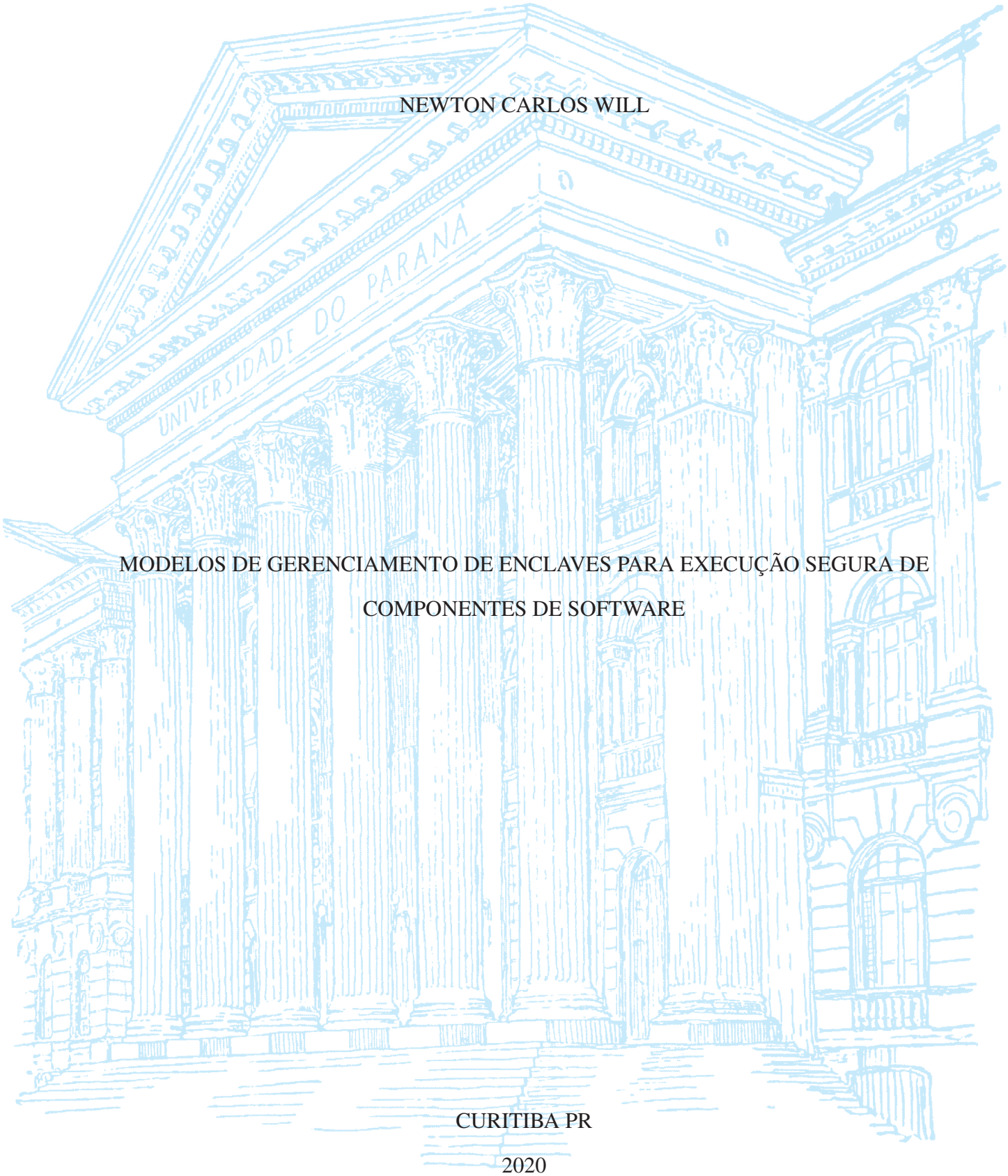
UNIVERSIDADE FEDERAL DO PARANÁ

NEWTON CARLOS WILL



MODELOS DE GERENCIAMENTO DE ENCLAVES PARA EXECUÇÃO SEGURA DE COMPONENTES DE SOFTWARE

CURITIBA PR

2020

NEWTON CARLOS WILL

MODELOS DE GERENCIAMENTO DE ENCLAVES PARA EXECUÇÃO SEGURA DE COMPONENTES DE SOFTWARE

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Informática, no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

CURITIBA PR

2020

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

W689m Will, Newton Carlos
Modelos de gerenciamento de enclaves para execução segura de componentes de software [recurso eletrônico] / Newton Carlos Will. – Curitiba, 2020.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2020.

Orientador: Carlos Alberto Maziero

1. Programação (Computadores). 2. Arquitetura de software. 3. Intel SGX. I. Universidade Federal do Paraná. II. Maziero, Carlos Alberto. III. Título.

CDD: 005.1

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **NEWTON CARLOS WILL** intitulada: **MODELOS DE GERENCIAMENTO DE ENCLAVES PARA EXECUÇÃO SEGURA DE COMPONENTES DE SOFTWARE**, sob orientação do Prof. Dr. CARLOS ALBERTO MAZIERO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 04 de Dezembro de 2020.

Assinatura Eletrônica
04/12/2020 10:11:55.0
CARLOS ALBERTO MAZIERO
Presidente da Banca Examinadora

Assinatura Eletrônica
04/12/2020 10:50:12.0
KEIKO VERONICA ONO FONSECA
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

Assinatura Eletrônica
04/12/2020 10:45:45.0
MARCOS DIDONET DEL FABRO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
04/12/2020 11:29:39.0
ANDREY ELISIO MONTEIRO BRITO
Avaliador Externo (UNIVERSIDADE FEDERAL DE CAMPINA GRANDE)

Aos meus pais.

AGRADECIMENTOS

Primeiramente agradeço ao meu pai, minha mãe e meu irmão, que me deram total apoio em todos os passos desta caminhada, fosse nos momentos fáceis ou difíceis, sempre estiveram ao meu lado para me incentivar a seguir em frente.

Ao meu orientador, Prof. Dr. Carlos Alberto Maziero, que sempre esteve à disposição nos diversos momentos em que precisei ao longo destes anos. Teve o desafio de receber um aluno vindo da Engenharia Elétrica, e soube me guiar no caminho da ciência em um campo que era totalmente novo para mim.

Aos amigos e colegas do Laboratório de Redes, Sistemas Distribuídos e Segurança (LaRSiS), que sempre estiveram abertos para a discussão, as quais foram muito proveitosas e auxiliaram na construção deste trabalho. Principalmente a Amanda Benites Viescinski e Tiago Heinrich pelas parcerias nas publicações, mas também a Marcus Botacin, Fabrício Ceschin, Vinícius Fülber Garcia, Alexandre Huff, José Wilson Vieira Flauzino e Rafael Gomes de Castro.

Aos colegas da Coordenação de Curso de Engenharia de Software, da Universidade Tecnológica Federal do Paraná, que auxiliaram neste caminho de diferentes formas. Principalmente os amigos André Roberto Ortoncelli, Evandro Kuszera, Franciele Beal e Marlon Marcon, e aos demais que cursaram as disciplinas juntamente conosco.

E por fim, mas provavelmente mais importante, a minha eterna companheira Aliciane de Almeida Roque, que esteve comigo ao longo de praticamente toda esta jornada, e com quem eu sempre pude contar, tanto nos momentos bons quanto nos momentos ruins. Te amo.

"It's not who I am underneath, but what I do that defines me."
(Batman, 2005)

"Não é quem eu sou por dentro, mas o que eu faço que me define."
(Batman, 2005)

RESUMO

A confidencialidade dos dados está se mostrando cada vez mais importante para os usuários de computadores, seja em um ambiente corporativo ou até mesmo em um ambiente doméstico. Atualmente, não somente dados sensíveis às empresas estão trafegando pela rede ou sendo manipulados pelos mais diversos programas de computador, mas também tem-se um intenso uso de aplicações para transações bancárias e outras aplicações de uso corriqueiro que manipulam dados sensíveis dos usuários, os quais devem ter sua confidencialidade e integridade garantidas. Nesse sentido, tem-se variadas soluções sendo propostas para manter a confidencialidade e integridade dos dados, dentre elas a arquitetura Intel SGX (*Software Guard Extensions*), a qual possui mecanismos para que as aplicações e os dados sejam encapsulados em uma área protegida da memória com acesso restrito, impossibilitando o acesso nessa região de memória a outras aplicações ou ao próprio sistema operacional. A utilização de tais mecanismos para prover a confidencialidade e integridade dos dados sensíveis da aplicação acarreta em um impacto de desempenho durante a sua execução, devido às restrições e verificações impostas pela arquitetura Intel SGX. O presente trabalho busca analisar os modelos de programação que são aplicados em soluções que utilizam a arquitetura Intel SGX e apresentar alternativas que buscam um uso mais eficiente dos recursos providos por tal arquitetura e também a redução do impacto de desempenho decorrente de sua utilização. Assim, são apresentados dois modelos de gerenciamento: (i) compartilhamento de enclaves; e (ii) *pool* de enclaves. Para a aplicação de tais modelos, é proposta uma arquitetura de um provedor de enclaves, que oferece um desacoplamento entre o enclave e a aplicação que o utiliza, permitindo aplicar os modelos de gerenciamento propostos e oferecer os recursos providos pelos enclaves às aplicações na forma de serviços. Um protótipo é construído para avaliar a arquitetura e modelos propostos, com os testes de desempenho demonstrando consideráveis reduções no impacto para requisição de enclaves, enquanto garante boa resposta para atender múltiplas requisições simultâneas. Assim, conclui-se que a utilização de modelos arquiteturais de software podem trazer benefícios no gerenciamento de recursos e ganho de desempenho na execução de aplicações seguras.

Palavras-chave: Intel SGX, modelos de programação, arquitetura de software, padrões de projeto, desempenho, otimização de recursos.

ABSTRACT

Data confidentiality is becoming increasingly important to computer users, whether in a corporate environment or even in a home environment. Not only are business-sensitive data currently being trafficked across the network or being handled by a variety of software, but there is also an intense use of applications for banking transactions and other commonly used applications that manipulate sensitive user data, which must have their confidentiality and integrity guaranteed. In this sense, there are several solutions being proposed to maintain the confidentiality and integrity of the data, among them the Intel SGX (Software Guard Extensions) architecture, which has mechanisms to encapsulate applications and data in a protected area of memory having restricted access, making it impossible to access this region of memory to other applications or to the operating system. The use of such mechanisms to provide the confidentiality and integrity of sensitive data results in a performance impact during the application execution, due to the restrictions and verifications imposed by the Intel SGX architecture. The present work aims to analyze the programming models that are applied in solutions that use the Intel SGX architecture and present alternatives that seek a more efficient use of the resources provided by this architecture and also the reduction of the performance impact due to its use. Thus, two management models are presented: (i) enclave sharing; and (ii) enclave pool. In order to apply such models, an architecture of an enclave provider is proposed, which offers a decoupling between the enclave and the application that uses it, allowing to apply the proposed management models and offering the resources provided by the enclaves to the applications in "as a service" format. A prototype is built to evaluate the proposed architecture and models, with the performance tests demonstrating considerable reductions in the impact for enclave requests, while guaranteeing good response to attend simultaneous requests. Thus, it is concluded that the use of architectural software models can bring benefits in resource management and performance gains in the execution of secure applications.

Keywords: Intel SGX, programming models, software architecture, design patterns, performance, resource optimization.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

ACM	Authenticated Code Module
AES	Advanced Encryption Standard
AEX	Asynchronous Exit
AMI	Asynchronous Method Invocation
API	Application Programming Interface
ATM	Caixa eletrônico
BGP	Border Gateway Protocol
BIOS	Basic Input/Output System
CAS	Configuration and Attestation Service
CDN	Content Distribution Network
CPU	Central Processing Unit
CRTM	Core Root of Trust Measurement
CVE	Common Vulnerabilities and Exposures
DE	Decryption Enclave
DMA	Direct Memory Access
DRAM	Dynamic RAM
DRM	Digital Rights Management
DRTM	Dynamic Root of Trust Measurement
EaaS	Enclave as a Service
ECDH	Elliptic Curve Diffie Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EDL	Enclave Definition Language
EID	Enclave Identifier
EK	Endorsement Key
EPC	Enclave Page Cache
EPCM	Enclave Page Cache Map
EPID	Enhanced Privacy ID
ERM	Enterprise Rights Management
FaaS	Function as a Service
FPGA	Field Programmable Gate Array
FE	Function Enclave
GPP	General Propose Processor
GPDR	General Data Protection Regulation
HVB	Hardware Validated Boot
IaaS	Infrastructure as a Service

IoT	Internet of Things
ISV	Independent Software Vendor
IXP	Internet eXchange Point
KME	Key Management Enclave
LCP	Launch Control Policy
LPC	Low Pin Count
LE	Launch Enclave
MAC	Message Authentication Code
MLE	Measured Launch Environment
MEE	Memory Encryption Engine
NFV	Network Function Virtualization
OS	Operating System
OTP	One-Time Password
P2P	Peer-to-Peer
PAM	Pluggable Authentication Module
PAVP	Protected Audion Video Path
PC	Personal Computer
PCR	Platform Configuration Register
PID	Process Identifier
PIN	Personal Identification Number
PRM	Processor Reserved Memory
PSP	Platform Security Processor
QE	Quoting Enclave
RAaS	Remote Attestation as a Service
RAM	Random Access Memory
RDTSCP	Read Time-Stamp Counter and Processor ID
ROM	Read Only Memory
RPC	Remote Procedure Call
RSA	Rivest-Shamir-Adleman
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SaaS	Software as a Service
SDK	Software Development Kit
SECaaS	Security as a Service
SECS	SGX Enclave Control Structure
SEV	Secure Encrypted Virtualization
SGX	Software Guard Extensions
S-NFV	Secure Network Function Virtualization
SoC	System-on-Chip

SRAM	Static RAM
SRK	Storage Root Key
SRTM	Static Root of Trust Measurement
SSL	Secure Socket Layer
SVM	Secure Virtual Machine
SVN	Security Version Number
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Plataform Module
TSX	Transactional Synchronization Extensions
TXT	Trusted Execution Technology
ULA	Unidade Lógica e Aritmética
XaaS	Everything as a Service
XML	Extensible Markup Language

1 INTRODUÇÃO

Neste capítulo é apresentada a contextualização do presente trabalho, buscando situá-lo no cenário atual, tanto em questões tecnológicas quanto conceituais. A Seção 1.2 descreve a motivação desta proposta de tese, com a Seção 1.3 apresentando a hipótese de pesquisa e, brevemente, a proposta deste trabalho, que aborda uma possível solução para reduzir alguns dos problemas identificados. Por fim, é detalhada a estrutura deste documento, trazendo uma breve descrição dos temas que serão abordados nos próximos capítulos.

1.1 CONTEXTUALIZAÇÃO

Atualmente, a tecnologia está presente na vida de muitas pessoas, as quais entregam seus dados aos seus dispositivos, como computadores, *smartphones*, *tablets*, e até mesmo a serviços de armazenamento *online*. Os dados e informações presentes nesses dispositivos, ou meios de armazenamento, são dos mais variados tipos, desde fotos de família, senhas para os mais diversos serviços, agendas pessoais, dados médicos e bancários, até informações de trabalho, as quais podem ser extremamente cruciais para os usuários. Nesse sentido, se faz necessário a especificação de mecanismos e práticas para garantir a confidencialidade e integridade de dados tão sensíveis.

Confidencialidade e integridade são dois conceitos distintos. Conforme Hou *et al.* (2004), a confidencialidade restringe a visualização dos dados apenas a quem detém os requisitos para visualizá-los, dificultando o acesso a esses dados a quem não tem essa permissão. A técnica mais comum para garantir a confidencialidade dos dados é a criptografia, onde os dados são cifrados utilizando uma chave e somente o detentor da mesma chave, para criptografia simétrica, ou de uma chave complementar, para criptografia assimétrica, poderá decifrar os dados (Katz e Lindell, 2014). Já a integridade é o fornecimento de proteção contra a corrupção ou modificação não autorizada dos dados, garantindo que os dados que estão sendo lidos são exatamente os mesmos que foram previamente gravados ou enviados. A forma mais comum de garantir a integridade dos dados é a utilização do MAC (*Message Authentication Code*). A utilização do MAC permite que qualquer alteração na mensagem seja detectada, visto que diferentes conjuntos de dados irão gerar diferentes *hashes* (Hou *et al.*, 2004).

As garantias de confidencialidade e integridade devem ser levadas em consideração também pelas empresas desenvolvedoras e fornecedoras de aplicações e/ou serviços que armazenam e manipulam os dados dos usuários. Conforme descrito por Hoekstra *et al.* (2013), as práticas para o desenvolvimento de aplicações seguras são padronizadas em diversas empresas, tendo o teste de segurança como parte chave na validação do software, mas ainda assim a confidencialidade e integridade dos dados dependerá da correta utilização de outros softwares que estarão sendo utilizados no mesmo ambiente, alguns dos quais podem ter privilégios para inspecionar a memória ou outros meios de armazenamento no dispositivo. Além disso, cabe ao usuário final o bom senso no uso das aplicações, o qual deverá seguir as boas práticas e adotar uma política cautelosa para a execução de diversas aplicações, seja no ambiente de trabalho ou para uso pessoal.

Adicionalmente, em maio de 2018 entrou em vigor, nos países membros da União Europeia, a GPDR (*General Data Protection Regulation*, ou Regulamento Geral de Proteção de Dados) que determina como as empresas, produtos e serviços devem tratar os dados privados dos usuários residentes nesses países, colocando os usuários como soberanos sobre seus dados e

prevendo uma série de obrigações para as empresas, e até mesmo sanções para as organizações que não cumprirem com tal legislação (Tikkinen-Piri *et al.*, 2018). De forma semelhante, em agosto de 2018 foi sancionada no Brasil a Lei Geral de Proteção de Dados (Lei Nº 13.709/2018) que altera o Marco Civil da Internet (Lei Nº 12.965/2014) e "dispõe sobre o tratamento de dados pessoais, inclusive nos meios digitais, por pessoa natural ou por pessoa jurídica de direito público ou privado, com o objetivo de proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural", seguindo os mesmo moldes da legislação europeia (Brasil, 2018).

Nesse cenário, a manutenção da confidencialidade e integridade dos dados dos usuários é algo crucial para as organizações e provedores de serviços e, em vista disso, diversas empresas estão desenvolvendo métodos para manter a confidencialidade dos usuários de seus aplicativos e serviços. Como exemplo, pode-se citar a Apple, a qual introduziu mecanismos de segurança em seu sistema operacional móvel, o iOS 9, que combina recursos de hardware, software e serviços para prover a confidencialidade de seus usuários, não só em operações locais, em seus dispositivos, mas também no tráfego de dados entre o dispositivo e serviços de Internet. Tais mecanismos já são habilitados por padrão nos dispositivos, e não há qualquer configuração que permita que o usuário desabilite-os, seja de forma acidental ou intencional (Apple, 2015). De forma similar a essa, dispositivos com sistema operacional Android também oferecem a possibilidade de criptografar os dados armazenados em memória secundária através de uma funcionalidade do sistema, que pode ser habilitada pelo usuário.

Além disso, diversos mecanismos foram sendo propostos e desenvolvidos ao longo dos anos com o intuito de oferecer suporte aos sistemas operacionais e demais aplicações que visam garantir uma execução e manipulação segura de dados sensíveis de usuários. Uma das mais recentes tecnologias nesse sentido é a Intel *Software Guard Extensions* (Intel SGX) (Intel, 2016a), que possibilita que a aplicação seja dividida em dois componentes: confiável e não-confiável. O Intel SGX traz uma série de novas instruções que visam garantir a confidencialidade e integridade dos dados que estão sendo manipulados dentro do enclave, que é o componente confiável da aplicação (McKeen *et al.*, 2013). Tal tecnologia está disponível no mercado desde 2015, com o lançamento dos processadores Intel Core de 6ª geração (*Skylake*), e está sendo utilizada em diversas áreas de aplicação.

1.2 MOTIVAÇÃO

A arquitetura Intel *Software Guard Extensions* (Intel SGX) possibilita que a aplicação seja dividida em dois componentes: confiável e não confiável. A parte confiável da aplicação, chamada de *enclave*, é executada em um ambiente seguro, com todas as instruções e dados contidos dentro do enclave sendo mantidos em uma região cifrada da memória, chamada de PRM (*Processor Reserved Memory*). A arquitetura Intel SGX possibilita também a gravação de informações cifradas em memória secundária através de um processo chamado de *selagem*. As chaves de criptografia de ambos os processos são mantidas pela CPU e nunca saem de seus limites (McKeen *et al.*, 2013).

Essas garantias de segurança acrescentam um *overhead* de desempenho considerável, tanto para a criação do enclave quanto para a execução de funções dentro do enclave. Além disso, a região de memória utilizada pela PRM é limitada a 128 MB, o que limita o tamanho e o número de enclaves que podem residir em memória ao mesmo tempo (Shaon *et al.*, 2017; Mofrad *et al.*, 2017; Fuhry *et al.*, 2017). Uma questão levantada é a possibilidade de reduzir o impacto de desempenho inerente à utilização de enclaves para a manipulação de dados sensíveis, principalmente no momento de sua instanciação (Fisch *et al.*, 2017). Outra questão a ser atacada

é a possibilidade de reduzir a utilização da PRM, que é um limitador para o número de enclaves que podem estar alocados simultaneamente. Tais questionamentos são extremamente importantes em aplicações que são executadas em um ambiente ou contexto em que devem atender a inúmeras requisições em um curto espaço de tempo, haja visto que a instanciação de um enclave para atender a cada requisição pode ser custosa, tanto do ponto de vista de desempenho quanto da utilização da PRM.

Alguns trabalhos presentes na literatura visam melhorar o desempenho de aplicações que utilizam a arquitetura Intel SGX, permitindo que o enclave efetue a alocação dinâmica de memória e, consequentemente, seja inicializado mais rapidamente (Silva *et al.*, 2017). Outros trabalhos também buscam reduzir o impacto de desempenho causado pela troca de contexto entre o enclave e a aplicação (Arnautov *et al.*, 2016; Orenbach *et al.*, 2017), e também para estender o tamanho da PRM, de forma que ela englobe toda a memória principal disponível (Taassori *et al.*, 2018). O ponto comum em tais abordagens está no refinamento de código em baixo nível e alterações nas ferramentas de software do SGX.

Assim, é necessário buscar alternativas para reduzir essa queda de desempenho na aplicação e para garantir uma utilização eficiente da região de memória da PRM, sem que isso comprometa as garantias de segurança providas pelo SGX e sem a necessidade de alterar as suas estruturas de funcionamento. Além disso, o uso de enclaves é recente e não há uma posição clara na comunidade sobre a forma correta de incorporá-los às aplicações, ou seja, sobre o modelo de programação mais adequado para construir programas usando enclaves.

1.3 PROPOSTA

Apesar da arquitetura Intel SGX ser relativamente nova no mercado, sendo disponibilizada comercialmente em 2015, há uma gama de trabalhos na literatura que exploram a utilização dos recursos providos por tal tecnologia nas mais diversas áreas de aplicação, como será detalhado no Capítulo 4. Tais soluções se utilizam de diferentes técnicas e modelos de programação para instanciar e utilizar os enclaves, obtendo acesso aos recursos providos para garantir a integridade e confidencialidade dos dados.

Algo que é inerente ao desenvolvimento e utilização de mecanismos de segurança em soluções de software é o impacto de desempenho causado pelas diversas verificações incluídas no processo de manipulação dos dados sensíveis, e até mesmo na instanciação das aplicações e requisições a diferentes serviços e recursos. Busca-se sempre um ponto ideal onde seja possível garantir um elevado índice de segurança da aplicação com o mínimo impacto possível, principalmente em soluções onde tem-se alta demanda ou espera-se alto desempenho.

Dessa forma, o presente trabalho tem como objetivo propor modelos de gerenciamento de enclaves, possibilitando uma alocação eficiente de recursos, visando reduzir a utilização a PRM e o tempo necessário para a requisição de um enclave. Além da proposição de um modelo de gerenciamento de enclaves, a proposta também apresenta a arquitetura de um serviço que se utilize desses modelos e forneça um gerenciamento de enclaves, permitindo que os enclaves sejam oferecidos às aplicações como recursos lógicos, tendo o seu ciclo de vida gerenciados por tal serviço. Assim, espera-se uma utilização mais eficiente da PRM, e uma redução no impacto de desempenho causado pela instanciação e alocação de recursos pelos enclaves através do gerenciamento do seu ciclo de vida.

Para tanto, é realizado um levantamento de diversos sistemas de software propostos na literatura que fazem uso da tecnologia Intel SGX com o intuito de mapear como os enclaves são utilizados por estas soluções, identificando modelos de programação e possíveis alternativas aos modelos utilizados. A partir das soluções avaliadas, e dos modelos de programação utilizados

por estas soluções, é realizada a especificação de novos modelos explicitando como estes podem ser implementados e utilizados em diferentes soluções, visando reduzir a utilização da PRM e o impacto de desempenho causado pela instanciação dos enclaves.

1.4 ESTRUTURA DO DOCUMENTO

A presente proposta de tese está organizada em 8 capítulos, com o presente capítulo trazendo a contextualização do trabalho, a descrição do problema abordado e uma breve descrição da proposta do trabalho. O Capítulo 2 apresenta uma visão geral sobre os diversos mecanismos de segurança baseados em hardware, os quais contribuíram, de forma direta ou indireta, para o desenvolvimento da arquitetura Intel SGX, descrita no Capítulo 3, onde também são apresentadas as suas limitações e questões em aberto acerca dessa nova arquitetura. O Capítulo 4 traz uma série de trabalhos que fazem uso da arquitetura Intel SGX para as mais variadas soluções, e o Capítulo 5 apresenta os diversos modelos de programação de software e busca identificar quais desses modelos são utilizados para o desenvolvimento das soluções apresentadas no capítulo anterior. Posteriormente, o Capítulo 6 formaliza a proposta deste trabalho, com o Capítulo 7 apresentando os detalhes arquiteturais para a implementação da proposta descrita no capítulo anterior, bem como a validação da proposta. Por fim, o Capítulo 8 faz o fechamento do trabalho, apresentando as contribuições obtidas e descrevendo os trabalhos futuros.

2 SEGURANÇA BASEADA EM HARDWARE

Este capítulo faz uma revisão dos mecanismos de segurança baseados em hardware que pavimentaram o caminho para o desenvolvimento da arquitetura Intel *Software Guard Extensions*, a qual será apresentada no Capítulo 3.

2.1 PROCESSADORES CRIPTOGRÁFICOS

Anderson *et al.* (2006) define um processador criptográfico como um dispositivo embarcado, resistente à violação, que comunica com o computador e fornece uma série de operações criptográficas utilizando chaves que são guardadas e protegidas pelo próprio dispositivo. Seu surgimento se deu no campo das aplicações militares, posteriormente expandindo para o uso nas redes de caixas eletrônicos (ATMs), sendo utilizados para a verificação e validação dos PINs (*Personal Identification Number*). Na década de 1990, os processadores criptográficos passaram a ser utilizados para o armazenamento de chaves SSL (*Secure Socket Layer*), e posteriormente também foram aplicados para uso em DRM (*Digital Rights Management*).

Bossuet *et al.* (2013) divide os processadores criptográficos em quatro categorias: processador de uso geral customizado; coprocessador criptográfico; processador criptográfico; e *array* criptográfico.

Um processador de propósito geral pode ser customizado para obter eficiência na implementação de algoritmos criptográficos adicionando uma Unidade Lógica e Aritmética (ULA) específica para executar operações de criptografia, as quais são chamadas durante a execução das aplicações. Esta solução, porém, ainda armazena as chaves utilizadas para criptografia na memória principal, juntamente com os dados, o que não evita ataques e a extração destas chaves. Esta abordagem pode ser utilizada para aplicações de áreas muito limitadas, principalmente em sistemas embarcados, já que o acréscimo de uma ULA dedicada ainda deixa o projeto relativamente simples, não aumentando demasiadamente os custos.

Um coprocessador criptográfico é um dispositivo lógico ou um hardware dedicado para a execução de funções de criptografia, contendo um ou mais elementos de processamento. Este não pode ser programado, mas pode ser controlado, configurado ou receber parâmetros vindos do processador principal, sendo utilizado para acelerar os cálculos referentes aos processos de criptografia. Neste caso, pode-se conseguir também um alto grau de flexibilidade quando utilizadas as capacidades de reconfiguração de alguns FPGAs para a construção deste dispositivo. A questão da segurança, entretanto, fica prejudicada, visto que, da mesma forma que acontece com os processadores de propósito geral customizados, as chaves de criptografia também são armazenadas na memória principal, juntamente com os dados.

Já um processador criptográfico, difere por ser um dispositivo ou hardware programável, com um conjunto de instruções dedicadas para implementar de forma eficiente as funções de criptografia, contendo uma ou mais ULAs projetadas especificamente para realizar os cálculos criptográficos. Desta forma, ele não pode funcionar como um processador de propósito geral, sendo necessária a inclusão deste segundo para executar as demais instruções do sistema. Gaspar *et al.* (2010) coloca que um processador criptográfico deve preencher quatro requisitos para atingir o objetivo de segurança: as chaves devem ser geradas dentro do processador criptográfico; as chaves devem ser armazenadas em uma memória dedicada, fisicamente separada da memória de dados; as chaves devem ser transportadas entre os processadores utilizando um barramento específico, o qual deve ser fisicamente separado do barramento de dados e; o processador

criptográfico pode ler e escrever as chaves na memória principal utilizando o processo de criptografia e descriptografia, sendo que as chaves nunca devem deixar o processador sem estar criptografadas.

Um *array* criptográfico é assemelhante à proposta de processadores criptográficos, necessitando ser utilizado em conjunto com um processador de uso geral. Esta arquitetura foi implementada principalmente para o algoritmo AES, se valendo da sua capacidade de paralelismo. O *array* criptográfico se diferencia dos processadores criptográficos multinúcleos pela sua finalidade, já que o segundo tem um uso mais genérico, implementando diversos algoritmos de criptografia.

A Figura 2.1 mostra os diagramas dos quatro tipos de processadores criptográficos apresentados.

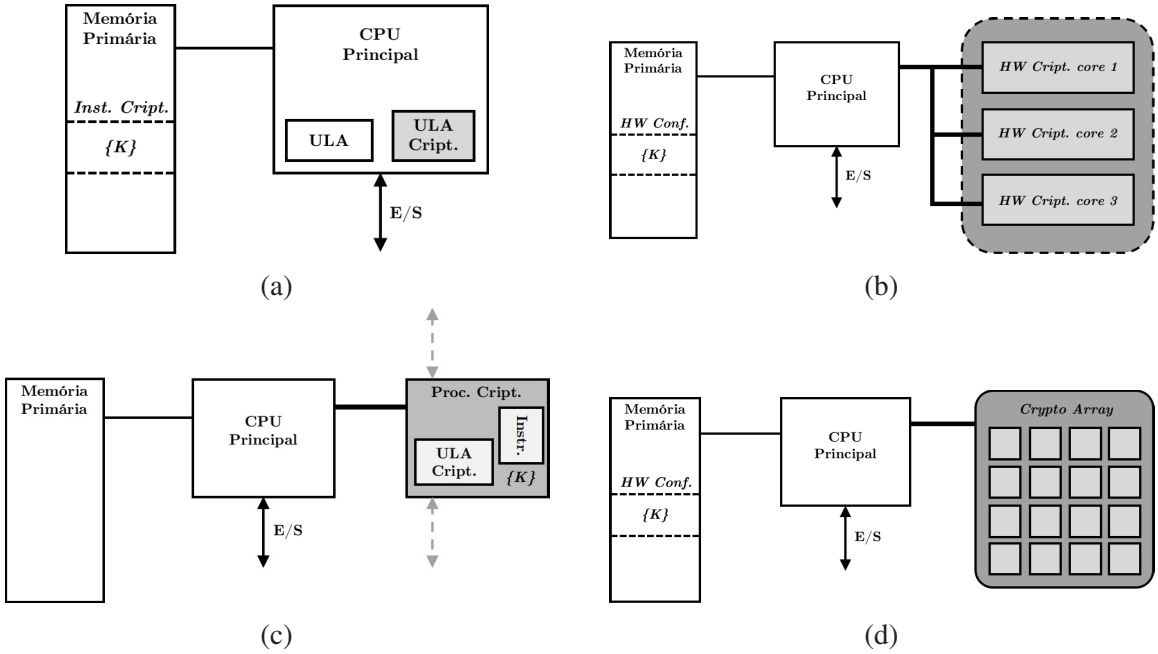


Figura 2.1: Diagrama de blocos de um: (a) processador de uso geral customizado; (b) coprocessador criptográfico; (c) processador criptográfico; (d) *array* criptográfico. Adaptado de Bossuet *et al.* (2013).

2.2 *TRUSTED PLATFORM MODULE*

O *Trusted Platform Module* (TPM) é um hardware seguro que atua como um auditor independente, contendo registradores especiais que são utilizados para armazenar informações sobre o *firmware* e configurações carregadas durante o processo de *boot*, podendo ter seus valores alterados somente em situações específicas, o que impede que estas informações possam ser sobrescritas arbitrariamente por qualquer software malicioso (Amin *et al.*, 2008; Weis, 2014).

Conforme o *Trusted Computing Group* (TCG, 2016), o TPM é um componente do sistema que tem um estado separado do sistema hospedeiro. A interação entre o sistema hospedeiro e o TPM é feita somente através de uma interface definida nas especificações do TPM. O TPM é construído em recursos físicos dedicados exclusivamente a ele. Normalmente são implementados em um único chip acoplado ao sistema (tipicamente um PC) e são compostos de processador, RAM, ROM e memória *flash*, e sua única interface é um barramento *Low Pin Count* (LPC). O sistema hospedeiro não pode alterar valores diretamente na memória do TPM, isto só pode ser feito através do *buffer* de E/S da interface, dedicado para este fim. A composição básica de um TPM é mostrada na Figura 2.2.

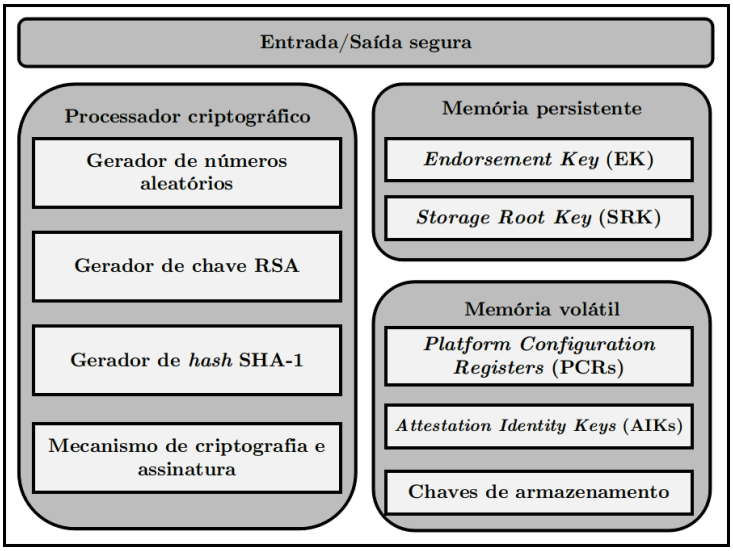


Figura 2.2: Arquitetura de um TPM. Adaptado de Vacca (2016).

O TPM verifica a integridade do sistema e o transforma em um sistema confiável através do *Core Root of Trust Measurement* (CRTM), que inclui um processo de *boot* seguro e autenticação remota. No princípio, o TPM utilizava um método chamado *Static Root of Trust Measurement* (SRTM), o qual valida o sistema apenas no momento do *boot*. Além de ser inflexível, o SRTM contém algumas falhas que permitem que um sistema atestado como seguro não o seja de fato, conforme colocado por Butterworth *et al.* (2013).

Como alternativa ao SRTM, tem-se o *Dynamic Root of Trust Measurement* (DRTM), sendo duas implementações deste o *Intel Trusted Execution Technology* (Intel TXT) e o *Secure Virtual Machine* da AMD (AMD SVM). O DRTM permite atestar o sistema a qualquer momento, sem a necessidade de reiniciá-lo, permitindo que o sistema seja iniciado em um estado inseguro e, após, sejam feitas as medições necessárias para atestar a confiabilidade do sistema, o que permite atestar não somente o *firmware*, mas também o sistema operacional. Teoricamente, o sistema operacional deveria estar totalmente isolado do *firmware* de *boot*, mas na prática isso não ocorre, o que deixa brechas para ataques de *malwares*, conforme colocado por Wojtczuk e Rutkowska (2009). Assim, mesmo na existência do DRTM, se faz necessário uma verificação do *firmware* através do SRTM, trazendo novamente as questões de inflexibilidade e insegurança do mesmo.

Além do CRTM, o TPM também traz o *Root of Trust for Storage* (RTS) e o *Root of Trust for Reporting* (RTR). O primeiro é usado para armazenar elementos como as medidas da plataforma, a *Endorsement Key* (EK), que é criada durante a fabricação e não pode ser alterada, sendo utilizada no processo de autenticação, e a *Storage Root Key* (SRK) que é usada para armazenamento cifrado, sendo gerada em tempo de execução. A memória do TPM tem seu acesso bloqueado a qualquer entidade, exceto o próprio TPM. No entanto, se o elemento de memória contém informações não sensíveis, como no caso de algum *Platform Configuration Register* (PCR), o TPM permite a leitura.

Já o RTR é responsável por reportar acerca do conteúdo do RTS para determinadas finalidades. Normalmente o *report* é um resumo criptográfico assinado digitalmente do conteúdo de valores selecionados do TPM. A interação entre o RTR e o RTS é crítica, sua implementação deve prevenir todas as formas de ataques físicos ou por software e prover os resumos criptográficos com precisão.

