIERO, 2017; INTEL, 2014).

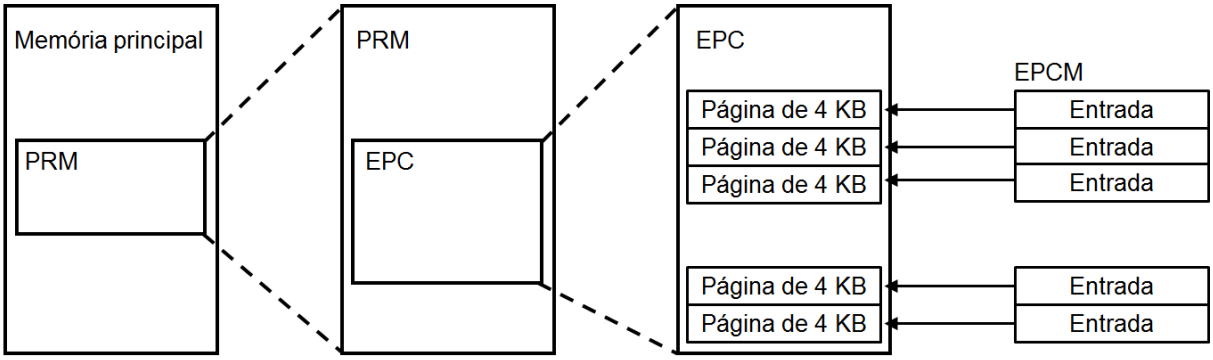

Figura 4: Os dados do enclave são armazenados no EPC, que é um subconjunto do PRM, que por sua vez é uma área contígua de memória que não pode ser acessada por outras aplicações ou por dispositivos via DMA.

Fonte: Will, Condé e Maziero (2017)

Para gerenciamento e verificações de segurança, o SGX armazena informações sobre a alocação de cada página do EPC no *Enclave Page Cache Map* (EPCM). O EPCM é uma matriz com uma entrada para cada página do EPC (conforme representado na Figura 4), a qual permite que a computação das mesmas sejam feitas apenas com uma operação de troca *bit* a *bit* e uma adição. A utilização do EPCM é transparente aos desenvolvedores, e não gera comportamento visível ao software em situações normais de execução devido que seu uso é feito apenas para verificações de segurança (WILL; CONDÉ; MAZIERO, 2017; INTEL, 2014).

Segundo Costan e Devadas (2016), cada entrada do EPCM armazena três informações sobre a página do EPC relacionada:

- Um campo de um único bit (*VALID*) indicando se a página do EPC está livre (*bit* 0) ou alocada (*bit* 1), evitando que os dados de outros enclaves sejam sobrescritos.
- Um segundo campo (*ENCLAVESECS*) indicando a qual enclave a página pertence,

desse modo garantindo que um enclave não tenha acesso à informações de outros enclaves.

- E por último, um campo (PT) identifica o tipo da página alocada, o qual determina sua finalidade de uso.

As páginas do EPC que armazenam o código e os dados de um enclave são consideradas regulares (PT_REG), as demais páginas são dedicadas ao armazenamento das estruturas de dados de suporte da tecnologia SGX e possuem tipos especiais. Um dos tipos especiais de páginas é o *SGX Enclave Control Structure* (SECS). Esse tipo armazena os metadados do enclave e não é mapeado no espaço de endereçamento do mesmo, sendo de uso exclusivo da CPU. Durante a inicialização de um enclave, a primeira página alocada no EPC irá armazenar seu SECS e o último passo para destruição do mesmo é desalocar ela. Devido a isso, o SECS pode ser considerado como um sinônimo da identidade de um enclave.

A tecnologia Intel SGX armazena informações confidenciais no SECS, como por exemplo a sua medição (a ser abordada na Seção 2.3.3). Visto que a identificação dos enclaves existentes no EPC é realizada através dos endereços SECS, a aplicação que o implementa deve possuir uma entrada na sua tabela de páginas apontando para o mesmo. Contudo, a aplicação não consegue acessar as páginas SECS, pois estão armazenadas na PRM, o que impede um software mal-intencionado de realizar alterações, como por exemplo, na medição do enclave, o que comprometeria a segurança da tecnologia Intel SGX.

Para garantir a segurança das informações contidas na PRM, as mesmas são criptografadas por uma unidade de hardware chamada de *Memory Encryption Engine* (MEE), de modo que só é possível acessar esses dados através das interfaces dos enclaves que a compõe. O MEE usa um Carter-Wegman MAC (*Message Authentication Code*) dedicado de alta velocidade combinado com um algoritmo AES-CTR para realizar a criptografia dos dados e garantir a integridade dos mesmos enquanto estão sendo transferidos entre a memória e o processador (AUMASSON; MERINO, 2016). Nesse cenário, a responsabilidade de carregar os códigos e dados iniciais do enclave para o EPC é transferida para a aplicação que o implementa. Para garantir a segurança dos dados residentes no EPC, o SGX valida todo o processo de inicialização de um enclave e recusa operações que possam comprometer as garantias de segurança.

Todo acesso à PRM executado por agentes externos são bloqueados tratando-

os como uma referência a uma memória não existente. O acesso a páginas de memória utilizando instruções, como *MOV*, são conferidas pelo hardware seguindo o fluxograma descrito na Figura 5, o qual libera o acesso somente se o processo está dentro de um enclave e a página que está sendo acessada pertence ao mesmo enclave, caso contrário também é tratado como um referência inexistente (MCKEEN et al., 2013; INTEL, 2014; COSTAN; DEVADAS, 2016).

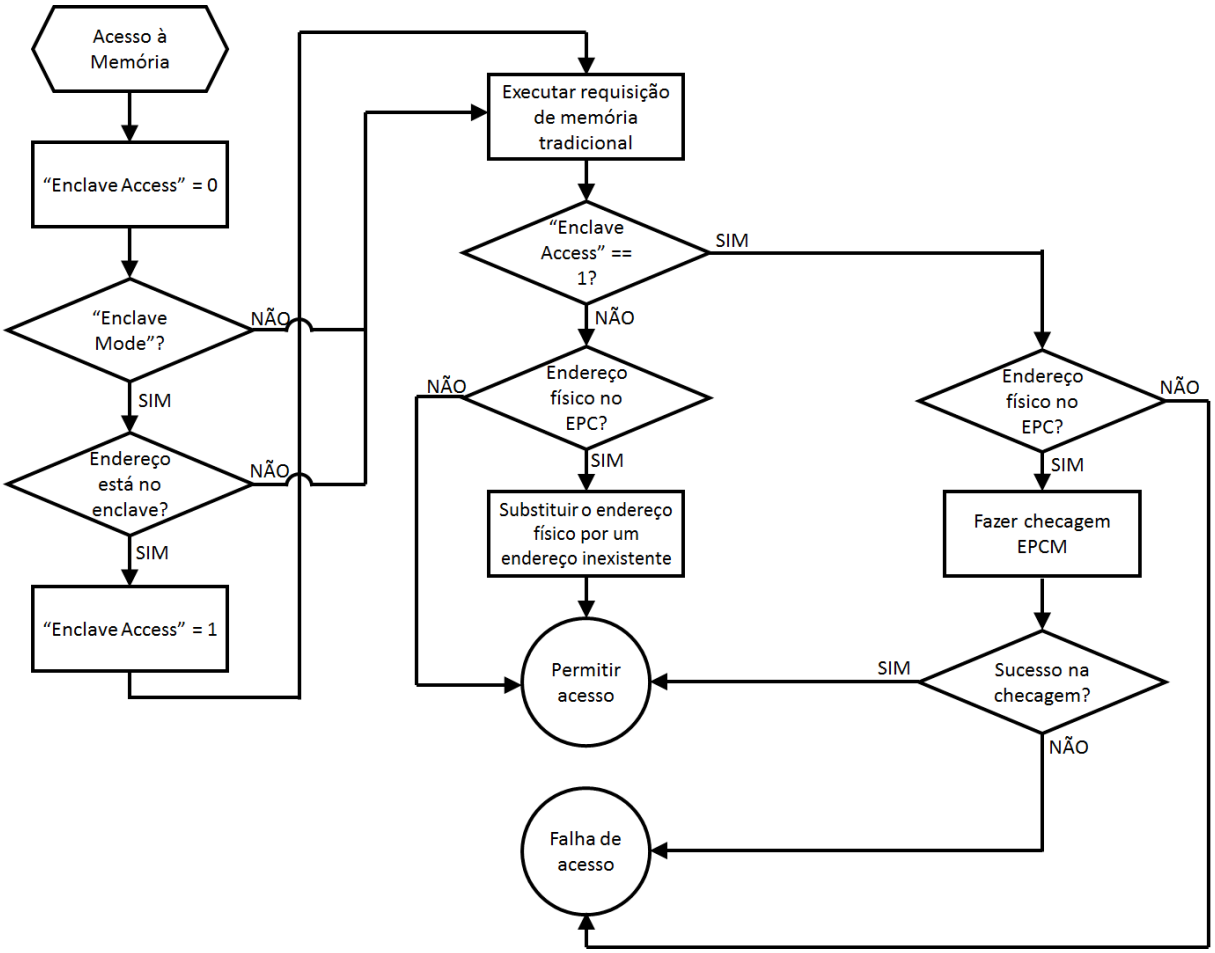

Figura 5: Fluxograma de acesso à memória do enclave.
Fonte: Will, Condé e Maziero (2017)

2.3.3 MEDIÇÃO E ASSINATURA DO ENCLAVE

Assinar um enclave consiste na criação de uma estrutura de assinatura chamada de *Enclave Signature* (SIGSTRUCT), a qual armazena informações sobre o mesmo e permite à tecnologia Intel SGX detectar adulterações e provar a integridade do enclave após o mesmo ser carregado no EPC. O algorítmo RSA-3072 PKCS#1 v1.5 com SHA-256 é utilizado para computar a assinatura do enclave (AUMASSON; MERINO, 2016).

A SIGSTRUCT é composta pelas seguintes informações:

- *Product ID* (ISVPRODID): O autor deve atribuir um identificador de produto com a sua identidade de autor (fornecida pela Intel) para cada enclave. Essa informação é registrada pela CPU após a inicialização e pode ser usada para atestação do mesmo.
- *Security Version Number* (ISVSVN): Cada enclave possui um SVN (*Security Version Number*) o qual identifica o controle de versão sobre as propriedades de segurança. Quando uma melhoria nas propriedades de segurança é realizada no enclave, o SVN deve ser incrementado. No entanto, se uma nova versão do enclave é gerada com correção de *bugs* e as propriedades de segurança não são alteradas, o SVN deve permanecer o mesmo. Assim como o *Product ID*, o SVN também é registrado na CPU após a inicialização e pode ser usado para atestação do enclave.
- Chave Pública do Autor: Um *hash* da chave pública do autor é armazenado pela CPU no registrador MRSIGNER após a inicialização do enclave. Esse *hash* servirá como identidade do autor do enclave, dessa forma outros enclaves do mesmo autor possuirão a mesma informação como identidade.
- *Enclave Measurement*: Um *hash* de 256 *bits* é gerado a partir do código, dos dados iniciais, sua a ordem e posição, e das propriedades de segurança das páginas que serão utilizadas pelo enclave. Após o carregamento do enclave para o EPC, a CPU realiza a medição do mesmo e armazena o valor no registrador MRENCLAVE. Logo após é realizado a comparação deste valor com o valor contido no SIGSTRUCT, e somente se ambos forem idênticos o enclave pode ser inicializado.

No entanto, o hardware apenas valida o enclave ao ser carregado, devido a isso a SIGSTRUCT também identifica o seu autor, a fim de evitar que qualquer desenvolvedor possa modificar um enclave e assiná-lo com sua própria chave.

O *Software Development Kit* (SDK) fornecido pela Intel inclui a ferramenta *Enclave Signing Tool*, que realiza o processo de assinatura e avaliação da imagem do enclave em busca de potenciais erros e problemas de segurança. O processo de assinatura do enclave, em sua versão de produção, deve ser efetuado em um ambiente separado e seguro. Para versões de teste e depuração, o SGX permite que a assinatura do enclave seja efetuada no mesmo ambiente de desenvolvimento (WILL; CONDÉ; MAZIERO, 2017).

2.3.4 ATESTAÇÃO ENTRE ENCLAVES

Em sistemas multiprocessados a troca de informações dentro dos diversos módulos da aplicação e entre aplicações, sejam remotas ou locais, é uma necessidade constante. Para que uma informação que está protegida em um enclave em execução possa ser compartilhada com outros enclaves é necessário garantir que ela estará segura em seu destino e durante a transmissão. Para isso a tecnologia Intel SGX fornece um recurso chamado de atestação, o qual permite que um enclave prove para terceiros (software em execução em uma plataforma habilitada para Intel SGX e protegida dentro de um enclave) que é legítimo, não foi adulterado e que foi carregado corretamente, permitindo criar um canal seguro para comunicação entre ambos.

A tecnologia Intel SGX fornece mecanismos que permitem duas formas de atestação: a atestação local e a atestação remota.

A atestação local, ou intra-plataforma, é realizada quando dois enclaves em um mesmo dispositivo precisam trocar informações entre si de forma segura. Neste caso um enclave deve provar para o outro sua identidade e autenticidade para que a comunicação seja iniciada. Esta forma de atestação utiliza um sistema de chaves simétricas, as quais são conhecidas somente pelo enclave que irá verificar a estrutura REPORT e o hardware que cria o REPORT. Para que que a atestação local seja realizada, 6 passos são necessários, os quais são representados pela Figura 6:

1. O enclave A envia seu MRENCLAVE para o enclave B usando comunicação aberta;
2. Este por sua vez pede à CPU para produzir uma credencial chamada REPORT através da instrução EREPORT, usando o MRENCLAVE recebido de A;
3. O enclave B envia a credencial gerada pela CPU para o enclave A ainda usando comunicação aberta;
4. O enclave A verifica com a CPU se B é um enclave local e legítimo;
5. O Enclave A gera uma credencial REPORT usando o MRENCLAVE da credencial que recebeu, e envia-a para o enclave B;
6. O enclave B verifica a credencial recebida do enclave A a fim de comprovar se é um enclave local e legítimo e, caso comprove, o processo de atestação é concluído;

A atestação remota, ou inter-plataforma, é realizada quando enclaves estão em execução em dispositivos distintos e precisam trocar informações entre si de forma segura.