2.3 INTEL *TRUSTED EXECUTION TECHNOLOGY*

O Intel *Trusted Execution Technology* (Intel TXT) utiliza componentes de hardware para criar ambientes de execução seguros contra ameaças físicas e virtuais. O Intel TXT necessita de alguns componentes fundamentais, como módulos de código autenticados (*Authenticated Code Modules* - ACMs), ferramentas de *Launch Control Policy* (LCP), um módulo TPM, BIOS e hipervisor ou sistema operacional habilitado para o Intel TXT. Se ocorrer a falta ou falha de qualquer componente da plataforma, esta será carregada no seu estado não-confiável (Greene, 2012).

A execução segura é obtida com inicializações verificadas pelo *Measured Launch Environment* (MLE), que permite que os elementos críticos da inicialização possam ser confrontados contra as medições destes elementos vindas de uma fonte confiável. A medição consiste na criação de um identificador criptográfico único para cada componente aprovado na inicialização. O provisionamento destas medições confiáveis é feito através de um microcódigo autenticado (ACM) embarcado; uma vez realizadas as medições, elas são gravadas e travadas dentro do TPM. A Intel provê, ainda, um mecanismo baseado em hardware para bloquear a inicialização de código que não confere com aqueles previamente aprovados.

Além da inicialização verificada, o Intel TXT dispõe de um mecanismo de políticas para a criação e implementação de listas de códigos executáveis aprovados ou sabidamente confiáveis. Este mecanismo é chamado de *Launch Control Policy* (LCP). Um mecanismo de proteção de segredos também é implementado através de métodos auxiliados por hardware, que removem dados residuais em reinicializações impróprias (ataques por *reset*). Por fim, o Intel TXT também tem a habilidade de produzir credenciais com as medições da plataforma para realizar atestação para usuários, ou sistemas locais ou remotos, de forma a completar o processo de verificação de confiança e suportar atividades de conformidade e auditoria.

Quando utilizado em ambiente virtualizado, o Intel TXT também faz uso extensivo da Tecnologia de Virtualização Intel (Intel VT) para prover proteção contra acesso direto à memória (DMA) não autorizadas e garantir o isolamento de dados no sistema.

2.4 ARM *TRUSTZONE*

A ARM *TrustZone* é uma arquitetura de hardware que estende o quesito de segurança a todo o *design* do sistema, permitindo que qualquer parte do sistema seja protegida. A tecnologia *TrustZone* fornece uma infraestrutura básica que permite aos projetistas de SoC's (*System-on-Chip*) escolher uma gama de componentes que podem auxiliar em funções específicas dentro de um ambiente seguro. O objetivo principal da arquitetura é habilitar a construção de um ambiente programável que permita a confidencialidade e integridade de quase todos os ativos a serem protegidos de ataques específicos, podendo ser utilizada para construir um conjunto de soluções de segurança que não são possíveis com os métodos tradicionais (ARM, 2009).

Com a arquitetura ARM *TrustZone* o sistema pode ser isolado em dois estados lógicos: um estado seguro e um estado inseguro (Figura 2.3). Estes estados também são sinalizados para todos os dispositivos periféricos, através do barramento do sistema, permitindo-lhes tomar decisões de controle de acesso com base no estado atual do sistema. O mecanismos responsável pela troca de contexto entre os dois estados é chamado de *monitor*.

Quando uma aplicação está sendo executada em um estado seguro, ela pode isolar partes da memória para seu próprio uso, impedindo que aplicações sendo executadas em um estado inseguro acessem estes locais. Isto é garantido pelo controlador de memória que utiliza as premissas da arquitetura *TrustZone*, que fornece controle de acesso para regiões de memória com

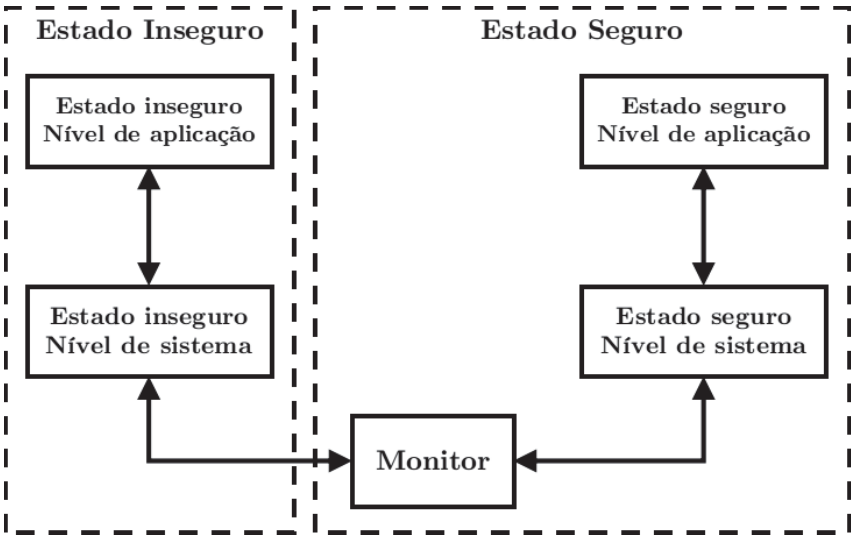


Figura 2.3: Modos de execução na arquitetura ARM *TrustZone*. Adaptado de ARM (2009).

base no estado atual do sistema. Este particionamento de memória pode ser definido de forma fixa, ou programável em tempo de execução.

Uma aplicação em estado seguro também pode forçar que certas interrupções ou exceções sejam capturadas somente em um estado seguro, sendo que este controle também fica a cargo do controlador de interrupções. Além disso, o sistema também pode bloquear o acesso a determinados dispositivos para aplicações que não estejam sendo executadas em um estado seguro, garantindo a exclusividade destes dispositivos somente a aplicações seguras (Lesjak *et al.*, 2015).

Os sistemas que suportam *TrustZone* vêm, ainda, com hardware complementar para suporte a chaves seguras não-voláteis, *Real Time Clock* seguro e aceleradores criptográficos.

Para a consolidação de um ambiente seguro, um sistema operacional e um mecanismo de *boot* seguros devem ser desenvolvidos, utilizando os recursos do *TrustZone*. A combinação do isolamento por hardware *TrustZone*, *boot* seguro e sistema operacional seguro constituem um ambiente de execução confiável (*Trusted Execution Environment* - TEE), com as propriedades de segurança do TEE suportando a confidencialidade e a integridade de múltiplas aplicações confiáveis (ARM, 2009).

2.5 AMD *SECURE PROCESSOR*

Também conhecido como *Platform Security Processor* (PSP), o AMD *Secure Processor* é um subsistema de segurança em hardware dedicado, integrado ao *chip* do processador, que executa de modo independente dos núcleos do processador principal. Esse subsistema proporciona um ambiente para tratamento de dados sensíveis isolado do sistema principal.

Conforme Arthur e Challener (2015), para ser uma base confiável em hardware, podendo ser usada para o estabelecimento da cadeia de confiança, o PSP faz uso de um microcontrolador ARM *TrustZone* de 32 *bits* mas, apesar da utilização da arquitetura *TrustZone*, o PSP não é um núcleo virtual, mas um processador real, fisicamente separado, integrado ao SoC, que dispõe de uma SRAM dedicada e acesso direto ao coprocessador criptográfico. Além disso, o PSP dispõe de acesso aos recursos de memória do sistema e uma DRAM criptografada, com isolamento implementado em hardware.

O PSP também contém um coprocessador criptográfico composto de um gerador de números aleatórios, mecanismos para processamento de algoritmos criptográficos (AES, RSA

e outros) e um bloco para armazenamento de chaves. Este bloco é composto de duas áreas de armazenamento: uma usada por software privilegiado, cuja leitura não é possível; e outra onde chaves podem ser carregadas, utilizadas e descartadas, em operações convencionais tanto de software executando no PSP ou no sistema operacional convencional. Também há uma lógica implementada em hardware para inicialização segura do núcleo da CPU e uma chave única da plataforma, que é distribuída para o bloco de armazenamento do coprocessador criptográfico durante o processo de inicialização.

Para garantir uma inicialização confiável, o PSP implementa o *Hardware Validated Boot* (HVB), que é uma forma de *boot* seguro não-modificável, implementado na ROM do SoC, que verifica a integridade da BIOS. A ROM faz a validação de uma chave de inicialização e então usa essa chave para validar o *firmware* do PSP que, por sua vez, é carregado e inicia a execução da aplicação do sistema.

2.6 AMD *SECURE ENCRYPTED VIRTUALIZATION*

O *Secure Encrypted Virtualization* (SEV) é um recurso de segurança disponível em processadores AMD que permite o isolamento de dados em ambientes virtualizados, cifrando o espaço de endereçamento da máquina virtual com uma chave de criptografia específica para aquela máquina virtual (Mofrad *et al.*, 2018). Para tal tarefa, o AMD SEV se utiliza do AMD *Memory Encryption Engine*, que permite trabalhar com diferentes chaves de criptografia em diferentes espaços de endereçamento. Ao carregar o conteúdo de uma máquina virtual para a *cache*, a CPU restringe o acesso aos dados da cache apenas para a máquina virtual proprietária de tais dados, garantindo a confidencialidade destes perante o sistema.

A cifragem dos dados em memória é efetuada utilizando um algoritmo AES de 128 *bits*, com as chaves de criptografia sendo geradas através do PSP, descrito na Seção 2.5. O PSP expõe uma API que deve ser utilizada pelo hipervisor para gerenciar as chaves de criptografia das máquinas virtuais que estão habilitadas para o AMD SEV. Por fim, o AMD SEV também provê mecanismos de atestação, permitindo que o *firmware* prove a identidade do hardware para os hospedeiros e que estes atestem a integridade da plataforma, garantindo que a mesma não foi alterada ou personificada por um agente malicioso (Buhren *et al.*, 2019).

2.7 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados uma série de mecanismos de segurança baseados em hardware, descrevendo as suas principais características e particularidades, com o objetivo de traçar um histórico das principais tecnologias e mecanismos que existem ou já existiram no mercado e que, de alguma forma, pavimentaram o caminho para a criação da arquitetura Intel SGX, que será explorada no capítulo seguinte.

3 INTEL *SOFTWARE GUARD EXTENSIONS*

O *Intel Software Guard Extensions*, ou *Intel SGX*, é uma extensão ao conjunto de instruções da arquitetura *x86* que permite que aplicações possam ser executadas em uma área protegida de memória, chamada de *enclave*, que irá conter o código e os dados da aplicação. Um enclave é uma área protegida no espaço de endereçamento da aplicação que garante a confidencialidade e integridade dos dados, evitando que esses dados sejam acessados por *malwares* que estejam em execução, e até mesmo de outros softwares com alto privilégio de execução, como monitores de máquinas virtuais, BIOS, e o próprio sistema operacional (McKeen *et al.*, 2013; Jain *et al.*, 2016; Will *et al.*, 2017). O principal objetivo da arquitetura SGX é reduzir o *Trusted Computing Base* (TCB) para uma pequena parte de hardware e software, conforme mostrado na Figura 3.1.

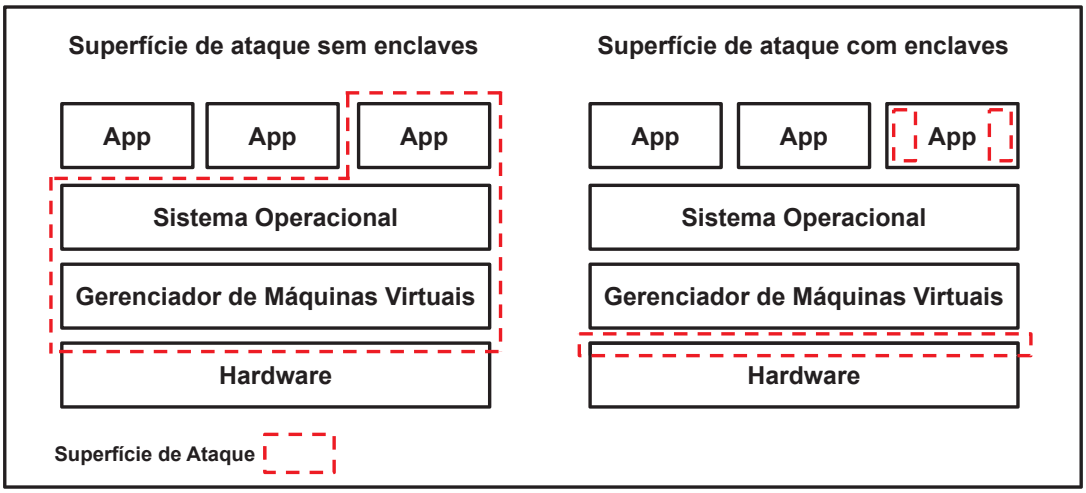


Figura 3.1: Superfície de ataque de um aplicativo sensível à segurança sem enclaves SGX (à esquerda) e com enclaves SGX (à direita). Adaptado de Sobchuk *et al.* (2018).

Para garantir a confidencialidade e integridade dos dados, a arquitetura SGX adicionou novas instruções, uma nova arquitetura de processador e um novo modelo de execução, que incluem o carregamento do enclave para uma área protegida de memória, o acesso de recursos via mapeamento de tabelas de páginas e o escalonamento de aplicações dentro de enclaves. Após a aplicação ser carregada dentro de um enclave, ela fica protegida de todo e qualquer acesso externo ao enclave, inclusive de acesso de outras aplicações que estejam em outros enclaves. As tentativas de alterações não autorizadas de conteúdo dentro de um enclave são detectadas e então são impedidas ou a sua execução é abordada. Enquanto os dados do enclave estão tramitando entre os registradores e os outros blocos do processador, o acesso não autorizado é evitado utilizando os mecanismos de controle de acesso do próprio processador. Quando os dados são escritos para a memória, eles são automaticamente encriptados e a integridade é mantida evitando sondagens de memória ou outras técnicas para visualizar, modificar ou substituir os dados contidos em um enclave (Costan e Devadas, 2016).

A cifragem de memória é feita utilizando algoritmos padrão de criptografia, contendo proteções contra ataques de repetição. O fato de conectar os módulos de memória DRAM em outro sistema apenas dará acesso aos dados em sua forma cifrada; além disso, a chave de criptografia é armazenada em registradores dentro da CPU, não sendo acessível a componentes

externos à mesma, e é alterada aleatoriamente em cada evento de hibernação ou reinício do sistema (Intel, 2016b).

3.1 CICLO DE VIDA DO ENCLAVE

O processo de criação de um enclave consiste em várias etapas: inicialização da estrutura de controle do enclave; alocação das páginas de memória no EPC (*Enclave Page Cache*) e carregamento do conteúdo do enclave para estas páginas; medição do conteúdo do enclave e; criação de um identificador para o enclave. Antes de iniciar a criação do enclave, o processo já estará residindo na memória principal, estando livre para qualquer inspeção e análise e, após ele ser carregado para dentro do enclave, seus dados e código estarão protegidos de quaisquer acessos externos. O ciclo de vida do enclave é apresentado na Figura 3.2, que mostra também as instruções responsáveis pelo gerenciamento do enclave.

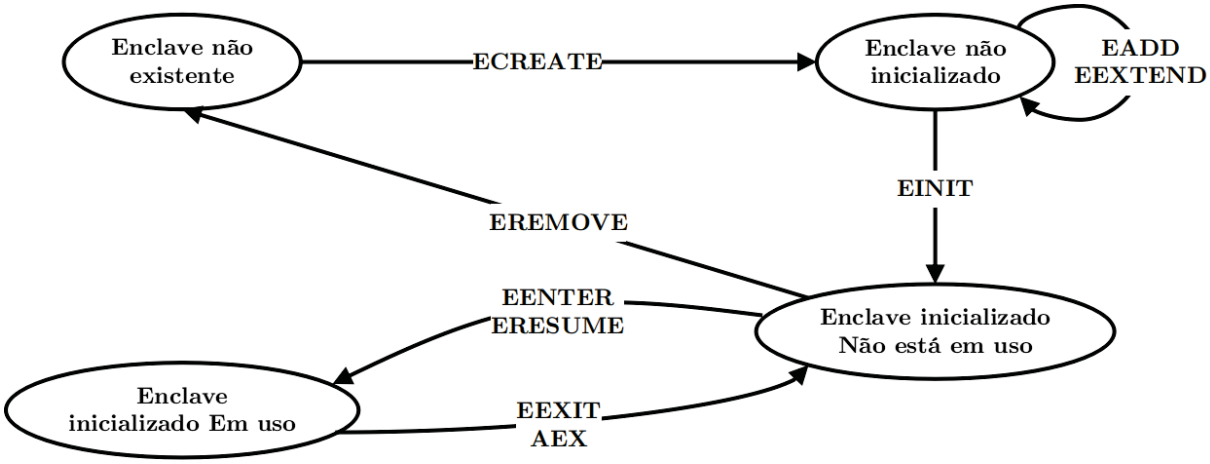


Figura 3.2: Instruções de gerenciamento do ciclo de vida do enclave e diagrama de transição de estados. Adaptado de Costan e Devadas (2016).

A instrução *ECREATE* inicia a criação do enclave e inicializa o *SGX Enclave Control Structure* (SECS) que irá conter informações globais sobre o enclave. Cada página de memória é adicionada ao enclave utilizando a instrução *EADD* e a instrução *EEXTEND* irá medir o conteúdo do enclave. Finalmente, a instrução *EINIT* completa o processo de criação do enclave e cria a sua identidade, permitindo então que ele seja utilizado.

Após esse processo, uma questão importante é o controle da transferência de execução ao entrar e sair do enclave. No processo de entrada no enclave, é necessário limpar quaisquer valores em *cache* que possam se sobrepor à região protegida pelo enclave, além de checar todos os endereços de memória protegidos pelo enclave, identificar a instrução dentro do enclave para a qual o processador deve transferir a execução e habilitar o modo de execução em enclave. Já ao sair de um enclave, novamente deve-se limpar os valores em *cache* que se referem a endereços de memória protegidos pelo enclave, de forma a evitar que outro software possa acessar essas informações. Para entradas e saídas efetuadas programaticamente no enclave, o SGX oferece as instruções *EENTER* e *EEXIT*, respectivamente. 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*.

Por fim, o enclave é destruído através da instrução *EREMOVE*, que irá liberar todas as páginas do EPC utilizadas pelo enclave, garantindo que nenhum processador lógico está

executando instruções dentro das páginas do EPC a serem removidas. O enclave é completamente destruído quando a página do EPC que contém a sua estrutura SECS é liberada (McKeen *et al.*, 2013; Intel, 2014; Costan e Devadas, 2016).

3.2 ORGANIZAÇÃO DE MEMÓRIA DO ENCLAVE

Para implementar a proteção dos dados em memória tem-se o conceito de *Enclave Page Cache* (EPC), onde as páginas de memória e as estruturas de controle do SGX são armazenadas, sendo essa região protegida do acesso externo, tanto via hardware como via software. Dentro do EPC residem os dados de diferentes enclaves, com cada enclave tendo a sua estrutura de controle, chamada SECS, e cada página de memória dentro do EPC pertence a um único enclave. Quando um enclave requisita acesso ao EPC, o processador decide se permitirá ou não esse acesso, mantendo a segurança e o controle de acesso aos dados dentro do EPC através de uma estrutura de hardware chamada *Enclave Page Cache Map* (EPCM).

O EPCM é composto de vários registros, sendo um registro para cada página de memória no EPC, e é gerenciado pelo processador através de um conjunto de instruções, não permitindo que este seja acessível diretamente via software ou outros dispositivos. O EPCM é utilizado pelo processador no processo de tradução de endereços de memória para garantir o controle de acesso às páginas residentes no EPC, fornecendo uma camada adicional de segurança no controle de acesso.

O armazenamento do EPC na memória principal é protegido criptografando os dados, de forma a possibilitar uma defesa contra ataques à memória, tanto por software quanto por hardware. Para isso, uma unidade de hardware chamada *Memory Encryption Engine* (MEE) cuida da criptografia e da integridade dos dados quando estes estão sendo transferidos entre a memória e o processador, sendo que a região de memória onde um MEE opera é chamado de região MEE. Um processador que suporta a arquitetura SGX e implementa o EPC em uma região de memória criptografada também dará suporte para a BIOS reservar um intervalo de memória chamado *Processor Reserved Memory* (PRM), sendo que o PRM poderá ser protegido por uma ou mais regiões MEE. Esta estrutura é mostrada na Figura 3.3.

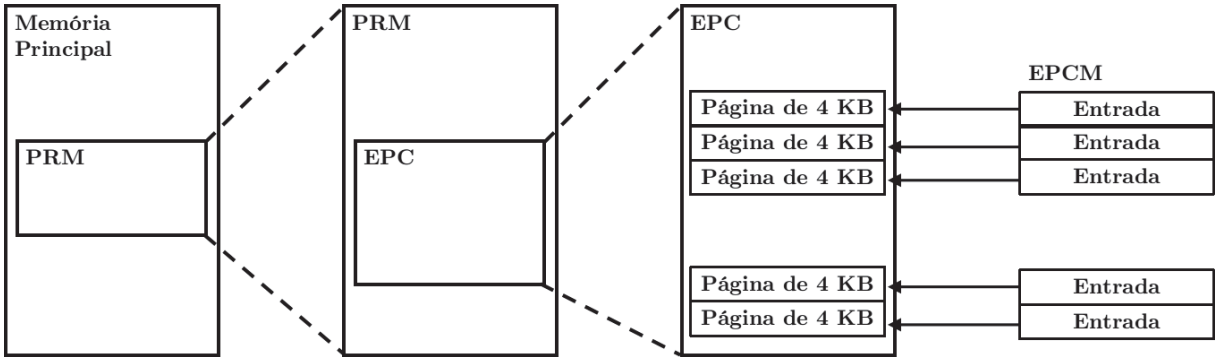


Figura 3.3: Os dados do enclave são armazenados no EPC, que é um subconjunto do PRM, que por sua vez é uma área contígua de memória que não pode ser acessada por outros softwares ou por dispositivos via DMA. Adaptado de Costan e Devadas (2016).

O processador bloqueia qualquer acesso ao PRM vindo de agentes externos, tratando esses acessos como uma referência não existente na memória. Já o acesso às páginas de memória dentro de um enclave utilizando instruções de memória, como MOV, são verificadas pelo hardware seguindo o fluxograma descrito na Figura 3.4, onde é verificado se o processo está sendo executado dentro de um enclave e se o mesmo pertence ao enclave que ele está querendo

acessar, em caso afirmativo, o acesso é liberado, caso contrário, o acesso é bloqueado tratando-o como uma referência a uma memória inexistente. Da mesma forma, as tentativas de acesso às páginas de memória dentro de um enclave por parte de um processo que não está sendo executado em um enclave, também são tratadas como referências inexistentes (McKeen *et al.*, 2013; Intel, 2014; Costan e Devadas, 2016).

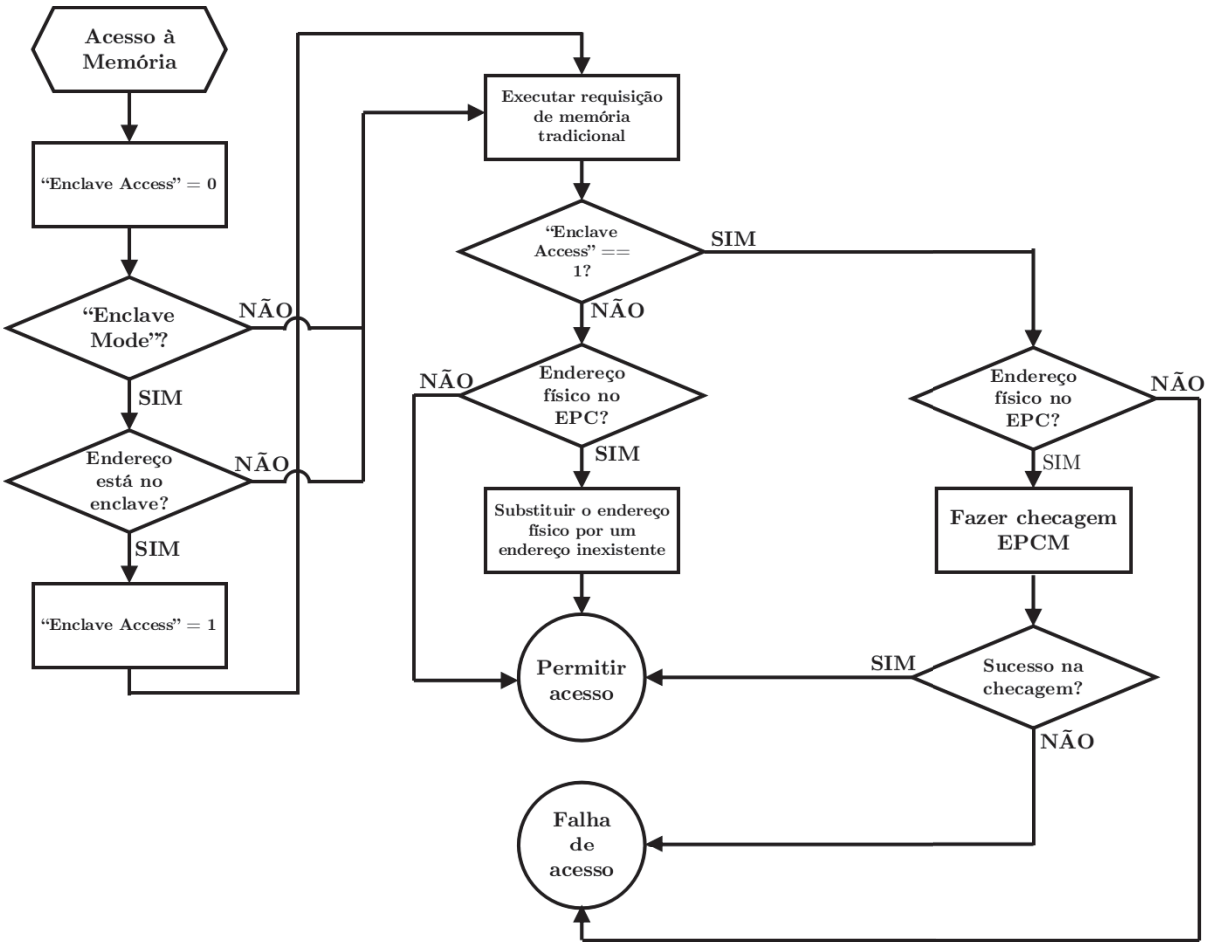


Figura 3.4: Fluxograma de acesso à memória do enclave. Adaptado de McKeen *et al.* (2013).

3.3 MEDIÇÃO E ASSINATURA DO ENCLAVE

Cada enclave possui um certificado auto-assinado pelo autor do enclave, chamado de *Enclave Signature* (SIGSTRUCT). A assinatura do enclave contém informações que permitem que a arquitetura Intel SGX detecte a adulteração de alguma parte do enclave, de forma a permitir que um enclave possa provar que ele foi corretamente carregado no EPC e pode ser confiável. No entanto, o hardware apenas verifica a medição do enclave quando este é carregado. Assim, qualquer pessoa pode modificar um enclave e assiná-lo com sua própria chave. Para evitar esse tipo de ataque, a assinatura do enclave também identifica o autor do enclave, contendo vários campos essenciais para que o enclave seja atestado por entidades externas.

O desenvolvedor deve fornecer o *Security Version Number* (SVN) e o *Product ID* de um enclave, bem como um par de chaves para gerar a assinatura do enclave. A CPU deriva a identidade do autor do enclave a partir de sua chave pública, já a chave privada é utilizada para assinar o enclave. O cálculo da medição do enclave (*Enclave Measurement*) é um *hash* de 256 *bits* único, que identifica o código e os dados iniciais a serem colocados dentro do enclave, bem

como a ordem e a posição na qual eles devem ser colocados, e as propriedades de segurança dessas páginas. Uma alteração em qualquer uma dessas variáveis resultará em uma medida diferente. Quando as páginas de código/dados do enclave são carregadas para o EPC, a CPU efetua a medição do enclave e armazena esse valor no registrador MRENCLAVE. Em seguida, a CPU compara o conteúdo do MRENCLAVE com o valor contido no SIGSTRUCT desse enclave e, somente se eles forem idênticos, a CPU permitirá que o enclave seja inicializado.

A chave de assinatura do desenvolvedor é parte da identidade do enclave e é fundamental para proteger seus segredos. Um intruso que comprometa a chave de assinatura privada de um ISV (*Independent Software Vendor*) poderá ser capaz de escrever um enclave malicioso que ateste com êxito a identidade dos enclaves legítimos e/ou escrever um *malware* que usa o enclave malicioso para comprometer dados selados em plataformas individuais (Intel, 2016a).

3.4 INTERFACE DO ENCLAVE: CHAMADAS ECALL E OCALL

Toda a funcionalidade contida em um enclave deve ser ligada estaticamente, em tempo de compilação, o que pode levar a um aumento do *Trusted Computing Base* (TCB) da aplicação. A solução proposta pela arquitetura SGX é fornecer uma camada para executar determinadas funções fora do enclave (*system calls*, por exemplo, que não podem ser executadas dentro do enclave), o que adiciona uma sobrecarga de desempenho na aplicação. A Figura 3.5 representa esta solução, dividindo a aplicação em duas partes, confiável e não confiável, com ambas tendo suas rotinas de borda, responsáveis por fazer a interligação entre as camadas (Intel, 2016a).

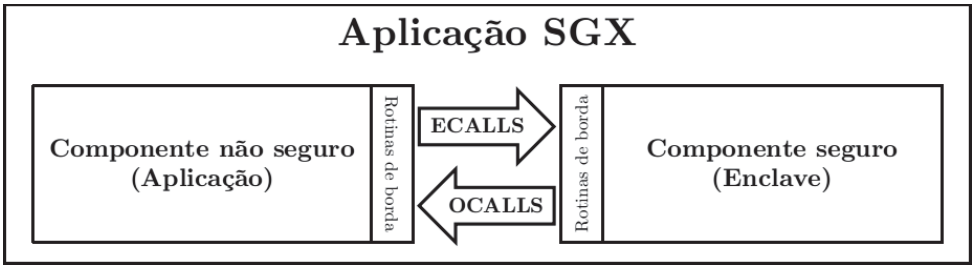


Figura 3.5: Divisão da aplicação em dois componentes, confiável e não confiável, e a comunicação entre os dois componentes. Adaptado de Intel (2016a).

A separação da aplicação em dois componentes, com a redução do TCB, traz algumas vantagens, com menos pontos de falha na parte confiável da aplicação, resultando em um software mais seguro. Ao reduzir o tamanho do enclave, este também será inicializado mais rapidamente, além de permitir que o *backup* dos seus dados seja feito de forma mais ágil quando ocorrer uma hibernação do sistema.

Desta forma, um enclave deve disponibilizar um conjunto de métodos e/ou funções que estarão disponíveis para que o aplicativo faça uso (ECALLs) e descrever quais serviços o aplicativo deve fornecer (OCALLs), sendo que essas funções devem ser definidas pelo desenvolvedor.

As funções ECALLs expõem a interface do enclave que a parte não confiável da aplicação pode utilizar e, por isso, devem ser limitadas para reduzir a superfície de ataque ao enclave. As funções ECALL somente podem ser chamadas após a inicialização do enclave, significando que as alocações de endereço necessárias foram executadas com êxito.

Os enclaves não podem acessar diretamente os serviços fornecidos pelo sistema operacional (*system calls*). Para isso, um enclave deve executar uma chamada OCALL para uma rotina no aplicativo não confiável, o que acrescenta uma sobrecarga de desempenho, mas não afeta a confidencialidade dos dados. Ao executar uma chamada OCALL, o domínio não confiável

da aplicação também pode efetuar chamadas ECALL, retornando a execução para o enclave durante a execução dessa chamada. Após a finalização de uma chamada OCALL, a execução retorna para o domínio do enclave, dando continuidade à função que chamou a OCALL.

3.5 SELAGEM DE DADOS

As aplicações também podem requisitar uma chave específica ao enclave para poder proteger suas chaves e dados quando desejar gravar estes fora da proteção do enclave, como, por exemplo, em disco. Isso é necessário porque as aplicações que executarão em enclaves não devem ser distribuídas com os dados confidenciais, mas sim, após a instalação, contatar um provedor de serviços para solicitar estes dados. Este processo é chamado de selagem de dados.

Ao selar os dados, o desenvolvedor deve especificar as condições necessárias para, posteriormente, abrir os dados selados e, para isto, existem duas opções disponíveis. A primeira opção envolve a criação de um selo para o enclave corrente, utilizando a medida do enclave (*MRENCLAVE*) para a criação do selo, indicando que apenas o enclave que selou os dados está apto a abrir esses dados, ou seja, se o código do enclave sofrer alterações, por qualquer motivo, ele não estará mais apto a abrir os dados que selou anteriormente. A segunda opção é utilizar a assinatura do autor do enclave (*MRSIGNER*) para gerar o selo do enclave, permitindo que diferentes versões do mesmo enclave ou até mesmo diferentes enclaves do mesmo autor possam abrir os dados selados.

Ao codificar o enclave, o desenvolvedor pode definir um número de versão para o mesmo, adicionando uma camada a mais de segurança. A versão do enclave é armazenada juntamente com os dados selados, permitindo que apenas as versões iguais ou superiores do enclave possam abrir esses dados. Este processo é descrito na Figura 3.6 (Anati *et al.*, 2013).

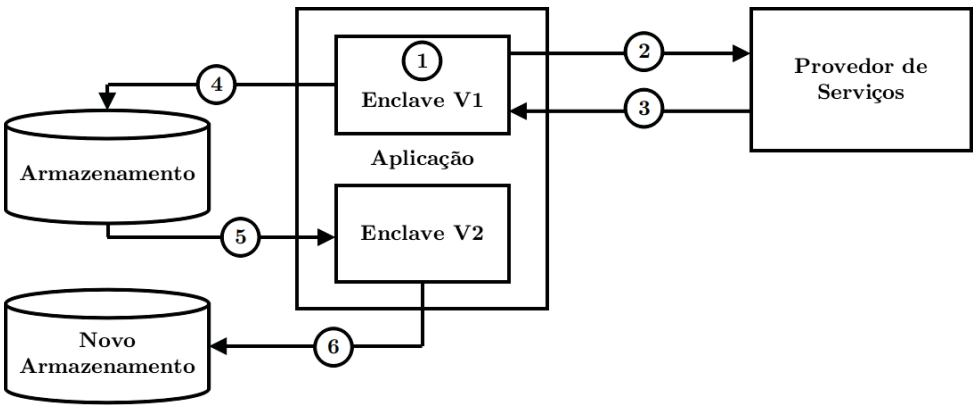


Figura 3.6: Fluxograma referente à integridade no armazenamento de dados. Adaptado de Anati *et al.* (2013).

No passo 1, o software inicia a criação do enclave para que possa ser executado de forma segura. No passo 2, o enclave contata o provedor de serviços para efetuar o *download* dos dados, nessa etapa o provedor irá gerar uma chave que identificará a máquina e o enclave que está executando a aplicação. Posteriormente, no passo 3, o provedor estabelece uma linha de comunicação segura com a aplicação e envia os dados para o enclave. Já no passo 4, o enclave utiliza uma chave de criptografia baseada em hardware para codificar e armazenar os dados confidenciais em um sistema de armazenamento permanente, garantindo que os dados serão acessados somente quando o enclave for restaurado. No passo 5 tem-se a situação de uma atualização de software, sendo que neste caso o enclave irá acessar os dados e gerar uma nova chave de criptografia, de forma que somente a nova versão do software consiga acessar os dados novamente, impedindo que versões anteriores da aplicação tenham acesso a estes dados.

Este processo tem como finalidade proteger os dados confidenciais até mesmo de versões anteriores da aplicação que possam conter brechas que possibilitem o acesso indevido aos dados. Desta forma, quando essas brechas forem detectadas, a empresa responsável pelo desenvolvimento do software pode corrigi-las e disponibilizar uma atualização, tendo também um controle que apenas os usuários que dispõem da última versão do software poderão acessar os dados e o provedor (Intel, 2014).

3.6 ATESTAÇÃO ENTRE ENCLAVES

Para permitir que as aplicações que utilizam os enclaves possam cooperar umas com as outras, elas precisam utilizar meios de efetuar uma atestação mútua.

Atestação é o processo de comprovar que um software foi corretamente estabelecido em uma plataforma. No caso da Intel SGX, é o mecanismo pelo qual uma terceira entidade estabelece que uma determinada entidade de software está em execução em uma plataforma habilitada para Intel SGX e protegida dentro de um enclave, antes de provisionar esse software com segredos e dados protegidos. A atestação depende da capacidade de uma plataforma produzir uma credencial que reflita com precisão a assinatura de um enclave, que inclui informações sobre as propriedades de segurança do enclave.

Essa atestação pode ser de duas maneiras: local, quando os dois enclaves estão sendo executados no mesmo dispositivo e; remota, com os enclaves executando em dispositivos diferentes.

3.6.1 Atestação Local

Na atestação local, a arquitetura SGX dispõe da instrução *EREPORT* que, quando invocada, cria uma estrutura que contém a identidade dos dois enclaves, os atributos associados ao enclave, quaisquer informações adicionais que o desenvolvedor da aplicação queira enviar ao enclave de destino, e um código de autenticação MAC (*Message Authentication Code*). Essa estrutura é conhecida como *REPORT*. O processo de atestação é efetuado em três etapas, como mostra a Figura 3.7 (Anati *et al.*, 2013).

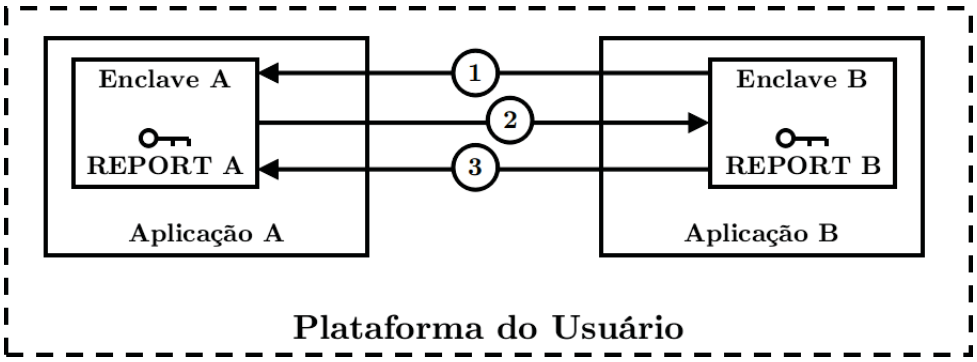


Figura 3.7: Diagrama de atestação local. Adaptado de Anati *et al.* (2013).