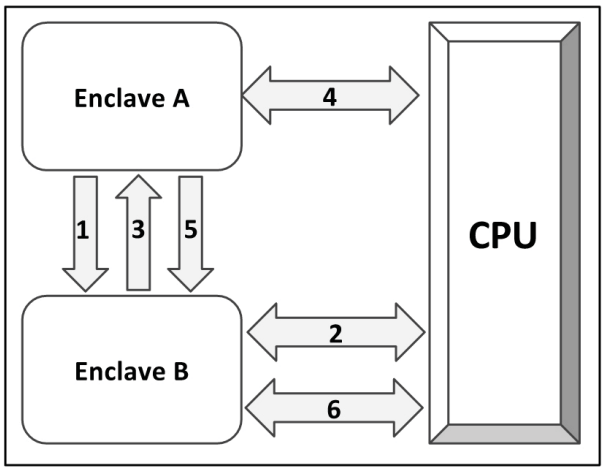

Figura 6: Atestação local entre enclaves.
Fonte: Condé (2017)

Esta forma de atestação requer o uso de criptografia assimétrica, de um enclave especial provido pela Intel, chamado de *Quoting Enclave*, e de uma credencial produzida pela CPU, chamada de *QUOTE*. A credencial *QUOTE* é gerada a partir da substituição do *Message Authentication Code* (MAC) dos REPORTs gerados pelos enclaves locais por uma assinatura criada através de uma chave assimétrica utilizando o Intel *Enhanced Privacy ID* (EPID). Para garantir a segurança da chave Intel EPID, somente o *Quoting Enclave* instanciado tem acesso à ela. A Figura 7 estabelece a sequência de passos a ser executada para que dois enclaves remotos realizem a atestação e se comuniquem de forma segura.

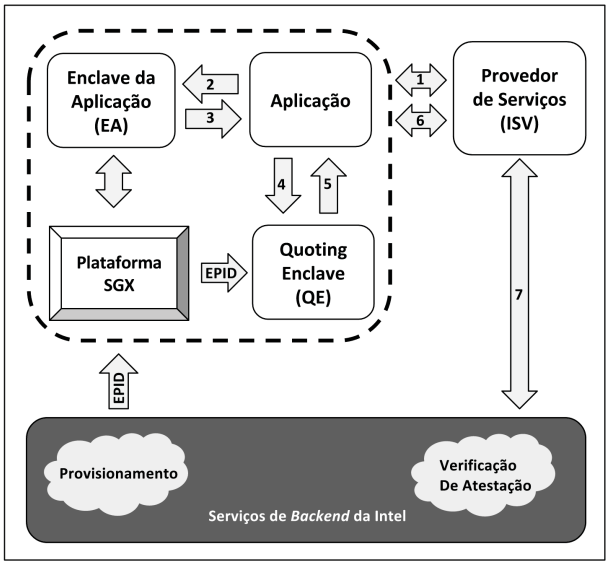

Figura 7: Atestação remota entre enclaves.
Fonte: Condé (2017)

1. A aplicação inicia a comunicação com o provedor de serviços e este solicita que ela prove que está executando os componentes necessários dentro de enclaves;
2. A solicitação do provedor de serviços é encaminhada para o enclave, juntamente com a identidade do *Quoting Enclave*;
3. Por sua vez, o enclave da aplicação gera uma chave pública efêmera, a qual será usada para as próximas comunicações, e então gera a credencial REPORT, enviando esta para a aplicação;
4. A aplicação envia a credencial para o *Quoting Enclave* para que ele assine-a;
5. O *Quoting Enclave* verifica a credencial recebida, cria a credencial *QUOTE*, assinando-a com a sua chave EPID, e a devolve à aplicação;
6. A aplicação envia a estrutura *QUOTE* para o provedor de serviços;
7. O provedor de serviços utiliza um verificador de autenticação a fim de identificar se a aplicação satisfaz os requisitos.

Após todas as verificações serem realizadas com sucesso, o servidor irá estabelecer um canal de comunicação seguro com o enclave usando a chave criptográfica gerada no passo 3 (CONDÉ, 2017; WILL; CONDÉ; MAZIERO, 2017).

2.3.5 SELAGEM DE DADOS

A tecnologia Intel SGX garante a confidencialidade e integridade dos dados enquanto os mesmos estão dentro de um enclave, de modo geral, quando o mesmo é destruído, toda informação é perdida. Para preservar essas informações, a tecnologia Intel SGX fornece um mecanismo que permite ao enclave criptografá-las utilizando a *Sealing Key*, de modo que a informação poderá ser armazenada em disco com segurança. A *Sealing Key* é uma chave única gerada pela CPU para aquele enclave e naquela plataforma específica, não sendo necessário o armazenamento da mesma para a descriptografia dos dados. As operações de criptografia e descriptografia são chamadas de *sealing* e *unsealing*, respectivamente.

Existem dois modos para selagem dos dados de um enclave, dos quais o desenvolvedor deve escolher o mais adequado para a aplicação.

O primeiro modo consiste em usar o valor contido no registrador MRENCLAVE para derivar a chave, de modo que somente esta versão do enclave, nessa mesma

plataforma, poderá de-selar os dados selados por ele, o que impede que novas versões do mesmo enclave acessem essas informações de forma direta, sendo necessário descriptografar os dados antes de atualizar e depois criptografá-los novamente com a nova versão.

No segundo modo, a chave é derivada a partir do valor contido no registrador MRSIGNER, o qual contém a identificação do autor, juntamente com o *Product ID* do enclave. Esse modo garante dois benefícios, o primeiro permite que um enclave atualizado acesse os dados selados por uma versão antiga do mesmo sem precisar de um processo complexo, e o segundo é que outros enclaves do mesmo autor podem compartilhar os dados selados. Um enclave deve fornecer o *Security Version Number* (SVN) quando solicita a *Sealing Key* para a CPU. Nessa situação, o SVN fornecido pode ser menor do que o especificado na assinatura do enclave (ISVSVN), o que garante acesso a dados selados em versões anteriores do mesmo, no entanto o fornecimento de um SVN superior ao da assinatura é bloqueado, garantindo que versões anteriores do mesmo não acessem dados criados por versões mais atuais. Esse mecanismo de bloqueio evita que uma versão antiga do sistema, que pode possuir brechas na segurança, seja utilizada para obter dados sigilosos em um ambiente seguro (INTEL, 2016; CONDÉ, 2017; WILL; CONDÉ; MAZIERO, 2017). O processo descrito é mostrado na Figura 8.

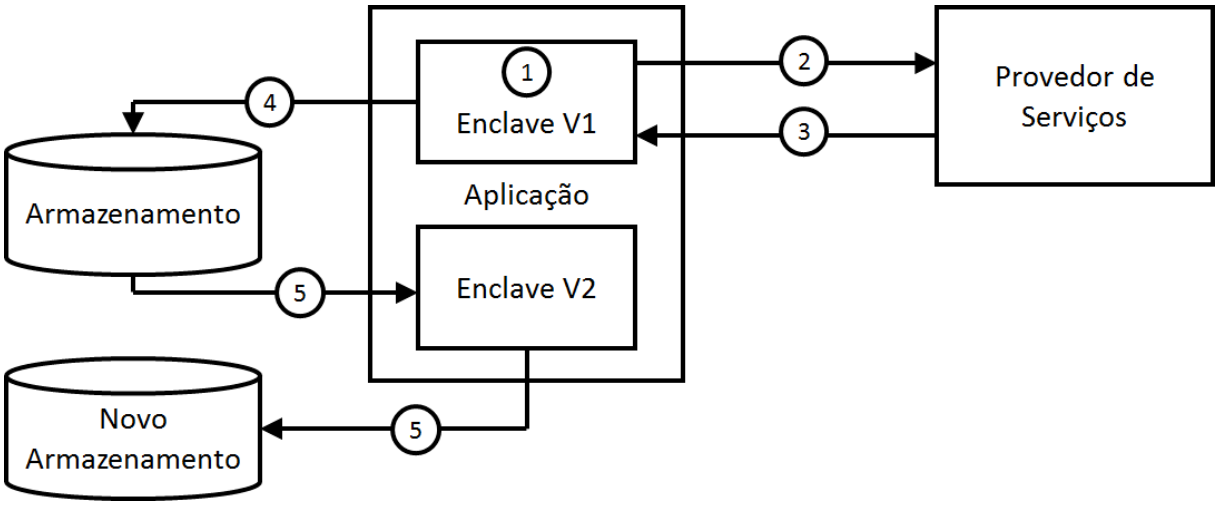

Figura 8: Fluxograma referente à integridade no armazenamento de dados. Fonte: Will, Condé e Maziero (2017)

A tecnologia SGX utiliza o algoritmo AES-GCM para selagem dos dados e o algoritmo AES-CMAC para derivação das chaves (AUMASSON; MERINO, 2016).

A Figura 9 demonstra o processo onde uma aplicação (parte não confiável) recebe

dados não criptografados e envia a um enclave (parte confiável) através de uma ECALL, este por sua vez solicita a *Sealing Key* para a CPU, criptografa a informação e devolve ela para a aplicação, a qual irá se encarregar de armazenar em disco a mesma. Para a desselagem de dados o processo é o mesmo, mudando apenas a ECALL executada, a qual irá usar a *Sealing Key* para descriptografar os dados e retornar para a aplicação.

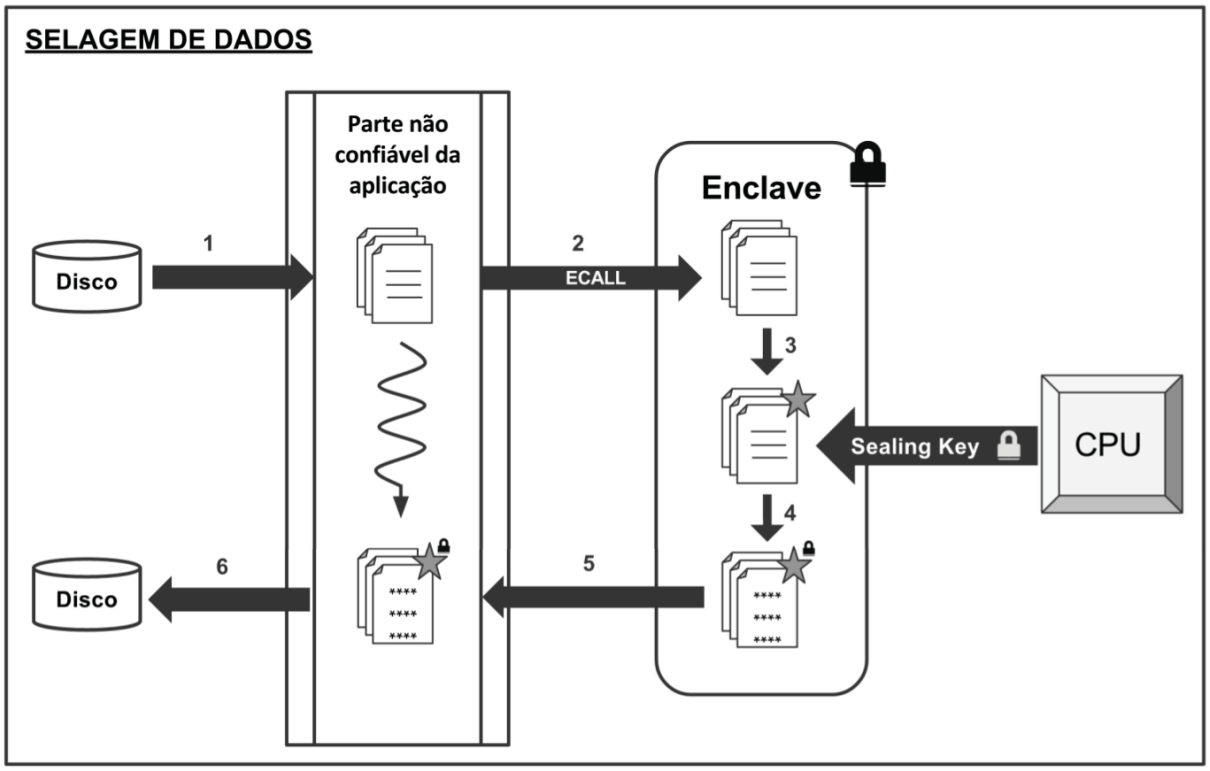

Figura 9: Selagem de dados: mecanismo de proteção dos dados sensíveis fora do enclave.

Fonte: Condé (2017)

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado uma breve descrição sobre o que é e como funciona a criptografia completa de disco, um resumo histórico sobre tecnologias que precedem a tecnologia SGX, além dos conceitos e detalhes de como ela trabalha internamente para garantir a segurança nas informações que estão sendo processadas. Tais conceitos são os alicerces da proposta deste trabalho, que será descrita no Capítulo 4.

3 ESTADO DA ARTE

O presente capítulo tem como objetivo elencar trabalhos relacionados que utilizam a tecnologia Intel SGX para criptografia de dados, um breve resumo sobre os mesmos e ao final uma comparação entre as aplicações apresentadas.

3.1 SELAGEM DE DADOS COMO UM MÓDULO DO *KERNEL*

Richter, Götzfried e Müller (2016) apresentam o conceito de isolar componentes do *kernel* do sistema operacional em enclaves a fim de impedir que vulnerabilidades em determinados módulos possam comprometer o sistema por completo.

Devido a restrições da tecnologia Intel SGX, o mesmo não pode ser executado diretamente pelo *kernel*, dessa forma, foi necessária a inclusão de um *daemon*[1] executando em modo usuário para se comunicar com um LKM (*Loadable Kernel Module* - Módulo carregável do *kernel*), conforme descrito na Figura 10.

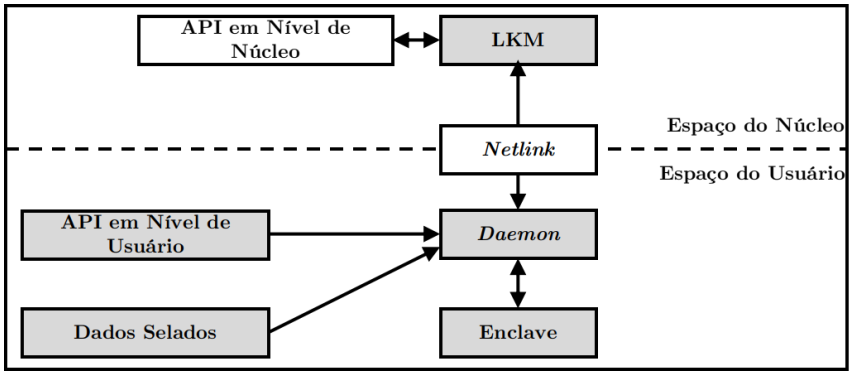

Figura 10: Um LKM utilizando um recurso provido por um enclave. O LKM se comunica com um *daemon* de modo de usuário que encaminha as solicitações para o enclave que reside no espaço do usuário.

Fonte: Adaptado de Richter, Götzfried e Müller (2016).

[1] *Daemon* é um processo de serviço que é executado em segundo plano e supervisiona o sistema ou fornece uma funcionalidade para outros processos.

Como prova de conceito do trabalho apresentado, os mesmos criaram um *LKM* que registra um novo modo dentro da API de criptografia do *kernel*, permitindo realizar o processo de criptografia em um enclave, que pode ser utilizado para a criptografia de discos. O algoritmo de criptografia que foi movido para dentro do enclave é o AES-NI (*Advanced Encryption Standard - New Instructions*), o qual utiliza instruções nativas do processador para realizar o processo, aumentando o desempenho do sistema (INTEL, 2019).

O fato de mover o processo de criptografia para dentro de um enclave não elimina todas as vulnerabilidades inerentes. Assim, a chave de criptografia utilizada é derivada a partir da senha fornecida pelo usuário e um *salt*[2] gerado para aquela plataforma, o qual é selado usando o Intel SGX, além de que a chave gerada nunca sai dos limites do enclave. Dessa forma, um invasor precisa além do acesso aos dados selados, o acesso à senha do usuário, e do *salt* descriptografado, o qual só é possível obter através da *CPU* em que ele foi gerado.

Para comparar a aplicação criada, foram realizados três testes de velocidade de gravação e leitura de dados em um mesmo HD, sendo eles: i) dados sem criptografia; ii) criptografia AES fornecida pelo *kernel* do Linux e, por último; iii) a criptografia através de enclaves usando AES-NI. Os testes realizados indicam que a criptografia utilizando enclaves, na forma em que o projeto foi desenvolvido, possui apenas 1% da performance em relação aos demais testes quando realizado leitura direta do disco. Quando realizado leitura através do cache, a aplicação forneceu 11,8% e 13,1% de performance em relação a leitura sem criptografia, e em relação ao modo AES, respectivamente. Os testes foram realizados pelas ferramentas *DD* e *Hdparm* e são exibidos na Tabela 1.

Tabela 1: Leitura e escrita, velocidades em MB/s de diferentes operações com acesso simples ao disco, a implementação AES padrão, e a implementação com SGX.

Teste	**Dados sem cript.**	**AES**	**SGX**
DD - gravação de 100MB	107,0	104,5	1,1
Hdparm - leitura de dados sem cache	110,1	113,7	1,1
Hdparm - leitura de dados em cache	13289,5	12004,3	1576,7

Fonte: Adaptdado de Richter, Götzfried e Müller (2016).

[2] *Salt* são dados aleatórios usados para modificar um *hash* de uma senha.

3.2 SELAGEM DE DADOS NO FUSE

A proposta apresentada por Burihabwa et al. (2018) consiste na inclusão da criptografia de dados dentro de um sistema de arquivos baseado no *FUSE* usando a tecnologia Intel SGX, a fim de garantir que os dados armazenados estejam seguros.

O FUSE (*Filesystem in Userspace*) é uma interface simples, a qual permite que programas executando no espaço do usuário possam exportar um sistema de arquivos virtual para o *kernel* do Linux, permitindo que usuários sem privilégios elevados possam criar e montar seus próprios sistemas de arquivos (KERRISK, 2019). A interface fornece ao sistema de arquivos desenvolvido o controle sobre as requisições vindas do sistema operacional, permitindo que os dados dos arquivos sejam tratados antes de serem enviados ao sistema operacional ou serem armazenados em disco, além de permitir a manipulação de dados que estão armazenados remotamente.

A Figura 11 descreve a aplicação desenvolvida para validação da proposta, na qual as requisições de arquivos feitas para o sistema operacional, através do sistema de arquivos virtual, são interceptadas pela biblioteca *FUSE* e enviadas para um enclave, o qual realiza o processo de criptografia e descriptografia dos dados, utilizando para isso o recurso nativo de selagem de dados fornecido pela tecnologia Intel SGX.

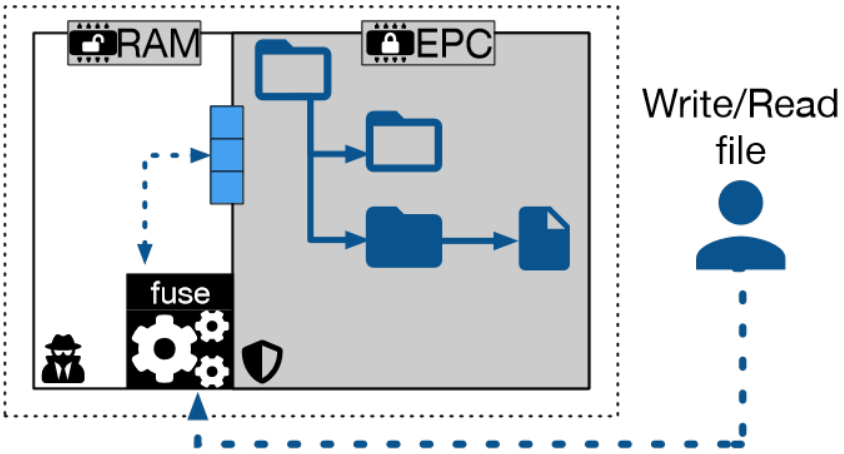

Figura 11: Arquitetura da solução SGX-FS.
Fonte: Adaptado de Burihabwa et al. (2018).

Para avaliação da aplicação foi realizado um conjunto de micro-*benchmarks* comparando três soluções baseadas na interface FUSE, onde, apesar do custo de entrada e saída nos enclaves SGX, a aplicação alcançou bom desempenho.

Nesse cenário, arquivos que são manipulados através da aplicação desenvolvida

possuem as garantias de segurança providas pela tecnologia Intel SGX (BURIHABWA et al., 2018).

3.3 SELAGEM DE DADOS RESISTENTE A ATAQUES DE CANAL LATERAL

O trabalho desenvolvido por Ahmad et al. (2018) possui foco em diminuir as chances de sucesso em um ataque de canal lateral sobre um sistema de arquivos baseado na tecnologia Intel SGX.

Ataques de canal lateral são ataques que fazem uso de informações vazadas durante a execução do processo de criptografia dos dados. Esses tipos de ataques utilizam variações no tempo de execução, no consumo de energia, emanações eletromagnéticas e outras características do dispositivo alvo a fim de obter dados sensíveis da aplicação em execução (LADEIRA et al., 2016).

De acordo com Ahmad et al. (2018), existem três principais tipos de ataques de canal lateral que afetam a tecnologia Intel SGX, sendo eles: *syscall*[3], *page fault*[4], e *cache*. A aplicação proposta limita-se em prevenir ataques baseados em *syscall* e *page fault* utilizando como base a divisão dos processos em dois enclaves e a implementação do protocolo ORAM.

ORAM (Oblivious RAM) consiste em um protocolo que ofusca as operações de leitura e gravação efetuadas em memória, seja ela primária ou secundária, usando para isso embaralhamento e criptografia dos dados com um novo *nonce*[5] a cada ação executada, fazendo com que o atacante não tenha certeza de qual região da memória foi acessada ou qual operação foi executada. Dessa forma, a técnica impede que atacantes possam aprender padrões sobre os dados acessados por um programa ou usuário (PINKAS; REINMAN, 2010).

Obliviate consiste em uma biblioteca que possui um sistema de arquivos executando dentro de um enclave, e uma aplicação executando em outro enclave fazendo requisições ao sistema de arquivos no enclave da biblioteca. A comunicação entre o enclave da biblioteca e o enclave da aplicação é realizada através de uma fila de mensagens enviadas por canais de comunicação criptografados entre os processos, evitando, dessa forma, o acesso indevido aos dados da aplicação através de acesso direto à memória.

[3] *Syscall* (*system calls*) são chamadas nas quais um programa solicita um serviço do *kernel*.

[4] *Page fault* ocorre quando um programa tenta acessar os dados (ou código) que está em seu espaço de endereço, mas não está carregado na memória primária do sistema no momento.

[5] O termo *nonce* significa “número usado uma vez” (number used once).

Para avaliação da aplicação foram realizados testes usando a ferramenta *Iozone*[6], a fim de medir o tempo necessário para as operações de abertura de arquivos do disco, preenchimento da estrutura ORAM e escrita do arquivo em disco novamente. Os dados obtidos são apresentados na Tabela 2 (AHMAD et al., 2018).

Tabela 2: Desempenho de operações abertura e de escrita no Obliviate (em milissegundos).

	16 MB	**128 MB**	**512 MB**	**1 GB**
Abertura	3,145	7,200	14,765	21,624
Preenchimento	1,990	5,100	7,878	12,323
Escrita	2,967	4,900	10,907	16,635

Fonte: Adaptado de Ahmad et al. (2018).

3.4 COMPARATIVO ENTRE AS SOLUÇÕES

Dentre as soluções apresentadas nas seções anteriores, todas tem em comum a preocupação com a segurança dos dados e o uso da tecnologia Intel SGX para fornecer um nível a mais de segurança. O Quadro 1 realiza uma comparação a partir dos dados apresentados por cada um dos projetos sobre três pontos principais que foram identificados.

3.5 CONSIDERAÇÕES FINAIS

Este capítulo abordou trabalhos que foram desenvolvidos a fim de utilizar a tecnologia Intel SGX com a finalidade de adicionar um nível a mais de segurança no processo de criptografia de dados a serem armazenados em mídias secundárias.

A partir dos estudos apresentados, fica evidente o crescente uso da tecnologia em diversas soluções, e o aumento na preocupação em relação a novos tipos de ataques, os quais a mesma ainda é vulnerável.

[6]Detalhes disponíveis em `http://www.iozone.org`.

Quadro 1: Comparativo entre as soluções apresentadas.

Solução	**Forma de implementação**	**Desempenho**	**Análise de segurança**
Selagem de dados no *kernel*	Selagem implementada através da execução do algoritmo AES-NI dentro de um enclave, o qual é acessado por um LKM através de um *daemon*	Obteve desempenho de apenas 1% ao comparar com o armazenamento sem criptografia ou criptografando através do modo AES padrão	Chave para criptografia é derivada de um *salt* gerado para aquela plataforma, e a mesma não sai dos limites do enclave
Selagem de dados no FUSE	Selagem implementada através do uso da biblioteca FUSE, criptografando e descriptografando os dados requisitados pelo sistema operacional	Os resultados não apresentam comparação da taxa de transferência em relação a outros modos, apenas o tempo em segundos para cópia de arquivos entre partições FUSE, os quais indicam que a implementação obteve um bom resultado	Uso da *sealing key* para selagem dos dados, fornecendo todas as garantias de segurança da tecnologia Intel SGX
Selagem de dados resistente a ataques de canal lateral	Selagem implementada em conjunto com o uso de um protocolo ORAM, a fim de reduzir as chances de sucesso de ataques de canal lateral baseadas em *syscall* e *page fault*	Os resultados idicam que a aplicação obteve bom desempenho nos testes de leitura, preenchimento da estrutura ORAM e gravação dos dados	Uso da tecnologia Intel SGX para fornecer as garantias de segurança durante o processo de selagem em conjunto com um protocolo ORAM, o qual tem a finalidade de diminuir as chances de sucesso em ataques de canal lateral

Fonte: Autoria própria

4 PROPOSTA

Este capítulo apresenta a proposta de alteração em um software de criptografia de disco para que o mesmo utilize o recurso de selagem de dados, provido pela tecnologia Intel SGX e que foi descrito na Seção 2.3.5, com o intuito de proporcionar maior segurança nos dados armazenados em disco.

A Seção 4.1 descreve a metodologia utilizada na escolha do software e como será realizada a alteração e validação do mesmo. A Seção 4.2 é destinada a detalhar o funcionamento da ferramenta escolhida, já na Seção 4.3 são detalhadas as alterações efetuadas em tal ferramenta, de forma a integrar a selagem de dados provida pelo SGX. Por fim, a Seção 4.4 apresenta uma visão geral da arquitetura proposta.

4.1 METODOLOGIA DE DESENVOLVIMENTO

Para a escolha do software que foi alterado, foram levados em conta os seguintes critérios:

- ***Open Source*:** O software escolhido deve ser *Open Source* para que permita alteração em seu código fonte.
- **Multiplataforma:** Preferencialmente, o software deve ser multiplataforma, de modo que a implementação do projeto não fique restrita a um único sistema operacional.
- **Projetos em constante manutenção:** Projetos encerrados não foram considerados, pois suas implementações podem conter erros que comprometam a validação da proposta.

Após a realização de buscas na internet a fim de localizar um software que melhor se adéque aos critérios estabelecidos anteriormente, foi encontrado o aplicativo

Cryptomator, um software *open source* que é multiplataforma e é um projeto recente, tendo iniciado seu desenvolvimento em 2014, e que está em constante manutenção.