No primeiro passo, o enclave A obtém a identificação do enclave B, utilizando uma comunicação aberta. No passo 2 o enclave A cria a estrutura *REPORT* utilizando a identificação do enclave B e envia esta estrutura ao enclave B, ainda utilizando uma comunicação aberta. No terceiro passo, o enclave B utiliza os dados contidos no *REPORT* enviado por A de forma a verificar que o enclave A está sendo executado no mesmo dispositivo que B e, confirmando estes dados, o enclave B também pode criar uma estrutura *REPORT* e enviar esta ao enclave A. Assim, os dois enclaves estão atestados e podem trocar mensagens de maneira segura.

3.6.2 Atestação Remota

Já a atestação remota requer o uso de criptografia assimétrica e de um enclave especial, chamado de *Quoting Enclave*, o qual irá verificar as estruturas *REPORT* dos outros enclaves da máquina utilizando a atestação local e então substituir o *Message Authentication Code* (MAC) destas estruturas com uma assinatura criada através de uma chave assimétrica, utilizando o *Intel Enhanced Privacy ID* (EPID), tendo como saída uma outra estrutura chamada *QUOTE*. O processo de atestação remota é descrito na Figura 3.8 (Anati *et al.*, 2013).

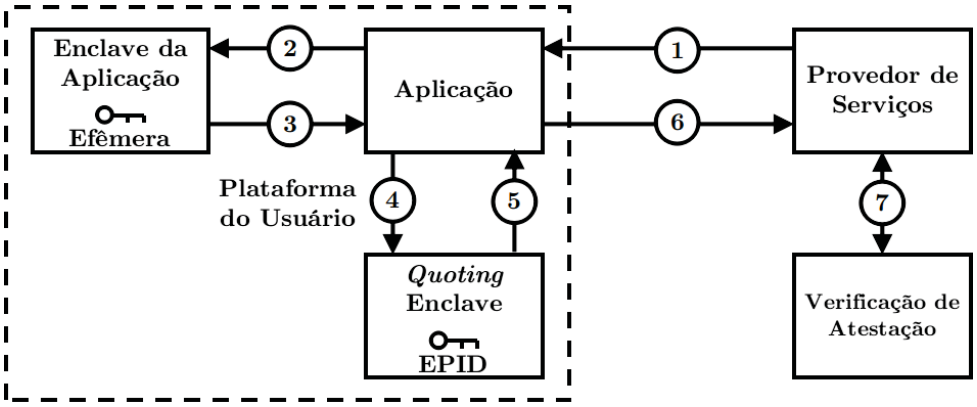


Figura 3.8: Diagrama de atestação remota. Adaptado de Anati *et al.* (2013).

Primeiramente, o provedor de serviços solicita que a aplicação prove que está executando os componentes necessários dentro de um ou mais enclaves. No segundo passo a aplicação repassa a solicitação do provedor de serviços, juntamente com a identidade do *Quoting Enclave*, para o enclave da aplicação. No passo 3, o enclave da aplicação irá gerar uma chave pública que deverá ser utilizada pelo provedor de serviços para as próximas comunicações e então gera a estrutura *REPORT*, enviando esta para a aplicação. No passo 4, a aplicação repassa a estrutura *REPORT* para que o *Quoting Enclave* assine. No passo 5 o *Quoting Enclave* verifica a estrutura *REPORT* e então cria a estrutura *QUOTE*, assinando-a com a sua chave EPID. No passo seguinte a aplicação envia a estrutura *QUOTE* para o provedor de serviços. Finalmente, o provedor de serviços pode utilizar um verificador de atestação para analisar a resposta recebida e verificar se a aplicação satisfaz os requisitos.

3.7 LIMITAÇÕES DA ARQUITETURA SGX

Uma das limitações da arquitetura SGX diz respeito ao uso da CPU para determinar a chave utilizada para a selagem dos dados. Isso implica que, no caso de substituição da CPU, seja por opção do usuário ou por problemas técnicos, o enclave não estará mais apto a abrir os dados previamente selados por ele. Essa é uma questão importante também no balanceamento de carga em servidores, seja na utilização de múltiplos servidores ou de servidores com múltiplos processadores. Em virtude dessa limitação, ainda não há disponíveis no mercado processadores multi-*socket* com arquitetura SGX. O Intel *Software Guard Extensions Developer Guide* (Intel, 2016a) também coloca que, na necessidade de efetuar a migração de dados selados para outro computador, isso deverá ser efetuado utilizando o processo de atestação remota entre ambos os enclaves.

Outro ponto, descrito por Hoekstra *et al.* (2013), diz respeito à entrada de dados das aplicações. Os usuários, inevitavelmente, terão que informar dados em algum momento para as aplicações que estão protegidas em seus enclaves, mas os dispositivos de entrada de dados (como

teclados, câmeras de vídeo, microfones, entre outros) não estão preparados para fornecer esses dados ao sistema de forma segura, o que possibilita sua interceptação antes que eles cheguem à segurança do enclave. Vale ressaltar também que, assim como os dispositivos de entrada, tem-se também os dispositivos de saída que, da mesma forma, não estão preparados para fornecer a segurança adequada no tratamento dos dados provenientes dos enclaves.

3.8 VULNERABILIDADES E QUESTÕES EM ABERTO DA ARQUITETURA SGX

Apesar da arquitetura Intel SGX prover mecanismos eficientes para garantir a segurança dos dados de uma aplicação, ainda há algumas questões a serem consideradas com maior atenção.

Davenport e Ford (2014) apontam uma preocupação quanto ao uso do SGX como um aliado à execução de *malwares*, visto que a Intel não fez nenhuma restrição a quais aplicações poderiam se utilizar da segurança dos enclaves. Schwarz *et al.* (2017) implementaram um ataque de canal lateral ao enclave através de um *malware* que, por executar dentro de outro enclave, fica protegido pelo isolamento provido pela arquitetura SGX e não pode ser identificado por anti-virus ou analisado pelo sistema operacional.

O Intel SGX não considera ataques por canal lateral nem ataques de engenharia reversa em seu modelo de ameaças. O Intel *Software Guard Extensions Developer Guide* (Intel, 2016a) ressalta que cabe aos desenvolvedores construir enclaves resistentes a esses tipos de ataque. O *malware* de Schwarz *et al.* (2017) é um ataque de canal lateral à memória *cache* do tipo *Prime and Probe*, capaz de extrair uma chave de um processamento RSA ocorrendo dentro do enclave da vítima. Este é o tipo de ataque mais comum ao enclave SGX encontrado até o momento. Moghimi *et al.* (2017) utiliza este mesmo método para extrair chaves AES de um processamento do enclave e Brasser *et al.* (2017) conseguiu, além de extrair uma chave RSA, detectar sequências específicas do genoma humano durante a indexação genômica ocorrendo dentro do enclave.

Nestes ataques o enclave da vítima e o *malware* executam em paralelo na mesma máquina física. Apesar do enclave estar protegido criptograficamente na EPC, na *cache* os dados estão em claro. Além disso, é o software do sistema operacional que gerencia a tradução dos endereços da *cache*, que por sua vez é compartilhada com os demais softwares em execução no computador. Desta forma, apesar do atacante não conseguir acesso direto ao conteúdo do *cache* usado pelo enclave, ele pode aplicar a técnica conhecida como *Prime and Probe* para inferir segredos do enclave.

Para obter sucesso com a técnica *Prime and Probe*, pelo menos dois requisitos devem ser atendidos: o *malware* deve ser capaz de sobrescrever determinados endereços da *cache* usados pela vítima, e também deve ser capaz de realizar medição de tempo com resolução boa o suficiente para distinguir *hits* e *misses* na *cache*. Em linhas gerais, no primeiro momento o atacante preenche a *cache* monitorada com seus dados. Então aguarda que a vítima execute algum código cujo acesso à memória é dependente da informação secreta (a chave por exemplo). Na sequencia o atacante faz uma requisição a dados seus e mede o tempo para recuperá-los. Este tempo determina se o dado se encontrava ou não na *cache* (*hit* ou *miss*). No caso de *miss*, sabe-se que aquele endereço foi utilizado pela vítima. Então o atacante se aproveita da dependência do acesso à memória para inferir os *bits* correspondentes ao segredo.

Buscando atenuar os efeitos causados por ataques de canal lateral em aplicações que fazem uso do SGX, Shih *et al.* (2017) propõem o T-SGX, que utiliza os recursos providos pelo *Transactional Synchronization Extensions* (TSX), para isolar as tentativas de acesso indevido aos dados do enclave, erradicando os efeitos das técnicas de ataque de canal lateral conhecidas.

Já início de 2018, um grupo de pesquisadores descobriu três variantes de um método de análise de canal lateral que, ao ser utilizado para fins maliciosos, tem o potencial de coletar

dados sensíveis em diversos processadores e diversos sistemas disponíveis no mercado. Estas vulnerabilidades são conhecidas como *Spectre* (variantes 1 e 2) e *Meltdown* (variante 3), e são descritas em Kocher *et al.* (2019). A exploração das vulnerabilidades *Spectre* para inferir segredos contidos em um enclave são demonstradas por Chen *et al.* (2019), onde os autores mostram que a predição de execução de código de um enclave pode ser influenciada por aplicações de fora do enclave, o que pode alterar temporariamente o fluxo de controle do enclave para executar instruções que levem a alterações de estado de *cache* observáveis, as quais um adversário pode utilizar para aprender segredos da memória do enclave ou de seus registradores. Como se trata de um ataque de canal lateral, estas vulnerabilidades não estão cobertas pelo modelo de ameaças da arquitetura Intel SGX e, assim sendo, a Intel divulgou o documento *Development Guidance for Potential Bounds Check Bypass (CVE-2017-5753) Side Channel Exploits* contendo medidas que devem ser seguidas para atenuar potenciais impactos destes ataques em aplicações que utilizam os mecanismos providos pelo SGX (Intel, 2018).

Outro tipo de ataque é descrito por Jang *et al.* (2017), sendo chamado pelos autores de *SGX-Bomb*, e consiste em lançar um ataque *rowhammer* contra a memória do enclave para desencadear o bloqueio do processador. O ataque resulta de um efeito colateral involuntário na memória dinâmica de acesso aleatório (DRAM) que faz com que as células de memória vazem suas cargas e interagem eletricamente entre elas, possivelmente alterando o conteúdo de linhas de memória próximas que não foram abordadas no acesso original à memória. Assim, efetua-se acesso repetitivo a um endereço de memória do enclave, ignorando o *cache*, e se os *bits* alterados involuntariamente estiverem na região de memória reservada a um enclave, qualquer tentativa de leitura da memória do enclave resultará em uma falha de verificação de integridade, fazendo com que o processador seja bloqueado e o sistema tenha de ser reiniciado.

Em nível de software, no entanto, várias bibliotecas criptográficas estão sendo robustecidas especificamente para evitar ataques à *cache*. Para cada acesso à memória dependente de segredos, o enclave pode gerar um conjunto de acessos à memória que irá se manifestar em mudanças em todos os conjuntos da *cache* monitorados, escondendo o endereço de memória efetivamente acessado do adversário. Outras abordagens buscam eliminar as dependências entre o segredo e o acesso à memória das implementações, o que também é efetivo para evitar este tipo de ataque.

Weichbrodt *et al.* (2016) também aborda problemas de sincronização de *threads* em enclaves, através da utilização de técnicas como *use-after-free* e *time-of-check-to-time-of-use*, permitindo que o atacante sequestre o fluxo de controle ou ignore controles de acesso do enclave, interrompendo *threads* e forçando falhas de segmentação em enclaves.

Costan e Devadas (2016) ressalta outras vulnerabilidades, como ataques por monitoramento do consumo de energia, cujas variações podem gerar informações suficientes para que o atacante infira a sequência de instruções sendo executadas, ou ainda, ataques passivos de tradução de endereçamento que podem dar informações do padrão de acesso à memória com a granularidade de páginas. A Intel menciona que o modelo de ameaças ao SGX considera que o atacante pode ter acesso à *flash* que armazena o *firmware* do computador, mas segundo Costan e Devadas (2016), a documentação oficial não aborda as implicações decorrentes desse fato. No entanto, até o presente momento, não foram encontrados trabalhos demonstrando o uso dessas vulnerabilidades para efetivamente descobrir segredos do enclave.

Por fim, além da Intel ter de ser considerada confiável, como, por exemplo, no provimento da *PROVISIONKEY* utilizada em enclaves que implementam a atestação, Costan e Devadas (2016) questionam a real necessidade do *Launch Enclave* (LE), que é um enclave emitido pela Intel e responsável por aprovar todos os enclaves antes da inicialização. Segundo os autores, o

LE poderia funcionar como um mecanismo de licenciamento, permitindo à Intel se posicionar como um intermediário na distribuição de todo software SGX.

3.9 CONSIDERAÇÕES FINAIS

Este capítulo apresentou em detalhes a arquitetura Intel SGX, disponibilizada em 2015 e que trouxe novos conceitos e instruções que buscam garantir a confidencialidade e integridade de dados sensíveis de aplicações. Foram apresentados detalhes desta arquitetura, como os processos de assinatura e medição de enclaves, além da atestação local e remota entre enclaves e a selagem de dados. Também foram pontuadas as limitações de tal arquitetura, além das limitações inerentes à sua utilização. Por fim, a Seção 3.8 apresentou os trabalhos que discutem sobre algumas das vulnerabilidades e questões que ainda estão em discussão sobre o SGX.

4 APLICAÇÕES DA ARQUITETURA INTEL SGX

Apesar da arquitetura SGX ser relativamente nova, já que foi, de fato, lançada no mercado em outubro de 2015, juntamente com os processadores Intel Core de 6^a geração (*Skylake*), desde o seu anúncio vários trabalhos foram sendo desenvolvidos de forma a validar a proposta. Este capítulo apresenta um estudo detalhado de diversas destas aplicações, analisando a arquitetura utilizada em cada uma delas.

4.1 CLASSIFICAÇÃO DOS TRABALHOS

Nesta seção classificamos os usos do SGX de acordo com seu contexto de aplicação, indicando onde cada solução está sendo utilizada. Assim, foi considerado dois contextos principais: **local** e **distribuído**. No contexto local, são definidos os seguintes subconjuntos: **aplicações seguras** (Seção 4.2), abrangendo o desenvolvimento de aplicações e bibliotecas seguras; ***runtime*** (Seção 4.3), no qual SGX é usado para fornecer segurança para aplicações sem a necessidade dessas sofrerem modificações em seu código-fonte; e *serviços de sistema operacional* (Seção 4.4), em que SGX é usado para melhorar a segurança dos componentes e serviços do sistema operacional.

O contexto distribuído abrange **rede**, **sistemas distribuídas** e **ambiente de nuvem**. No conjunto de redes (Seção 4.5), existem soluções para melhorar a segurança da infraestrutura básica (como roteamento, gateways e serviços de nomenclatura) e software de rede. As aplicações distribuídas (Seção 4.6) abrangem *blockchain*, dispositivos IoT e sistemas *peer-to-peer*. Finalmente, no contexto do ambiente de nuvem (Seção 4.7), as soluções apresentadas abrangem infraestrutura, gerenciamento, virtualização, serviços, análise de dados e soluções de Gestão de Direitos Digitais e Empresariais (DRM e ERM).

Um diagrama com uma visão geral da organização utilizada para a apresentação dos trabalhos é mostrado na Figura 4.1, dividindo as aplicações nos dois eixos principais (local e distribuído), e contendo as ramificações com suas subcategorias.

4.2 APLICAÇÕES SEGURAS

Um campo de uso óbvio para SGX é construir aplicações mais seguras, como a solução apresentada por Kim *et al.* (2017), com o objetivo de aumentar a segurança do navegador anônimo *Tor*, chamada de *SGX-Tor*. Os autores propõem a migração de componentes sensíveis do Tor, como criptografia e operações de roteamento, para um enclave SGX, e também utilizam o recurso de selagem para armazenas chaves privadas e documentos de consenso. Hoekstra *et al.* (2013) buscam manter o sigilo dos dados de áudio e vídeo em um sistema de videoconferência, codificando-os para, posteriormente, serem transmitidos ao receptor através de um canal seguro, conseguido através da atestação entre os enclaves, conforme demonstrado na Figura 4.2. Além disso, é utilizado um servidor de videoconferência para que os usuários possam se autenticar, efetuando a atestação entre cada *host*, para o provisionamento de segredos necessários para a comunicação entre as partes.

A segurança de credenciais de usuários é abordada por Brekalo *et al.* (2016), que propõem a utilização da arquitetura SGX para a adição de uma chave à senha e ao *salt* antes da geração do *hash* e, efetuando o cálculo do *hash* dentro do enclave, essa chave nunca cruzará seus limites, inviabilizando assim ataques *offline*. Uma validação adicional é realizada através de um

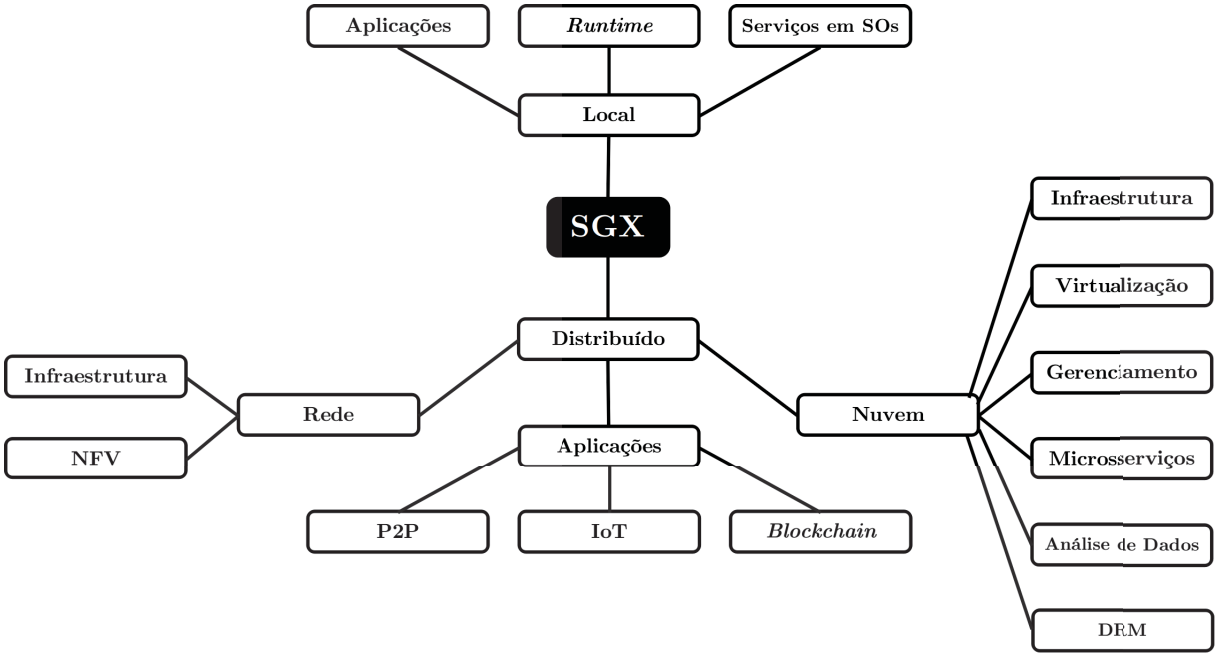


Figura 4.1: Classificação das aplicações da arquitetura SGX apresentadas neste capítulo.

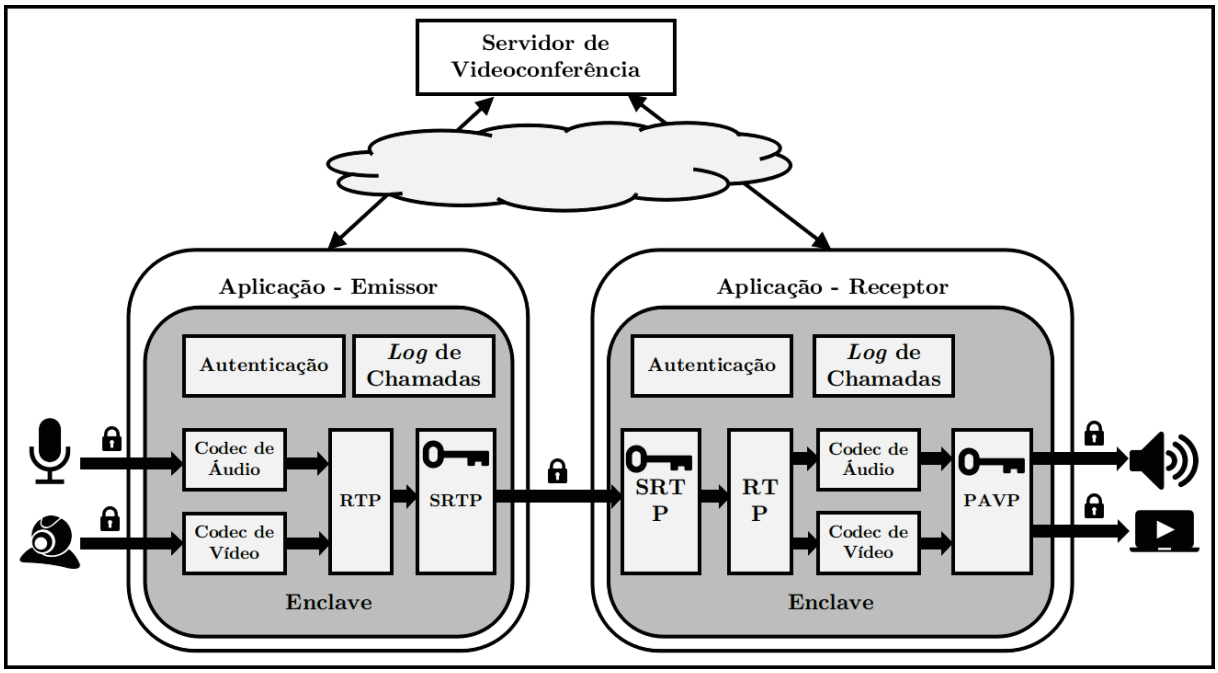


Figura 4.2: Arquitetura do sistema de videoconferência seguro. Adaptado de Hoekstra *et al.* (2013).

enclave provedor, com o qual o enclave cliente deverá realizar a atestação local para assegurar que está sendo executado em um provedor confiável. Liang *et al.* (2017) utilizam os mecanismos de selagem e atestação providos pelo SGX para garantir a confidencialidade e integridade das credenciais do usuário, tanto no seu armazenamento local, quanto na autenticação com o servidor. A arquitetura da solução é descrita na Figura 4.3 e consiste em duas etapas. A primeira etapa é a recuperação das credenciais armazenadas localmente, que é realizada com a utilização de um enclave para acessar os dados selados. A segunda etapa consiste na atestação remota com o servidor de armazenamento, para a validação das credenciais e posterior sincronização de dados.

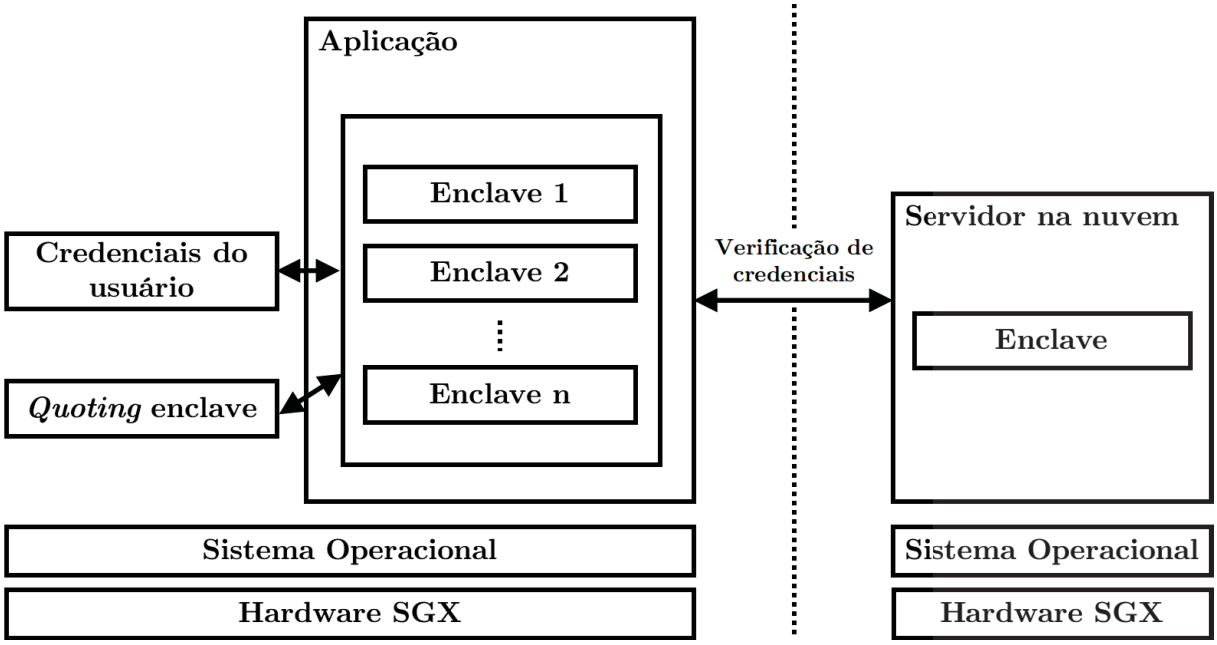


Figura 4.3: Arquitetura da solução *MITC Defender*. Adaptado de Liang *et al.* (2017).

Aplicações OTP (*One-Time Password*) também são objeto de estudo de Hoekstra *et al.* (2013), que utilizam um enclave para gerar *tokens* de maneira segura, que podem ser utilizados como um segundo fator de autenticação. Para isso, o cliente faz a comunicação com um enclave sendo executado no servidor OTP através de um canal seguro estabelecido no processo de autenticação remota, conforme apresentado na Figura 4.4. Outra abordagem tem foco no processo de autenticação em ambientes *Web*. A solução apresentada pelos Krawiecka *et al.* (2018) é composta por um *plugin* que é executado no navegador *Web* do usuário, cujo propósito é estabelecer um canal seguro de comunicação com o servidor para o envio das credenciais. No lado servidor, a senha é armazenada de forma criptografada e é manipulada somente pelo enclave, que possui a chave para cifrar e decifrar seu conteúdo. A estrutura da solução é mostrada na Figura 4.5.

A arquitetura SGX pode ser utilizada para aumentar a segurança na execução de funções criptográficas, como demonstram Mofrad *et al.* (2017), que utilizam a arquitetura SGX na criação de uma biblioteca criptográfica segura, utilizando enclaves para armazenar as chaves de criptografia e atuando como uma API (*Application Program Interface*) para as funções criptográficas implementadas. Na solução proposta pelos autores, as chaves de criptografia são geradas e mantidas dentro do enclave, utilizando a função de geração de números pseudo-aleatórios provida pelo SGX.

Fisch *et al.* (2017) também propõem a utilização da arquitetura Intel SGX para a construção de uma arquitetura de criptografia funcional, a qual facilita o controle de acesso refinado e não interativo a dados criptografados. A arquitetura proposta, chamada de *Iron*, é mostrada na Figura 4.6, e se utiliza de um enclave gerenciador de chaves (KME) que detém a chave mestra e é responsável por configurar um sistema de criptografia de chave pública e um esquema de assinatura padrão. A função a ser executada sobre os dados é interpretada em um enclave separado, o enclave de função (FE), sendo que cada função é executada em um FE separado. Para executar a decifragem, o usuário executa o enclave de decifragem (DE) na plataforma SGX, o qual irá obter a chave secreta do KME, através de um canal seguro obtido após o processo de atestação, para decifrar os dados.

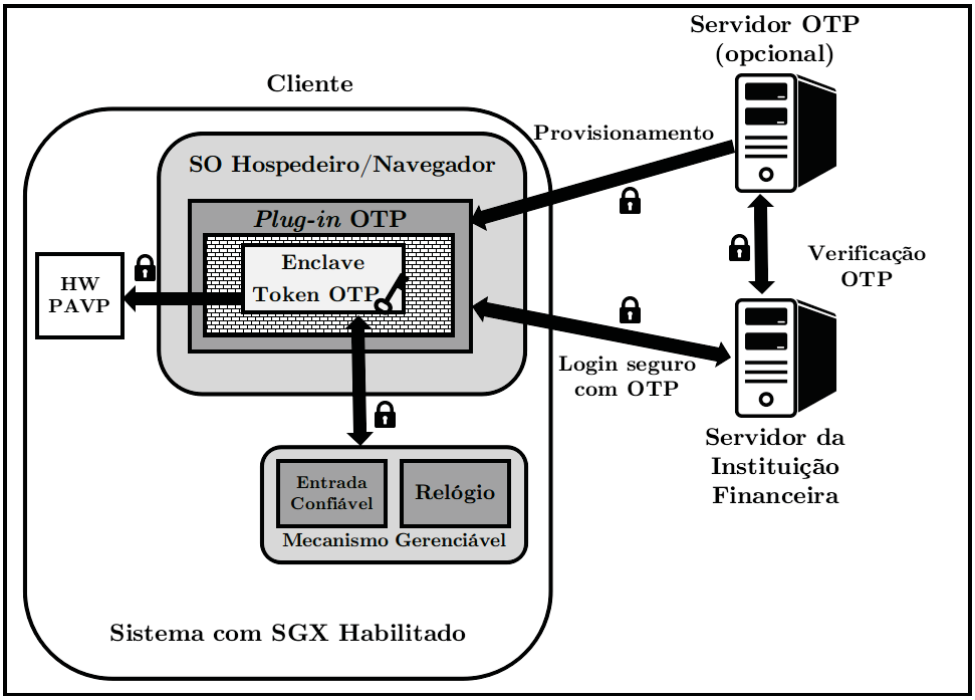


Figura 4.4: Execução de uma solução OTP com o uso de enclaves. Adaptado de Hoekstra *et al.* (2013).

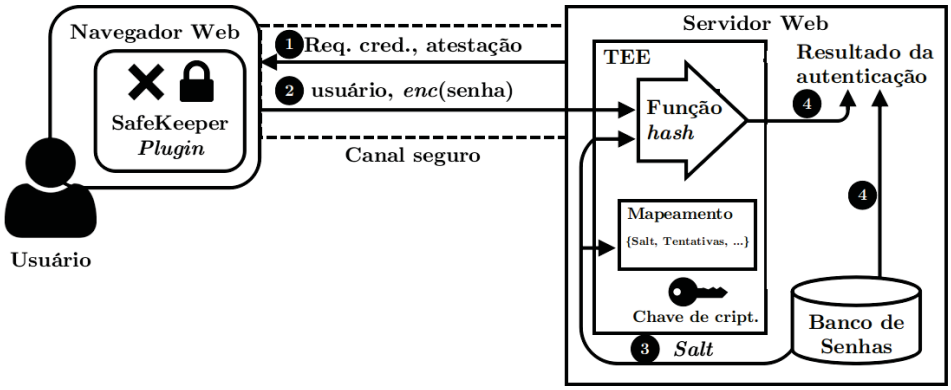


Figura 4.5: Estrutura da solução SafeKeeper. Adaptado de Krawiecka *et al.* (2018).

Já Li *et al.* (2018) propõem o uso do SGX em um sistema alternativo para substituir os sistemas de chave pública atuais, utilizando um par de enclaves para cifrar e decifrar o conteúdo. Um enclave atua como remetente, efetuando as operações análogas à chave privada, e outro atua como destinatário, com as operações análogas à chave pública. Assim, utilizando as premissas de segurança do SGX, os enclaves podem compartilhar uma chave secreta, mantendo tal chave nos limites de ambos os enclaves, e efetuar as operações de criptografia de maneira mais eficiente.

Outra questão abordada com o uso da arquitetura Intel SGX é a criptografia homomórfica, sendo tratada por Wang *et al.* (2019b). Os autores então propõem uma solução híbrida, utilizando enclaves SGX para a etapa de *bootstrapping*[1], que é um dos principais obstáculos no projeto de esquemas FHE práticos, incluindo um *thread* de serviço, que enfileira as requisições recebidas dos clientes, e um escalonador, que controla quando cada requisição será processada, fazendo um balanceamento de carga e evitando sobrecarga no processo de *bootstrapping*. Cada requisição é processada por um *thread*, executando dentro de um enclave, e esses *threads* são mantidos

[1]No contexto de criptografia homomórfica, *bootstrapping* é o processo de remover o ruído no conteúdo cifrado, que é adicionado após cada operação realizada.

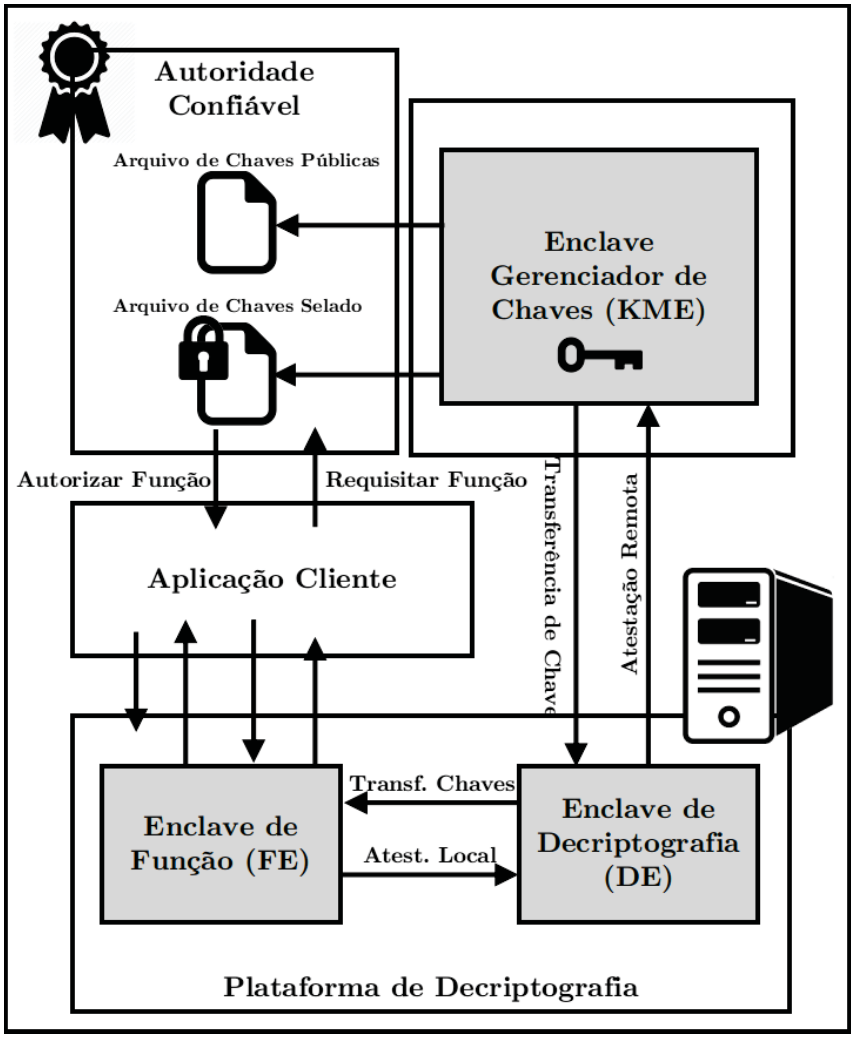


Figura 4.6: Arquitetura da solução de criptografia funcional com SGX. Adaptado de Fisch *et al.* (2017).

em um *pool*, podendo também ser formado um *pool* de enclaves para atender à solicitação. A arquitetura da solução é mostrada na Figura 4.7.

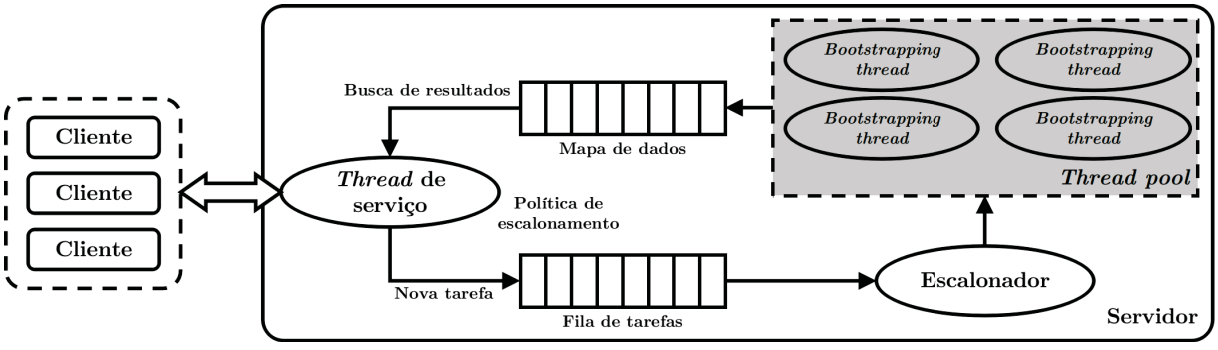


Figura 4.7: Arquitetura da solução de criptografia homomórfica *TEEFHE*. Adaptado de Wang *et al.* (2019b).

4.3 *RUNTIME*

Um campo de estudos na utilização de enclaves SGX se concentra em possibilitar que aplicações legadas utilizem as garantias de segurança providas pela arquitetura Intel SGX,

resultando em alternativas para a execução nativa dessas aplicações com pouca ou nenhuma modificação em seu código-fonte original.