Por ser um software bem estruturado, permite a inclusão da criptografia de dados provida pela tecnologia Intel SGX sem a necessidade de remover o modo de criptografia existente, mantendo dessa forma a compatibilidade do mesmo com plataformas que não possuem um processador com a tecnologia SGX disponível, além de permitir que correções efetuadas no projeto principal possam ser facilmente incluídas nesta variante do mesmo. Essa alteração visa a utilização do ambiente seguro que os enclaves fornecem e a criptografia dos dados utilizando a *Sealing Key*, de modo que a informação armazenada em disco somente possa ser acessada através da plataforma que selou os dados.

4.2 ARQUITETURA DO SOFTWARE CRYPTOMATOR

O Cryptomator trabalha com a criptografia de contêineres de arquivos, ou seja, fornece ao sistema operacional do usuário um sistema de arquivos virtual onde os dados são lidos e escritos em outro local.

A aplicação é divida em três módulos principais, sendo eles:

1. **Cryptomator:** interface gráfica que fornece ao usuário o controle dos contêineres;
2. **CryptoFS:** biblioteca que implementa um sistema de arquivos virtual e é responsável pela leitura e gravação dos dados dentro dos contêineres através do sistema operacional, fornecendo ao sistema de arquivos os dados descriptografados e recebendo dados do mesmo e criptografando-os antes de armazenar em uma mídia secundária;
3. **CryptoLib:** responsável por prover funções para a criptografia e descriptografia dos dados dos arquivos, as quais são utilizadas pelo módulo CryptoFS.

O funcionamento dos processos de leitura e gravação em um ambiente Linux, onde a biblioteca FUSE é responsável por prover o sistema de arquivos virtual, é descrito nas Seções 4.2.1 e 4.2.2, respectivamente.

4.2.1 LEITURA DOS DADOS CRIPTOGRAFADOS ATRAVÉS DO CRYPTOMATOR

O processo de leitura dos arquivos provido pela aplicação pode ser descrito em 12 passos e é exemplificado pela Figura 12:

1. O usuário solicita ao sistema operacional que um arquivo seja aberto;
2. O sistema operacional solicita ao FUSE os dados do arquivo;
3. O FUSE solicita ao Cryptomator os dados do arquivo;
4. O Cryptomator, por sua vez, através da biblioteca CryptoFS, solicita ao sistema operacional que os dados do arquivo que estão armazenados em uma mídia secundária sejam carregados;
5. O sistema operacional localiza os dados na mídia e,
6. Carrega as informações para a memória principal,
7. E envia as mesmas para a CryptoFS;
8. A CryptoFS envia os dados criptografados para a CryptoLib;
9. A CryptoLib descriptografa os dados recebidos e retorna-os para a CryptoFS,
10. A CryptoFS envia os dados ao FUSE;
11. O FUSE, por sua vez, envia os dados ao sistema operacional;
12. O sistema operacional disponibiliza ao usuário o arquivo descriptografado.

4.2.2 GRAVAÇÃO DE NOVOS DADOS ATRAVÉS DO CRYPTOMATOR

Para salvar um arquivo, o processo inverso é realizado, e pode ser descrito também em 12 passos, sendo exemplificados pela Figura 13:

1. O usuário solicita ao sistema operacional que um arquivo seja salvo;
2. O sistema operacional envia os dados do arquivo ao FUSE;
3. O FUSE envia ao Cryptomator os dados do arquivo;

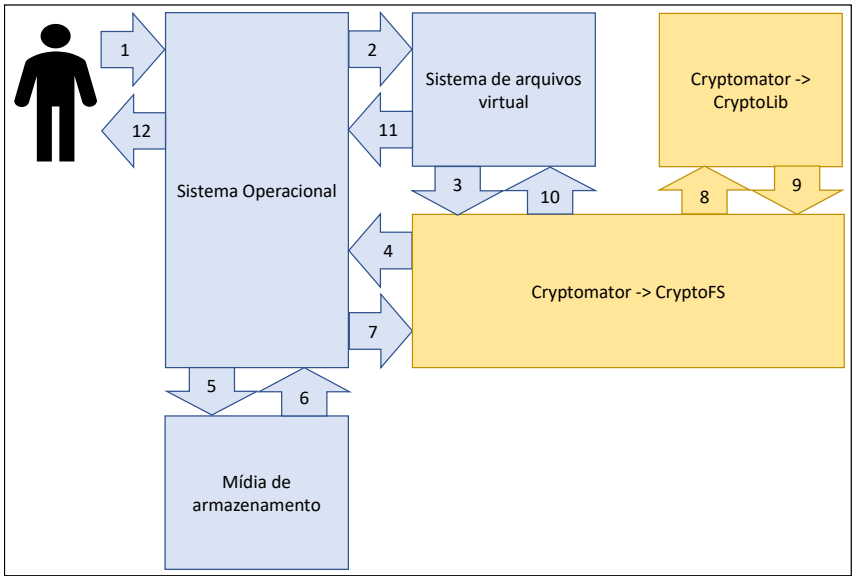

Figura 12: Fluxograma do processo realizado para leitura e descriptografia dos dados armazenados em mídia secundária executados pelo Cryptomator.

Fonte: Autoria própria.

4. O Cryptomator, por sua vez, através da biblioteca CryptoFS, envia os dados para a CryptoLib para serem criptografados;
5. A CryptoLib criptografa os dados do arquivo e envia para a CryptoFS novamente;
6. A CryptoFS solicita ao sistema operacional que os mesmos sejam armazenados em uma mídia;
7. O sistema operacional armazena os dados na mídia;
8. A mídia retorna ao sistema operacional os dados da localização onde o arquivo foi salvo;
9. O sistema operacional envia os dados da localização para o Cryptomator;
10. O Cryptomator armazena essas informações e informa ao FUSE que a operação foi concluída;
11. O FUSE informa ao sistema operacional que o arquivo foi salvo;
12. O sistema operacional informa ao usuário que a operação foi concluída.

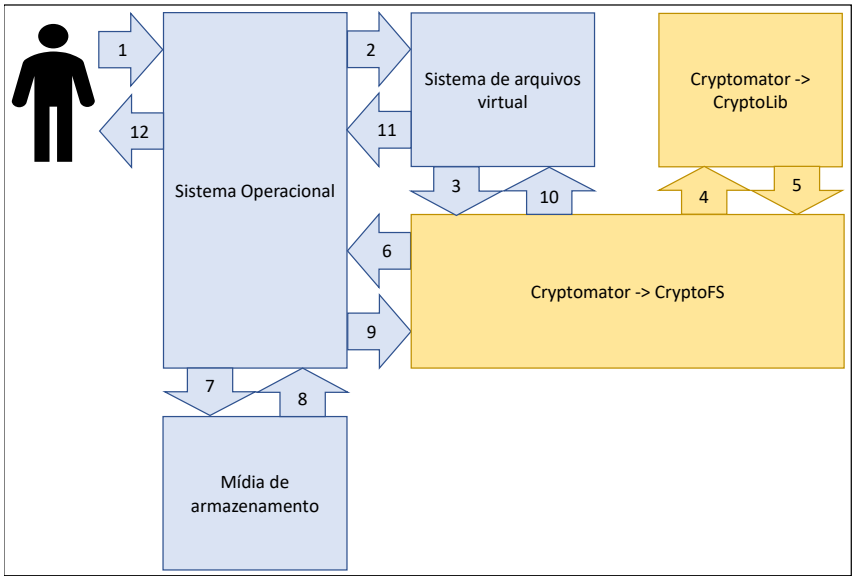

Figura 13: Fluxograma do processo realizado para criptografia e armazenamento dos dados em mídia secundária executados pelo Cryptomator.

Fonte: Autoria própria.

4.3 ARQUITETURA DO SOFTWARE CRYPTOMATOR COM A UTILIZAÇÃO DA SELAGEM DE DADOS

A proposta deste trabalho consiste na inclusão do mecanismo de selagem de dados provido pela tecnologia Intel SGX dentro da biblioteca CryptoLib em paralelo com a criptografia AES existente, e a alteração da biblioteca CryptoFS para que utilize a implementação realizada.

A alteração proposta garante, através do uso da criptografia AES existente, que a biblioteca CryptoLib ainda possa ser utilizada em ambientes onde a tecnologia Intel SGX não está disponível, mantendo assim a compatibilidade do projeto com alterações no projeto principal.

A Figura 14 descreve o processo de envio e recebimento dos dados entre a CryptoFS e a CryptoLib anterior à inclusão da selagem de dados do SGX, onde a comunicação se dava em dois passos, e como é o processo após a implementação do novo recurso, onde são incluídos dois passos adicionais para comunicação da CryptoLib com o enclave SGX.

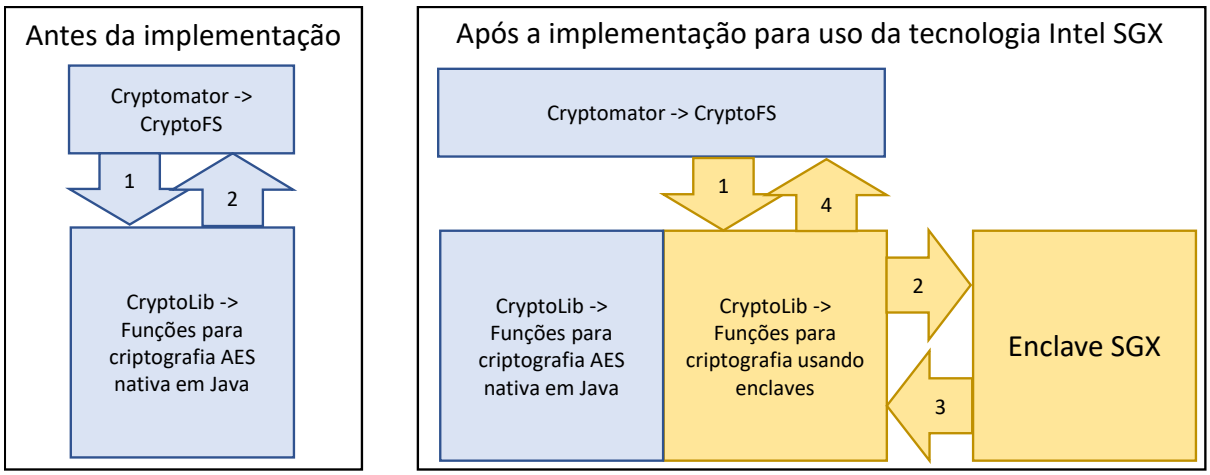

Figura 14: Fluxograma do processo realizado para criptografia e descriptografia dos dados antes e depois da implementação para uso da tecnologia Intel SGX.

Fonte: Autoria própria.

4.3.1 IMPLEMENTAÇÃO

Por ser um projeto recente e bem estruturado, o Cryptomator foi desenvolvido para suportar vários modos de criptografia sem que um afete a implementação do outro, ou que precise de mudanças no projeto principal para suportá-lo. A biblioteca CryptoLib é composta por uma interface do Java, chamada de "api", a qual descreve todas as classes e os métodos que cada modo de criptografia deverá implementar para que ele possa ser usado pelo software, e pela implementação do modo de criptografia AES com base na interface descrita, o qual é chamado de "v1" dentro da biblioteca.

A implementação realizada consistiu na criação de um novo modo chamado de "sgx", implementando todas as classes e métodos solicitados pela interface da biblioteca, e a alteração da biblioteca CryptoFS para utilizar o modo "sgx" no lugar do modo "v1". Foi necessário também a criação de uma biblioteca em C++ para comunicação com os enclaves, chamada de "SgxLib".

O modo "sgx" possui uma classe Java com 4 métodos principais utilizados por todas as demais classes do modo, sendo eles:

- **InitializeEnclave**: Método responsável pela inicialização do enclave SGX.
- **SgxEncryptBytes**: Método responsável por criptografar um *array* de *bytes*.
- **SgxDecryptBytes**: Método responsável por descriptografar um *array* de *bytes*.

- **DestroyEnclave**: Método responsável por finalizar o enclave após ele não ser mais necessário.

Os 4 métodos citados não possuem implementação nativa em Java, sendo utilizados apenas para invocar os métodos correspondentes dentro da biblioteca "SgxLib".

Devido ao fato que Java não é uma linguagem compilada nativamente, para que os métodos em Java se comuniquem com a biblioteca "SgxLib" foi necessário criar uma uma interface JNI (*Java Native Interface*), contendo os 4 métodos principais da classe em Java. Essa interface tem a finalidade de receber os dados da aplicação em Java, converte-los para a linguagem C++, realizar o processo descrito pelo método e então converter os resultados e enviá-los para o Java. O processo descrito é exemplificado pela Figura 15.

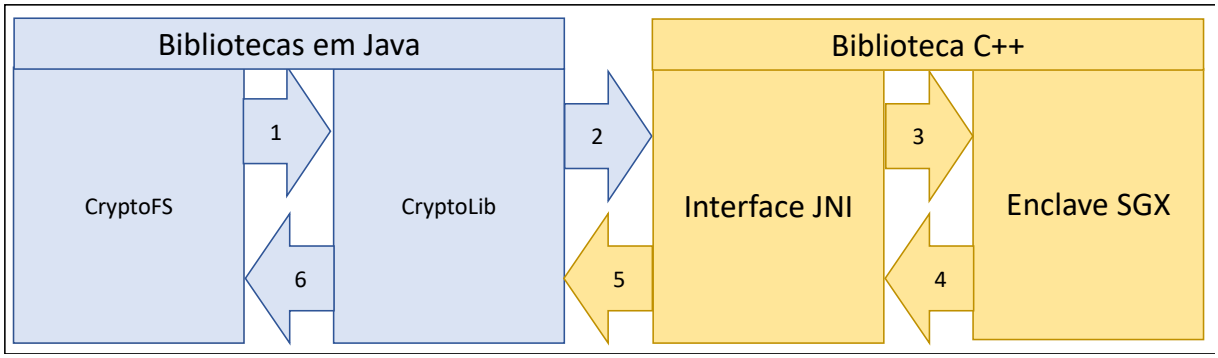

Figura 15: Fluxograma do processo realizado para comunicação entre as bibliotecas em Java e o enclave SGX.