Nesse contexto, Baumann *et al.* (2015) trazem a utilização da arquitetura SGX para encapsular aplicações legadas, sem a necessidade de realizar modificações no código-fonte dessas aplicações, através da arquitetura chamada *Haven*. A arquitetura proposta é apresentada na Figura 4.8, tendo um enclave criado dentro de um picoprocesso[2] *Drawbridge*[3], o qual contém toda a aplicação e o *Library OS*.

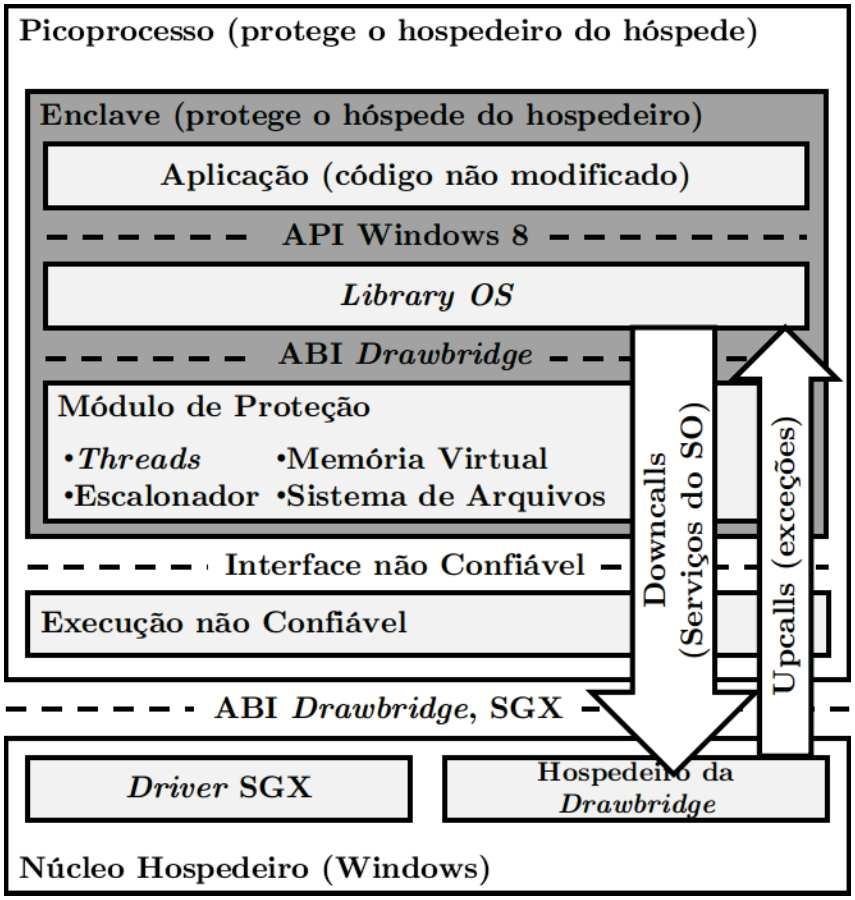


Figura 4.8: Arquitetura da solução Haven. Adaptado de Baumann *et al.* (2015).

Apesar da possibilidade de execução de serviços como o SQL Server e o Apache utilizando essa técnica, tem-se um considerável *overhead* de desempenho e um aumento do TCB, devido à migração das bibliotecas do C para dentro do enclave.

Tsai *et al.* (2017) utilizam a mesma abordagem, rebatendo conclusões de trabalhos anteriores que afirmam que essa abordagem acaba degradando o desempenho do sistema e aumenta o tamanho do TCB. Para comprovar suas afirmações, os autores portam o sistema operacional *Graphene*, descrito por Tsai *et al.* (2014), para a execução dentro de um enclave, oferecendo suporte à bibliotecas de carregamento dinâmico e multi-processos. O *Graphene-SGX* estende a solução proposta por Baumann *et al.* (2015), permitindo a execução de binários Linux

[2]Um picoprocesso é um contêiner de isolamento baseado em processo, com uma superfície mínima da API do sistema operacional. Ele é construído a partir de um espaço de endereçamento do processo do sistema operacional, mas com todos os serviços tradicionais do sistema operacional removidos.

[3]*Drawbridge* é um projeto de pesquisa da Microsoft Research para uma nova forma de virtualização de aplicações em uma *sandbox* através da utilização de picoprocessos e um sistema operacional baseado em uma biblioteca, sendo desenvolvido para funcionar de forma eficiente dentro de um picoprocesso.

sem qualquer modificação de código, permitindo o carregamento dinâmico de bibliotecas, bem como a ligação dinâmica dessas na aplicação.

Tian *et al.* (2017) também adotam a abordagem de *Library OS*, quantificando o impacto de desempenho dessa abordagem, e identificando as transições entre as aplicações SGX como um gargalo para todos os *Library OSes* que têm o intuito de oferecer suporte à execução de aplicações dentro de enclaves. Os autores então propõem o *SGXKernel*, um *Library OS* que é executado dentro de um enclave com um projeto que evita a necessidade de transição entre enclaves, implementando uma comunicação assíncrona e compartilhando estruturas de dados entre o enclave e a parte não confiável, além de um mecanismo *multi-threading* implementado totalmente dentro de um enclave. Assim, os autores obtiveram resultados de desempenho superiores aos apresentados por Tsai *et al.* (2017) com o *Graphene-SGX*. A arquitetura do *SGXKernel*, mostrada na Figura 4.9, é muito semelhante à arquitetura do *Haven*, proposta por Baumann *et al.* (2015).

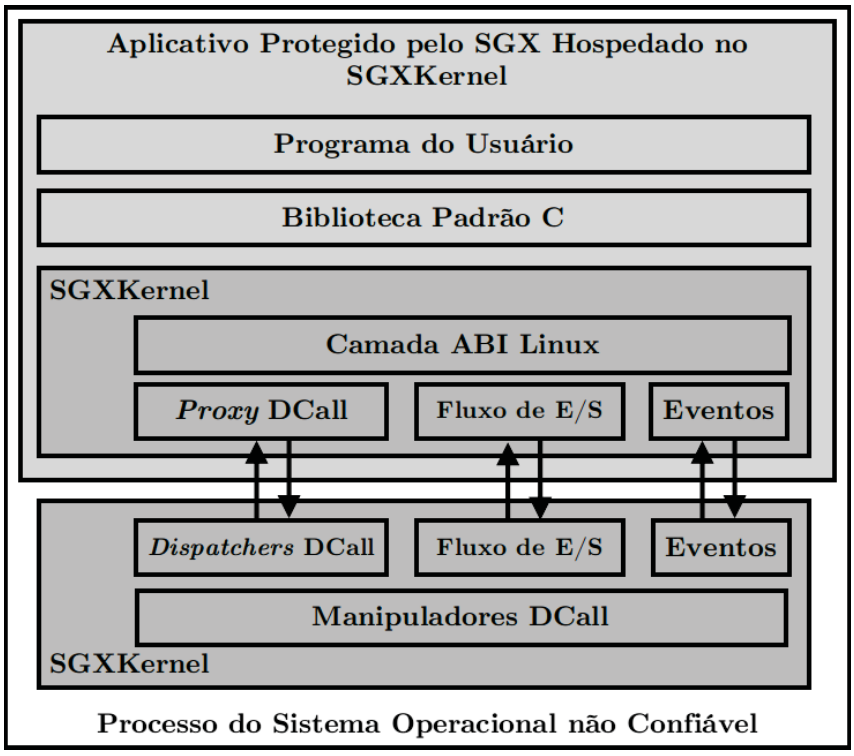


Figura 4.9: Arquitetura do sistema SGXKernel, que consiste em um componente confiável e um não confiável, executando em um enclave e no modo usuário, respectivamente. Adaptado de Tian *et al.* (2017).

Outra proposta é apresentada por Goltzsche *et al.* (2017), fazendo uso da arquitetura SGX para prover um ambiente confiável para a execução de códigos *JavaScript* em navegadores de Internet. A solução, chamada de *TrustedJS*, utiliza um enclave que é executado no mesmo contexto do processo do navegador, sendo adicionado ao mesmo através de um *plug-in*. Com isso, é oferecido um ambiente confiável para descarregar e executar códigos *JavaScript* no lado cliente, permitindo validações de dados diretamente no navegador, sem a necessidade de efetuar revalidações no lado servidor. A arquitetura da solução é apresenta na Figura 4.10.

4.4 SERVIÇOS DO SISTEMA OPERACIONAL

A arquitetura SGX também já é utilizada em soluções que visam aumentar a segurança em aplicações ou módulos de sistemas operacionais existentes. Nesse contexto encontra-se a solução proposta por Richter *et al.* (2016), que utiliza a arquitetura SGX para isolar módulos do

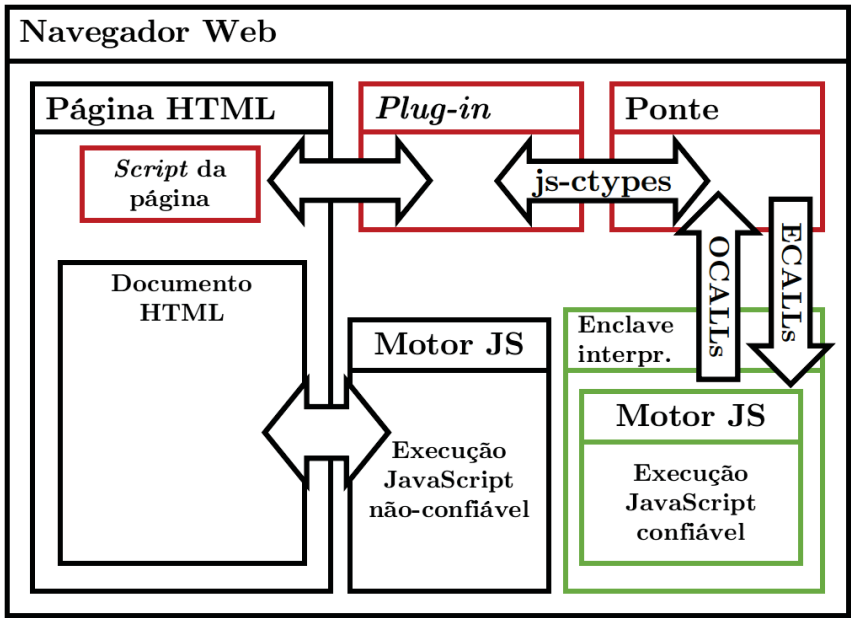


Figura 4.10: Arquitetura da solução *TrustedJS* no lado cliente. Adaptado de Goltzsche *et al.* (2017).

núcleo do Linux em enclaves, evitando que uma falha em um módulo do núcleo se propague ao restante do sistema. Para isso, partes do módulo são migradas para o espaço do usuário, visto que os enclaves não podem ser executados em modo núcleo. O enclave é implementado por um *daemon* no espaço do usuário que o instanciará, permitindo que o *daemon* acesse as funcionalidades do enclave, conforme mostrado na Figura 4.11.

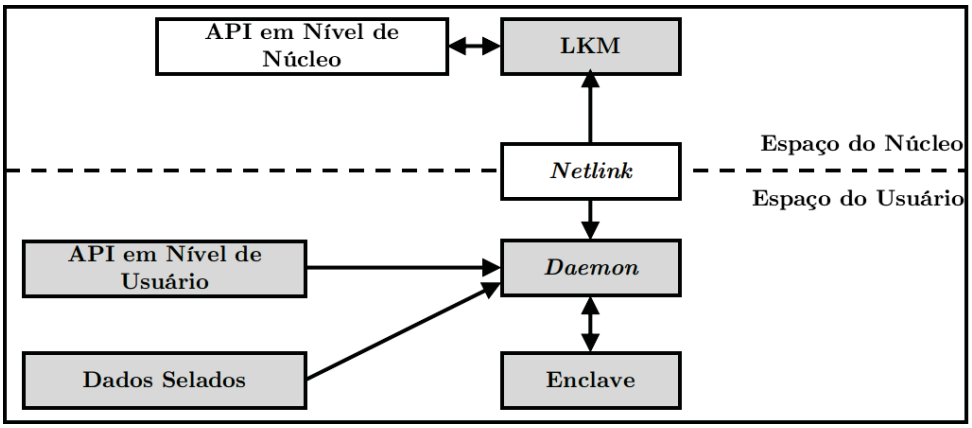


Figura 4.11: Um LKM utilizando um recurso provido por um enclave. O LKM se comunica com um *daemon* de modo de usuário que encaminha as solicitações para o enclave que reside no espaço do usuário. Adaptado de Richter *et al.* (2016).

Karande *et al.* (2017) utilizam uma arquitetura cliente/servidor, onde o cliente é um componente de solicitação de *log*, que emite várias mensagens de *log*, enquanto o servidor de *log* executa os serviços de *log* seguro. O servidor é dividido em dois componentes: a parte confiável que manipula os dados confidenciais, e a parte não confiável, como mostrado na Figura 4.12. Dessa forma, diversos clientes podem emitir requisições ao servidor de *log*, o qual, por sua vez, se utiliza de um enclave para efetuar a gravação dos registros.

Por fim, Condé *et al.* (2018) fazem uso do recurso de selagem provido pela arquitetura SGX para cifrar o conteúdo do arquivo de credenciais (`/etc/shadow`) utilizado pelo *framework* de autenticação PAM (*Pluggable Authentication Module*) em sistemas Linux, apresentando

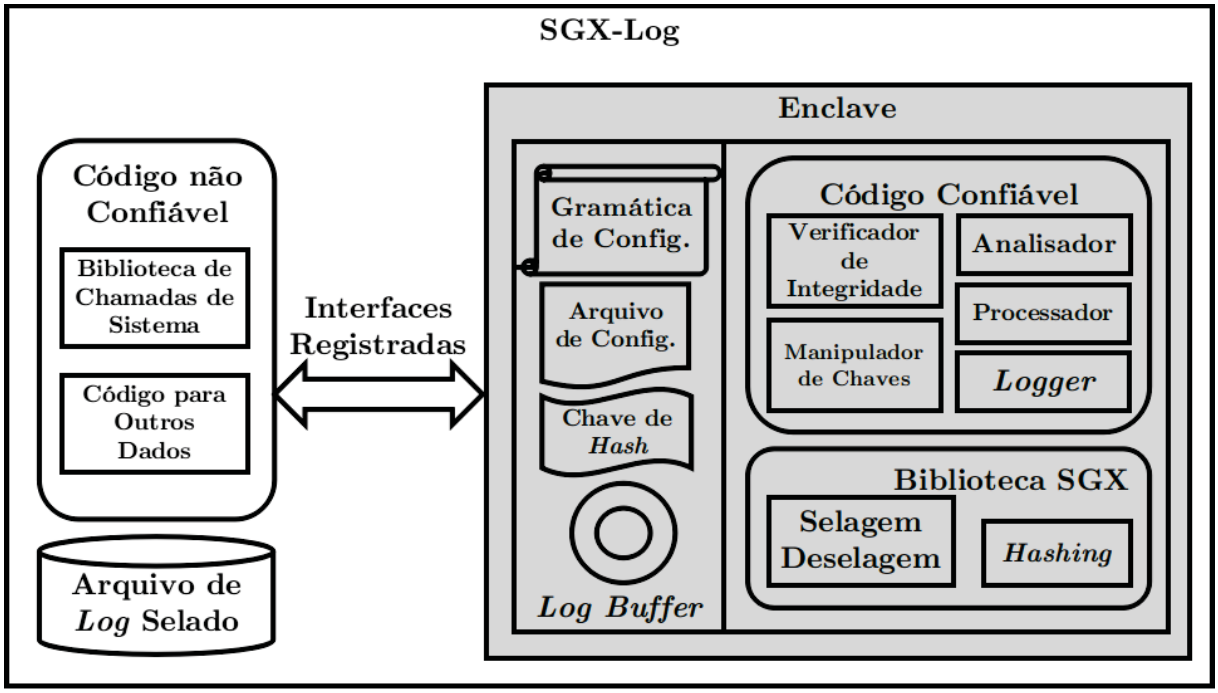


Figura 4.12: Arquitetura do servidor de *log* SGX. Adaptado de Karande *et al.* (2017).

a solução *UniSGX*. Os autores implementam uma modificação no módulo `pam_unix.so`, permitindo que esse módulo utilize o arquivo de credenciais selado, utilizando um enclave para efetuar a verificação das credenciais providas pelo usuário. A Figura 4.13 mostra o fluxo de controle do processo de autenticação com a utilização da solução *UniSGX*. Ao iniciar o processo de autenticação, a aplicação requisitante inicia a conversação com o PAM para informar as credenciais ou solicitar que o usuário as informe. Ao receber as credenciais, o PAM inicia o módulo *UniSGX*, o qual irá instanciar um enclave para receber as credenciais e validar com o arquivo `/etc/shadow` selado, indicando sucesso ou não na autenticação.

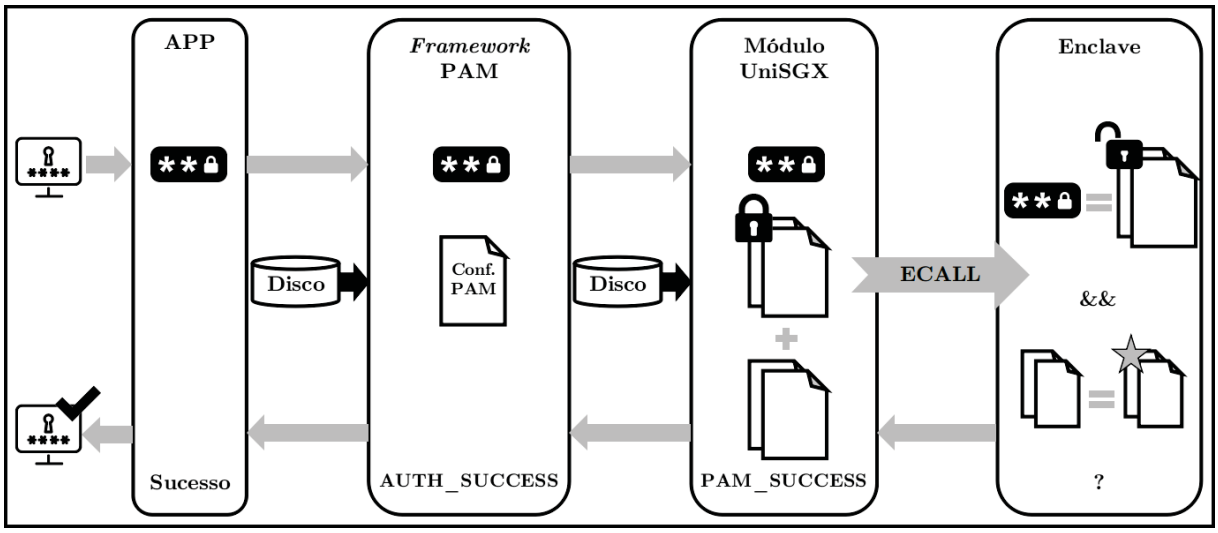


Figura 4.13: Fluxo de controle da autenticação utilizando o módulo UniSGX. Adaptado de Condé *et al.* (2018).

4.5 APLICAÇÕES DE REDE

Esta seção apresenta as aplicações da arquitetura Intel SGX situadas no tema de redes de computadores. Para tanto, os trabalhos são agrupados em duas subseções, sendo a primeira delas relacionada à infraestrutura de redes, com a segunda trazendo soluções de software de rede.

4.5.1 Infraestrutura

O SGX é utilizado para garantir a privacidade em IXPs (*Internet eXchange Points*), os quais servem como pontos de referência onde muitos provedores de serviços de rede se encontram para obter conectividade recíproca. Na solução proposta por Chiesa *et al.* (2017), o BGP (*Border Gateway Protocol*) é executado dentro de um enclave, garantindo a segurança das políticas de roteamento, permitindo acesso a essas informações somente ao membro do IXP que configurou a política e permitindo que o servidor de rotas aplique as políticas solicitadas e cada membro do IXP para verificar se essas políticas foram implementadas corretamente. As requisições são sempre solicitadas através de um canal seguro de comunicação, estabelecido pelo processo de atestação.

Aublin *et al.* (2018) apresentam uma biblioteca para auditoria de serviços de Internet, que cria um registro de auditoria com não-repúdio das operações, além de verificações para descobrir violações de integridade do serviço. A solução, chamada pelos autores de *LibSEAL* (*SEcure Audit Library*), foi projetada como uma substituta ao TLS e, para tanto, observa e registra todas as solicitações e respostas de serviço, como mostrado na Figura 4.14. Os registros de *log* são armazenados em um banco de dados relacional, permitindo que qualquer violação seja descoberta utilizando-se de consultas SQL. Os autores se utilizam da arquitetura Intel SGX para efetuar as operações de cifragem e decifragem dos dados transmitidos e garantir a confidencialidade e integridade de tais dados. Além disso, periodicamente, um cliente pode invocar o analisador de *log* para executar verificações invariantes específicas do serviço.

4.5.2 *Network Function Virtualization*

As aplicações NFV (*Network Function Virtualization*) são baseadas em estados como, por exemplo, o *Content Distribution Network* (CDN) que armazena em *cache* conteúdos de servidores remotos na Web e distribui a seus clientes quando necessário. Vulnerabilidades nestas aplicações podem permitir que invasores roubem e manipulem os estados internos destas aplicações, fornecendo dados errôneos aos clientes.

Com o intuito de garantir a segurança do estado de aplicações NFV, Shih *et al.* (2016) fazem uso da arquitetura Intel SGX para a construção da solução *S-NFV*. Na arquitetura proposta pelos autores, a aplicação NFV original é dividida em duas partes: o enclave S-NFV e o *host* S-NFV, mostradas na Figura 4.15. O enclave S-NFV contém os estados e o código de processamento de estado, sendo separado da aplicação NFV original e colocado dentro de um enclave SGX. O enclave S-NFV também não deve depender da memória fora do enclave, para garantir o isolamento do ambiente em que está sendo executado. Já o *host* S-NFV contém o restante do código de processamento.

Também nesse sentido, Coughlin *et al.* (2017) estendem o roteador modular *Click* para permitir o processamento dos pacotes dentro de um enclave SGX. No modelo apresentado pelos autores, as aplicações NFV são movidas para uma nuvem, e os dados são trafegados através de um túnel. Para a solução, o *gateway* no cliente cria um canal seguro de comunicação, através da atestação remota, para enviar a chave de criptografia ao enclave na nuvem, necessária para decifrar os dados que serão enviados na sequência. Em posse da chave e dos dados, o enclave contido no

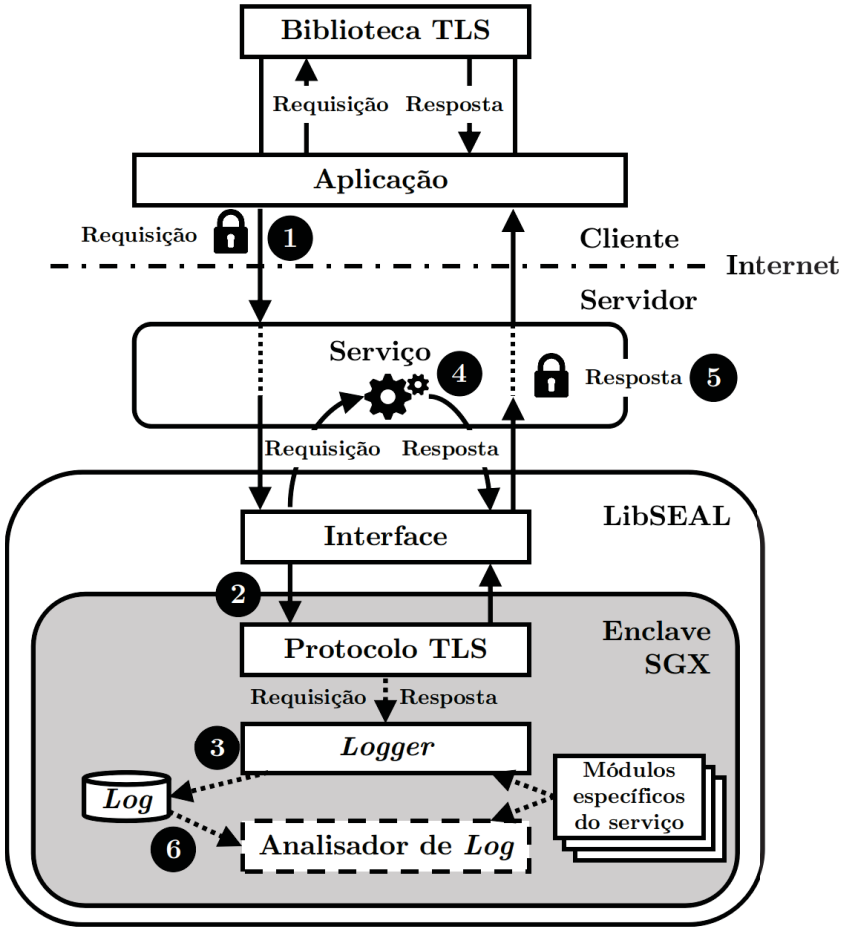


Figura 4.14: Arquitetura da solução *LibSEAL*. (1) cliente efetua a requisição; (2) serviço envia a requisição para a *LibSEAL* para decifragem; (3) requisição decifrada é enviada ao *logger*; (4) requisição é processada; (5) resposta é enviada para a *LibSEAL* para cifragem e encaminhamento ao cliente; (6) verificação do *log*. Adaptado de Aublin *et al.* (2018).

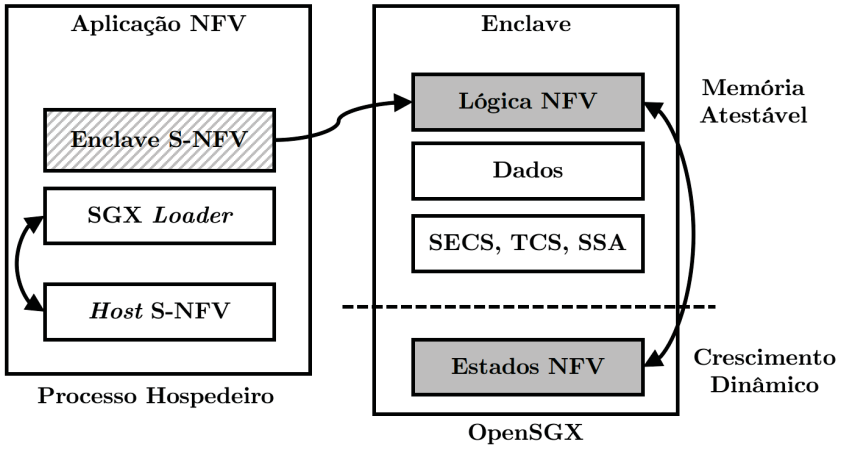


Figura 4.15: Arquitetura de aplicações NFV utilizando SGX. Adaptado de Shih *et al.* (2016).

módulo *Click* pode então decifrar e processar tais dados, garantindo a confidencialidade de tais dados mesmo durante o processamento. A arquitetura da solução, nomeada de *TrustedClick*, é apresentada na Figura 4.16.

De forma semelhante, Trach *et al.* (2018) fazem uso da plataforma *SCONE* para garantir a segurança na execução de funções de rede. Todos os elementos não utilizados no contexto da

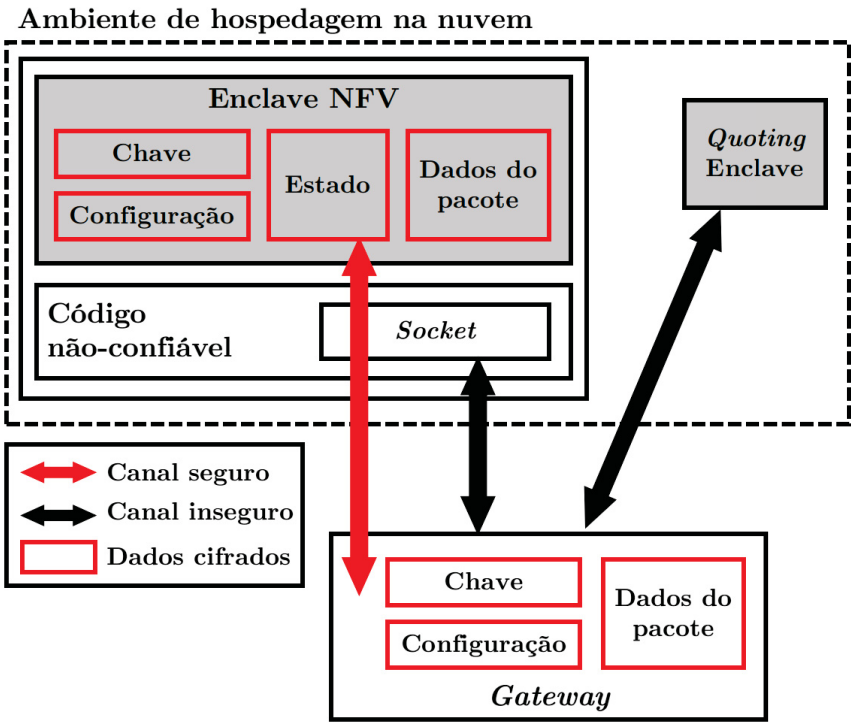


Figura 4.16: Arquitetura da solução *TrustedClick*. Adaptado de Coughlin *et al.* (2017).

aplicação foram removidos do *Click*, com o intuito de reduzir o *footprint* e, por consequência, a superfície de ataque. Isso também permitiu portar todo o *Click* para dentro de um enclave, sendo executado através do *SCONE*, como demonstrado na Figura 4.17.

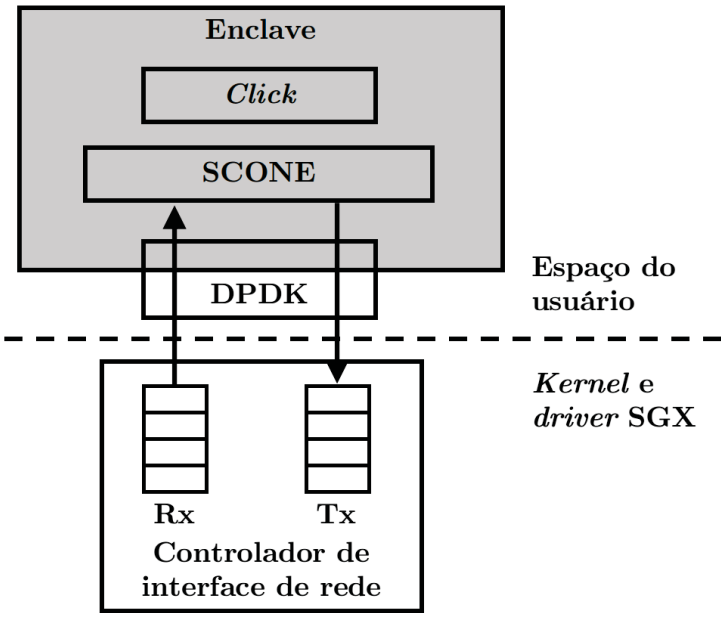


Figura 4.17: Arquitetura básica da solução ShieldBox. Adaptado de Trach *et al.* (2018).

Por fim, nesse mesmo contexto, Duan *et al.* (2019) apresentam a solução *LightBox*, que garante proteção total não somente aos dados do pacote, mas também aos seus metadados, como o cabeçalho, tamanho, saltos, etc. Tal preocupação se deve ao fato de que os metadados também podem conter informações valiosas para um atacante, e podem ser explorados. A Figura 4.18 apresenta a arquitetura do *LightBox*, que é constituído de dois componentes principais:

dispositivo de rede virtual (*etap*) e; gerenciador de estados. O primeiro permite que as funções de rede executem o empacotamento sem a necessidade de deixar os limites do enclave, já o segundo permite que as funções de rede executem o processamento dos pacotes, também dentro dos limites do enclave, de maneira eficiente.

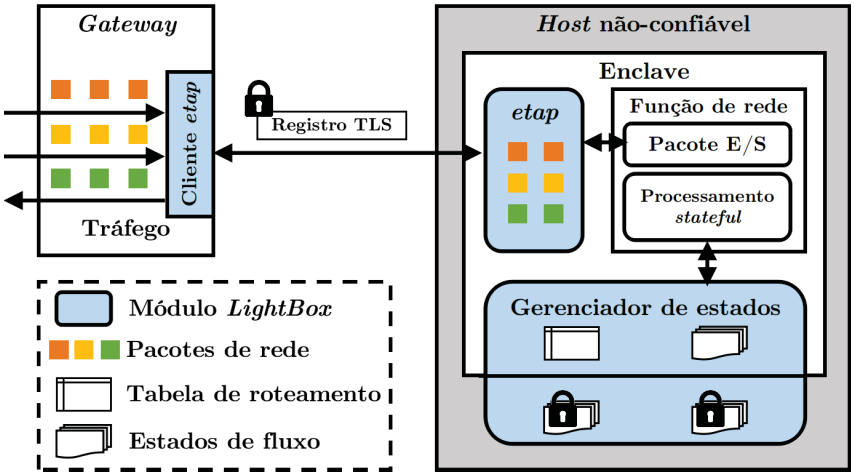


Figura 4.18: Visão geral dos principais componentes da solução LightBox. Adaptado de Duan *et al.* (2019).

4.6 APLICAÇÕES DISTRIBUÍDAS

Aplicações distribuídas são executadas em vários computadores, cooperando e compartilhando dados pela rede. A confidencialidade e integridade dos dados compartilhados é uma preocupação central de tais sistemas. Assim, diversas soluções propõem a utilização da arquitetura Intel SGX para garantir premissas de segurança no compartilhamento e processamento de dados em sistemas distribuídos

4.6.1 *Peer-to-Peer*

No contexto de redes *peer-to-peer*, ou P2P, Jia *et al.* (2017) avaliam a preocupação com ataques bizantinos, pontuando que uma contramedida amplamente estudada é o modelo de omissão geral, que produz protocolos simples com boa eficiência, mas é considerado impraticável ou irrealizável, uma vez que limita artificialmente o adversário apenas a omitir mensagens. Assim, os autores propõe a utilização de enclaves SGX, com cada nodo possuindo um enclave para a execução segura das aplicações e com a troca de mensagens entre nodos sendo efetuada com o uso de atestação. Tal solução, segundo os autores, reduz o ataque bizantino ao modelo de omissão geral, eliminando qualquer fonte de vantagem para um adversário bizantino além daquela obtida pela omissão de mensagens, tornando assim, o modelo de omissão geral realizável.

4.6.2 *Internet of Things*

A utilização do SGX também abrange dispositivos IoT, com Nguyen *et al.* (2016) apresentando uma solução para armazenar os registros de *log* de dispositivos médicos e de informações de pacientes em um ambiente de nuvem, de maneira segura, garantindo a confidencialidade, integridade e detecção de violação. A solução proposta pelos autores, apresentada na Figura 4.19, consiste de um *dongle* ligado ao dispositivo médico que envia os registros aos servidores de armazenamento, utilizando-se de uma conexão segura provida pelo

SGX e por um TPM. Os dados recebidos são cifrados dentro de um enclave e armazenados na memória secundária. Um *hash* do conteúdo também é gerado, assinado e armazenado no TPM, construindo uma cadeia de *hashes* que pode ser utilizada para detectar violações no conteúdo armazenado.

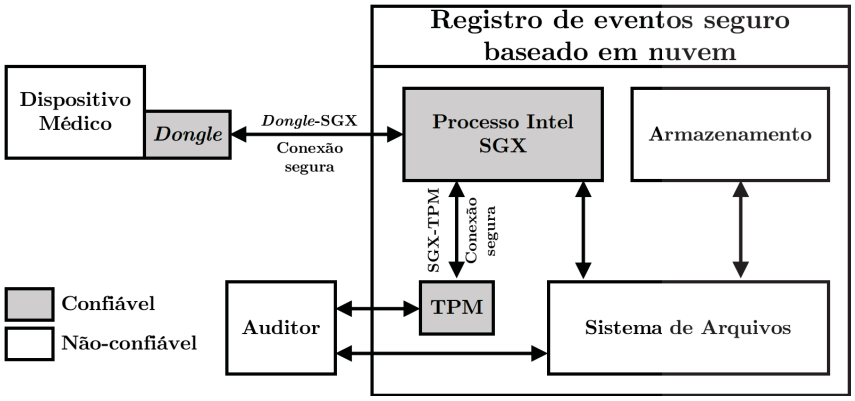


Figura 4.19: Arquitetura da solução de *log* seguro para dispositivos médicos. Adaptado de Nguyen *et al.* (2016).

4.6.3 *Blockchain*

A arquitetura SGX também é aplicada à execução de contratos inteligentes. Contratos inteligentes são programas que são executados de forma autônoma em *blockchains*, e comumente é exigido que estes consumam dados de fora do *blockchain*. Nesse contexto, é extremamente necessário a utilização de *feeds* de dados confiáveis, para evitar a manipulação de contratos inteligentes com dados errôneos. Assim, Zhang *et al.* (2016) apresentam um sistema de *feed* de dados autenticado, chamado *Town Crier*, que tem por objetivo atuar como uma ponte entre os contratos inteligentes e provedores de dados, acessando tais dados na Web, via HTTPS, e autenticando-os. A solução também suporta confidencialidade, permitindo solicitações com parâmetros criptografados e uso seguro de credenciais para capturar dados protegidos por controles de acesso, com a execução em um ambiente confiável.

Já Milutinovic *et al.* (2016) abordam a utilização de um ambiente de execução confiável nas primitivas de consenso de *blockchain*, aplicando tal ambiente a esquemas de prova de trabalho, prova de tempo e prova de propriedade, para tornar o processo de mineração equitativo. Além disso, os autores propõem um esquema de prova de sorte, que se utiliza da geração de números aleatórios em um enclave, onde o maior número é eleito, obtendo um baixo consumo energético e uma mineração equitativa.