Fonte: Autoria própria.

Nesse cenário, ao utilizar os métodos da CryptoLib, ela invoca os métodos implementados na biblioteca "SgxLib", a qual utiliza a interface JNI para acesso aos dados a serem processados, através de ponteiros, realiza as operações necessárias e envia os dados novamente para a máquina virtual Java.

4.4 CONSIDERAÇÕES FINAIS

O presente capítulo abordou a sistemática utilizada para a escolha do software que foi alterado, o recurso que foi customizado, como foi realizada a implementação, a validação da implementação e os resultados esperados a partir da mesma, além de uma breve descrição de como a aplicação escolhida trabalha em conjunto com o sistema operacional para fornecer ao usuário segurança aos arquivos armazenados em mídia secundária.

5 VALIDAÇÃO DA PROPOSTA

Este capítulo tem como objetivo descrever as etapas realizadas para a validação da proposta e os objetivos de cada uma, e, também, detalhar as limitações existentes na solução.

5.1 ANÁLISE DE DESEMPENHO

Um dos pontos levados em consideração por um usuário ao escolher um software para criptografia de disco é a perda de desempenho nos processos de gravação e leitura dos dados ao utilizá-lo.

Para melhor descrever os resultados obtidos a partir da solução implementada, foram realizados testes de performance comparando 5 modos de armazenamento dos dados, sendo eles:

- **Sem criptografia**: Operações de leitura e escrita de dados na mídia de armazenamento sem nenhum tipo de criptografia;
- ***Linux Unified Key Setup* (LUKS)**: Operações de leitura e escrita de dados na mídia de armazenamento utilizando o modo nativo de criptografia existente no Ubuntu, LUKS[1];
- **VeraCrypt**: Operações de leitura e escrita de dados na mídia de armazenamento através do aplicativo VeraCrypt[2] utilizando a solução de armazenamento em contêineres;
- **Cryptomator**: Operações de leitura e escrita de dados na mídia de armazenamento através do aplicativo Cryptomator, sem nenhuma alteração;

[1]Detalhes disponíveis em: `https://gitlab.com/cryptsetup/cryptsetup/wikis/home`
[2]Detalhes disponíveis em `https://www.veracrypt.fr/en/Home.html`

- **Cryptomator-SGX**: Operações de leitura e escrita de dados na mídia de armazenamento através da solução implementada, com a utilização da selagem de dados SGX no aplicativo Cryptomator.

A comparação de desempenho do Cryptomator-SGX com as aplicações LUKS e VeraCrypt se deu pelo fato de a primeira ser uma solução nativa do sistema operacional e a segunda ser um sistema multiplataforma de amplo uso, também sendo desenvolvido em linguagem compilada, gerando código nativo ao sistema.

Para cada um dos modos descritos anteriormente, foram executados quatro testes distintos, sendo dois em operações de gravação de dados, e dois em operações de leitura de dados. Os testes consistiram na transferência de um arquivo único, e na transferência de diversos arquivos em sequência. Para a tarefa, foi utilizada a imagem DVD ISO do sistema operacional CentOS[3] 7, que possui 4,27 GB de tamanho, como conjunto de dados para a transferência de arquivo único, e também os arquivos e pastas obtidos ao extrair a mesma imagem, usando o modo de transferência recursiva de diretórios, caracterizando a transferência de diversos arquivos.

A ferramenta utilizada para efetuar a transferência de dados foi a RSync[4], utilizando como parâmetros de execução os modos "-avhh" para arquivo simples e "-rvhh" para diretórios. Mais detalhes sobre os parâmetros utilizados são expostos abaixo:

- **-a**: Modo de cópia de arquivo único;
- **-r**: Modo de cópia recursiva de diretórios, onde todos os sub-diretórios e arquivos são copiados também;
- **-v**: Imprime no prompt de comando o que está sendo executado;
- **-h**: Modo em que os resultados são exibidos em um formato mais fácil de ler. Isso faz com que os números grandes saiam usando unidades maiores, com um sufixo K, M ou G. Se esta opção foi especificada uma vez, estas unidades são K (1000), M (1000 * 1000) e G (1000 * 1000 * 1000), se a opção for repetida, as unidades são potências de 1024 em vez de 1000.

Também foram utilizados dois processadores distintos, ambos com SGX habilitado, sendo eles:

[3]Detalhes disponíveis em: `https://www.centos.org/download/`
[4]Detalhes disponíveis em: `https://help.ubuntu.com/community/rsync`

- **Processador 1:** Intel i7 9700K, 8 núcleos e 8 *threads*, a 3.60 GHz;
- **Processador 2:** Intel Pentium G5500, 2 núcleos e 2 *threads*, a 3.80 GHz.

Além disso, os dois processadores citados foram combinados, cada um, com três dispositivos de armazenamento diferentes. Em todas as mídias, foi utilizada uma partição com o sistema de arquivos Ext4, sendo que cada uma delas contém características distintas no que diz respeito ao desempenho para leitura e gravação de dados:

- **Dispositivo 1:** HDD Samsung ST1000LM024, 1 TB de armazenamento, 5400 RPM, taxa máxima de transferência de dados de 145 MB/s;
- **Dispositivo 2:** SSD SanDisk PLUS, 240 GB de armazenamento, velocidade de leitura sequencial de até 530 MB/s, velocidade de gravação sequencial de até 440 MB/s;
- **Dispositivo 3:** SSD NVMe M.2 Samsung 970 EVO Plus, 250 GB de armazenamento, velocidade de leitura sequencial de até 3.500 MB/s, velocidade de gravação sequencial de até 2.300 MB/s.

Para o ambiente de testes foi montado um computador com 16 GB de memória RAM a 2666 MHz, placa mãe com *chipset* Z390, a qual suporta os 2 processadores utilizados, e o sistema operacional Ubuntu 16.04.6 LTS[5], utilizando a versão 4.15.0-51 do *kernel*.

Para obtermos um resultado mais estável, foram desativados todos os recursos de *boost* em ambos os processadores, tais como *TurboBoost* e *HyperThreading*. Além disso, os testes foram repetidos um total de 10 vezes em cada cenário, limpando os caches do sistema através do comando "sh -c "sync && echo 3 > /proc/sys/vm/drop_caches"" em cada execução, a fim de reduzir variações provindas do sistema operacional.

Devido ao fato de uma das mídias de armazenamento ser de alta performance (SSD M.2), e que em uma transferência de dados entre duas mídias o maior desempenho obtido é definido pela mídia com menor performance, foi optado por realizar os testes de gravação utilizando como origem dos dados e como destino para os testes de leitura um *RAMDisk*, a fim de que não houvesse limitação da velocidade pela origem ou destino dos dados respectivamente.

[5]Detalhes disponíveis em: `http://releases.ubuntu.com/16.04/`

RAMDisk consiste em alocar parte da memória RAM do computador e utilizá-la como mídia temporária de armazenamento, com a vantagem de possuir acesso mais rápido aos dados do que se utilizado uma mídia física. Para os testes foi criado um *RAMDisk* de 10 GB utilizando o parâmetro "ramfs" para evitar que fosse utilizado a memória SWAP[6] do Ubuntu durante os testes.

Para melhor detalhar os resultados obtidos, os mesmos foram separados em testes de gravação, Seção 5.1.1, e testes de leitura, Seção 5.1.2.

5.1.1 ANÁLISE DE DESEMPENHO NA GRAVAÇÃO DE ARQUIVOS

A análise de desempenho na gravação de dados foi separada em 4 gráficos, sendo 2 para gravação de arquivo simples, e 2 para gravação de arquivos múltiplos. Em cada gráfico estão agrupados os três dispositivos de armazenamento, juntamente com os cinco modos de transferência de dados.

A primeira análise efetuada diz respeito à gravação de arquivo único, a saber, a imagem ISO do sistema operacional CentOS descrita anteriormente. A Figura 16 apresenta os diferentes resultados obtidos, através dos dispositivos de armazenamento e transferência de dados, utilizando o processador Intel Core i7 9700K. Já a Figura 17 apresenta os resultados para o processador Intel Pentium G5500.

Analisando tais resultados, pode-se notar que os modos de transferência que se utilizam da capacidade de multiprocessamento alcançam taxas de transferência maiores, em relação ao Cryptomator e Cryptomator-SGX, que se limitam a utilizar apenas um núcleo de processamento. Também destaca-se que, em relação ao processador Intel Pentium G5500, tanto o Cryptomator quanto o Cryptomator-SGX alcançam maiores taxas de transferência, devido ao maior *clock* de processamento, o que reduz a diferença em relação aos outros três métodos.

Também nota-se um desempenho superior da solução Cryptomator-SGX, em relação à implementação original do Cryptomator, devido ao fato da primeira se utilizar das funções de criptografia nativas do processador, visto que a execução dentro do enclave é a partir de código nativo, escrito em C++, e otimizado pelo compilador. Vale destacar também que as operações de transferência de dados pela interface JNI, para a comunicação com o enclave, e as trocas de contexto geradas pela entrada e saída do enclave têm um custo computacional considerável, que, nesses dois cenários, é totalmente suprimido pelo

[6]SWAP é uma quantidade de memória alocada dentro de uma mídia de armazenamento secundária, a qual é usada pelo sistema operacional quando a memória (RAM) disponível foi totalmente utilizada.

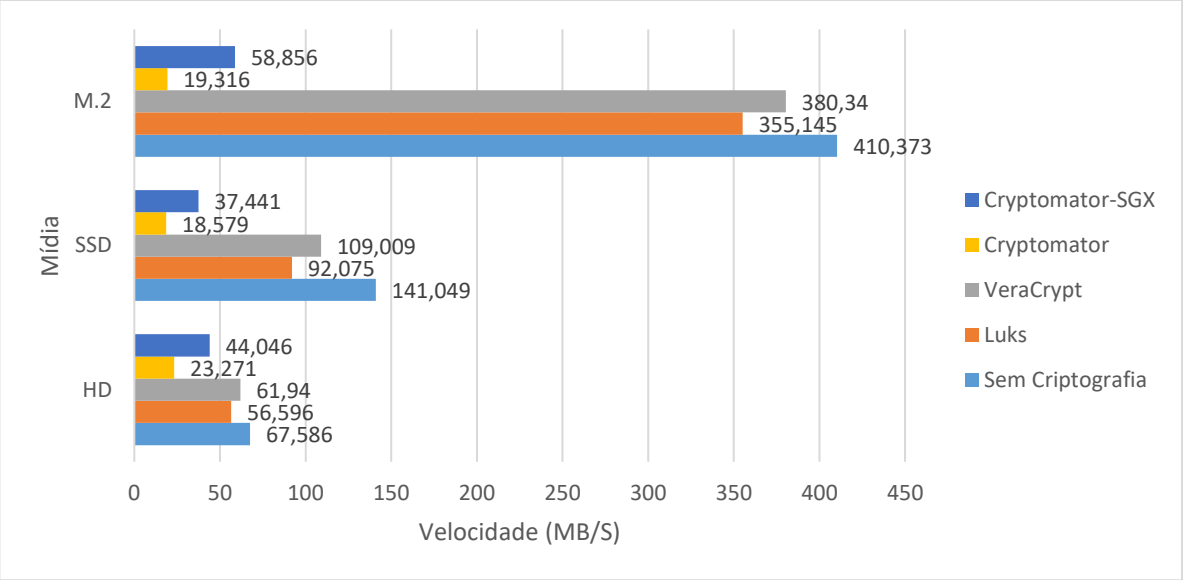

Figura 16: Gravação de arquivo único usando o processador Intel Core i7 9700K. Fonte: Autoria Própria.

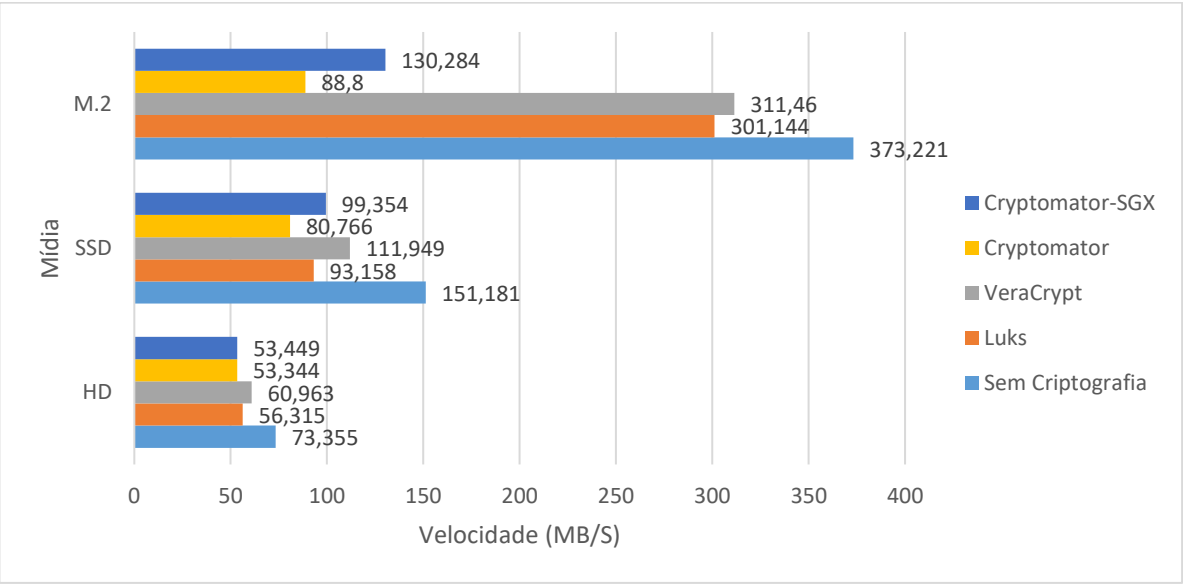

Figura 17: Gravação de arquivo único usando o processador Intel Pentium G5500. Fonte: Autoria Própria.

ganho obtido na operação de selagem dos dados.