No subconjunto da criptomoeda, o SGX é aplicado em um misturador de *bitcoin*, o que ajuda a desassociar qualquer *bitcoin* da identidade do proprietário, misturando *bitcoins* de vários usuários e garantindo uma maior privacidade do usuário. Na proposta apresentada por Tran *et al.* (2018), chamada de *Obscuro*, o SGX é usado para garantir que as operações de mistura sejam efetuadas de maneira correta, proteger dados confidenciais do usuário e para impedir que os depósitos sejam misturados mais de uma vez. Ainda no contexto de *bitcoin*, outro campo de pesquisa são as redes de pagamento, que superam a necessidade de consenso global por meio de transações fora da cadeia, mas permitem que uma parte mal-intencionada roube fundos da *blockchain*. Assim, na solução *Teechain*, descrita por Lind *et al.* (2019), o SGX é usado para criar tesourarias, que estabelecem canais de pagamento fora da cadeia entre as partes e mantém fundos colaterais para trocar transações sem interagir com o *blockchain* subjacente.

4.7 APLICAÇÕES EM NUVEM

O ambiente de nuvem é abordado de forma ampla no desenvolvimento de soluções com a arquitetura Intel SGX. Nuvens de terceiros podem colocar em risco a integridade e a confidencialidade de dados de usuários não criptografados, porque os administradores de nuvem têm acesso total a eles; o mesmo se aplica a nuvens privadas, nas quais dados confidenciais podem ser expostos a administradores de sistemas de infraestrutura não confiáveis. Além disso, as vulnerabilidades em ambientes de execução da nuvem podem expor dados confidenciais a terceiros.

4.7.1 Infraestrutura

Os mecanismos de confidencialidade e integridade fornecidos pela arquitetura Intel SGX também podem ser usados em infraestrutura de nuvem, como em sistemas de valor-chave. Fuhry *et al.* (2017) apresentam um índice de banco de dados criptografado e de alto desempenho, chamado de *HardIDX*, onde a chave e o valor correspondente são armazenados de maneira cifrada, utilizando chaves de criptografia distintas. Para efetuar consultas, o cliente deve primeiro realizar o processo de atestação com o enclave no servidor, a fim de criar um canal de comunicação seguro para o envio da chave de criptografia utilizada na cifragem das chaves do conjunto chave-valor. A consulta é realizada dentro dos limite do enclave, no servidor, utilizando o conjunto de chaves armazenado, sendo que o conjunto de valores é retornado ao cliente ainda cifrados, com o cliente tendo que decifrar tais dados utilizando a sua chave de criptografia adequada. Todo o processo é mostrado na Figura 4.20

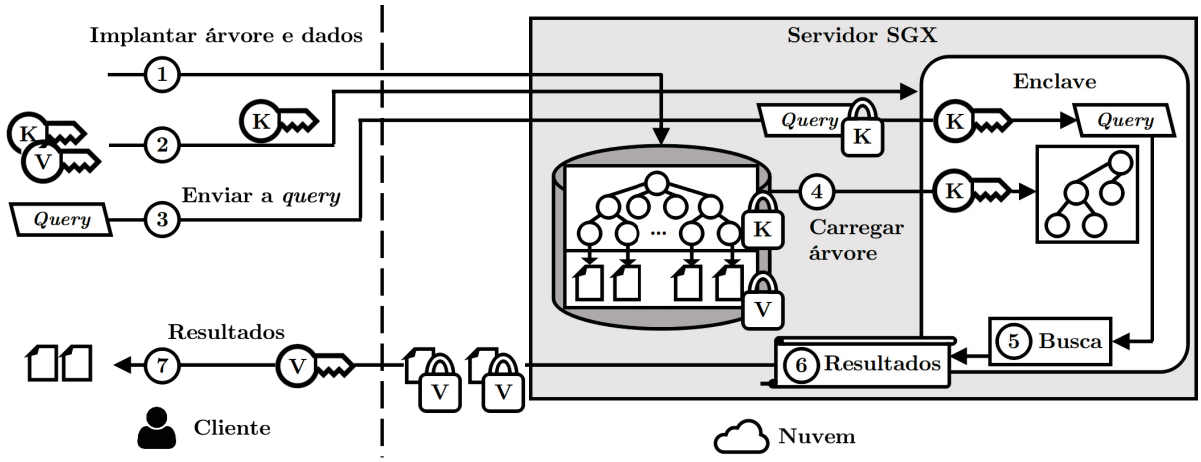


Figura 4.20: Arquitetura da solução *HardIDX*. Adaptado de Fuhry *et al.* (2017).

Nesse contexto, Sun *et al.* (2018) apresentam a solução *REAGUARD*, que permite efetuar consultas a documentos criptografados em servidores de terceiros, garantindo a privacidade da consulta. Os autores apresentam um cenário onde um cliente possui *n* documentos relacionados a *m* palavras-chave, que são armazenados na nuvem. De forma semelhante à proposta de Fuhry *et al.* (2017), os autores propõem que os clientes atestem o servidor, de forma a criar um canal de comunicação seguro entre o cliente e o enclave no servidor, para que seja enviada a chave de criptografia ao enclave, de forma que esse possa decifrar o conteúdo em um ambiente confiável, para executar as consultas solicitadas e enviar o resultado cifrado para o cliente.

Sinha e Christodorescu (2018) apresentam a solução *VeritasDB*, que consiste em um *proxy* que faz a mediação da comunicação entre os clientes e o servidor de banco de dados *NoSQL*. O código e o estado do *proxy* são protegidos através de enclaves e, ao mediar todas as

interações, o *proxy* pode executar a contabilidade necessária para rastrear versões de objetos gravados para cada chave. A transmissão de dados entre os clientes e o *proxy* é efetuada através de um canal seguro, o que garante também a integridade das requisições.

Há ainda a preocupação em armazenar um conjunto de pares do tipo chave-valor em memória de maneira segura, com Kim *et al.* (2019) pontuando que a arquitetura Intel SGX pode auxiliar nessa questão, desde que seja contornada a limitação de uso da PRM. Assim, os autores propõem um novo mecanismo para o armazenamento de conjuntos chave-valor em memória, projetado especificamente para se utilizar dos recursos providos pelo SGX, chamando-o de *ShieldStore*. A solução proposta mantém as principais estruturas de dados na memória desprotegida, com cada par chave-valor sendo criptografado individualmente e tendo sua integridade garantida pelo enclave. Uma visão simplificada da arquitetura da solução é apresentada na Figura 4.21.

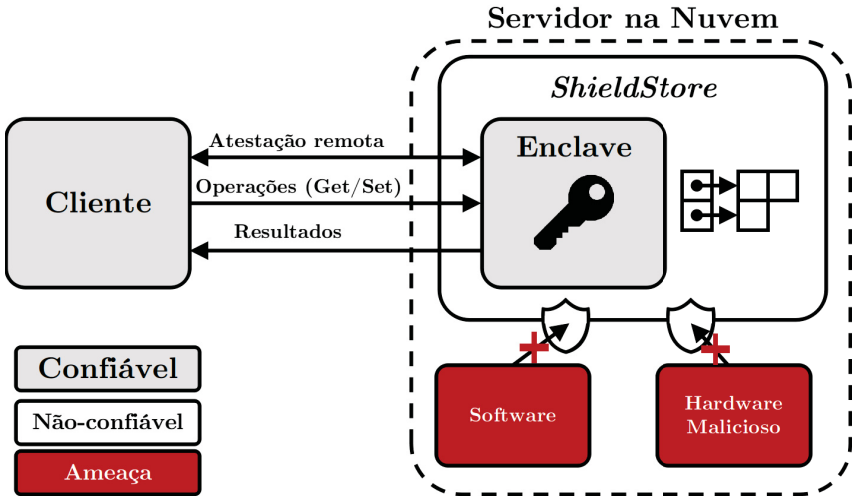


Figura 4.21: Arquitetura simplificada da solução *ShieldStore*. Adaptado de Kim *et al.* (2019).

Bailleu *et al.* (2019) apresentam a solução *SPEICHER*, que garante a integridade e confidencialidade dos dados em estruturas LSM utilizando o SCONE. O SPEICHER estende a parte confiável para além do enclave, para garantir que as propriedades de segurança também sejam preservadas em configurações *stateful* de um meio de armazenamento não confiável, podendo se recuperar de falhas no sistema, reinicialização ou migração.

Por fim, no contexto de bancos de dados, Priebe *et al.* (2018) apresentam a solução *EnclaveDB*, que garante confidencialidade, integridade e atualidade em dados e consultas, mesmo quando o administrador do banco de dados é malicioso ou o sistema operacional ou hipervisor está comprometido. Todos os dados sensíveis, como tabelas, índices e outros metadados, são manipulados dentro de um enclave. As consultas a serem realizadas na base de dados são pré-compiladas pelo cliente, sendo assinadas, cifradas e então enviadas ao servidor que hospeda a base de dados. O servidor valida a autenticidade da requisição, decifra seu conteúdo, executa a consulta e retorna os resultados ao cliente, também cifrados. A arquitetura geral da solução é apresentada na Figura 4.22.

4.7.2 Virtualização

O uso de SGX também é estendido para virtualização na nuvem, permitindo que o *Docker* suporte a execução segura fornecida por SGX, conforme apresentado por Arnautov *et al.* (2016). Essa abordagem é importante, visto que contêineres oferecem um alto desempenho em

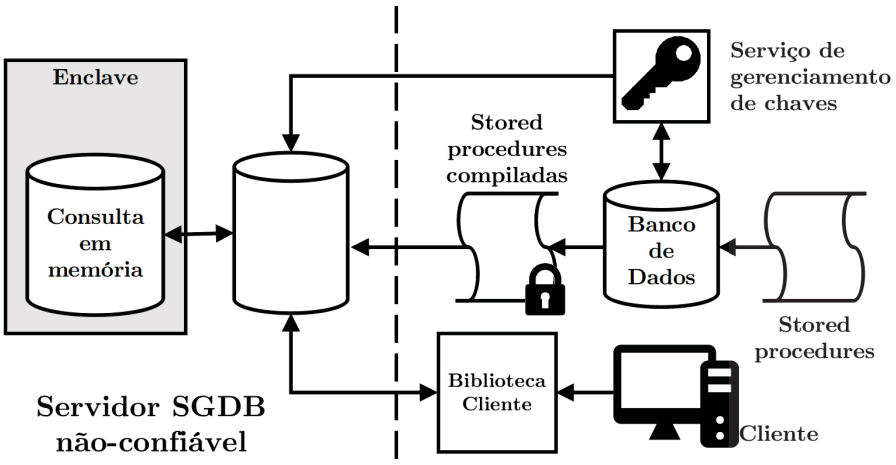


Figura 4.22: Arquitetura da solução *EnclaveDB*. Adaptado de Priebe *et al.* (2018).

operações de entrada e saída e uma rápida inicialização, quando comparados com máquinas virtuais, mas pecam ao oferecer garantias de isolamento, por compartilhar o mesmo núcleo, permitindo que atacantes possam facilmente comprometer a integridade e confidencialidade dos dados contidos dentro desses contêineres.

A solução proposta pelos autores expõe uma interface externa baseada em chamadas de sistema para o sistema operacional hospedeiro, e executa verificações de integridade constantes para se proteger de ataques. Também evita o custo desnecessário de transições entre enclaves com *M* enclaves sendo multiplexados em *N threads* do sistema operacional. Assim, quando um *thread* emite uma chamada de sistema, o *SCONE* verifica se há outro *thread* que pode ser ativado e executado até que o resultado da chamada do sistema esteja disponível. As chamadas de sistema também são executadas de forma assíncrona, utilizando uma memória compartilhada para passar argumentos para a chamada de sistema e receber os valores de retorno, evitando a operação de sair do enclave para executar tais chamadas, reduzido assim o impacto no desempenho causado pela transição entre o enclave a aplicação. A arquitetura do SCONE é apresentada na Figura 4.23.

Nesse mesmo contexto, Shinde *et al.* (2017) apresentam a solução *Panoply*, que fornece uma nova abstração, chamada pelos autores de *micro contêiner*, ou *micron*, disponibilizando às aplicações abstrações no padrão POSIX, para acesso ao sistema de arquivos, redes, multi-processamento e sincronização de *threads*. Os autores também destacam a possibilidade de comunicação entre os enclaves de maneira confiável mesmo quando o sistema operacional apresente um comportamento anormal, o que permite a divisão de uma aplicação em vários *microns*. A arquitetura da solução Panoply é mostrada na Figura 4.24, onde todas as partes inseridas dentro do enclave são consideradas confiáveis, enquanto as partes destacadas em preto são consideradas como não confiáveis. Além disso, as partes destacadas em cinza foram adicionadas ou modificadas para a construção da solução proposta.

Também existe a questão dos TPMs virtuais, ou vTPM, que são largamente utilizados para prover um *root-of-trust* virtual para as máquinas virtuais de um hipervisor. Mas, de acordo com Wang *et al.* (2019a), as atuais implementações de vTPMs não fornecem proteção completa contra o vazamento de dados, além de ter um alto impacto no desempenho de execução. Nesse sentido, os autores apresentam a solução *SvTPM*, que utiliza a arquitetura Intel SGX na implementação de um vTPM, fornecendo um forte isolamento no armazenamento e manipulação de chaves e dados sensíveis. A Figura 4.25 apresenta a arquitetura geral da solução, onde cada máquina virtual tem o seu vTPM correspondente, o qual está contido dentro de um enclave SGX.

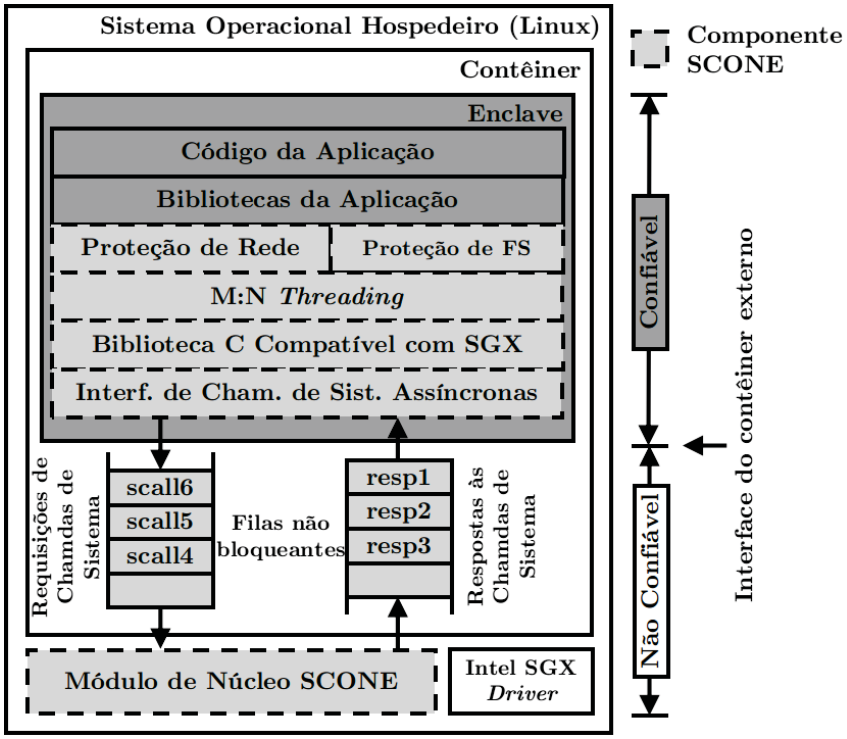


Figura 4.23: Arquitetura do SCONE. Adaptado de Arnautov *et al.* (2016).

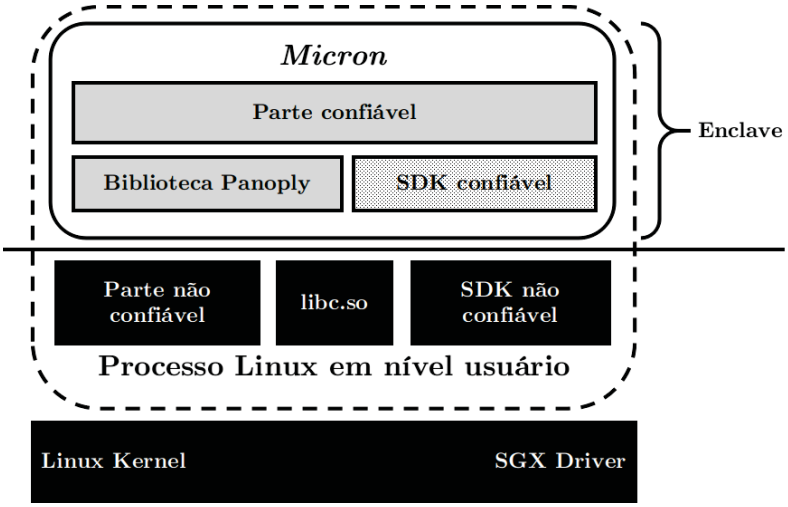


Figura 4.24: Arquitetura do sistema Panoply, que fornece às aplicações abstrações seguindo o padrão POSIX através de micro contêineres. Adaptado de Shinde *et al.* (2017).

Além disso, é fornecido uma biblioteca (*SvTPM 2.0*) para que as aplicações possam acessar os recursos providos pelo vTPM.

4.7.3 Gerenciamento

Outra preocupação em ambientes de nuvem é um atacante explorar vulnerabilidades no escalonador do hipervisor, de modo a tomar o tempo de CPU destinado a outras máquinas virtuais para si. Para mitigar esse tipo de problema, Leach *et al.* (2017) apresentam a solução *Scotch*, que é um sistema de contabilização transparente do consumo de recursos em um hipervisor, garantindo a integridade dos dados coletados mesmo quando o hipervisor está comprometido. A arquitetura da solução é apresentada na Figura 4.26, onde um sistema protegido executa um monitor de

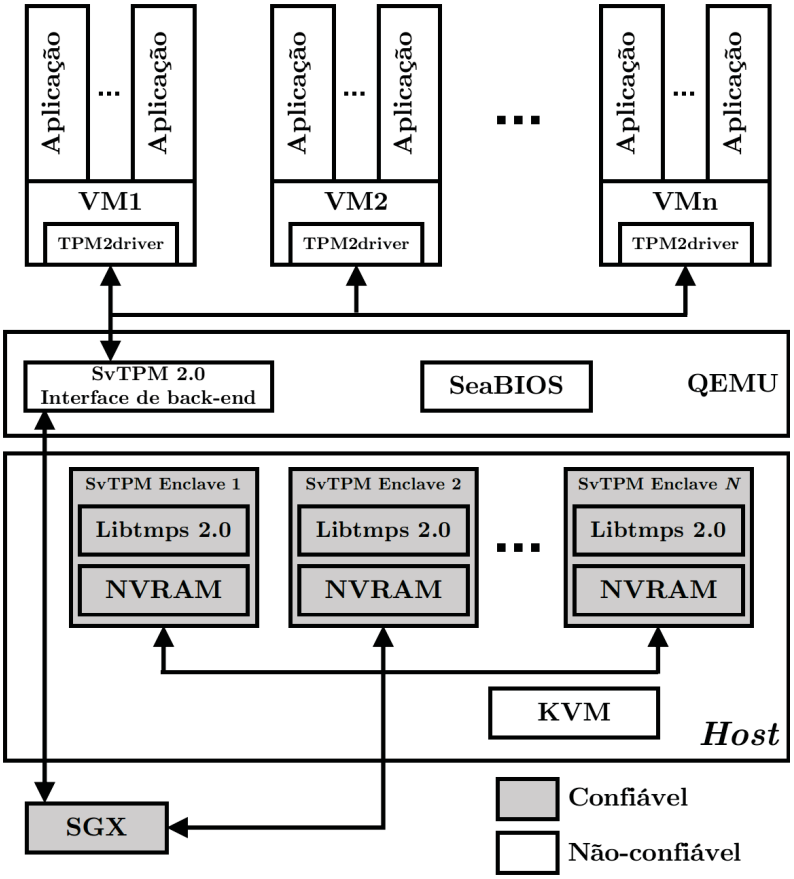


Figura 4.25: Arquitetura geral da solução *SvTPM*. Adaptado de Wang *et al.* (2019a).

máquinas virtuais que pode hospedar uma ou mais máquinas virtuais, potencialmente infectadas. O sistema irá coletar as informações de consumo de recursos periodicamente (a cada interrupção ou troca de contexto) e reportar essas informações a um enclave SGX, o qual irá armazenar essas informações para posterior processamento, que também é realizado de maneira protegida, evitando a intervenção do sistema operacional ou do hipervisor, potencialmente comprometidos.

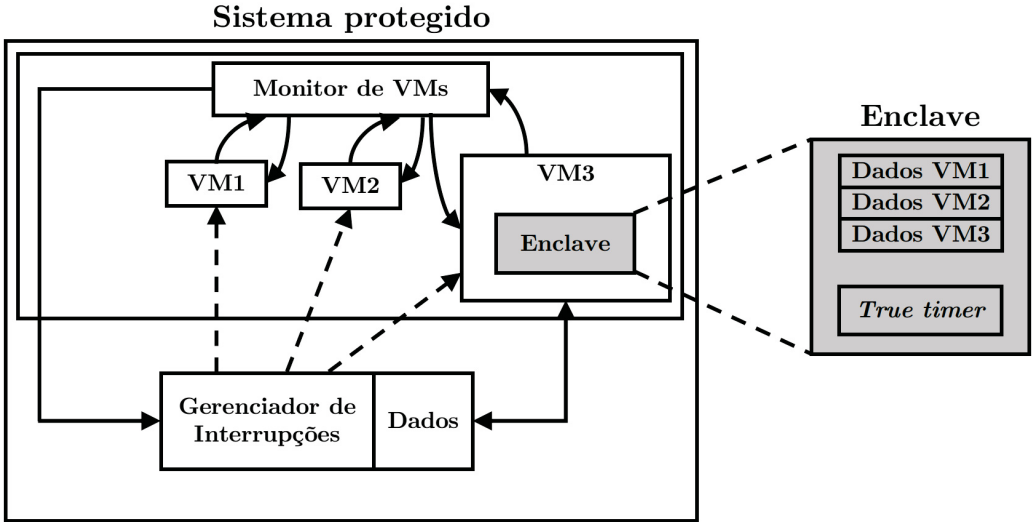


Figura 4.26: Arquitetura geral da solução *Scotch*. Adaptado de Leach *et al.* (2017).

O SGX também é utilizado para prover mecanismos de segurança em serviços de coordenação na nuvem, com Brenner *et al.* (2016) utilizando enclaves SGX para preservar a confidencialidade e a integridade dos dados gerenciados pelo serviço *ZooKeeper*, criando a solução *SecureKeeper*. Na arquitetura proposta pelos autores, apresentada na Figura 4.27, são utilizados vários pequenos enclaves para garantir que os dados fornecidos pelo usuário no *ZooKeeper* sejam sempre mantidos criptografados enquanto estiverem fora do enclave e que as operações sobre tais dados sejam executadas de maneira segura, dentro de um enclave SGX. Cada cliente tem um enclave individual associado a ele na réplica onde está conectado, criando um canal de comunicação seguro entre o cliente e a réplica através de TLS. Cada réplica também dispõe de um *enclave contador*, que se mantém ativo apenas na réplica lider e é utilizado na eleição de um novo líder em caso de falha do atual.

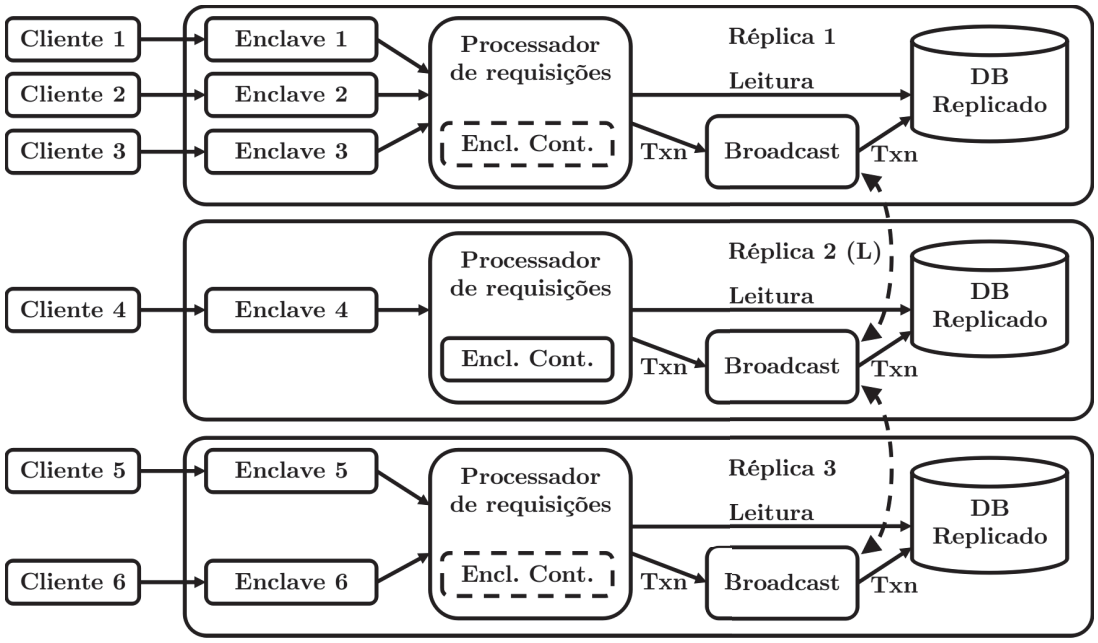


Figura 4.27: Arquitetura geral da solução *SecureKeeper*. Adaptado de Brenner *et al.* (2016).

4.7.4 Microsserviços

A arquitetura SGX também é aplicada na execução de microsserviços na nuvem. Fetzer (2016) apresenta uma arquitetura que se utiliza dos conceitos de microsserviços para minimizar o TCB de tais serviços, apresentada na Figura 4.28. Cada microsserviço é executado dentro de um enclave SGX que está contido em um contêiner seguro. A execução segura é garantida utilizando a solução SCONE, apresentada na Seção 4.7.2.

Brenner *et al.* (2017) propõem uma alteração do *Eclipse Vert.x* para permitir que microsserviços seguros sejam executados juntamente com os serviços regulares, sendo interconectados através do barramento de eventos do *Vert.x*, o que possibilita a construção de grandes aplicações que podem conter componentes confiáveis. No *Vert.x*, cada microsserviço é chamado de *vértice*, e deve incluir uma pequena, e bem definida, parte da aplicação. A Figura 4.29 ilustra a conexão entre um vértice seguro e um vértice regular, utilizando o barramento de eventos do *Vert.x*. Assim, um vértice seguro é um vértice com um enclave integrado, que pode ser acessado por outros vértices através do barramento de eventos. Essa estratégia permite que os desenvolvedores criem suas aplicações com um conjunto de serviços seguros e regulares.

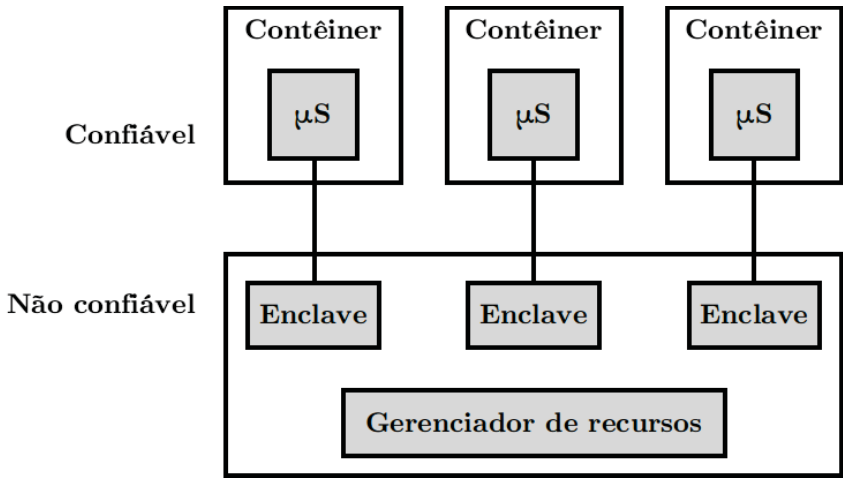


Figura 4.28: Uma abordagem baseada em microsserviços para reduzir o estado dentro de enclaves. Adaptado de Fetzer (2016).

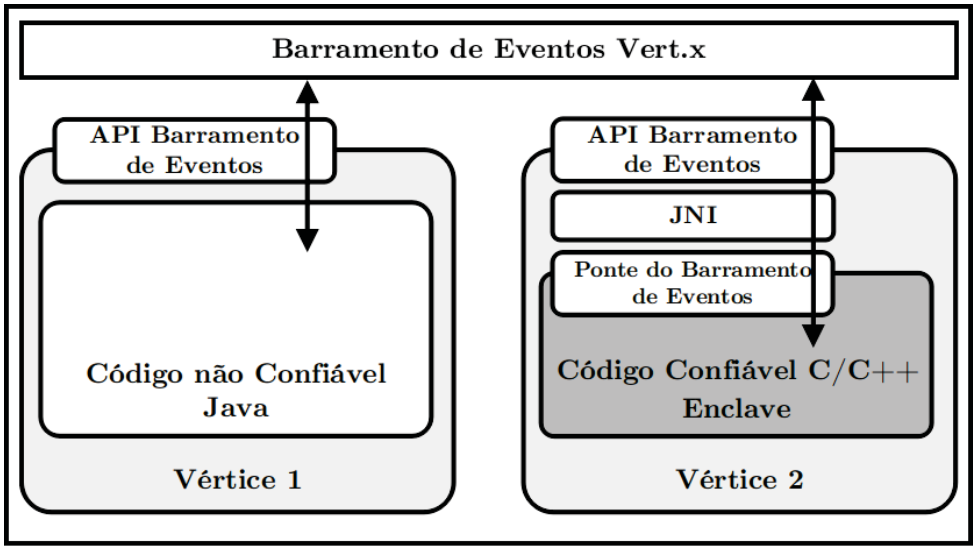


Figura 4.29: Arquitetura do *Vert.x* e a conexão de dois vértices via barramento de eventos do *Vert.x*. Adaptado de Brenner *et al.* (2017).

4.7.5 Análise de Dados

As garantias de integridade e confidencialidade providas pela arquitetura Intel SGX também atendem às necessidades de aplicações que visam efetuar o processamento e análise de dados sensíveis em ambientes de nuvem pública. Schuster *et al.* (2015) utiliza a arquitetura SGX na implementação do *VC3*, uma aplicação que permite executar operações *MapReduce*[4] mantendo a confidencialidade e integridade dos dados. Para a solução proposta, os autores utilizam o SGX para isolar regiões de memória em computadores individuais e na construção de novos protocolos que garantem a segurança das operações *MapReduce* distribuídas. Tanto o sistema operacional quanto o *Hadoop*[5] são mantidos fora do TCB, garantindo a confidencialidade e integridade dos dados mesmo que estes componentes estejam comprometidos.

[4] *MapReduce* é um modelo de programação desenhado para processar grandes volumes de dados em paralelo, dividindo o trabalho em um conjunto de tarefas independentes, com funções *Map* e *Reduce*, e a execução de ambas as funções é automaticamente paralelizada e distribuída.

[5] *Hadoop* é uma plataforma de software de computação distribuída voltada para clusters e processamento de grandes volumes de dados, inspirada no *MapReduce* e no *Google File System*, com atenção a tolerância a falhas.

Nesse mesmo contexto, Shaon *et al.* (2017) apresentam um *framework*, chamado de *BigMatrix*, que permite a utilização da arquitetura Intel SGX para efetuar a análise de dados científicos, que inclui uma linguagem simples, e de alto nível, para o tratamento de dados, com sintaxe semelhante ao *Python* e *Matlab* e a compilação automática de programas escritos nessa linguagem é que gerencia questões de mais baixo nível para simplificar parte do processamento de dados e remove algumas construções de linguagem para reduzir a probabilidade de que partes do código possam revelar informações sigilosas, ocultando, assim, padrões de acesso. A arquitetura do *BigMatrix* pode ser vista na Figura 4.30, onde nota-se a distinção entre as partes confiável e não confiável constituindo um mesmo processo.

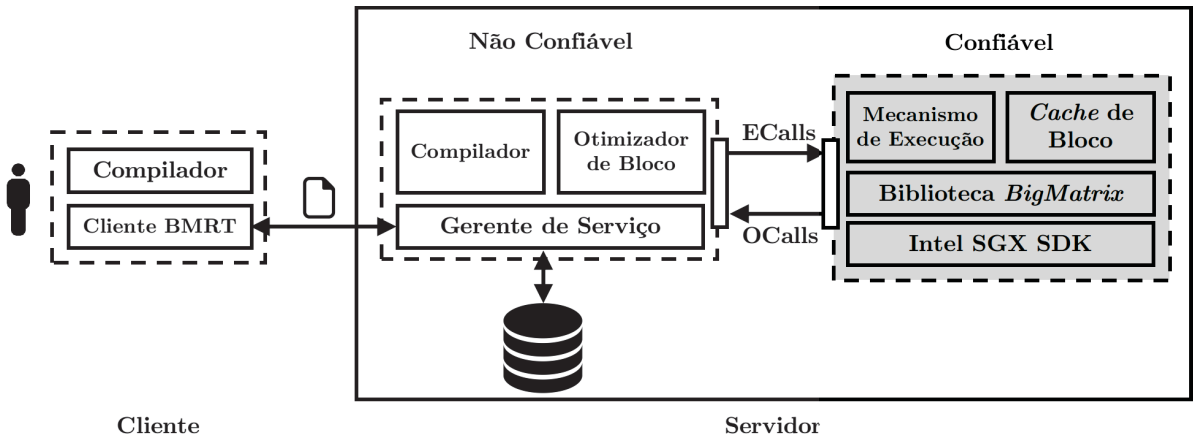


Figura 4.30: Representação do *framework* BigMatrix. Adaptado de Shaon *et al.* (2017).

Já Zheng *et al.* (2017) propõem a solução *Opaque*, baseada no *Spark SQL*, que introduz novos operadores relacionais que ofuscam o padrão de acesso aos dados, aumentando as garantias de segurança para análise de dados distribuída. A Figura 4.31 representa a arquitetura da solução *Opaque*. Os autores optaram por mover o otimizador de consultas para o lado cliente, tendo em vista que uma nuvem não confiável controlando tal mecanismo pode resultar em uma execução incorreta da requisição enviada. A otimização da consulta é complementada com um conjunto de regras que são definidas em um enclave, com o intuito de ofuscar dados sensíveis sobre a consulta. Além disso, a manipulação dos dados pelos *workers* também é realizada dentro de um enclave, mantendo a confidencialidade desses dados.

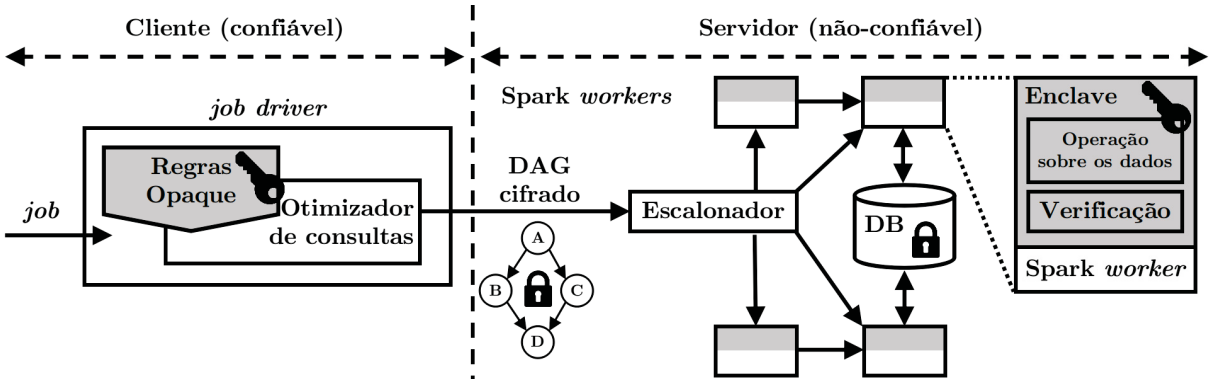


Figura 4.31: Arquitetura da solução Opaque. Adaptado de Zheng *et al.* (2017).

Semelhante à proposta de Zheng *et al.* (2017), Le Quoc *et al.* (2019) apresentam a solução *SGX-PySpark*, que faz a integração entre o PySpark e o SCONE, proposto por Arnautov *et al.* (2016) e descrito na Seção 4.7.2. Diferentemente do Opaque, o SGX-PySpark executa

apenas os processos Python dentro de um enclave, já que esses processos irão efetuar a análise sobre dados criptografados. A utilização do SCONE também elimina a necessidade de qualquer alteração no código do PySpark para a execução dentro de enclaves. A arquitetura da solução é apresentada na Figura 4.32. O SGX-PySpark efetua toda a transmissão, tanto de dados de entrada quanto de código Python, de maneira cifrada, os quais serão decifrados dentro de um enclave, utilizando uma chave fornecida pelo serviço de configuração e atestação (CAS).