A segunda análise de gravação de dados se utiliza de diversos arquivos de tamanhos distintos, buscando identificar o impacto causado pelas operações realizadas pelo sistema de arquivos para construir as referências a tais arquivos. Para tal cenário,

foram extraídos os arquivos contidos na imagem ISO apresentada anteriormente, e efetuada a cópia de todo o seu conteúdo, conforme já mencionado.

A Figura 18 apresenta os resultados obtidos com a utilização do processador Intel Core i7 9700K. Nesse cenário nota-se uma queda considerável de desempenho, tanto na implementação original do Cryptomator, quanto na solução proposta nesse trabalho. Apesar disso, os resultados obtidos com o uso do Cryptomator-SGX apresentam pouca diferença em relação ao Cryptomator não modificado.

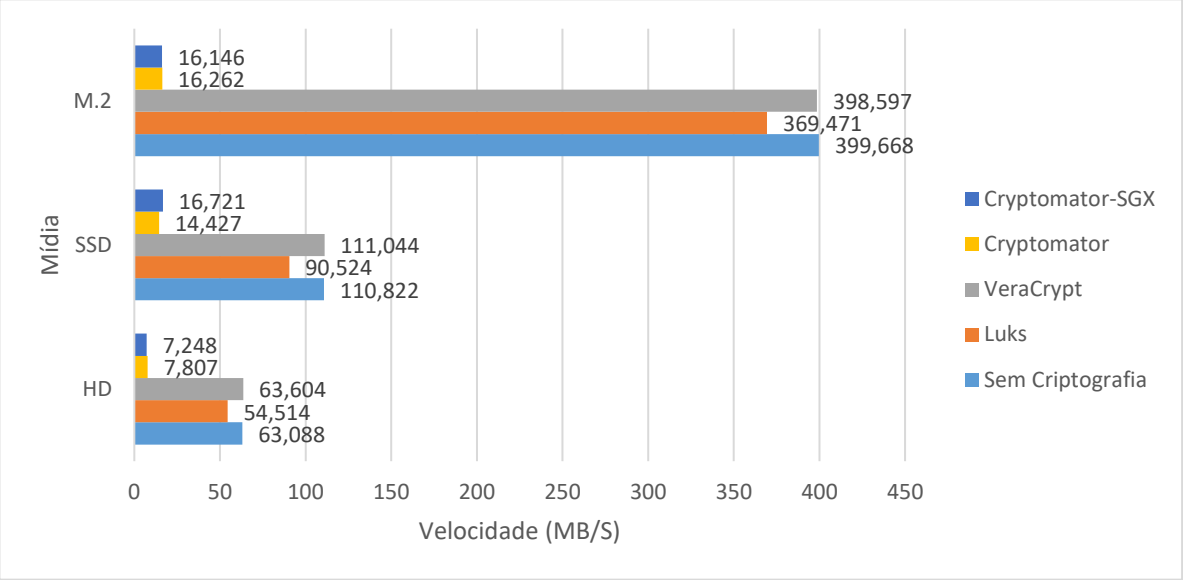

Figura 18: Gravação de múltiplos arquivos usando o processador Intel Core i7 9700K.

Fonte: Autoria Própria

Na Figura 19 são apresentados os dados obtidos com a utilização do processador Intel Pentium G5500. Novamente nota-se um desempenho superior do Cryptomator e Cryptomator-SGX, quando comparado com os resultados obtidos no processador Intel Core i7 9700K. Apesar da solução Cryptomator-SGX apresentar uma queda considerável de desempenho quando utilizado o dispositivo de armazenamento M.2 juntamente com o processador Intel Pentium G5500, quando comparado diretamente à implementação original do Cryptomator, com as outras mídias de armazenamento foram obtidas taxas de transferência superiores a 80%, na comparação direta.

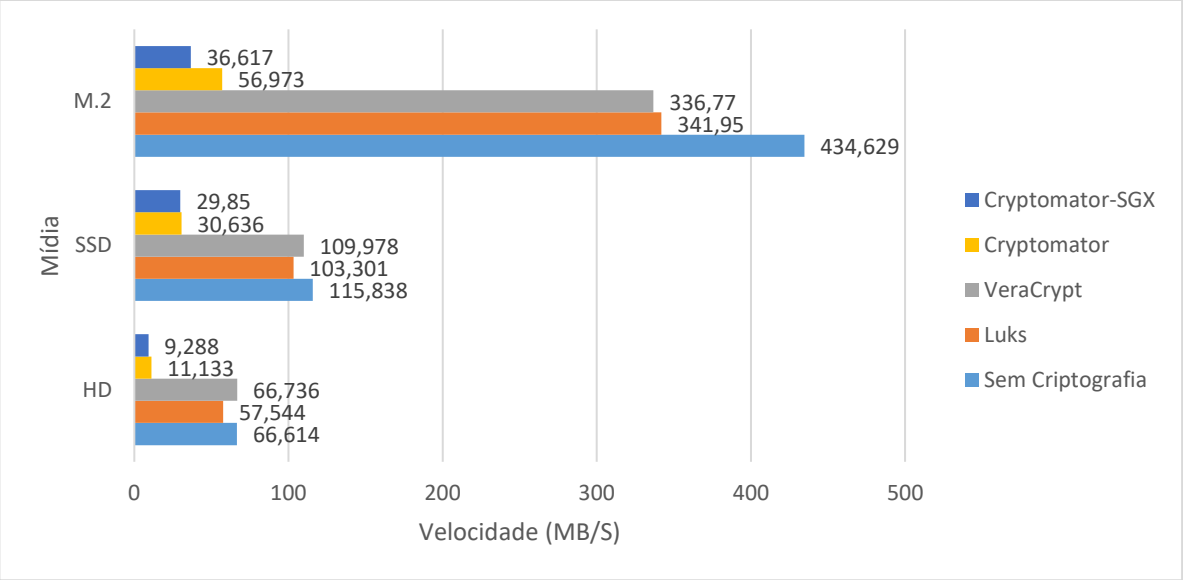

Figura 19: Gravação de múltiplos arquivos usando o processador Intel Pentium G5500.

Fonte: Autoria Própria.

5.1.2 ANÁLISE DE DESEMPENHO NA LEITURA DE ARQUIVOS

Um segundo conjunto de testes foi efetuado com o objetivo de analisar o desempenho na leitura de dados previamente armazenados em cada um dos dispositivos. Da mesma forma que a gravação de dados, foram testadas a leitura de arquivo único, e de um conjunto de arquivos distintos.

A Figura 20 traz os resultados obtidos na leitura de arquivo único, utilizando o processador Intel Core i7 9700K. Pode-se observar, nesse cenário, que o aplicativo Cryptomator obteve taxas de transferência próximas ou superior ao LUKS, com a solução Cryptomator-SGX alcançando um desempenho superior aos dois nos três dispositivos de armazenamento testados. Novamente, o desempenho superior da solução Cryptomator-SGX em relação à implementação original pode ser atribuído à utilização das funções de criptografia nativas do processador. Nota-se também um desempenho aquém do esperado quando utilizado o aplicativo LUKS com o dispositivo M.2.

Da mesma forma, a Figura 21 apresenta os resultados obtidos na leitura de arquivo único, nos três dispositivos de armazenamento, utilizando o processador Intel Pentium G5500. Novamente, nota-se um desempenho superior da solução Cryptomator-SGX em relação à sua implementação original, com exceção do disco rígido, onde a taxa de transferência obtida ficou um pouco abaixo da observada na implementação não

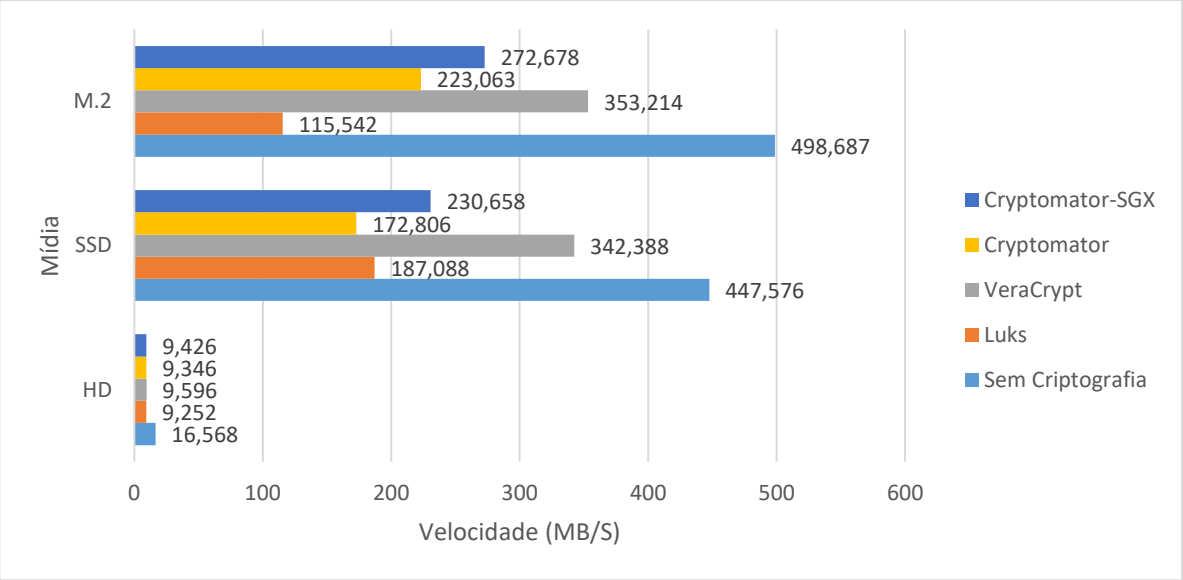

Figura 20: Leitura de arquivo único usando o processador Intel Core i7 9700K. Fonte: Autoria Própria.

modificada do Cryptomator, mas ainda assim muito próxima a essa. Também observa-se que, nesse cenário, a solução proposta obteve taxas de transferência muito próximas ou acima daquelas apresentadas pelo LUKS, exceto no dispositivo de armazenamento M.2.

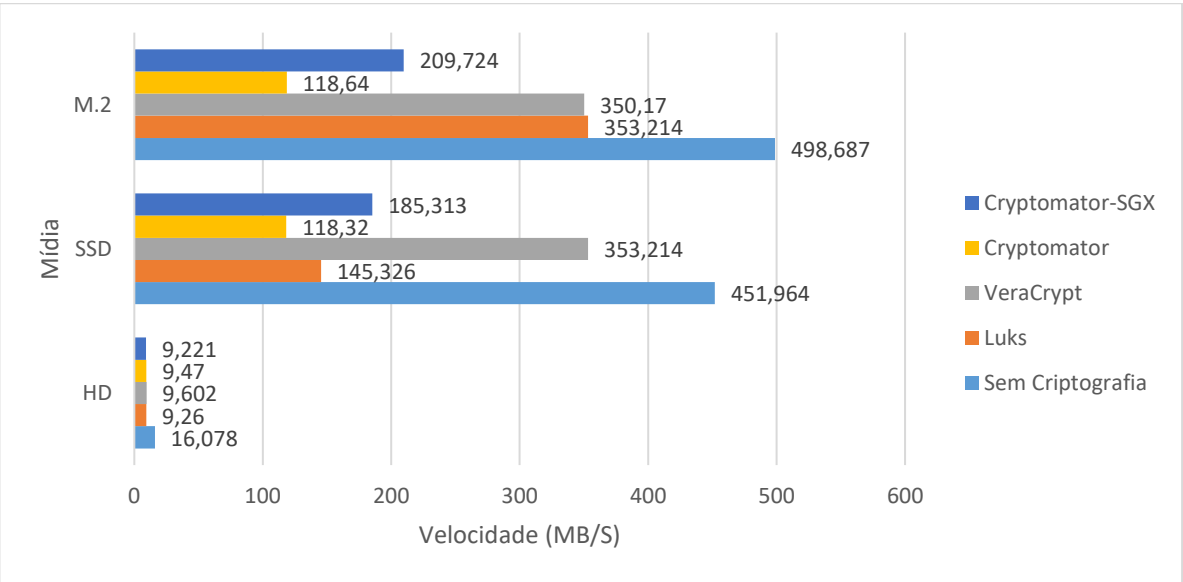

Figura 21: Leitura de arquivo único usando o processador Intel Pentium G5500. Fonte: Autoria Própria.

Na avaliação dos resultados de leitura, quando utilizado um conjunto de diversos

arquivos, o bom desempenho da solução proposta se repete. A Figura 22 apresenta os dados referentes ao uso do processador Intel Core i7 9700K, onde nota-se um desempenho superior do Cryptomator-SGX, se comparado com a implementação original.