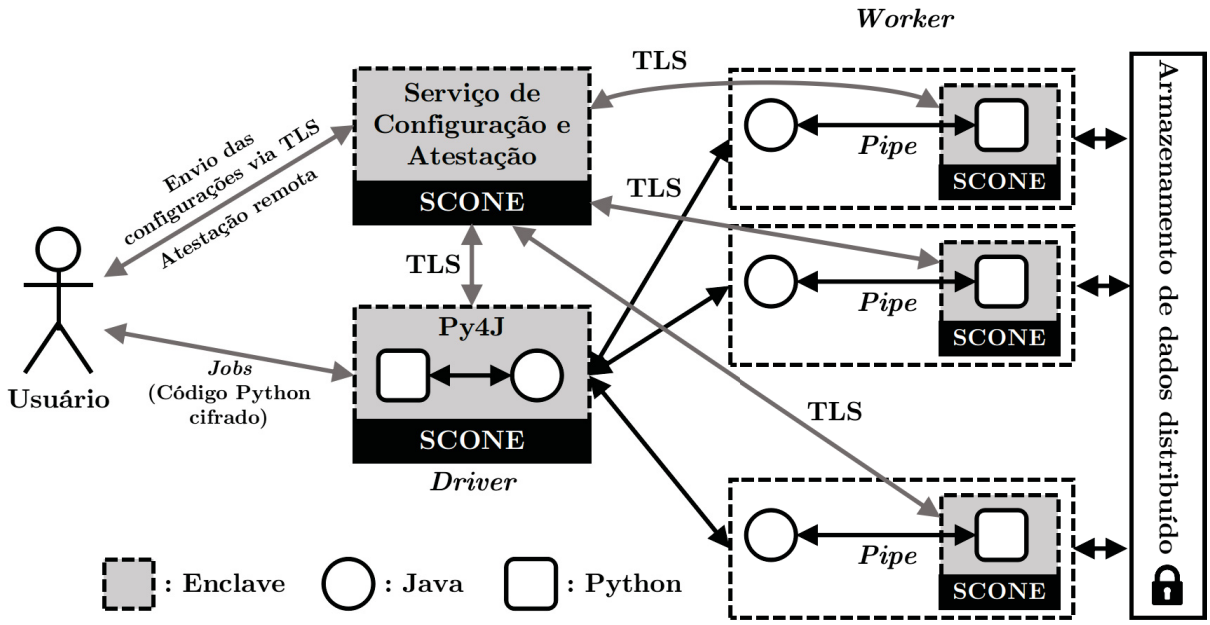


Figura 4.32: Arquitetura da solução SGX-PySpark. Adaptado de Le Quoc *et al.* (2019).

4.7.6 Gestão de Direitos Digitais e Empresariais

A confidencialidade de documentos empresariais e proteção de direitos autorais também são abordadas com soluções desenvolvidas com a arquitetura Intel SGX. Hoekstra *et al.* (2013) apresentam um ERM (*Enterprise Rights Management*), mostrado na Figura 4.33, onde as partes confiáveis no lado cliente, que são responsáveis por operar os ativos da aplicação que necessitam de proteção, são colocadas dentro de um enclave SGX. A transferência de conteúdo entre servidor e cliente é feita através de um canal seguro, criado após a atestação entre os enclaves que estão sendo executados no servidor e no cliente. O cliente especifica a política de uso e o controle de acesso ao documento através do módulo *Policy Engine* e efetua a leitura do documento por meio de um leitor seguro, implementado dentro do enclave, o qual gera *bitmaps* das páginas do documento para que estas sejam renderizadas no vídeo. Como o caminho entre a memória da aplicação e o *buffer* do vídeo é inseguro, essa transmissão deve ser protegida com o uso de alguma outra tecnologia, com os autores sugerindo a utilização da tecnologia PAVP (*Protected Audio Video Path*), descrita em Intel (2013).

Outra solução dentro desse contexto é apresentada por Bauman e Lin (2016), com o intuito de garantir a confidencialidade e integridade de dados e código de jogos de computador. A verificação da integridade dos dados e do próprio código-fonte do jogo é importante para evitar que jogadores carreguem dados alterados, ou façam modificações no modo de execução do jogo, com a finalidade de conseguir vantagens sobre seus adversários, o que é conhecido como *cheating*. Para essa tarefa, o código do jogo é carregado para um enclave, o qual irá contactar um servidor remoto através do processo de autenticação, possibilitando que este verifique a

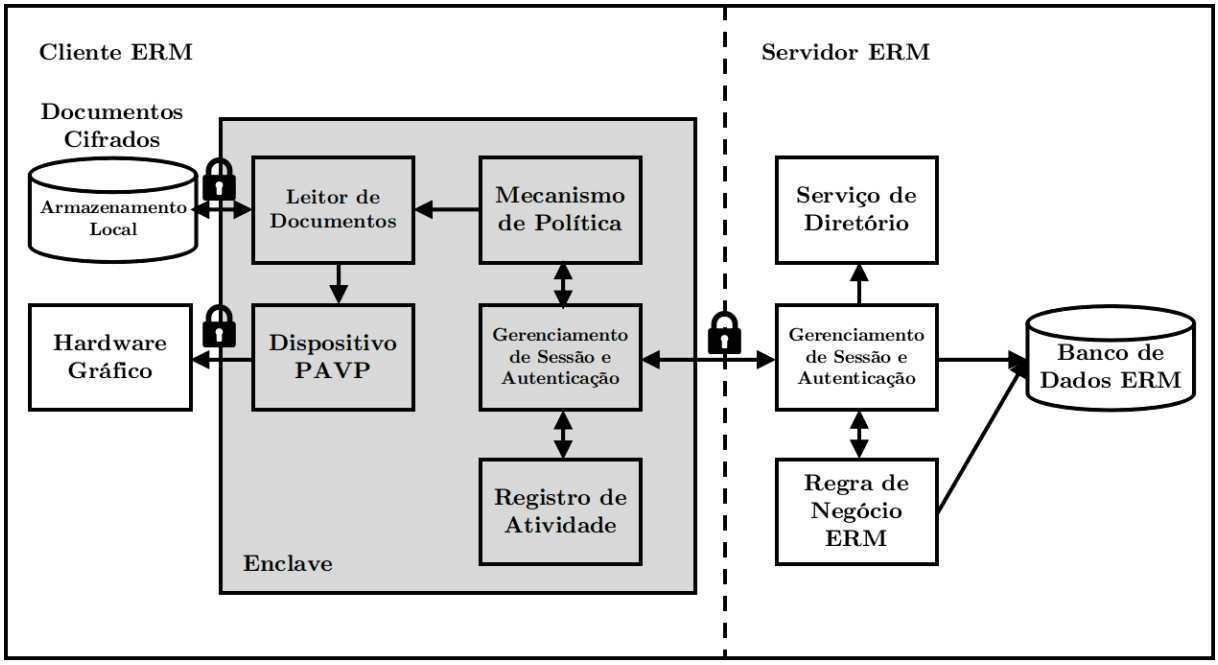
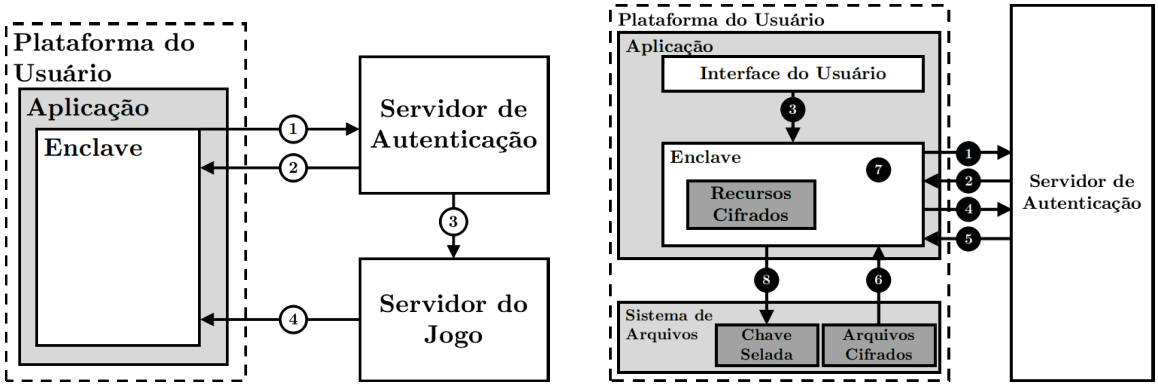


Figura 4.33: Componentes de uma arquitetura ERM cliente/servidor. Adaptado de Hoekstra *et al.* (2013).

integridade dos dados e código carregados pelo jogador (Figura 4.34(a)). De forma semelhante, os autores também propõem um esquema para garantir a confidencialidade de dados e códigos sensíveis do jogo, bem como a verificação da chave de licença, para evitar a disseminação de cópias não autorizadas. Tais dados podem ser selados por um enclave, e validados por um servidor de autenticação sempre que necessário, como demonstra a Figura 4.34(b).



(a) (b)

Figura 4.34: Esquema proposto para: (a) garantir integridade do código e dados; (b) garantir confidencialidade de código e dados. Adaptado de Bauman e Lin (2016).

4.8 CONSIDERAÇÕES FINAIS

O presente capítulo fez um apanhado de diversas soluções de software, dos mais variados tipos, que fazem uso da arquitetura SGX para prover segurança, integridade e confidencialidade dos dados. É possível notar que a utilização da arquitetura Intel SGX se dá nos mais diversos segmentos, tendo, em várias das soluções apresentadas, uma preocupação quanto ao impacto de desempenho causado pela utilização de tal tecnologia.

As soluções apresentadas nesse capítulo serão analisadas de forma mais detalhada no capítulo seguinte, sob o ponto de vista dos modelos de programação utilizados no gerenciamento dos enclaves, com o intuito de identificar quais são os modelos utilizados e quais são os impactos causados por tais modelos de gerenciamento.

5 IDENTIFICAÇÃO DE PADRÕES DE PROJETO NA UTILIZAÇÃO DE ENCLAVES

O advento da arquitetura Intel SGX traz a possibilidade de dividir a aplicação em dois componentes distintos: o enclave, que irá conter os dados sensíveis e as instruções que manipulam tais dados e; um componente não confiável, que contém os demais dados e instruções da aplicação. As garantias de segurança providas pela arquitetura Intel SGX para os dados e instruções contidas em um enclave invariavelmente causam um impacto de desempenho na execução da aplicação, o qual pode ser amenizado com a utilização de técnicas e modelos de programação adequados.

O presenta capítulo visa analisar os modelos de programação utilizado nas soluções apresentadas no capítulo anterior e mapear a forma que se dá a utilização dos enclaves por tais aplicações e serviços.

5.1 PADRÕES DE PROJETO

De acordo com Coplien (2003), padrões são soluções comprovadas para problemas recorrentes de projeto, que suportam métodos e práticas de design existentes, e têm um sentido especial no projeto de software contemporâneo. Os padrões geralmente descrevem estruturas sutis que unem módulos produzidos por métodos comuns de projeto. Conforme Buschmann *et al.* (2013), um padrão descreve um problema particular do projeto que é específico em determinado contexto e apresenta um esquema genérico e comprovado para solucionar tal problema. Essa solução é especificada descrevendo os componentes que a constituem, suas responsabilidades e relacionamentos entre si, de quais maneiras tais componentes colaboram uns com os outros nessa solução. Em tal definição, o contexto é uma situação que dá origem ao problema, sendo o problema uma situação recorrente que surge nesse contexto, ao passo que a solução é um método comprovado que resolve o problema. Essas três partes estão intimamente acopladas, tendo uma relação estabelecida entre um determinado contexto, um determinado problema que surge nesse contexto, e uma solução apropriada para o problema.

Os padrões se encontram em vários níveis no desenvolvimento de software. Alguns padrões auxiliam na estruturação do software ou de seus subsistemas, outros suportam o refinamento desses subsistemas e seus componentes, bem como a relação entre eles. Por fim, também existem padrões que auxiliam na implementação de aspectos específicos de projeto em uma linguagem de programação específica.

Um padrão de projeto fornece um esquema para refinar os subsistemas ou componentes de um software, ou os relacionamentos entre eles, descrevendo uma estrutura de comunicação que resolve um problema geral de projeto dentro de um contexto particular. Os padrões complementam as técnicas de desenvolvimento e devem oferecer suporte ao desenvolvimento, manutenção e evolução de sistemas complexos e de grande escala. Uma boa descrição de um padrão também inclui diretrizes para sua implementação, e grande parte dessas implementações são descritas utilizando linguagens orientada a objetos, o que leva a crer que a única maneira de implementar tais padrões é utilizando tais linguagens. Os recursos de orientação a objetos não são, entretanto, essenciais para a implementação desses padrões, já que maioria dos padrões requer apenas certos recursos de abstração de uma linguagem de programação, como módulos ou abstração de dados, o que possibilita implementá-los com quase qualquer paradigma de programação e em praticamente qualquer linguagem de programação. Além disso, toda linguagem de programação tem padrões específicos próprios, que são as expressões idiomáticas dessa linguagem, que

capturam a experiência de programação existente com a linguagem e definem um estilo de programação para ela.

Os padrões também fornecem uma boa abordagem para o desenvolvimento de aplicações, documentando o conhecimento do projeto e auxiliando a encontrar soluções apropriadas aos problemas. Os padrões abrangem diversas áreas no desenvolvimento de software, abordando aspectos importantes da arquitetura da aplicação, e podem ser combinados uns com os outros, podendo até mesmo refinar um determinado padrão com a utilização de outros. Pode-se afirmar que os padrões fornecem ferramentas lógicas que auxiliam na construção de aplicações que atendam aos requisitos funcionais e não funcionais (Buschmann *et al.*, 2013).

A utilização de padrões de projeto é amplamente difundida no desenvolvimento de software orientado a objetos, tendo diversos destes padrões descritos por Gamma *et al.* (2004), que definem 23 padrões que são agrupados, de acordo com suas finalidades, em três categorias: criacionais, estruturais e comportamentais. Além dessas três, outra categoria de padrões que também merece destaque é a de padrões arquiteturais.

Um padrão criacional abstrai ou adia o processo de criação de um objeto, ajudando a tornar o sistema independente de como os objetos são criados, compostos e representados, dando flexibilidade no que é criado, quem cria, como e quando é criado. De acordo com Gamma *et al.* (2004), "os padrões estruturais se preocupam com a forma como os objetos são compostos para formar estruturas maiores", facilitando o processo de identificar uma forma simples para construir as relações entre diferentes entidades. Os padrões comportamentais se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles também descrevem como será a comunicação entre os objetos, caracterizando os fluxos de controle e permitindo que o desenvolvedor foque apenas na maneira como os objetos são interconectados.

5.2 PADRÕES ARQUITETURAIS E DE COMUNICAÇÃO

Um padrão arquitetural, segundo Buschmann *et al.* (2013), expressa um esquema de organização estrutural fundamental para sistemas de software, fornecendo um conjunto de subsistemas predefinidos, especificando suas responsabilidades e incluindo regras e diretrizes para organizar os relacionamentos entre eles. Tais padrões são modelos para arquiteturas de software concretas, que especificam propriedades estruturais de todo o sistema e também tem impacto na arquitetura de seus subsistemas e, por isso, a escolha de um padrão arquitetural é uma decisão fundamental na fase de projeto de um sistema.

As próximas subseções apresentam os padrões arquiteturais e de comunicação amplamente utilizados no desenvolvimento de aplicações, e que foram identificados nas aplicações que utilizam a arquitetura Intel SGX, descritas no Capítulo 4.

5.2.1 *Client-Server*

A arquitetura Cliente-Servidor, ou *Client-Server*, divide uma aplicação em duas partes: os provedores de recursos e serviços, chamados de *servidores* e; os requisitantes de serviços, chamados de *clientes*. Tipicamente, o servidor irá prover um ou mais serviços e/ou recursos a diversos clientes, podendo compartilhar recursos entre esses clientes. O cliente e o servidor podem constituir processos distintos sendo executados na mesma plataforma ou em plataformas distintas, efetuando a comunicação via protocolos de rede.

Uma extensão dessa arquitetura é o padrão *Client-Dispatcher-Server*, que adiciona uma camada intermediária entre os clientes e os servidores para prover mecanismos transparentes para a localização de serviços e ocultar os detalhes da comunicação entre cliente e servidor (Buschmann *et al.*, 2013).

5.2.2 *Peer-to-Peer*

O *Peer-to-Peer* (*P2P*) é um padrão arquitetural para aplicações distribuídas onde cada participante acessa os demais de forma direta, sem a necessidade de entidades mediadoras. Cada participante atua simultaneamente como cliente e servidor para os demais participantes (Schollmeier, 2001).

A principal característica de uma arquitetura *P2P* é a ausência de um provedor de recursos central. Assim, na ausência de um dos nós, os recursos necessários podem ser solicitados aos demais. Isso também faz com que se obtenha uma alta resiliência e escalabidade da solução.

5.2.3 *Broker*

O padrão *Broker* tem o intuito de estruturar sistemas distribuídos utilizando componentes desacoplados que se utilizam de invocações remotas para efetuar as interações. Um componente *Broker* coordena as comunicações, tanto das solicitações de encaminhamento como dos resultados e exceções retornados. Os clientes acessam os serviços providos pelos servidores através do envio de requisições ao *Broker*, que se encarregará de localizar o servidor apropriado, encaminhar a requisição e retornar os resultados ao cliente.

A utilização do padrão *Broker* simplifica o acesso a serviços distribuídos, tirando o foco da comunicação inter-processos em baixo nível, além de permitir alterações, adições, exclusões e realocações dinâmicas de objetos no sistema, tornando a arquitetura mais flexível (Buschmann *et al.*, 2013).

5.2.4 *Publish-Subscribe*

Publish-Subscribe é um padrão de comunicação em sistemas onde os remetentes das mensagens (*publishers*) não as enviam diretamente aos destinatários (*subscribers*), sendo as mensagens categorizadas em diferentes classes sem o conhecimento de quais destinatários existem. Os destinatários podem manifestar interesse em uma ou mais classes de mensagens e então passam a receber tais mensagens, sem o conhecimento de quais remetentes existem. Geralmente esse padrão está inserido dentro de um sistema orientado a mensagens de maior porte.

Uma característica do padrão *Publish-Subscribe* é que ele oferece maior escalabilidade e suporte a topologias mais dinâmicas mas, em contrapartida, é menos flexível quando há a necessidade de alterar as estruturas de dados que são publicadas (Buschmann *et al.*, 2013).

5.2.5 *Asynchronous Method Invocation*

O padrão *Asynchronous Method Invocation* (AMI), também referenciado como *Event-Based Asynchronous*, descreve uma maneira assíncrona para efetuar chamadas a métodos que exigem um elevado tempo de processamento. Nesse padrão, o método chamador não é bloqueado enquanto aguarda uma resposta, mas notificado ao final da execução da tarefa desejada, podendo então consultar a resposta gerada. Esse padrão se utiliza das vantagens providas por sistemas *multithread*, ocultando muitos dos problemas complexos inerentes à programação *multithread* (Microsoft, 2017).

5.2.6 *Proxy*

O padrão *Proxy* pode ser utilizado para definir um objeto que irá como um substituto para outro objeto, permitindo que se criem representações do objeto a ser substituído. Dessa

forma, o padrão pode prover um nível de proteção de acesso a um objeto sensível ou uma interface de comunicação com um objeto que está localizado em um espaço remoto. Ele pode ser utilizado também para prover um objeto que servirá como um intermediário para a comunicação entre outros dois objetos.

5.3 PADRÕES DE PROJETO NO GERENCIAMENTO DE ENCLAVES

Analisando as soluções apresentadas no Capítulo 4, pode-se notar uma grande variedade de aplicações e serviços, que atendem aos mais diversos propósitos, utilizando os recursos providos pela arquitetura Intel SGX. Tais soluções fazem uso de enclaves de diversas maneiras, buscando utilizar as propriedades desses de uma maneira concisa.

Atrelada à utilização dos enclaves, está uma decisão de projeto, que é a definição do modelo de programação e padrões de projetos utilizados para instanciar e gerenciar o ciclo de vida do enclave, além da escolha dos dados que serão manipulados dentro do enclave e em que momentos a aplicação fará a comunicação com o enclave, e vice-versa, com a utilização das ECALLs e OCALLs.

Conforme exposto no Intel *Software Guard Extensions Developer Guide* (Intel, 2016a), o modelo de programação de uma aplicação que se utiliza de enclaves não difere do modelo de programação utilizado por uma aplicação que não utiliza enclaves. Dessa forma, utilizando os conceitos de programação orientada a objetos, pode-se tratar o enclave como um objeto, o qual pode ser instanciado para a utilização dos métodos que possui e, quando este não for mais necessário, pode ser destruído e liberar a memória que estava utilizando. Fazendo uso dessa relação, é possível dizer que há uma grande gama de padrões de projeto que podem ser utilizados para o gerenciamento de enclaves, o que abre possibilidades para otimizações na instanciação de enclaves e na utilização de recursos por parte desses.

A forma mais natural de uma aplicação utilizar os recursos providos por um enclave é instanciá-lo quando necessário, utilizar as garantias providas por esse enclave para manipular os dados sensíveis, realizando as operações necessárias em um ambiente confiável e, após a execução de tais operações, o enclave é destruído, sendo novamente instanciado quando for necessário em outro momento. Esse modelo de programação é a aplicação do que é disposto no ciclo de vida do enclave, mostrado na Figura 3.2, no qual o enclave é instanciado, os seus dados e código são carregados para a PRM e, após a sua utilização, o enclave é destruído.

Ao analisar os padrões de projeto utilizados pelas soluções descritas no Capítulo 4, nota-se que a maioria dessas aplicações fazem uso dos enclaves sob demanda, instanciando o enclave quando necessário e destruindo-os assim que finalizam a sua utilização. Além disso, outra forma comum de utilização dos enclaves é tratá-los como uma parte da aplicação, ficando o enclave diretamente ligado à aplicação que o utiliza. Uma síntese das soluções estudadas e dos modelos de gerenciamento de enclaves utilizados por estas soluções é apresentada no Quadro 5.1.

Apesar de tais técnicas serem as mais utilizadas, elas não caracterizam a única forma de utilização e gerenciamento do ciclo de vida de enclaves. A solução *Iron*, proposta por Fisch *et al.* (2017), se utiliza de mais de um enclave por processo, utilizando diferentes enclaves para diferentes funcionalidades, o que traz um maior grau de independência entre os enclaves e garante que os dados processados por determinado enclave sejam inacessíveis aos demais enclaves em execução por tal aplicação (a não ser, é claro, quando tem-se a transferência de dados explícita através de um processo de atestação entre os enclaves), o que pode ser um requisito importante em determinadas situações. Tal solução também utiliza um enclave (*Key Management Enclave* - KME) que é oferecido como um serviço de distribuição de chaves aos demais enclaves.

Quadro 5.1: Modelos de programação adotados nas aplicações SGX apresentadas.

Proposta	**Aplicação**	**Enclaves / Processo**	**Separação entre Parte Confiável e Não-Confiável**	**Instanciação de Enclaves**	**Padrões Identificados**
SGX-Tor (Kim *et al.*, 2017)	Aplicações Seguras (Seção 4.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
Videoconferência (Hoekstra *et al.*, 2013)	Aplicações Seguras (Seção 4.2)	1:1	Cada aplicação instancia um enclave para efetuar a atestação com a outra parte	Sob demanda	Cliente-Servidor
Banco de Senhas (Brekalo *et al.*, 2016)	Aplicações Seguras (Seção 4.2)	2:1	Há a existência de um enclave provedor em cada servidor, responsável por atestar os demais enclaves	Sob demanda (com exceção do enclave provedor)	Cliente-Servidor
MITC Defender (Liang *et al.*, 2017)	Aplicações Seguras (Seção 4.2)	N:1	A aplicação cliente gerencia vários enclaves para a leitura de dados confidenciais e efetua atestação com um enclave no servidor para o envio desses dados	Sob demanda	Cliente-Servidor
OTP (Hoekstra *et al.*, 2013)	Aplicações Seguras (Seção 4.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
SafeKeeper (Krawiecka *et al.*, 2018)	Aplicações Seguras (Seção 4.2)	1:N	Utiliza uma arquitetura cliente/servidor *stateful*, com as informações essenciais sendo carregadas na primeira inicialização do enclave e podendo ser recuperadas em inicializações seguintes	O enclave é mantido instanciado enquanto o processo servidor está ativo	Cliente-Servidor
Biblioteca Criptográfica (Mofrad *et al.*, 2017)	Aplicações Seguras (Seção 4.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
Iron (Fisch *et al.*, 2017)	Aplicações Seguras (Seção 4.2)	3+:1	São criados processos distintos, com enclaves remotos, e utilizado o recurso de atestação para a comunicação entre os enclaves	Sob demanda (com exceção do KME)	Cliente-Servidor; *Proxy*
PoS (Li *et al.*, 2018)	Aplicações Seguras (Seção 4.2)	2:1	Cada processo possui dois enclaves, um responsável pela chave pública, e outro responsável pela chave privada	Sob demanda	Cliente-Servidor; *Proxy*
TEEFHE (Wang *et al.*, 2019b)	Aplicações Seguras (Seção 4.2)	N:M	Um *pool* de enclaves é responsável por responder às requisições, sendo orquestrados por um escalonador	Na criação do *pool*	Cliente-Servidor; *Broker*
Haven (Baumann *et al.*, 2015)	*Runtime* (Seção 4.3)	1:1	Ambas as partes constituem o mesmo processo	Junto com o processo	*Proxy*

Continua na próxima página

Quadro 5.1 – *Continuação da página anterior*

Proposta	**Aplicação**	**Enclaves / Processo**	**Separação entre Parte Confiável e Não-Confiável**	**Instanciação de Enclaves**	**Padrões Identificados**
Graphene-SGX (Tsai *et al.*, 2017)	*Runtime* (Seção 4.3)	1:1	Ambas as partes constituem o mesmo processo	Junto com o processo	*Proxy*
SGXKernel (Tian *et al.*, 2017)	*Runtime* (Seção 4.3)	1:1	Ambas as partes constituem o mesmo processo	Junto com o processo	*Proxy*
TrustJS (Goltzsche *et al.*, 2017)	*Runtime* (Seção 4.3)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
Módulos do Núcleo Seguros (Richter *et al.*, 2016)	Serviços do Sistema Operacional (Seção 4.4)	1:1	O enclave é separado do processo principal, sendo gerenciado por um *daemon*, que é responsável pela comunicação com o enclave	Sob demanda	*Proxy*
SGX-Log (Karande *et al.*, 2017)	Serviços do Sistema Operacional (Seção 4.4)	1:N	Utiliza uma arquitetura cliente/servidor, com diversos clientes se comunicando com um processo servidor que é encarregado de efetuar as operações com o enclave	Mantido instanciado enquanto o processo servidor está ativo	Cliente-Servidor
UniSGX (Condé *et al.*, 2018)	Serviços do Sistema Operacional (Seção 4.4)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
PrIXP (Chiesa *et al.*, 2017)	Infraestrutura de Rede (Seção 4.5.1)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	Cliente-Servidor; *Proxy*
LibSEAL (Aublin *et al.*, 2018)	Infraestrutura de Rede (Seção 4.5.1)	1:N	Utiliza uma arquitetura cliente/servidor, com o servidor sendo responsável pelo registro das requisições recebidas	Mantido instanciado enquanto o processo servidor está ativo	Cliente-Servidor
S-NFV (Shih *et al.*, 2016)	*Network Function Virtualization* (Seção 4.5.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
TrustedClick (Coughlin *et al.*, 2017)	*Network Function Virtualization* (Seção 4.5.2)	1:1	O enclave faz parte de um processo sendo executado na nuvem	Sob demanda	Cliente-Servidor; *Proxy*
ShieldBox (Trach *et al.*, 2018)	*Network Function Virtualization* (Seção 4.5.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*; *Asynchronous Method Invocation*
LightBox (Duan *et al.*, 2019)	*Network Function Virtualization* (Seção 4.5.2)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*

Continua na próxima página

Quadro 5.1 – *Continuação da página anterior*

Proposta	**Aplicação**	**Enclaves / Processo**	**Separação entre Parte Confiável e Não-Confiável**	**Instanciação de Enclaves**	**Padrões Identificados**
P2P (Jia *et al.*, 2017)	*Peer-to-Peer* (Seção 4.6.1)	1:1	Cada nodo possui um enclave para a execução segura de aplicações	Sob demanda	*Proxy*
MedLog (Nguyen *et al.*, 2016)	*Internet of Things* (Seção 4.6.2)	1:N	Utiliza uma arquitetura cliente/servidor, com o dispositivo cliente se comunicando com um enclave no servidor para o envio de informações	Mantido instanciado enquanto o processo servidor está ativo	Cliente-Servidor
Town Crier (Zhang *et al.*, 2016)	*Blockchain* (Seção 4.6.3)	1:1	Cada instância de contrato inteligente se comunica com um enclave específico	Sob demanda	Cliente-Servidor; *Proxy*
Proof of Luck (Milutinovic *et al.*, 2016)	*Blockchain* (Seção 4.6.3)	1:1	Cada nodo na *blockchain* possui um enclave responsável pela geração de números aleatórios	Sob demanda	*Proxy*
Obscuro (Tran *et al.*, 2018)	*Blockchain* (Seção 4.6.3)	1:1	O enclave faz parte do processo do mixer	Sob demanda	*Proxy*
Teechan (Lind *et al.*, 2019)	*Blockchain* (Seção 4.6.3)	1:1	Cada nodo da rede de pagamentos dispõe de um enclave para a execução de tarefas confiáveis	Sob demanda	*Proxy*; P2P
HardIDX (Fuhry *et al.*, 2017)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave se localiza no servidor, sendo responsável pela manipulação de chaves e execução de consultas	Permanece instanciado enquanto o servidor está ativo	Cliente-Servidor; *Proxy*
REARGUARD (Sun *et al.*, 2018)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave se localiza no servidor, sendo responsável pela manipulação de chaves e execução de consultas	Permanece instanciado enquanto o servidor está ativo	Cliente-Servidor; *Proxy*
VeritasDB (Sinha e Christodorescu, 2018)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave se localiza no servidor, sendo responsável pela manipulação de chaves e execução de consultas	Permanece instanciado enquanto o servidor está ativo	Cliente-Servidor; *Proxy*
ShieldStore (Kim *et al.*, 2019)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave se localiza no servidor, sendo responsável pela manipulação de chaves e execução de consultas	Permanece instanciado enquanto o servidor está ativo	Cliente-Servidor; *Proxy*
SPEICHER (Bailleu *et al.*, 2019)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave é encapsulado dentro de um contêiner, gerenciado pelo SCONE	Junto com o contêiner	Cliente-Servidor; *Asynchronous Method Invocation*

Continua na próxima página

Quadro 5.1 – *Continuação da página anterior*

Proposta	**Aplicação**	**Enclaves / Processo**	**Separação entre Parte Confiável e Não-Confiável**	**Instanciação de Enclaves**	**Padrões Identificados**
EnclaveDB (Priebe *et al.*, 2018)	Infraestrutura de Nuvem (Seção 4.7.1)	1:N	O enclave se localiza no servidor, sendo responsável pela manipulação de chaves e execução de consultas	Permanece instanciado enquanto o servidor está ativo	Cliente-Servidor; *Proxy*
SCONE (Arnautov *et al.*, 2016)	Virtualização em Nuvem (Seção 4.7.2)	M:N	O enclave faz parte do processo que gerencia o contêiner, mas há um segundo processo que efetua as requisições ao sistema operacional hospedeiro, através da multiplexação de *threads* para reduzir o impacto de desempenho	Junto com o contêiner	*Asynchronous Method Invocation*
Panoply (Shinde *et al.*, 2017)	Virtualização em Nuvem (Seção 4.7.2)	1:1	Ambas as partes constituem o mesmo processo	Junto com o processo	*Proxy*
SvTPM (Wang *et al.*, 2019a)	Virtualização em Nuvem (Seção 4.7.2)	1:1	Cada VM tem um TPM virtual associado a ela	Sob demanda	*Proxy*
Scotch (Leach *et al.*, 2017)	Gerenciamento de Nuvem (Seção 4.7.3)	1:N	O enclave é executado em uma VM e é responsável por armazenar as informações de todas as VMs gerenciadas	Junti com a máquina virtual	Cliente-Servidor
SecureKeeper (Brenner *et al.*, 2016)	Gerenciamento de Nuvem (Seção 4.7.3)	1+:1	Cada cliente tem um enclave individual associado a ele no servidor onde está conectado	Sob demanda	Cliente-Servidor
Microsserviços Seguros (Fetzer, 2016)	Microsserviços (Seção 4.7.4)	2:1	Um dos enclaves constitui um contêiner para a execução da aplicação e outro armazena o estado e provê a comunicação segura com o gerenciador de recursos	Junto com o processo	Cliente-Servidor
Microsserviços Seguros (Brenner *et al.*, 2017)	Microsserviços (Seção 4.7.4)	1:1	Ambas as partes constituem o mesmo processo	Junto com o processo	*Proxy*
VC3 (Schuster *et al.*, 2015)	Análise de Dados na Nuvem (Seção 4.7.5)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	*Proxy*
BigMatrix (Shaon *et al.*, 2017)	Análise de Dados na Nuvem (Seção 4.7.5)	1:1	Ambas as partes constituem o mesmo processo	Sob demanda	Cliente-Servidor; *Proxy*
Opaque (Zheng *et al.*, 2017)	Análise de Dados na Nuvem (Seção 4.7.5)	2+:1	Desconsiderando a separação inerente ao Spark, ambas as partes constituem o mesmo processo	Sob demanda	Cliente-Servidor; *Broker*

Continua na próxima página

Quadro 5.1 – *Continuação da página anterior*

Proposta	**Aplicação**	**Enclaves / Processo**	**Separação entre Parte Confiável e Não-Confiável**	**Instanciação de Enclaves**	**Padrões Identificados**
SGX-PySpark (Le Quoc *et al.*, 2019)	Análise de Dados na Nuvem (Seção 4.7.5)	2+:1	Desconsiderando a separação inerente ao Spark, ambas as partes constituem o mesmo processo	Sob demanda	Cliente-Servidor; *Broker*
ERM (Hoekstra *et al.*, 2013)	Gestão de Direitos Digitais e Empresariais (Seção 4.7.6)	1:1	Utiliza uma arquitetura cliente/servidor, com a aplicação cliente responsável por criar o enclave para efetuar a comunicação com o servidor	Sob demanda	Cliente-Servidor
Games (Bauman e Lin, 2016)	Gestão de Direitos Digitais e Empresariais (Seção 4.7.6)	1:1	Utiliza uma arquitetura cliente/servidor, com a aplicação cliente responsável por criar o enclave para efetuar a comunicação com o servidor	Sob demanda	Cliente-Servidor

Outro modelo arquitetural para a utilização de enclaves é utilizado no serviço *SGX-Log*, proposto por Karande *et al.* (2017). Tal serviço utiliza uma arquitetura cliente/servidor, centralizando as operações sobre os arquivos de *log* em uma única instância de enclave, sendo que as aplicações que desejam efetuar leituras ou escritas no arquivo de *log* devem solicitar tais operações à aplicação servidora. Dessa forma, o gerenciamento do enclave se dá de forma centralizada, reduzindo o impacto causado por várias instâncias de um mesmo enclave em execução. Nessa arquitetura, o servidor gerencia um único enclave, que pode ser utilizado por várias instâncias do serviço de *log*, as quais têm a finalidade de escrever ou consultar as informações contidas nos registros de *log* do sistema.

Além da arquitetura cliente/servidor utilizada pelo serviço *SGX-Log*, destaca-se também a arquitetura utilizada pelo serviço *SGXKernel*, apresentada por Richter *et al.* (2016), o qual tem a intermediação de um *daemon* que permite que módulos do núcleo do sistema operacional utilizem enclaves. O enclave, devido às restrições impostas pela arquitetura Intel SGX, somente pode executar as suas rotinas em modo usuário e, para que serviços do núcleo do sistema operacional possam fazer uso dos recursos providos pelos enclaves, deve-se ter um serviço mediador sendo executado em modo usuário, recebendo e repassando as requisições ao enclave. Esse é o papel desempenhado pelo *daemon* do serviço *SGXKernel*. Dessa forma, cada enclave é gerenciado por um *daemon* específico, que é responsável por intermediar as requisições vindas do serviço que é executado no núcleo do sistema operacional.

Por fim, destaca-se também a arquitetura proposta pela solução *SCONE*, descrita por Arnautov *et al.* (2016), que utiliza uma relação onde cada enclave é multiplexado em diversos *threads* no sistema operacional, executando as chamadas de sistema de forma assíncrona e reduzindo o impacto de desempenho imposto pelas chamadas ECALL e OCALL. Nessa proposta, o enclave não é instanciado sob demanda, mas sim na inicialização do processo, já que o processo irá se utilizar dos recursos providos pelo enclave durante todo o seu ciclo de vida.

Em cada uma dessas arquiteturas de gerenciamento de enclaves, busca-se não somente utilizar-se das garantias providas por tal arquitetura, mas fazê-la com o menor impacto possível na execução das aplicações, no que diz respeito ao desempenho dessas.

5.4 ASPECTOS DE DESEMPENHO NA UTILIZAÇÃO DE ENCLAVES

A busca por aumentar a segurança das aplicações e confidencialidade de dados sensíveis acaba por impactar no desempenho dessas aplicações, e com a utilização da arquitetura SGX isso não é diferente. Fisch *et al.* (2017) pontuam que a inicialização de diversos enclaves pela aplicação pode ter um custo computacional elevado, visto que, a princípio, o enclave deve alocar toda a memória de que necessitará. A Figura 5.1 mostra o custo de inicialização do enclave, variando o tamanho da *heap* e da *stack*. Ao variar a *stack*, a *heap* é fixada em 1 MB e, ao variar a *heap*, a *stack* é fixada em 256 KB, sendo que esses são os valores padrões definidos pela documentação do SGX para tais parâmetros. É possível verificar que o aumento da *stack* causa um maior impacto na inicialização do enclave, o que reforça a necessidade de selecionar cuidadosamente quais partes da aplicação deverão ser executadas dentro de um enclave, mantendo esse o menor possível.

O custo de criação do enclave pode ser reduzido com a utilização do SGX 2, conforme descrito por McKeen *et al.* (2016). A segunda versão do SGX adiciona seis novas instruções que permitem gerenciar dinamicamente o tamanho da *heap* e da *stack* do enclave, possibilitando alocar novas páginas de memória e liberar páginas de memória que não são mais necessárias dentro do EPC. Todavia, o SGX 2 ainda não está disponível comercialmente, e também não há uma previsão de quando será disponibilizado.

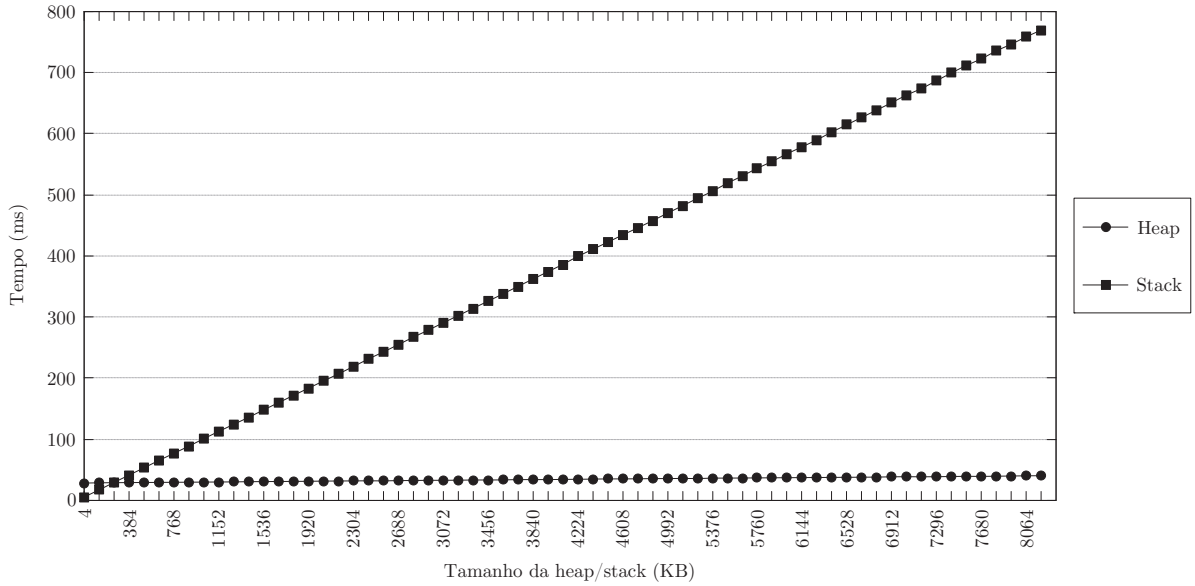


Figura 5.1: Impacto do tamanho da *heap* e da *stack* no tempo de inicialização de um enclave.

Outro ponto a ser considerado na construção de aplicações é a queda de desempenho que essas terão ao fazerem uso de enclaves. Essa queda de desempenho se dá por diversos fatores: desde a inicialização do enclave até a sua própria utilização, através das chamadas ECALL, e também pelo fato da CPU necessitar decifrar e cifrar os dados contidos na PRM. Zhao *et al.* (2016) apresentam um estudo experimental sobre o custo de desempenho dos enclaves, demonstrando que o chaveamento da CPU entre o modo enclave e o modo normal, quando se tem a execução de uma ECALL ou de uma OCALL, tem um alto custo, chegando a ser até 35 vezes mais custoso que uma chamada de sistema.

Uma das maiores necessidades de efetuar uma chamada OCALL, saindo dos domínios do enclave, é para a execução de uma chamada de sistema, visto que esse tipo de chamada não pode ser executada de dentro do enclave, já que esse é executado no espaço do usuário (*ring-3*). Com o intuito de contornar o *overhead* de desempenho causado pela chamadas de funções OCALLs, Orenbach *et al.* (2017) apresentam uma solução para a execução de chamadas de sistema sem a necessidade de alterar o contexto de execução. Para isso, é utilizada uma segunda *thread*, executando fora do enclave, que recebe as requisições através de uma chamada de procedimento remoto (RPC - *Remote Procedure Call*), e então executa a chamada de sistema requerida.

Além do custo da entrada e saída do enclave, também há o custo de manipular a memória cifrada dentro da PRM, como mostrado por Zhao *et al.* (2016). A velocidade para efetuar a cópia de dados entre duas regiões de memória fora da PRM é de 4 GB/s, enquanto que a cópia de dados entre duas regiões de memória dentro da PRM é de $1,2$ GB/s. Além disso, a operação de cópia de dados de uma região fora da PRM para dentro da PRM, ou vice-e-versa, é feita a uma taxa de $2,6$ GB/s, o que deve ser considerado quando há a necessidade de transferir grandes quantidades de dados para dentro ou para fora do enclave, ou então quando há um fluxo constante de dados atravessando as fronteiras do enclave. Göttel *et al.* (2018) apresentam um estudo completo sobre as sobrecargas impostas pelo SGX nas operações realizadas sobre memória cifrada, e Costa *et al.* (2018) apresentam uma análise de desempenho no uso das funções de criptografia da arquitetura Intel SGX, utilizando a distribuição de vídeo sob demanda como caso de uso.

Também deve ser considerado o tamanho da PRM, a qual é limitada a 128 MB e que, de acordo com Shaon *et al.* (2017), Mofrad *et al.* (2017) e Fuhry *et al.* (2017), somente

cerca de 90 MB a 96 MB estão efetivamente disponíveis para o armazenamento de dados e código dos enclaves. Essa limitação da PRM faz com que seja necessário pensar em alternativas para situações em que os enclaves necessitem trabalhar com grandes quantidades de dados simultaneamente, além de pode trazer algumas consequências quando houver várias instâncias de um mesmo enclave, ou vários enclaves em memória, podendo fazer com que seja necessário que alguns destes enclaves sejam trazidos para a memória secundária, a fim de liberar espaço na PRM. A operação de *swap* de memória é suportada pela arquitetura Intel SGX mas, obviamente, impacta no desempenho.

Diante desses fatores, é importante buscar alternativas para uma utilização eficiente de enclaves por parte das aplicações, principalmente em um ambiente onde existe uma alta demanda na instanciação e utilização de enclaves. Uma alternativa para o gerenciamento de enclaves é o compartilhamento de uma mesma instância de enclave por diversas instâncias de uma mesma aplicação, ou até mesmo por diversas aplicações distintas. Este modelo é uma generalização da arquitetura cliente/servidor proposta por Karande *et al.* (2017) e pode ser utilizado em soluções de software que tratam diversas requisições visando ler ou alterar um determinado conteúdo selado por um enclave ou manipular um segredo em comum. Outra alternativa é a manutenção de um *pool* de enclaves, com cada enclave sendo utilizado por uma única instância de aplicação, mantendo a confidencialidade dos dados da aplicação requisitante, mas, ao final da requisição, em vez do enclave ser destruído, os dados sensíveis podem ser removidos e o enclave se tornar disponível para responder a outra requisição. Esse modelo pode ser utilizado em serviços Web ou cliente/servidor, em que o servidor trata pequenas requisições mas em grande número e, mantendo uma quantidade considerável de enclaves instanciados pode-se diminuir a sobrecarga de desempenho causada pela frequente instanciação de novos enclaves. Tais alternativas são analisadas em detalhe no capítulo seguinte.

5.5 CONSIDERAÇÕES FINAIS

Esse capítulo apresentou os conceitos dos padrões de programação e fez uma análise dos modelos de programação utilizados pelos sistemas de software descritos no capítulo anterior. A utilização de tais padrões visa solucionar problemas e/ou dificuldades na implementação de determinadas tarefas de uma aplicação, com a utilização de uma metodologia bem documentada e já comprovada.

Nota-se que muitas dessas soluções fazem a utilização dos enclaves sob demanda, com o enclave sendo instanciado para atender a uma requisição e, após efetuadas as operações necessárias, o enclave é destruído. Outro ponto a ser destacado é que, na maioria das vezes, cada aplicação faz uso de um ou mais enclaves específicos, fato que, tendo uma alta utilização de enclaves, pode-se ter uma saturação da área da PRM. Essas implicações levam à proposta de técnicas de gerenciamento de enclaves, as quais serão apresentadas no próximo capítulo.

6 UM PROVEDOR DE SERVIÇOS CONFIÁVEIS

O presente capítulo tem o objetivo de discutir modelos eficientes de gerenciamento de enclaves, do ponto de vista da utilização de recursos, e apresentar um novo paradigma para o gerenciamento do ciclo de vida e disponibilização de enclaves às aplicações. Além disso, é apresentada uma descrição em alto nível de um provedor de enclaves, que engloba os requisitos necessários para a aplicação do paradigma proposto, com os detalhes técnicos e de implementação da proposta sendo expostos e discutidos no capítulo seguinte.

6.1 INTERAÇÃO ENTRE PROCESSOS E ENCLAVES

A arquitetura SGX oferece aos desenvolvedores a possibilidade de garantir a confidencialidade e integridade de porções de código e dados em suas aplicações, através de uma execução encapsulada e protegida pelo hardware. O modelo proposto pelo SGX SDK tem base no particionamento estático da aplicação, definindo em tempo de projeto os componentes que são encapsulados em enclaves, os quais devem ser instanciados e gerenciados pela própria aplicação. A aplicação de tal modelo pode trazer algumas dificuldades na construção de aplicações que dependem umas das outras, ou que precisem compartilhar algum conjunto de dados entre si.

A comunicação direta entre diferentes enclaves é impossibilitada devido às premissas de isolamento impostas pelo SGX, sendo necessária a mediação por parte de uma aplicação não confiável para realizar a troca de dados entre dois enclaves distintos, o que acarreta em várias trocas de contexto e acaba por impactar no desempenho da aplicação. Alguns trabalhos apresentam soluções monolíticas para contornar tais limitações, fazendo uso de um único enclave para encapsular todos os componentes sensíveis da aplicação, eliminando a necessidade de comunicações inter-enclaves e facilitando o acesso aos dados, sendo uma abordagem amplamente utilizada em soluções de *library OSes* (Shen *et al.*, 2018).

A aplicação de um modelo monolítico pode reduzir a complexidade de comunicação entre diferentes enclaves, mas traz consigo um aumento expressivo do TCB e acaba aumentando o impacto de desempenho inicial devido à instanciação do enclave. Além disso, quanto maior for o enclave, maior também será a superfície de ataque exposta a um adversário, aumentando assim o risco à integridade dos dados e métodos nele contidos. De acordo com Gjerdrum *et al.* (2017), é necessária uma atenção especial na fase de particionamento da aplicação, definindo quais partes devem participar ou não do enclave, de forma a reduzir a quantidade de dados que deverão ser copiados para tais enclaves, sendo recomendável que cada um deles não ultrapasse o tamanho de 64 KB e optando pela reutilização das instâncias, desde que isso não incorra em vazamento de dados sensíveis. Brenner *et al.* (2018) apresenta as diretrizes para o particionamento de aplicações, levando em consideração os aspectos de desempenho e segurança, para a manipulação de dados sensíveis em ambiente confiável.

Para reduzir o impacto de desempenho causado pelas trocas de contexto, algumas soluções propõem fornecer uma interface de comunicação entre o enclave e o aplicativo sem a necessidade de tais trocas, como o uso de *threads* em execução fora do enclave para receber as solicitações e depois executar as chamadas de sistema necessárias. A comunicação pode ser feita através de chamadas de procedimento remoto (RPC), como apresentado por Orenbach *et al.* (2017), ou utilizando uma memória compartilhada, alternativa apresentada por Arnautov *et al.* (2016) e Weisse *et al.* (2017). A melhoria de desempenho das chamadas de sistema por enclaves

sem troca de contexto é investigada por Tian *et al.* (2018), buscando efetuar a integração de tal recurso no Intel SGX SDK.

Também são utilizados outros paradigmas em busca de reduzir os impactos causados na comunicação e sincronização de enclaves, que são necessários em aplicações que utilizam diversos enclaves para realizar suas operações. Um paradigma utilizado é o modelo de atores, onde cada ator é auto-contido e se comunica com outros atores através de troca de mensagens, permitindo a execução de operações paralelas e não bloqueantes. Tal paradigma é utilizado por Sartakov *et al.* (2018), tratando cada enclave como um ator e reduzindo a necessidade de trocas de contexto e sincronização entre enclaves em execuções paralelas.

Além da interação entre enclaves e da redução na ocorrências de trocas de contexto para execução de rotinas dentro ou fora do enclave, outro ponto que merece atenção é a forma como os enclaves são gerenciados pelas aplicações que os detêm, visto que há um elevado custo computacional na sua criação, além da reduzida disponibilidade de memória protegida aos enclaves, devido às restrições de tamanho da PRM. Para tanto, busca-se maneiras eficientes para a criação e utilização de enclaves, a fim de reduzir os custos computacionais inerentes.

6.2 MODELOS DE GERENCIAMENTO DE ENCLAVES

A forma como a aplicação gerencia os enclaves que utiliza pode impactar em seu desempenho, sendo importante a utilização de técnicas de gerenciamento eficientes para minimizar tal impacto. Além disso, como já mencionado anteriormente, as limitações impostas pela arquitetura SGX na utilização da PRM podem fazer com que esse impacto seja acentuado, devido às necessidades de paginação para permitir a execução de um grande número de enclaves. Assim, modelos de gerenciamento que visam reduzir e número de instanciações e de enclaves ativos no sistema, através do compartilhamento ou reutilização desses, podem trazer uma redução significativa no impacto de desempenho causado pela utilização de enclaves. Dessa forma, são propostos a seguir dois modelos de gerenciamento de enclaves que norteiam a definição de um serviço provedor de enclaves, o qual será apresentado na Seção 6.4.

6.2.1 Compartilhamento de Enclaves

Em algumas situações, pode-se ter várias instâncias de um mesmo enclave, que estão executando a mesma função e utilizando o mesmo conjunto de dados. Tal cenário pode ser encontrado, por exemplo, no trabalho proposto por Condé *et al.* (2018), onde cada requisição por autenticação cria uma nova instância do enclave para efetuar a leitura do arquivo de credenciais. Já no trabalho apresentado por Karande *et al.* (2017), que poderia ter uma situação semelhante, é utilizada uma arquitetura cliente/servidor, centralizando as operações com os enclaves.

O problema de ter diversas instâncias de um mesmo enclave manipulando os mesmos dados pode ser recorrente, não somente em situações onde se utilizam arquivos de credenciais ou *logs*, mas informações centralizadas de qualquer sistema computacional, principalmente em um servidor que recebe um grande número de requisições.

Uma alternativa para reduzir o número de enclaves instanciados é permitir que tais instâncias sejam compartilhadas por diversos processos em execução, possibilitando que uma mesma instância de um enclave responda às requisições de diferentes fontes, sem a necessidade de criar uma nova instância para atender àquela requisição. Essa solução também possibilita que o enclave mantenha, quando possível, os dados necessários na memória principal, visando reduzir o tempo de resposta para a aplicação requisitante.

O compartilhamento de enclaves permite que uma mesma instância de um enclave seja utilizada por diversas aplicações ou diversas instâncias de uma mesma aplicação, eliminando a

necessidade de cada instância da aplicação criar uma instância de um enclave. O compartilhamento de enclaves visa reduzir a quantidade de memória necessária para a execução desses. Outro ponto é que, mantendo uma instância do enclave ativa para ser utilizada por outras aplicações posteriormente, o tempo de inicialização do enclave, a partir da segunda requisição, pode ser desprezado, o que traz um aumento de desempenho médio, quando toma-se uma média de várias requisições para o mesmo enclave.

A utilização de um enclave compartilhado entre várias aplicações traz benefícios consideráveis para a execução de tais aplicações, principalmente no que diz respeito ao desempenho na execução de tarefas sensíveis que se utilizam desses enclaves. A utilização de um único enclave contendo os dados sensíveis a serem manipulados reduz drasticamente o processamento necessário na instanciação do enclave e no carregamento dos dados e, por consequência, o tempo gasto na execução da tarefa, quando levado em consideração um intervalo de tempo com várias requisições. Isso se deve ao fato de que, a partir da segunda requisição, o enclave já estará instanciado e, possivelmente, os dados necessários já estarão carregados no enclave.

Além disso, em um cenário onde há expectativa de que várias instâncias de aplicações estejam executando e manipulando os mesmos dados sensíveis em um determinado intervalo de tempo, a aplicação do modelo de compartilhamento de enclaves também reduzirá o percentual de utilização da PRM. Tal redução ocorre em virtude da utilização de uma única instância do enclave contendo os dados a serem manipulados, em vez de de várias instâncias de um mesmo enclave tendo os mesmos dados carregados em memória.

Shen *et al.* (2018) também enfatiza que o modelo proposto pelo SGX reforça um isolamento entre os enclaves, mas que, em muitos casos, o compartilhamento de dados entre enclaves é desejável, podendo até ser necessário, com o compartilhamento de código entre enclaves podendo reduzir a utilização da PRM. A troca de dados contidos em enclaves também pode se fazer necessária em um ambiente onde diversas aplicações são executadas de forma a colaborar umas com as outras, e podendo compartilhar seus segredos entre si. Tal tarefa é dificultada pelo modelo de programação padrão para enclaves, podendo ser utilizado um gerenciamento centralizado para compartilhar um único enclave, ou um conjunto de enclaves, com diversas aplicações que necessitam deles.

O compartilhamento de enclaves é adotado por Will e Maziero (2020) para centralizar a manipulação de um arquivo de credenciais em um sistema de autenticação. O modelo proposto pelos autores, com a utilização de uma arquitetura cliente/servidor, possibilitou a aplicação das garantias de segurança do SGX sem grandes impactos de desempenho.

6.2.2 *Pool* de Enclaves

Em um ambiente com um alto número de requisições por um recurso (neste contexto, um enclave), e que se faz necessário que este recurso tenha garantida a sua exclusão mútua, pode-se pensar em algumas alternativas para reduzir o tempo necessário para disponibilizar tal recurso. Uma dessas alternativas é o *pooling pattern*, que é definido por Kircher e Jain (2002) como um padrão que "descreve como a aquisição e a liberação de recursos dispendiosas podem ser evitadas através da reciclagem dos recursos que não são mais necessários". Tal padrão pode ser utilizado em um contexto onde sistemas continuamente adquirem e liberam recursos do mesmo tipo ou de tipo semelhante e precisam atender a altas demandas de escalabilidade e eficiência, ao mesmo tempo em que busca garantir que o desempenho e a estabilidade do sistema não sejam comprometidos.

O *pooling pattern* gerencia várias instâncias de um tipo de recurso em um *pool*, permitindo que esses recursos sejam adquiridos, utilizados, e então liberados para retornar ao *pool*. Para garantir a eficiência do sistema, esse *pool* mantém um número estático de recursos

instanciados, e pode criar novas instâncias quando a demanda aumenta. Tal padrão é utilizado em alguns domínios específicos, como o *connection pool* para gerenciar conexões com bancos de dados, amplamente utilizado em servidores Web. Outra aplicação é o *thread pool*, que mantém um determinado número de *threads* instanciadas para aumentar o desempenho do sistema e evitar a latência na execução devido à criação e destruição frequentes de *threads* para tarefas de curta duração. Destaca-se também o *object pool* que gerencia a reutilização de objetos de um tipo específico, os quais são custosos para instanciar ou apenas um número limitado desse tipo pode ser instanciado (Kircher e Jain, 2002).

A ideia chave do *object pool* vai de encontro com a limitação imposta pela arquitetura SGX no que diz respeito ao tamanho da PRM, que acaba por limitar o número de instâncias de enclaves que podem residir na memória em um mesmo intervalo de tempo, bem como com o custo na inicialização dos enclaves, que cresce proporcionalmente ao tamanho desses. Manter um *pool* de enclaves pode diminuir o tempo de resposta de requisições Web ou locais com demandas frequentes a enclaves, situação que pode ocorrer em determinadas implementações, como na solução apresentada por Brenner *et al.* (2017).

A utilização de um *pool* de enclaves é abordada por Li *et al.* (2019), sendo aplicada em um ambiente de requisições via Web. Na solução apresentada pelos autores, cada enclave no servidor está vinculado a um *thread* que é responsável pela comunicação direta com o enclave, evitando problemas decorrentes de condições de corrida dentro de enclaves, além de uma fila de requisições para distribuir cada requisição a um enclave do *pool*. As tarefas a serem executadas em enclaves são enfileiradas e associadas a enclaves disponíveis logo que possível, mantendo um controle sobre o fluxo de execução. A utilização de um *pool* de enclaves possibilita, além da redução no impacto decorrente da criação do enclave a cada nova requisição, uma redução na utilização da CPU, em comparação com o modelo padrão de utilização de enclaves.

A principal diferença entre um *pool* e um compartilhamento de enclaves é a garantia de que na primeira solução um enclave não será disponibilizado para dois ou mais processos ao mesmo tempo, o que é essencial quando os dados contidos dentro do enclave não podem ser compartilhados entre diferentes processos em execução. Outra distinção é o fato de um *pool* manter um número pré-determinado de enclaves instanciados, aguardando requisições para utilizar esses enclaves.

6.3 O MODELO DE ENCLAVE COMO SERVIÇO

O enclave pode ser entendido como um componente para a execução confiável de alguma rotina sensível, e que está diretamente vinculado à aplicação que o utiliza. Por outro lado, o enclave pode também ser considerado como um prestador de serviços para a aplicação, na execução de rotinas confiáveis e pela disponibilização de um ambiente encapsulado e protegido para a manipulação de dados sensíveis.

O conceito de oferecer recursos ou rotinas como serviço é bastante difundido em ambientes de computação na nuvem, onde é denotado pelo termo *Everything as a Service* (XaaS). Esse conceito permite a integração entre aplicações heterogêneas, com os recursos sendo empacotados em serviços que são acessados pelas aplicações clientes. Os serviços são itens fundamentais, e totalmente independentes das aplicações, permitindo a sua utilização sob demanda (Robison, 2008; Li e Wei, 2014).

Em ambientes de nuvem, existem diversos recursos que são entregues às aplicações na forma de serviços, como recursos de software, englobado pelo conceito de *Software as a Service* (SaaS), e recursos de hardware, com o conceito de *Infrastructure as a Service* (IaaS). No contexto de segurança computacional, também é apresentado o modelo *Security as a Service* (SECaaS), o

qual provê, na forma de serviços, mecanismos de autenticação, detecção de intrusões, verificação de software malicioso, entre outros. Uma ampla classificação de tais serviços é apresentada por Duan *et al.* (2015a) e Duan *et al.* (2016). Apesar de ser bastante difundido e utilizado em ambientes de nuvem, o paradigma XaaS não está restrito a esses ambientes, podendo também ser utilizado localmente, através da utilização de processos do tipo *daemon* para atender a requisições de usuários do sistema ou de outros processos (Duan *et al.*, 2015b).

O conceito chave do paradigma XaaS, de desvincular os recursos das aplicações, pode ser aplicado à utilização de enclaves, oferecendo-os como um serviço às aplicações. Assim, pode-se utilizar um paradigma *Enclave as a Service* (EaaS), que é caracterizado por um conjunto de enclaves independentes que são disponibilizados a diversas aplicações para a execução de rotinas confiáveis. Tal conceito já é parcialmente explorado com a execução de microsserviços seguros, descritos na Seção 4.7.4, que oferecem um ambiente confiável para a execução de pequenos serviços na nuvem.

Além disso, a utilização de enclaves no paradigma XaaS também é explorada por Alder *et al.* (2019), integrando a arquitetura SGX em um ambiente *Function as a Service* (FaaS), permitindo a execução de funções e a computação da utilização de recursos de maneira segunda dentro de um enclave. Outra proposta apresenta o conceito de atestação remota como um serviço (*Remote Attestation as a Service* - RAaS), possibilitando clonar um dispositivo IoT em um ambiente de nuvem, permitindo realizar as custosas operações de atestação em nome do dispositivo real (Conti *et al.*, 2019). Tal abordagem permite reduzir o número de operações custosas a serem realizadas pelo dispositivo, permitindo assim uma redução no consumo energético do dispositivo e também mais agilidade nas operações, haja vista as capacidades limitadas de processamento destes.

A presente proposta visa fornecer rotinas pré-estabelecidas a um conjunto de aplicações, sem a necessidade de tais aplicações enviarem qualquer tipo de código ao provedor de serviços, bastando apenas enviar os dados necessários para processar a requisição. Assim, é proposta a utilização do paradigma EaaS em um escopo local, provendo uma gama de diferentes enclaves a diversas aplicações, com toda a execução e troca de mensagens entre os participantes ocorrendo em uma única plataforma. Os enclaves podem ser providos utilizando os modelos de gerenciamento descritos na Seção 6.2, com o ciclo de vida de cada um deles sendo gerido por um provedor de enclaves.

6.4 PROVEDOR DE ENCLAVES

Após serem definidos os modelos de gerenciamento de enclaves a serem utilizados (Seção 6.2) e a forma como os enclaves serão disponibilizados para as aplicações (Seção 6.3), propõem-se a definição de uma solução que contemple tais características, com os detalhes arquiteturais sendo apresentados nesta seção.

6.4.1 Modelo de Ameaças

Primeiramente, o projeto do provedor de enclaves deve levar em consideração diversos aspectos de segurança e conter proteções contra possíveis atacantes, seja para evitar que esses monitorem os dados trafegados ou que adulterem informações sensíveis ao funcionamento do provedor.

O modelo de ameaças que baliza a presente proposta leva em consideração que o atacante pode monitorar o mecanismo de comunicação entre as aplicações, podendo coletar os dados trocados entre elas. Assim, os dados sensíveis trocados entre as aplicações e o provedor

de enclaves devem ter a sua confidencialidade garantida. O controle total do mecanismo de comunicação por parte do atacante, permitindo que este altere ou substitua os dados trafegados, não é considerado.

Os enclaves gerenciados pelo provedor são previamente registrados, conforme será apresentado na Seção 6.4.3, e o atacante pode ter acesso ao arquivo de registro e adulterar seu conteúdo ou substituí-lo. Nesse caso, mecanismos de proteção devem ser aplicados para garantir a confidencialidade e integridade desse conteúdo. Além disso, o atacante pode efetuar a substituição dos arquivos de código dos enclaves registrados, o que faz necessário o uso de mecanismos para validação de tais arquivos antes de carregá-los.

Por fim, estruturas de dados internas do provedor de enclaves podem ser manipuladas por um atacante, visando mapear requisições de aplicações legítimas e encaminhá-las a enclaves maliciosos. Tais estruturas devem ser mantidas em uma área protegida, evitando que sejam adulteradas deliberadamente em memória.

6.4.2 Arquitetura Geral da Solução

A utilização de um enclave está restrita ao processo que o instanciou, com os demais processos tendo o acesso negado ao tentar fazer requisições a um enclave de terceiros. Assim, para permitir que um determinado enclave seja utilizado por diversas aplicações, de forma simultânea ou coordenada, é necessária a inclusão de um mediador entre as aplicações requisitantes e os enclaves provedores de serviços.

Tal mediador, ou provedor de enclaves, tem como objetivo gerenciar o ciclo de vida dos enclaves, desonerando as aplicações de tal tarefa, além de prover meios para que as aplicações possam se comunicar com os enclaves disponíveis de forma segura. O provedor de enclaves gerencia, de forma coordenada, a requisição de N aplicações para M enclaves, obedecendo os parâmetros contidos em cada requisição, os quais são apresentados na Seção 6.4.4. Para tanto, o provedor de enclaves centraliza todas as requisições provindas das aplicações em um *broker*, que é encarregado de verificar os parâmetros da requisição e encaminhá-la ao enclave correspondente, além de encaminhar a resposta da execução à aplicação requisitante. A arquitetura da proposta, em alto nível, é mostrada na Figura 6.1.

As informações sensíveis para a configuração e gerenciamento do provedor são mantidas seladas, e manipuladas dentro de um *enclave gerente*, evitando que as mesmas sejam modificadas indevidamente e garantindo a sua integridade. Tais informações incluem a lista de enclaves registrados no provedor, com suas respectivas configurações, e a relação de aplicações e enclaves ativos e os canais de comunicação estabelecidos entre eles. Os detalhes de projeto da arquitetura são descritos nas seções seguintes.

6.4.3 Registro de Enclaves

O primeiro passo para que um enclave esteja disponível para que seja utilizado pelas aplicações através do provedor de enclaves é o seu registro na plataforma. O registro do enclave é necessário para que o provedor tome conhecimento do enclave e de suas configurações adicionais, para que possa instanciá-lo de forma adequada quando esse for requisitado.

A partir do registro do enclave no provedor, o enclave passará a ser identificado por um nome único, que será utilizado pelas aplicações para efetuar as requisições a este enclave. É nessa etapa, que são definidos os critérios para o gerenciamento do enclave, identificando se o enclave será gerenciado na forma de um *pool* ou de forma compartilhada entre as aplicações.

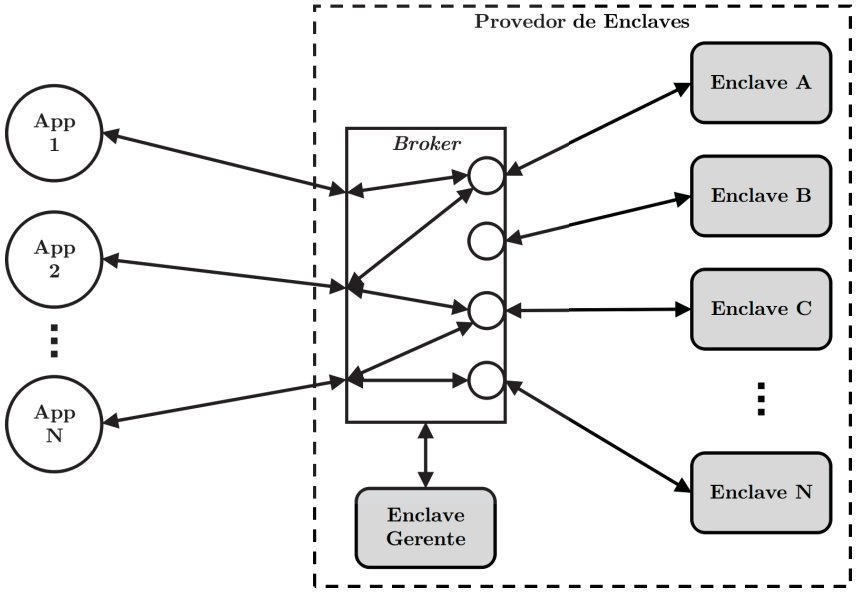


Figura 6.1: Arquitetura geral do provedor de enclaves, contendo diversas aplicações comunicando-se com enclaves distintos, com a comunicação mediada por um *broker*, que trata as requisições, encaminha para o enclave correspondente, e retorna o resultado à aplicação requisitante.

6.4.4 Comunicação com Enclaves

Para que uma aplicação possa utilizar um enclave disponibilizado pelo provedor, ela precisa ter algum meio de requisitar uma instância desse enclave. Esse processo é descrito na Figura 6.2, e é composto pelas seguintes etapas:

1. A aplicação inicia um processo de acordo de chaves com o provedor de enclaves, visando estabelecer uma chave secreta entre ambas as partes para criptografar os dados enviados. Ao final dessa etapa, tanto a aplicação quanto o provedor detêm uma chave secreta, SK_1, a qual será utilizada nas comunicações seguintes entre essas partes;

2. É efetuado um acordo de chaves entre a aplicação e o enclave de destino para estabelecer uma chave secreta (SK_2) para a comunicação entre a aplicação e o enclave, com essa operação sendo mediada pelo provedor de enclaves;

3. A primeira etapa dessa comunicação, entre a aplicação e o provedor, já é cifrada com a chave SK_1 estabelecida anteriormente, com o provedor decifrando o conteúdo do pacote e coletando as informações necessárias para a identificação do enclave requisitado;

4. A requisição é encaminhada ao enclave correspondente para prosseguir com o estabelecimento das chaves;

5. As demais comunicações entre aplicação e enclave são efetuadas com criptografia fim-a-fim, com os dados necessários ao provedor sendo cifrados com a chave SK_1, e os dados que serão encaminhados ao enclave cifrados com a chave SK_2;

6. O provedor utiliza a chave SK_1 para decifrar o cabeçalho da requisição, a fim de identificar o enclave a que esta se destina;

7. O *payload* da requisição é enviado à função ECALL requisitada pela aplicação cliente.

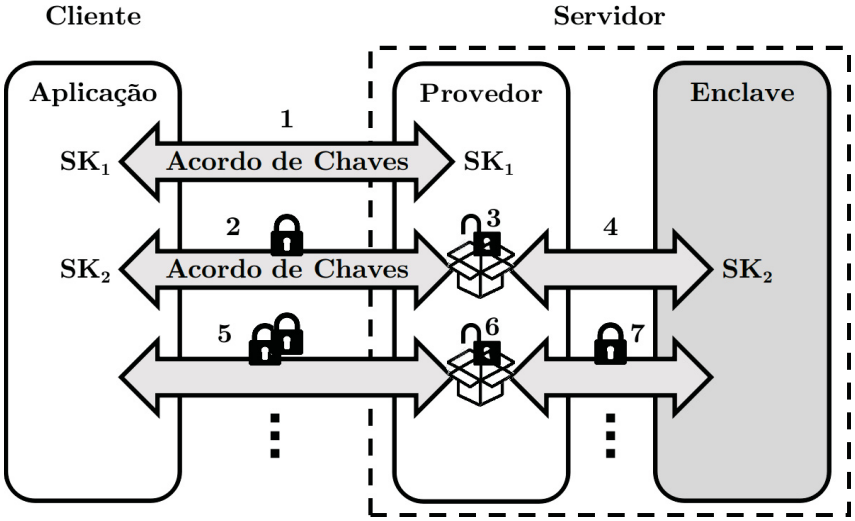


Figura 6.2: Comunicação entre a aplicação e o enclave, mediada pelo provedor de enclaves, contendo criptografia fim-a-fim.

A decifragem de parte dos dados recebidos no provedor de enclaves é essencial, visto que esse deve tomar conhecimento de qual enclave está sendo requisitado e qual método deve ser invocado. Assim, estabelece-se um protocolo de comunicação utilizando o empacotamento de dados, à semelhança do que é feito na pilha de rede. Os dados a serem enviados ao enclave compõem um *payload*, o qual será cifrado utilizando a chave SK_2, mantendo o sigilo de tais dados mesmo para o provedor. Esse *payload* será então encapsulado em um pacote, juntamente com os demais dados necessários para que o provedor identifique a requisição, com esse pacote sendo cifrado com a chave SK_1, permitindo que o provedor decifre tais dados e proceda com a requisição. É esperado que o provedor tenha conhecimento da identidade do processo que efetuou a requisição, podendo ser utilizado o *process identifier* (PID) deste; o identificador do enclave requisitado (EID) que é recebido pela aplicação ao final do acordo de chaves, sendo uma identidade única para a comunicação entre a aplicação e o enclave; o nome da função ECALL a ser executada pelo enclave e; o tamanho do *payload* em *bytes*. Esse conjunto de dados permitirá que o provedor efetue a chamada ao enclave solicitado e encaminhe a resposta à aplicação requisitante, a qual conterá um *payload* com os dados retornados pelo enclave, quando for o caso, e um *status* indicando o sucesso ou a ocorrência de algum erro na execução da operação solicitada.

6.4.5 Ciclo de Vida do Enclave

Além de mediar a comunicação entre as aplicações e os enclaves disponíveis, o provedor de enclaves também fica responsável por gerenciar o ciclo de vida dos enclaves registrados, buscando manter a eficiência na utilização dos recursos necessários por cada enclave. Do ponto de vista da aplicação requisitante, a utilização de enclaves mediada pelo provedor se assemelha à utilização de um enclave de forma direta pela própria aplicação, mantendo as mesmas etapas de solicitação do enclave, chamadas ECALL e liberação do enclave.

Na primeira etapa, a aplicação solicita ao provedor uma instância de determinado enclave para ser utilizada, o que ocorre de forma concomitante ao acordo de chaves entre a aplicação e o provedor. A instância do enclave poderá ser exclusiva para a aplicação requisitante, no caso de um *pool* de enclaves, ou compartilhada, sendo essas características especificadas no momento do registro do enclave no provedor. Nessa chamada, a aplicação deve fornecer o

identificador do enclave registrado no provedor e os dados necessários para iniciar o acordo de chaves com o provedor. Caso o enclave solicitado esteja registrado e disponível para utilização, o provedor responderá com os dados para finalizar o acordo de chaves e estabelecer a chave SK_1. Caso contrário, será retornado um código de erro indicando o problema ocorrido. Essa etapa é análoga à chamada do método `sgx_create_enclave`, que uma aplicação efetua para criar uma instância de enclave.

O processo detalhado de requisição de um enclave é mostrado no fluxograma da Figura 6.3. Primeiramente o gerenciador verifica se já existe alguma instância ativa do enclave requisitado. Caso não exista alguma instância ativa, o gerenciador iniciará o processo de criar uma nova instância do enclave. Caso tenha alguma instância ativa, é verificado se o número máximo de aplicações aptas a compartilhar uma mesma instância daquele enclave já foi atingida. Se esse número ainda não foi atingido, o gerenciador retornará para a aplicação a instância existente, caso contrário, uma nova instância do enclave será criada para atender à requisição.

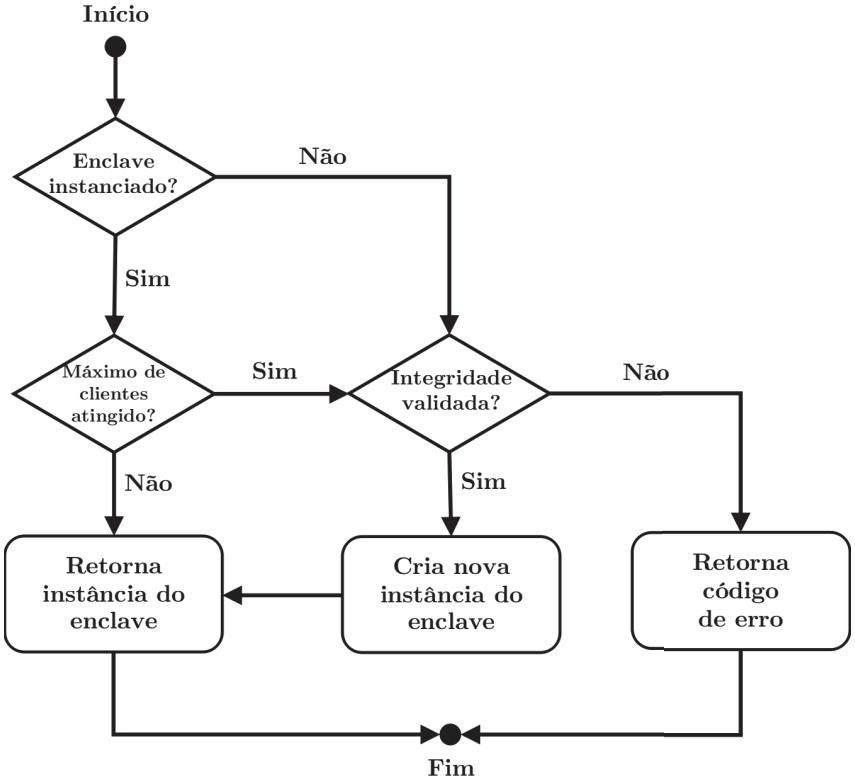


Figura 6.3: Fluxograma seguido para a responder a uma requisição de enclave.