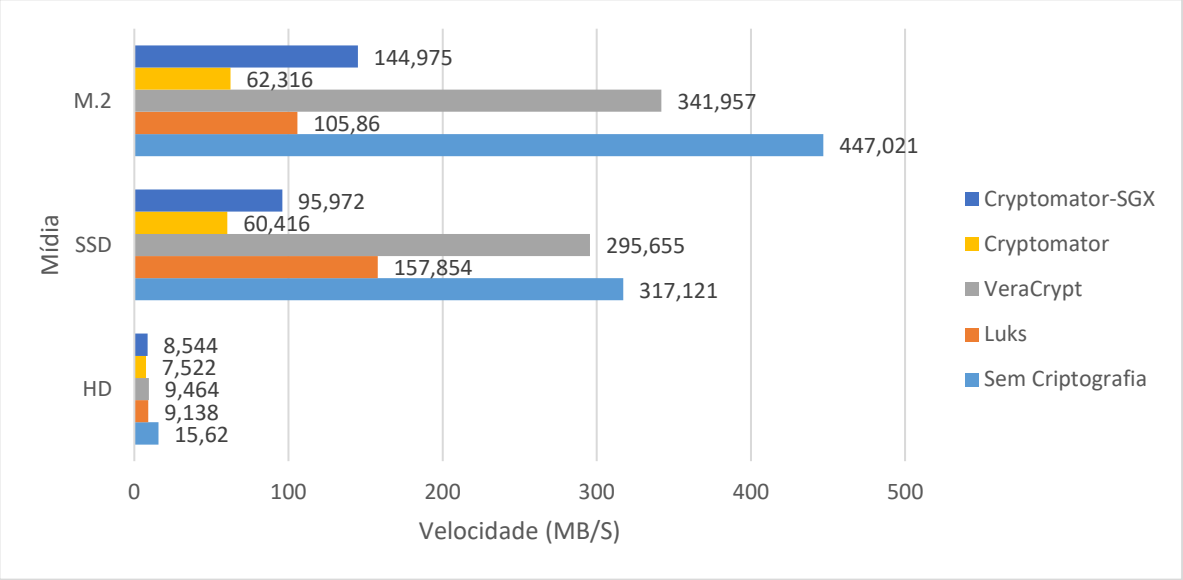

Figura 22: Leitura de múltiplos arquivos usando o processador Intel Core i7 9700K. Fonte: Autoria Própria

Já com a utilização do processador Intel Pentium G5500, observa-se novamente um desempenho superior da solução apresentada nesse trabalho em relação ao Cryptomator, exceto na utilização do dispositivo de armazenamento SSD, onde obteve-se uma queda de cerca de 15% na taxa de transferência, conforme apresentado na Figura 23.

5.2 ANÁLISE DE SEGURANÇA

Outro ponto que é levado em consideração pelos usuários ao escolher uma aplicação para criptografia de dados é a segurança que a mesma proporciona e suas vulnerabilidades. Essa seção tem o objetivo de apresentar o modelo de ameaças utilizado, a análise de segurança da tecnologia Intel SGX e a análise de segurança da solução implementada.

5.2.1 MODELO DE AMEAÇAS

No modelo de ameaças é considerado que o adversário tem como objetivo acessar informações confidenciais armazenadas no disco rígido do computador, e que o mesmo

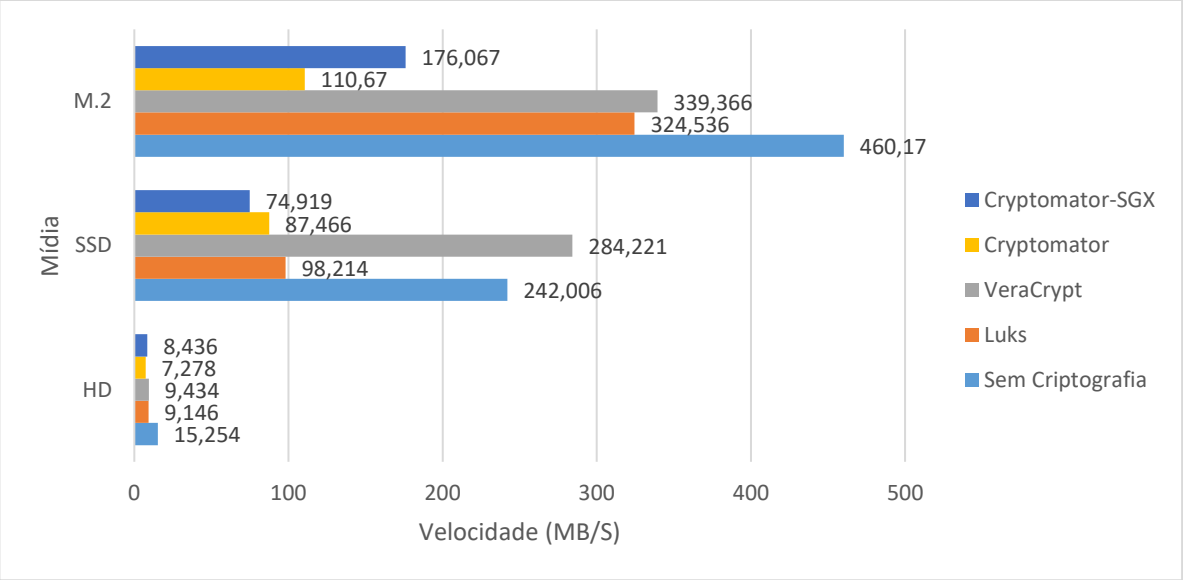

Figura 23: Leitura de múltiplos arquivos usando o processador Intel Pentium G5500. Fonte: Autoria Própria.

tem acesso físico ao computador para tal tarefa, podendo remover o disco e instalá-lo em outra máquina com maior poder computacional, a fim de aplicar técnicas para descobrir a senha utilizada na geração da chave, ou até mesmo a própria chave de criptografia. Além disso, supõe-se que o atacante tenha instalado algum software malicioso na máquina do usuário com o intuito de obter a chave utilizada na criptografia através de um *dumping* de memória, ou obter a senha utilizada para abrir o contêiner ao capturar os dados inseridos no teclado através de um *keylogger*.

É levado em consideração que a tecnologia Intel SGX funciona de forma adequada e de acordo com suas especificações, e que o ambiente de desenvolvimento da solução proposta é confiável.

5.2.2 ANÁLISE DE SEGURANÇA DA TECNOLOGIA INTEL SGX

Para construção de um sistema que seja considerado seguro, a base da computação confiável do mesmo (TCB) deve ser reduzida ao máximo, a fim de reduzir as chances de sucesso em um ataque. Na tecnologia Intel SGX a TCB é composta pela CPU e seus elementos internos, como a lógica do hardware, microcódigo, registradores e memória *cache*, e por alguns elementos de software utilizados na atestação remota, como *quoting enclave*.

Na solução implementada, a responsabilidade pela criptografia dos dados foi passada inteiramente para o enclave. Desse modo, todas as garantias fornecidas pela tecnologia estão em uso durante a criptografia dos dados. Algumas dessas garantias são de que a chave criptográfica nunca sai dos limites do processador, e a memória em que o mesmo está em execução, a PRM, é encriptada, além de possuir mecanismos contra ataques diretos à memória e a proteção contra ataques externos aos enclaves, mesmo que provindos de componentes com alto privilégio de execução, como BIOS ou *hypervisor*, conforme especificado na Seção 2.3.

No entanto, mesmo que a tecnologia possua vários mecanismos para proteção dos dados, há algumas formas de ataques as quais ela é suscetível. A documentação da Intel faz uma relação das limitações da tecnologia, especificando que o SGX não é capaz de evitar ataques de canal lateral, exploração de padrões de acesso aos dados e à *cache*, ou ataques físicos contra a CPU, como a injeção de falhas ou reprogramação das funcionalidades do código de máquina (CONDÉ, 2017).

Um desses modos foi abordado por Ahmad et al. (2018) em seu trabalho, o qual propõe uma forma de reduzir as chances de sucesso em um ataque de canal lateral baseado em *syscall* e *page fault* ao utilizar um protocolo ORAM em conjunto com a tecnologia.

Além das vulnerabilidades descritas pela Intel, a tecnologia pode ser comprometida em casos onde o ambiente de desenvolvimento do enclave foi comprometido, ou se o SDK utilizado não for uma versão mais recente fornecida pela Intel por seu canal oficial.

A segurança do enclave também depende do desenvolvedor, onde o mesmo deve tomar precauções durante a manipulação dos dados dentro do enclave, evitando assim problemas com vazamentos de dados através da manipulação de ponteiros ou chamadas para fora do enclave.

5.2.3 ANÁLISE DE SEGURANÇA DA SOLUÇÃO PROPOSTA

Para validação da segurança da implementação, foi considerado a tecnologia Intel SGX como segura, sendo validado apenas a alteração efetuada no Cryptomator.

A validação consistiu na instalação do Cryptomator em um ambiente e a utilização de dois processadores com suporte a tecnologia Intel SGX, criptografando os dados em um deles e tentando descriptografar no outro, de modo que para o teste ter sucesso, a operação de descriptografia no outro processador deve falhar.

Ao realizar o teste proposto, foi possível descriptografar apenas o nome dos arquivos utilizando a senha do contêiner. Os dados de cada um dos arquivos ficaram inacessíveis, assim como o esperado.

Ao considerar que a tecnologia Intel SGX é segura, é garantido que os dados somente poderão ser descriptografados na plataforma em que foram criptografados, ou que o atacante consiga obter a chave para descriptografia através de um ataque baseado em força bruta. Nesse cenário, é preciso garantir que somente o proprietário dos dados possua acesso aos mesmos na plataforma em que foram criptografados, para isso a senha utilizada para abertura do contêiner foi mantida e é utilizada na criptografia do nome dos arquivos.

Dessa forma, se um atacante conseguir acesso físico à mídia e à plataforma utilizada, ele ainda precisará da senha para acessar os dados. Caso ele não possua acesso à plataforma, ele precisará da senha do usuário e da chave utilizada pelo enclave para a criptografia dos dados.

Em um ataque à memória do computador, a chave utilizada pelo enclave não estará disponível, visto que ela nunca sai dos limites do processador. No entanto, com esse modo de ataque, é possível obter a senha utilizada pelo usuário para abrir o contêiner.

Devido ao fato do software Cryptomator ser desenvolvido em Java e estar separado em 3 módulos principais, um atacante pode fazer uso da biblioteca CryptoLib e implementar sua própria solução, burlando mecanismos de segurança existentes dentro do Cryptomator. Assim, se faz necessário uma segunda camada de segurança, visando autenticar a solicitação para abertura dos dados selados. Tal proposta é elencada nos trabalhos futuros.

5.3 LIMITAÇÕES DA SOLUÇÃO

A solução implementada consiste na criptografia apenas dos dados dos arquivos que são armazenados dentro do contêiner criado através da aplicação. Devido a isso, algumas limitações foram identificadas na aplicação.

Pelo fato da selagem de dados adicionar 560 *bytes* por bloco criptografado, já que inclui dados de autenticação ao criptograma (AES-GCM), o tamanho final do arquivo dentro do contêiner sempre será superior ao tamanho do arquivo original. Desse modo, fazendo com que a solução implementada sempre use mais espaço em disco para os mesmos arquivos em relação a outros modos de armazenamento.

O tamanho final do arquivo, em *bytes*, pode ser obtido através da equação:

$$t = o + (\lceil o \div 32768 \rceil * 560) \quad (1)$$

onde t e o correspondem ao tamanho final e original do arquivo, respectivamente, em *bytes*. Divide-se o pelo tamanho máximo de um bloco, definido como 32768 *bytes* (32 KB), aplicando a função teto para obter um valor inteiro, que corresponderá ao número de blocos utilizados. Após, multiplica-se por 560, número de *bytes* adicionados pela selagem de dados, que, somados ao número de *bytes* do arquivo original, resultará no número de *bytes* do arquivo final.

Para realizar o processo de criptografia de cada arquivo, o Cryptomator armazena um arquivo com nome criptografado dentro do contêiner, o qual corresponde ao arquivo de origem. Nesse cenário, não foi possível utilizar a solução implementada para criptografar o nome dos arquivos dentro do contêiner devido a limitações no sistema operacional em relação à quantidade máxima de caracteres que podem existir no nome dos mesmos, sendo definido o máximo de 255 caracteres no nome do arquivo na maior parte dos sistemas operacionais, e ao tamanho dos dados após serem criptografados pelo enclave SGX, conforme descrito anteriormente, o qual sempre gera 1120 caracteres a mais ao converter os dados criptografados para hexadecimal. Para contornar essa limitação, foi mantida a criptografia usando o modo AES já existente, o qual depende do usuário informar uma senha para realizar o processo.

Essa limitação acabou por adicionar uma camada a mais de proteção na solução ao impedir a abertura dos dados do contêiner na plataforma em que foram criptografados sem que a senha do usuário seja informada.

A limitação que foi abordada na Seção 5.1 em relação ao desempenho da aplicação, é devido ao fato da mesma utilizar apenas um núcleo para processamento. Devido a isso, somente um enclave é utilizado no processo de criptografia e descriptografia dos dados, diminuindo a performance do sistema. Outro ponto analisado é a necessidade constante de conversão dos dados entre a linguagem Java e a linguagem C++ através da interface JNI, gerando um custo alto de processamento. Alterar a arquitetura da solução para utilizar melhor os recursos disponíveis dentro do sistema será objeto de trabalhos futuros, no entanto a necessidade de conversão dos dados entre as linguagens não poderá ser removida devido à necessidade de utilização de código nativo para execução do enclave SGX.

A selagem dos dados utilizando a *sealing key* pode gerar transtornos e imprevistos,

como a necessidade de acesso remoto ou a queima do processador, visto que os dados só estarão acessíveis se a aplicação for executada através do processador que foi utilizado para selagem dos mesmos. Uma forma de contornar essa limitação é a criação de um mecanismo, dentro da aplicação, para transferência segura dos dados entre dois computadores em que a mesma esteja em execução, permitindo acesso remoto e *backup* dos dados sempre que necessário. Tal mecanismo será objeto de trabalhos futuros.

5.4 CONSIDERAÇÕES FINAIS

O presente capítulo abordou a sistemática para realização dos testes de performance da solução, comparando-a com outros modos de armazenamento de dados e em diversos cenários, a sistemática utilizada para realização dos testes de segurança, as ameaças a qual a solução está vulnerável e as limitações da mesma.

A partir dos dados de desempenho apresentados nas Seções 5.1.1 e 5.1.2, identificamos que, na maior parte dos casos, a solução implementada obteve desempenho superior em relação ao aplicativo original, tanto para leitura, quanto para gravação de dados. Esse aumento de desempenho pode ser atribuído ao fato de que o processo de criptografia passou a ser realizado diretamente pelo processador, enquanto no aplicativo original era executado através da máquina virtual do Java. Ao compararmos a solução com os modos multiprocessados, identificamos que a solução implementada e o aplicativo original possuem desempenho muito inferior em mídias de alto desempenho, indicando uma limitação por parte da implementação do projeto.

Considerando as garantias de segurança fornecidas pela tecnologia Intel SGX, as quais são descritas na Seção 2.3, o modo como o Cryptomator trabalha em sua implementação original, e a forma utilizada para combinar ambos, a solução desenvolvida oferece um nível extra de segurança ao selar os dados utilizando a *sealing key* em conjunto com a senha do usuário para a abertura dos contêineres. Desse modo, em um ataque aos dados criptografados, o atacante precisará descobrir, além da senha do usuário, a *sealing key*, ou conseguir acesso físico ao processador utilizado para a selagem.

Para evitar que em um ataque à memória a senha do usuário possa ser obtida, é necessário alterar o software para que a mesma seja armazenada dentro do enclave e não saia dos seus limites. Tal alteração será objeto de trabalhos futuros.

6 CONCLUSÃO

A partir do cenário atual apresentado no Capítulo 1, o qual descreve o aumento na quantidade de informações armazenadas e também o aumento no número de ameaças que surgem diariamente, fazendo com que empresas invistam cada vez mais em pesquisas para desenvolvimento de tecnologias de segurança baseadas em hardware, o contexto histórico apresentado na Seção 2.1 e o estado da arte apesentado no Capítulo 3, os quais nos mostram que a busca pela segurança no armazenamento de informações é um assunto que a anos é pesquisado e está em constante evolução, o presente trabalho buscou contribuir ao aplicar a tecnologia lançada recentemente pela Intel a um software de criptografia de disco.

As tecnologias que precedem o SGX, descritas na Seção 2.2, são exemplos dos avanços obtidos através de estudos realizados por diversas empresas. Nesse contexto, a Intel desenvolveu a tecnologia SGX, que permite encapsular partes da aplicação para que o processamento dos dados ocorra de maneira segura, utilizando uma região de memória criptografada e controlada pelo próprio processador, provendo também mecanismos para realizar o armazenamento dos dados e comunicação entre sistemas de forma segura, através de um ambiente criptografado e de critérios de segurança que são aplicados e controlados exclusivamente pelo processador.

Os detalhes referentes ao funcionamento da tecnologia SGX, abordados a partir da Seção 2.3, são de suma importância para o desenvolvimento deste trabalho, o qual teve como objetivo implementar o uso da tecnologia SGX no software *Cryptomator*, de modo que o mesmo faça uso das garantias de segurança providas pelo recurso de selagem de dados, o qual é provido pelo SGX.

Ao final do desenvolvimento, a aplicação obteve um nível adicional de segurança para os dados de pessoas e empresas que a utilizem e possuam um dispositivo compatível com a tecnologia SGX.

6.1 CONTRIBUIÇÕES E RESULTADOS OBTIDOS

Os trabalhos apresentados no Capítulo 3 são exemplos de projetos que também fizeram uso da tecnologia Intel SGX a fim de obter um nível de segurança adicional no processo de criptografia das informações armazenadas em mídia secundária. Dentre os 3 trabalhos apresentados, apenas a selagem de dados no FUSE, apresentada na Seção 3.2, possui maior semelhança entre os objetivos com a proposta apresentada por esse trabalho, com a diferença de que a solução apresentada por eles é exclusiva para ambientes em Linux.

Ao comparar a solução implementada com os demais trabalhos, pode-se identificar diferentes preocupações em relação à segurança, onde a proposta desse trabalho foi de adicionar as garantias de segurança da tecnologia Intel SGX em uma solução que possa ser utilizada em diferentes sistemas operacionais, enquanto os demais trabalhos focaram em outros pontos, como fornecer ao *kernel* do Linux um ambiente seguro para criptografia de dados, selagem de dados como um módulo do *kernel*, (apresentado na Seção 3.1), ou reduzir as chances de sucesso em um ataque de canal lateral, selagem de dados resistente a ataques de canal lateral, (apresentado na Seção 3.3).

O presente trabalho implementou o uso da tecnologia Intel SGX para a criptografia de dados em um software *open-source*. As principais contribuições obtidas durante o desenvolvimento do mesmo são listadas a seguir:

- Um estudo completo sobre a tecnologia e suas funcionalidades.
- Compilado de testes de performance, mostrando as vantagens e desvantagens com o uso da tecnologia em relação a outros modos de armazenamento em cenários distintos.
- Análise da segurança obtida através da implementação.
- Biblioteca para criptografia de dados em Java que permite o uso do recurso de selagem de dados fornecido pela tecnologia.
- Código fonte da solução, o qual serve de objeto para estudos e implementações futuras.

A solução implementada está disponível no GitHub, através do endereço `https://github.com/utfpr-gprsc/cryptomator-sgx`.

6.2 TRABALHOS FUTUROS

A tecnologia Intel SGX pode ser utilizada para diversas finalidades. No trabalho apresentado, ela foi utilizada exclusivamente para a selagem de dados, no entanto foram identificados outros possíveis usos para a mesma, e alguns pontos de melhoria da aplicação durante o desenvolvimento da mesma.

6.2.1 MECANISMO PARA TRANSFERÊNCIA DE DADOS ENTRE DUAS MÁQUINAS ATRAVÉS DE CANAIS SEGUROS

O uso da selagem de dados fornecida pela tecnologia Intel SGX trouxe benefícios para a aplicação, no entanto, devido à constante necessidade da realização de backups e acesso remoto aos dados, a solução atual pode ser alterada para que utilize o recurso de atestação remota e permita transferência de dados dos contêineres entre duas máquinas, que estejam executando a aplicação, através de canais seguros.

6.2.2 CRIPTOGRAFIA COMPLETA DO ARQUIVO DE CONFIGURAÇÃO DO CONTÊINER

Devido à necessidade de manter o modo de criptografia AES existente no Cryptomator, foi optado por não alterar a forma que o arquivo de configuração dos contêineres é criptografado.

Assim como foi possível a selagem dos dados dos arquivos do usuário através da tecnologia Intel SGX, é possível selar os dados da configuração também. No entanto, para manter a compatibilidade, é necessário adicionar outro identificador em aberto para que a aplicação possa escolher entre os modos disponíveis automaticamente (v1 ou sgx). Tal alteração demanda ajustes na estrutura da biblioteca CryptoFS, podendo remover a compatibilidade com o projeto principal.

6.2.3 UTILIZAÇÃO DE ENCLAVES PARA MANIPULAÇÃO DE CHAVES

Em muitas situações, utilizar um modo de criptografia em que os dados só estarão acessíveis na plataforma em que foram criptografados pode não ser vantagem para determinados usuários. Desse modo, a tecnologia Intel SGX pode ser utilizada para a manipulação de chaves para criptografia ao fazer com que um algoritmo criptográfico existente seja executado dentro de um enclave.

Para tornar isso possível, é necessário utilizar o mesmo conceito aplicado por Richter, Götzfried e Müller (2016) em seu trabalho, o qual foi apresentado na Seção 3.1, onde o modo AES-NI é executado dentro de um enclave, no entanto, fazer com que a chave criptográfica seja gerada apenas a partir da senha do usuário, e nunca saia dos limites do enclave. Nesse cenário, as garantias de segurança providas pela tecnologia Intel SGX evitam que em um ataque a memória do computador seja possível obter a senha do usuário ou a chave utilizada pela criptografia, reduzindo as chances de sucesso do ataque, além de permitir a migração dos dados para outra plataforma de forma transparente.

6.2.4 MELHORIA NO USO DOS RECURSOS DO AMBIENTE

De acordo com os dados que foram expostos na Seção 5.1, apesar da solução implementada obter performance melhor que o aplicativo original, o uso de apenas um núcleo do processador e apenas um enclave limitam o desempenho máximo que ela pode atingir.

Tal implementação demanda alteração na estrutura principal do aplicativo, a qual trata as requisições vindas do SO, sendo necessário adicionar o uso de filas e processamento em paralelo, permitindo assim, que enquanto um bloco é lido ou gravado na mídia de armazenamento, outro bloco é processado pelo enclave, evitando assim que um componente necessite a conclusão do processo em execução pelo outro para que ele possa continuar.

REFERÊNCIAS

AHMAD, A.; KIM, K.; SARFARAZ, M. I.; LEE, B. Obliviate: A Data Oblivious File System for Intel SGX. In: **Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS**. [S.l.: s.n.], 2018. p. 18–21.

ANATI, I.; GUERON, S.; JOHNSON, S.; SCARLATA, V. Innovative Technology for CPU Based Attestation and Sealing. In: **Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy**. New York, NY, USA: ACM, 2013. (HASP 2013, v. 13). ISBN 978-1-4503-2118-1.

ANDERSON, R.; BOND, M.; CLULOW, J.; SKOROBOGATOV, S. Cryptographic Processors - A Survey. **Proceedings of the IEEE**, IEEE, v. 94, n. 2, p. 357–369, 2006. ISSN 0018-9219.

ARM. **ARM Security Technology Building a Secure System using TrustZone Technology (white paper)**. 2009.

ARTHUR, W.; CHALLENER, D. **A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security**. 1st. ed. Berkely, CA, USA: Apress, 2015. ISBN 978-1-4302-6583-2.

AUMASSON, J.; MERINO, L. SGX Secure Enclaves in Practice: Security and Crypto Review. **Black Hat**, 2016.

BURIHABWA, D.; FELBER, P.; MERCIER, H.; SCHIAVONI, V. SGX-FS: Hardening a File System in User-Space with Intel SGX. In: **Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2018. p. 67–72. ISSN 2330-2186.

CONDÉ, R. C. R. **Proteção de Dados de Autenticação em um Sistema Operacional Usando Enclaves SGX**. Dissertação (Mestrado) — Universidade Federal do Paraná, Curitiba, PR, BR, 2017.

COSTAN, V.; DEVADAS, S. **Intel SGX Explained**. 2016. Cryptology ePrint Archive, Report 2016/086.

CRYPTOMATOR. 2019. Disponível em: <https://cryptomator.org/>. Acesso em: 23/01/2019.

GREENE, J. **Intel Trusted Execution Technology (white paper)**. 2012.

HOEKSTRA, M.; LAL, R.; PAPPACHAN, P.; PHEGADE, V.; CUVILLO, J. D. Using Innovative Instructions to Create Trustworthy Software Solutions. In: **Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy**. New York, NY, USA: ACM, 2013. (HASP 2013). ISBN 978-1-4503-2118-1.

INTEL. **Intel Software Guard Extensions Programming Reference**. 2014.

INTEL. **Intel Software Guard Extensions Developer Guide**. [S.l.]: Intel Corporation, 2016.

INTEL. **Intel Software Guard Extensions SDK for Linux OS Developer Reference**. [S.l.], 2016.

INTEL. **Intel Data Protection Technology with AES-NI and Secure Key**. Intel Corporation, 2019. Disponível em: <https://www.intel.com.br/content/www/br/pt/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html>.

KERRISK, M. **Linux Programmer's Manual**. [S.l.], 2019. Disponível em: <http://man7.org/index.html>.

LADEIRA, L.; NASCIMENTO, E.; VENTURA, P. F.; DAHAB, R.; ARANHA, D.; HERNáNDEZ, J. C. L. Canais laterais em criptografia simétrica e de curvas elpticas: Ataques e contramedidas. p. 82–141, 11 2016.

LESJAK, C.; HEIN, D.; WINTER, J. Hardware-security technologies for industrial IoT: TrustZone and security controller. In: **Proceedings of the IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society**. [S.l.]: IEEE, 2015. p. 002589–002595. ISBN 978-1-4799-1762-4.

MCKEEN, F.; ALEXANDROVICH, I.; ANATI, I.; CASPI, D.; JOHNSON, S.; LESLIE-HURD, R.; ROZAS, C. Intel Software Guard Extensions (Intel SGX) Support for Dynamic Memory Management Inside an Enclave. In: **Proceedings of the Hardware and Architectural Support for Security and Privacy 2016**. New York, NY, USA: ACM, 2016. (HASP 2016). ISBN 978-1-4503-4769-3.

MCKEEN, F.; ALEXANDROVICH, I.; BERENZON, A.; ROZAS, C. V.; SHAFI, H.; SHANBHOGUE, V.; SAVAGAONKAR, U. R. 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 2013). ISBN 978-1-4503-2118-1.

MEIJER, C.; GASTEL, B. van. **Self-encrypting deception: weaknesses in the encryption of solid state drives (SSDs)**. Radboud University, 2018. Disponível em: <https://www.ru.nl/english/news-agenda/news/vm/icis/cyber-security/2018/radboud-university-researchers-discover-security/>. Acesso em: 2019-04-05.

MÜLLER, T.; FREILING, F. C. A Systematic Assessment of the Security of Full Disk Encryption. **IEEE Transactions on Dependable and Secure Computing**, v. 12, n. 5, p. 491–503, Setembro 2015. ISSN 1545-5971.

PINKAS, B.; REINMAN, T. Oblivious ram revisited. In: RABIN, T. (Ed.). **Advances in Cryptology – CRYPTO 2010**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 502–519. ISBN 978-3-642-14623-7.

RICHTER, L.; GÖTZFRIED, J.; MÜLLER, T. Isolating Operating System Components with Intel SGX. In: **Proceedings of the 1st Workshop on System Software for Trusted Execution**. New York, NY, USA: ACM, 2016. p. 8:1–8:6. ISBN 978-1-4503-4670-2.

TCG. **Trusted Platform Module (TPM) Summary**. 2008.

TRUECRYPT. 2014. Disponível em: <http://truecrypt.sourceforge.net/>. Acesso em: 10/06/2018.

VERACRYPT. 2018. Disponível em: <http://www.veracrypt.fr/en/Home.html>. Acesso em: 20/04/2018.

WILL, N. C.; CONDÉ, R. C. R.; MAZIERO, C. A. Mecanismos de Segurança Baseados em Hardware: Uma Introdução à Arquitetura Intel SGX. In: NUNES, R. C.; CANEDO, E. D.; JÚNIOR, R. T. S. (Ed.). **Minicursos do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**. Brasilia, DF, BR: Sociedade Brasileira de Computação, 2017. cap. 2, p. 49–98. ISBN 978-8-5766-9410-6.