O processo de criação de uma nova instância de enclave se inicia com a verificação estática do enclave, com o intuito de verificar se o arquivo não sofreu modificações desde que foi registrado no provedor. Caso essa verificação falhe, é retornado um código de erro para a aplicação indicando que o arquivo do enclave sofreu alterações ou está corrompido. Caso contrário, o gerenciador cria uma nova instância do enclave, através da chamada da função `sgx_create_enclave`, e disponibiliza essa instância para ser utilizada pela aplicação requisitante.

A execução de chamadas ECALL ocorre da maneira descrita na Seção 6.4.4, com os dados encapsulados e cifrados sendo enviados ao provedor, que irá decifrar e ler os dados necessários para identificar o enclave e função requisitados, efetuando a chamada requisitada e encaminhando os dados de retorno à aplicação. Todo o processo de requisição e utilização de

uma instância de enclave gerenciada pelo provedor é demonstrado no fluxograma da Figura 6.4, sendo tal processo é muito semelhante à utilização de um enclave diretamente pela aplicação.

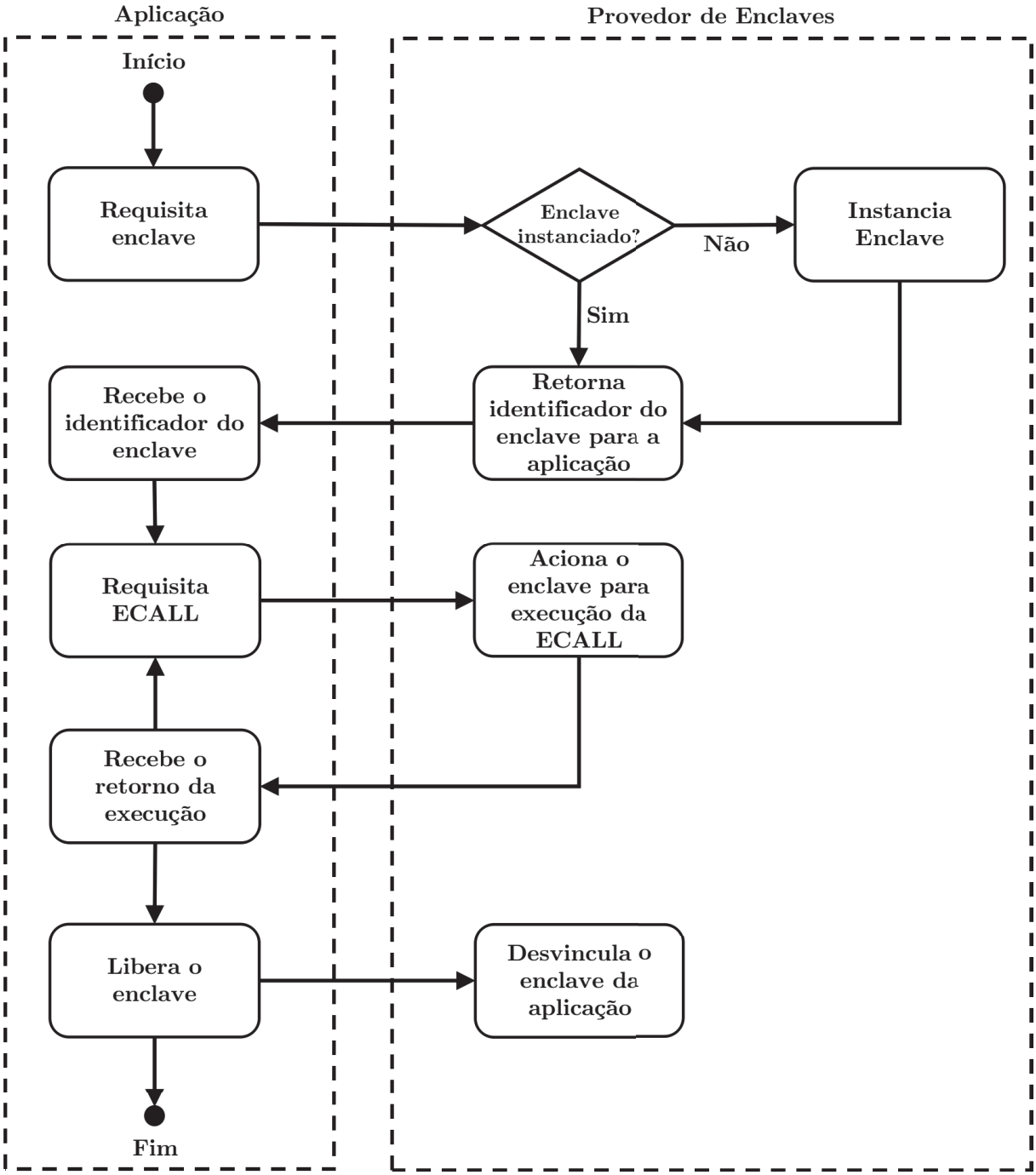


Figura 6.4: Fluxograma demonstrando a sequência de execução de uma aplicação utilizando os serviços providos pelo gerenciador de enclaves.

O gerenciador de enclaves também dispõe de uma estrutura que mantém uma lista de todos os enclaves instanciados e quais aplicações estão utilizando essas instâncias, sendo que cada aplicação recebe um identificador único para cada enclave que está utilizando (EID). Dessa forma, é possível validar a quantidade de aplicações clientes que estão utilizando determinada instância de enclave e também validar requisições posteriores das aplicações para a execução de chamadas ECALL.

6.4.6 Liberação de Enclaves

Após a utilização do enclave, quando esse não é mais necessário, a aplicação que o requisitou deverá sinalizar ao provedor que não mais necessita do enclave. Essa operação é análoga à chamada da função `sgx_destroy_enclave` e, para isso, a aplicação efetua uma chamada ao provedor de enclaves fornecendo o identificador que ela detém.

Ao receber a requisição para a liberação de um enclave, por parte da aplicação que o estava utilizando, o provedor de enclaves remove a entrada correspondente àquela aplicação na sua lista de enclaves instanciados, e essa instância do enclave continuará disponível para ser utilizado por outras aplicações. Mesmo que não exista outra aplicação utilizando a mesma instância do enclave, essa instância ainda é mantida, estando apta a atender novas requisições. Caso o enclave em questão esteja configurado para atender apenas uma aplicação por vez, isto é, evitando o compartilhamento do mesmo, o provedor notifica-o para que quaisquer dados sensíveis sejam excluídos, para evitar o vazamento de tais dados quando esta mesma instância de enclave for reutilizada por outra aplicação.

O provedor de enclaves está constantemente verificando a utilização dos enclaves instanciados e, caso exista alguma instância inutilizada por um longo período de tempo, essa instância é então removida, com o intuito de liberar espaço na PRM. Além disso, o provedor de enclaves deve garantir uma utilização responsável dos enclaves e, dessa forma, caso alguma aplicação tenha solicitado uma instância de enclave e, após um longo período de tempo, não está mais fazendo uso da mesma, o provedor também se encarrega de remover a sua entrada na lista de instâncias e, caso não haja nenhuma outra aplicação utilizando aquela instância de enclave, a mesma é então removida.

6.5 CONSIDERAÇÕES FINAIS

Este capítulo discutiu as formas de interação entre aplicação e enclave e entre diferentes enclaves que compõem um mesmo ecossistema, ponderando as dificuldades e custos computacionais implícitos na troca de dados em tais interações. Considerando também os impactos e custos descritos no capítulo anterior, apresenta-se a discussão de modelos eficientes para o gerenciamento de enclaves (compartilhamento e *pool*), visando reduzir a utilização de recursos computacionais, como a memória cifrada reservada do SGX, e os custos na inicialização de enclaves, que oferece grande impacto de desempenho.

A proposta de compartilhamento de enclaves foi elaborada e detalhada no desenvolvimento deste trabalho, já o *pool* de enclaves foi descrito por Li *et al.* (2019) e também utilizado por Wang *et al.* (2019b), sendo ambos os casos voltados para aplicações específicas. O presente trabalho visa prover tais modelos de gerenciamento de enclaves de uma maneira mais abrangente, permitindo a criação de um ecossistema composto por aplicações e enclaves heterogêneos, onde tais enclaves possam ser acessados e utilizados por essas aplicações de forma dinâmica.

Assim, é apresentada a proposta de um provedor de enclaves, responsável por gerenciar o ciclo de vida de diversos enclaves simultaneamente, disponibilizando os mesmos para diversas aplicações clientes, as quais requisitam a utilização dos enclaves. Toda a arquitetura do provedor de enclaves é descrita em alto nível, dando suporte para a implementação da solução e dos detalhes técnicos de protocolos, os quais serão apresentados no capítulo que segue.

7 VALIDAÇÃO DA PROPOSTA

Após a definição da arquitetura geral para um provedor de enclaves, foi definida a implementação de um protótipo de tal serviço, com o intuito de validar a proposta e analisar os possíveis impactos causados por ela, tanto do ponto de vista de desempenho quanto de segurança. Assim, este capítulo apresenta os detalhes técnicos do protótipo implementado, sendo uma prova de conceito das premissas apresentadas no capítulo anterior, e também análises quantitativas e qualitativas do referido protótipo, fazendo uma comparação com o modelo padrão de utilização de enclaves.

7.1 IMPLEMENTAÇÃO DO PROTÓTIPO

A implementação do protótipo seguiu as especificações apresentadas no Capítulo 6, sendo o objetivo da presente seção apresentar os detalhes técnicos de tal implementação. Como prova de conceito, optou-se pela construção de um provedor local de enclaves, o qual deverá fornecer às aplicações acesso a enclaves disponíveis na mesma máquina, com alternativas e outras propostas de implementação sendo discutidas na Seção 8.4.

O provedor de enclaves foi construído utilizando a linguagem de programação C, com o intuito de obter melhor desempenho de execução e uma interação mais direta com os enclaves gerenciados, através do SDK (*Software Development Kit*) disponível para o SGX. Para a comunicação entre as aplicações clientes e o provedor, utilizou-se o *framework* D-BUS (freedesktop.org, 2018).

7.1.1 Registro de Enclaves

O registro do enclave requer, além do próprio arquivo do enclave, um arquivo de manifesto que deverá conter os parâmetros de configuração para o registro, conforme apresentado na Listagem 7.1.

O elemento `FileName` contém o nome do arquivo do enclave, o qual deve ser compilado e assinado conforme as especificações contidas no *Intel Software Guard Extensions Developer Reference* (Intel, 2016b), e deve ser obrigatoriamente fornecido. O segundo elemento, `RegistrationName`, também obrigatório, contém o nome ao qual o enclave será referenciado no provedor de enclaves. O nome de registro será utilizado posteriormente para solicitar instâncias desse enclave, sendo que esse nome deve ser único na plataforma.

Listagem 7.1: Exemplo de um arquivo de configuração para o registro de enclaves.

```
<EnclaveRegistration>
      <FileName>enclave.so</FileName>
      <RegistrationName>Enclave</RegistrationName>
      <Description>Enclave description</Description>
      <EnclaveHash>xxxx</EnclaveHash>
      <MaxClients>0</MaxClients>
      <CreateOnStart>false</CreateOnStart>
      <PoolSize>0</PoolSize>
      <ECALLList>
            <ECALL>
                  <Index>0</Index>
                  <Name>ECALL_Name</Name>
            <ECALL>
      </ECALLList>
      <InitECALL>ECALL_Name</InitECALL>
      <DestroyECALL>ECALL_Name</DestroyECALL>
</EnclaveRegistration>
```

O elemento `Description` é opcional, e contém a descrição do enclave, podendo descrever os serviços que o enclave provê ou outras informações relevantes acerca do enclave que está sendo registrado. Caso nenhuma descrição seja fornecida, o registro do enclave se dará com uma descrição em branco. Já o elemento `EnclaveHash` traz um *hash* SHA-256 do arquivo do enclave, o qual será utilizado para validar o arquivo fornecido, tanto no ato do registro do enclave quanto no ato de sua instanciação, processo que será descrito na Seção 7.1.2.

Também existem outros três elementos opcionais, `MaxClients`, `CreateOnStart`, `PoolSize`, cujos valores padrões são 0, *false* e 0, respectivamente. O elemento `MaxClients` pode ser utilizado para limitar o número de aplicações clientes que uma mesma instância do enclave pode atender. Dessa forma, caso o número máximo de aplicações clientes for atingido, uma nova requisição será atendida por uma nova instância desse enclave. Qualquer valor menor ou igual a zero fornecido para esse elemento indicará que uma única instância do enclave será utilizada para atender a todas as requisições, independente de quantas forem. Já o elemento `CreateOnStart` pode ser utilizado para sinalizar que uma instância daquele enclave deve ser criada na inicialização do gerenciador de enclaves. O elemento `PoolSize`, por sua vez, indica quantas instâncias do enclave devem ser criadas e disponibilizadas para utilização.

Também se faz necessário informar quais chamadas ECALL estão disponíveis no referido enclave, através do elemento `ECALLList`, o qual apresenta todas as ECALLs públicas do enclave, definidas no arquivo EDL (*Enclave Definition Language*), trazendo o seu indexador, de acordo com a ordem em que consta no arquivo EDL, e uma *string* que será utilizada para fazer referência à ECALL. Por fim, podem ser definidas chamadas ECALL específicas para serem executadas no momento da instanciação do enclave (`InitECALL`) e destruição do enclave (`DestroyECALL`).

Caso todos os parâmetros obrigatórios sejam fornecidos, o provedor de enclaves iniciará o processo de validação do enclave para o registro. Nesse processo é verificado se há algum enclave registrado com o mesmo `RegistrationName` e, caso exista, o registro será negado. Também é verificado o *hash* fornecido, comparando-o com o *hash* gerado a partir do arquivo de enclave e, caso não sejam iguais, o registro também será negado. Caso não ocorra nenhum problema na validação do registro, o enclave é registrado no gerenciador e estará disponível para ser utilizado pelas aplicações.

7.1.2 Instanciação de Enclaves

Após o enclave estar devidamente registrado no provedor, ele pode ser instanciado e disponibilizado para as aplicações clientes, o que pode ocorrer logo na inicialização do provedor, ou sob demanda, conforme definido na propriedade `CreateOnStart` apresentada anteriormente.

O processo de instanciação de um novo enclave pelo provedor, descrito na Figura 7.1, consiste em, primeiramente, carregar o conteúdo do arquivo fonte do enclave para o enclave gerente, onde será calculado o *hash* SHA-256 de tal conteúdo para comparação com o *hash* fornecido no momento do registro do enclave. Tal verificação visa garantir que o arquivo fonte do enclave não foi adulterado ou substituído após ser registrado no provedor.

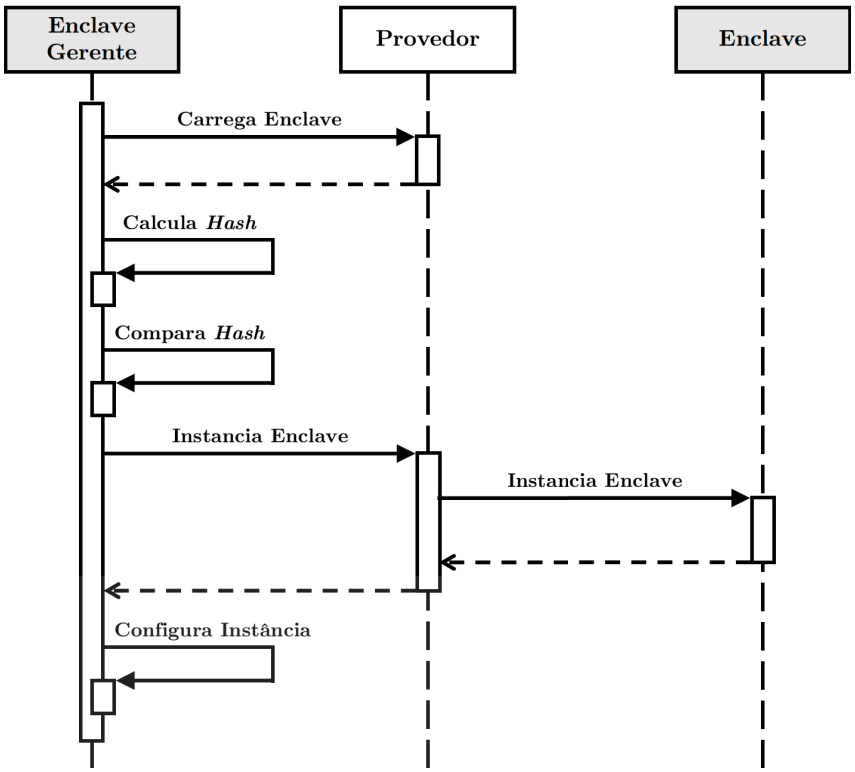


Figura 7.1: Sequência de operações realizadas na instanciação de enclaves pelo provedor.

Após a verificação do conteúdo do arquivo fonte, é iniciado o processo de instanciação do enclave, com o identificador dessa nova instância sendo encaminhado ao enclave gerente para que tal instância seja registrada e esteja disponível para as aplicações requisitantes.

7.1.3 Requisição de Enclaves

Após o enclave estar devidamente registrado no provedor, ele é disponibilizado para atender as requisições oriundas das aplicações clientes e, para que a aplicação possa fazer uso do enclave, o primeiro passo é solicitar ao provedor a permissão para utilização desse enclave. Esse processo se inicia com a criação de um canal de comunicação entre a aplicação cliente e o provedor de enclaves e, posteriormente, com um acordo de chaves entre as partes para possibilitar a transferência de dados cifrados. Todo o processo de requisição de enclave é demonstrado no diagrama da Figura 7.2.

O primeiro passo é a criação de um canal de comunicação com o provedor de enclaves, através de uma requisição pelo D-BUS. Após criar o canal de comunicação, a aplicação cliente

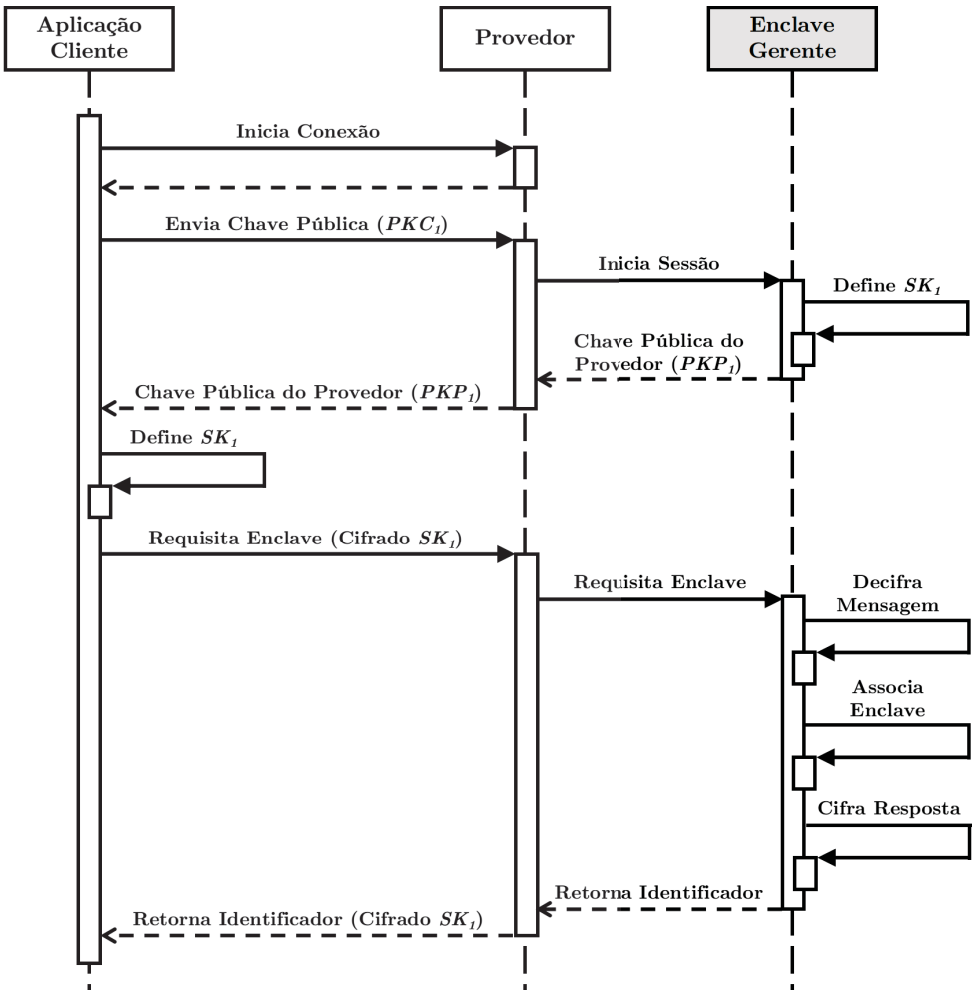


Figura 7.2: Sequência de operações realizadas para iniciar a comunicação com o provedor de enclaves e efetuar a requisição de enclaves.

deve gerar um par de chaves e enviar a sua chave pública (PKC_1) ao provedor de enclaves, o qual irá se utilizar do enclave gerente para também definir um par de chaves, definir a chave secreta (SK_1) a ser utilizada nas comunicações posteriores, e enviar a sua chave pública (PKP_1) à aplicação cliente para que essa também defina a chave secreta (SK_1).

O acordo de chaves é efetuado através de um algoritmo Diffie-Hellman de curva elíptica (ECDH), utilizando a curva X25519 proposta por Bernstein (2006), a qual foi originalmente projetada para essa finalidade e não está sob nenhuma patente, tendo sido adotada como curva padrão nas implementações do OpenSSH e também incluída na especificação TLSv1.3 (Rescorla, 2018). Para a criptografia dos dados trafegados entre a aplicação e o provedor de enclaves utiliza-se o algoritmo AES-CTR, com uma chave de 128 *bits*.

Após definida a chave secreta (SK_1), a aplicação cliente então requisita o enclave desejado, através do nome registrado no provedor (`RegistrationName`, conforme descrito na Seção 7.1.1), enviando essa informação devidamente cifrada pelo canal de comunicação. Ao receber a requisição, o provedor encaminha os dados para o enclave gerente, que então decifrará o pacote e buscará uma instância do enclave requisitado para associá-la à aplicação requisitante. Caso não haja instâncias do enclave disponíveis, será iniciado o processo de criação de uma nova instância para responder à requisição, conforme apresentado na Seção 7.1.2.

Por fim, o enclave gerente retorna um identificador de acesso ao enclave requisitado, cifrando tal informação com a chave secreta SK_1. Tal identificador é recebido pela aplicação

cliente, que deverá utilizá-lo posteriormente nas chamadas seguintes para execução de ECALLs ou liberação do enclave.

7.1.4 Execução de ECALLs

De posse de um identificador de enclave, a aplicação cliente pode efetuar requisições para a execução de rotinas ECALL do referido enclave. A requisição para a execução de uma chamada ECALL é enviada pela aplicação cliente para o provedor de enclaves, contendo os dados apresentados na Figura 7.3. O atributo `AppID` identifica a aplicação requisitante, já o atributo `EnclaveID` identifica o enclave requisitado. Os próximos dois atributos apresentam o tamanho do *payload* de entrada e saída, respectivamente, em *bytes*.

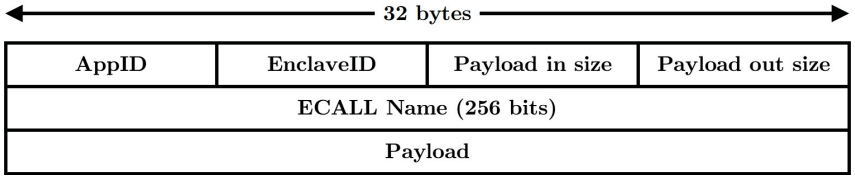


Figura 7.3: Pacote de dados enviado ao provedor de enclaves para a execução de uma chamada ECALL.

Também, é necessário informar o nome da ECALL que está sendo requisitada, sendo um nome válido e informado no momento do registro do enclave, conforme apresentado na Seção 7.1.1. Por fim, o pacote contém o *payload* de entrada, contendo os dados necessários para a execução da chamada, que será enviado para a ECALL requisitada.

Ao receber a requisição para a execução de uma ECALL, o provedor de enclaves encaminha o pacote de dados para o enclave gerente, o qual será responsável por decifrar e verificar os dados contidos no pacote, e posteriormente buscar qual instância do enclave requisitado está associada à aplicação requisitante. Após isso, o enclave gerente retorna ao provedor o identificador da instância que irá responder a requisição e o *payload* de entrada para a execução da ECALL, e o provedor fará então a chamada da função ECALL no enclave correspondente. Esse retorno ao provedor de enclaves se faz necessário já que não é possível que um enclave faça uma chamada ECALL diretamente para outro enclave.

Após a execução da chamada ECALL, o enclave requisitado retorna o resultado da execução para o provedor de enclaves, que então efetua uma chamada ao enclave gerente para cifrar a resposta, utilizando a chave secreta SK_1, e então encaminha a resposta para a aplicação requisitante. O diagrama da Figura 7.4 apresenta a sequência de operações realizadas na requisição de uma chamada ECALL através do provedor de enclaves.

Os dados trafegados entre a aplicação cliente e o enclave requisitado (*payload* de entrada e *payload* de saída) podem ser cifrados obedecendo parâmetros definidos por ambos, com o provedor de enclaves não tendo nenhuma influência para a escolha do algoritmo de criptografia ou do tamanho da chave a ser utilizada. O acordo de chaves entre a aplicação e o enclave também não está limitado ao ECDH, podendo ser utilizado qualquer algoritmo disponível para tal tarefa. Além disso, o acordo de chaves entre a aplicação cliente e o enclave requisitado pode ser efetuado através de requisições de chamadas ECALL disponíveis no enclave requisitado, podendo ser definida uma chave para ser utilizada em todas as requisições de ECALLs subsequentes, ou uma chave efêmera válida exclusivamente para uma chamada ECALL.

A definição de uma segunda chave secreta, entre a aplicação cliente e o enclave requisitado por ela, visa garantir a confidencialidade dos dados trafegados entre ambos, evitando que o provedor de enclaves tenha acesso a tais dados em claro. As implicações de tal procedimento serão debatidas em mais detalhes na Seção 7.2.

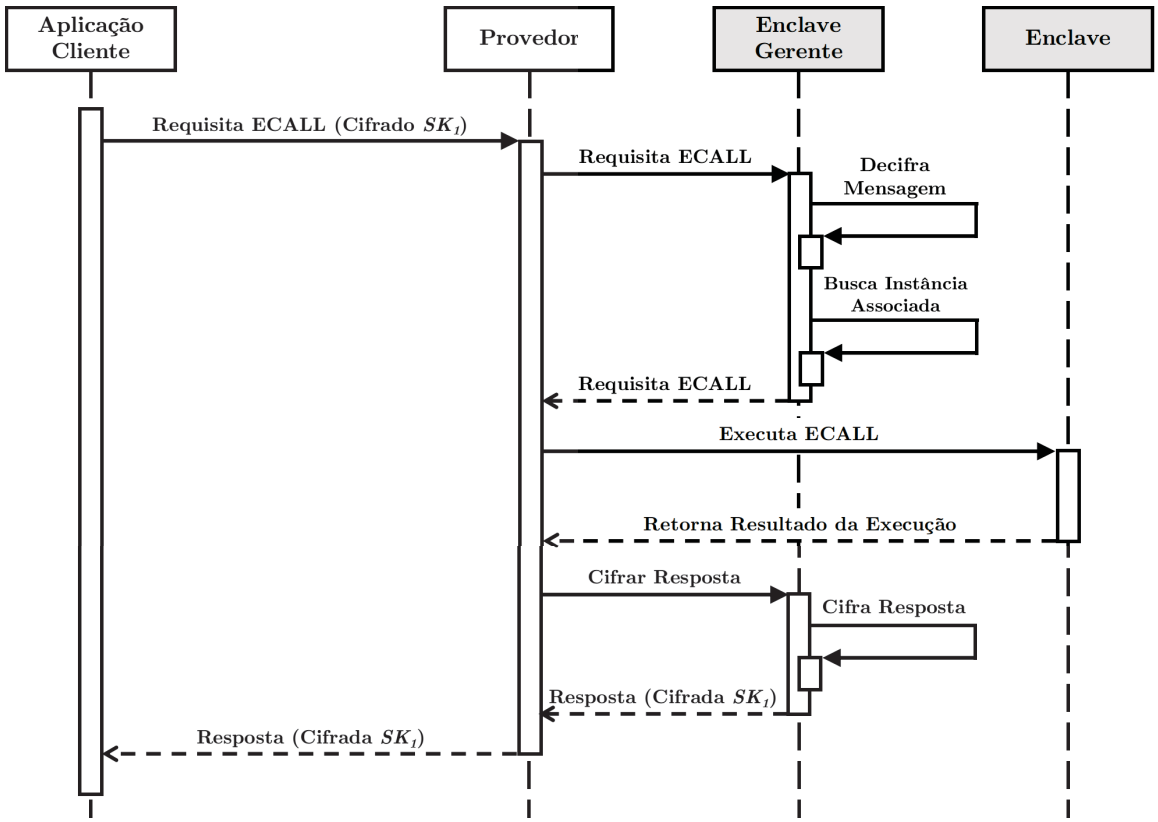


Figura 7.4: Sequência de operações realizadas na execução de chamadas ECALL do enclave requisitado.

7.1.5 Execução de OCALLs

Conforme já mencionado, a execução de instruções dentro do enclave está limitada ao *ring* 3. Assim, os enclaves precisam de uma interface de comunicação para executar chamadas de sistemas e outras rotinas em nível de *kernel*. Para possibilitar a execução de tais rotinas, o provedor de enclaves disponibiliza uma interface para a execução de chamadas OCALL, garantindo ao enclave meios de acessar recursos externos, como *sockets* de rede, arquivos, etc.

Por se tratar de uma execução local, as chamadas OCALL são executadas diretamente pelo provedor de enclaves, o qual fornece uma gama de chamadas padrão para acesso a arquivos e outros recursos que podem ser utilizadas pelos enclaves. Dessa forma, não há a necessidade do provedor de enclaves solicitar qualquer execução de tarefa à aplicação que requisitou o uso do enclave, o que também elimina a necessidade de tal aplicação estar constantemente aguardando alguma solicitação do provedor.

7.1.6 Liberação de Enclaves

Após a aplicação efetuar as suas chamadas ECALL, e quando não necessitar mais dos recursos providos pelo enclave, ela deverá enviar uma notificação ao provedor de enclaves, indicando a liberação do enclave que estava sendo utilizado. Ao receber a solicitação para a liberação de um enclave, o provedor irá encaminhá-la para o enclave gerente, que irá desfazer a associação entre a aplicação cliente e o enclave que estava sendo utilizado, podendo iniciar o processo de remoção da instância do enclave, e então retornar o resultado da operação para a aplicação cliente. A sequência de operações realizadas para a liberação de um enclave pelo provedor é apresentada na Figura 7.5.

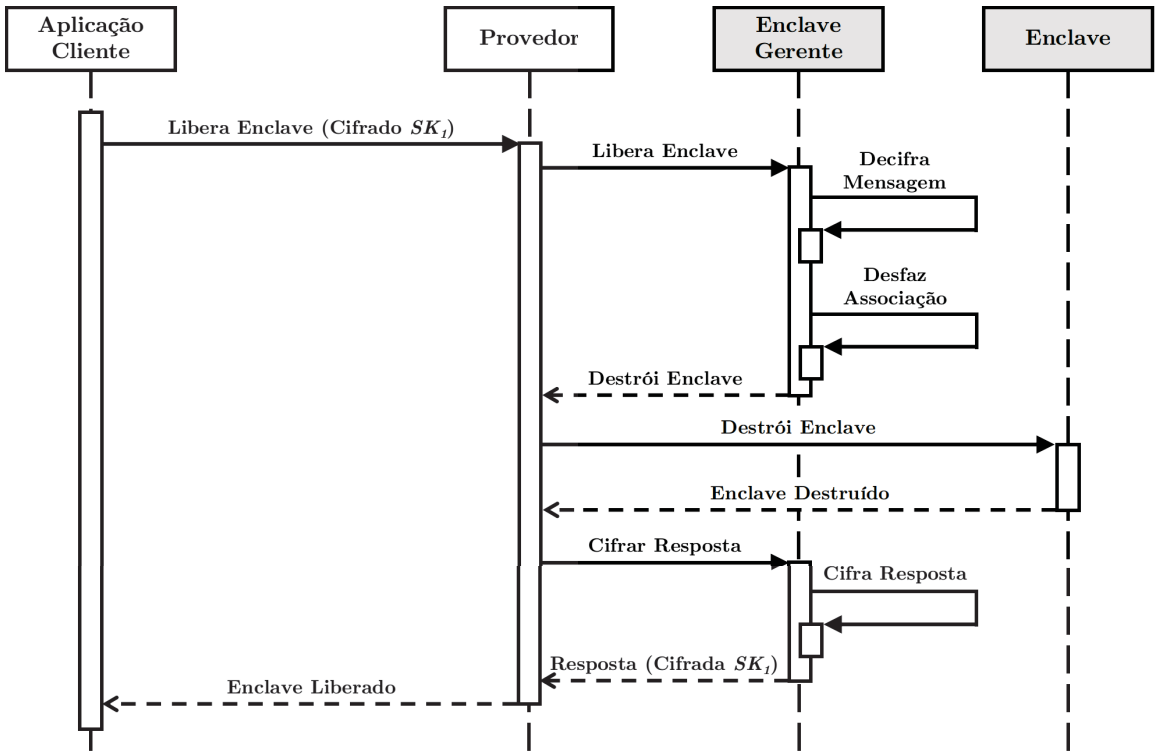


Figura 7.5: Sequência de operações realizadas na liberação de um enclave.

Quando o enclave está registrado no provedor para ser gerenciado em forma de *pool*, todas as instâncias que excederem o número mínimo de instâncias do *pool* sempre serão removidas, assim que a aplicação que a estava utilizando solicitar a sua liberação. Quando o enclave é utilizado de forma compartilhada, a instância permanece ativa após a liberação, aguardando para atender a outras requisições provindas de outras aplicações.

Também é possível definir na inicialização do provedor de enclaves um período de ociosidade para as instâncias de enclaves e, sempre que esse período for excedido, a instância ociosa será removida, garantindo a liberação de recursos utilizados por ela.

7.2 AVALIAÇÃO DO PROTÓTIPO

Com o intuito de descrever as implicações quantitativas e qualitativas do protótipo construído, foram avaliadas questões relativas ao desempenho da solução, no espectro quantitativo, e questões relacionadas à confidencialidade e integridade dos dados sensíveis trafegados entre os diversos atores, além de um estudo de escalabilidade da solução, buscando uma análise qualitativa. Tais análises são descritas nas seções que seguem.

7.2.1 Análise de Desempenho

A análise de desempenho foi efetuada buscando mensurar o impacto causado pela utilização do provedor de enclaves, seja no uso de enclaves compartilhados ou em *pool*, comparando com a instanciação e utilização do enclave diretamente pela aplicação. As definições de hardware e software utilizados para a execução dos testes são descritas na Seção 7.2.1.1, com os critérios avaliados sendo apresentados na Seção 7.2.1.2. Dois testes de desempenho foram executados, sendo o primeiro para avaliar o tempo médio de resposta de cada requisição (Seção

7.2.1.3), e o segundo para avaliar o comportamento do provedor de enclaves para atender diversas requisições simultâneas (Seção 7.2.1.4).

7.2.1.1 Configurações dos Equipamentos Utilizados

Os testes de desempenho foram executados utilizando um *notebook* Dell Inspiron 7460, com processador Intel Core i7-7500U, de 7^a geração, dispondo de dois núcleos de processamento, podendo executar quatro *threads* em simultâneo a uma frequência de operação de 2.7 GHz. Além disso, o computador dispunha de 16 GB de memória principal, com um disco rígido de 1 TB e um disco de estado sólido com 128 GB de armazenamento.

Em termos de software, o computador executava o sistema operacional Ubuntu 18.04 LTS, com *kernel* 4.15.0-112-generic e a versão 2.9.101.2 do SDK do SGX, com o tamanho máximo da PRM definido na BIOS como 128 MB. Além disso, foram desabilitados na BIOS as opções de *TurboBoost*, Intel *SpeedStep* e o *HyperThread* para a redução de oscilações na medição dos resultados. Por fim, todos os processos não essenciais foram desabilitados.

7.2.1.2 Critérios Avaliados

Com o intuito de verificar as diferenças no tempo de resposta e taxa de resposta, considerou-se a execução de aplicações sem a utilização do provedor de enclaves e fazendo requisições para o provedor com a utilização de enclaves compartilhados e em *pool*.

Conforme apresentado na Seção 5.4, o tamanho definido para a *stack* do enclave tem grande impacto no tempo necessário para a sua inicialização, devido ao fato de ser necessário alocar toda a memória no ato da criação do enclave. Assim, os testes foram executados considerando diversos tamanhos de *stack*, sendo definidas como 8 KB, 16 KB, 32 KB, 64 KB e 128 KB.

7.2.1.3 Tempo Médio de Resposta

A avaliação do tempo de resposta foi dividida em duas etapas: requisição do enclave e execução de chamada ECALL. A medição de tempo para a requisição do enclave visa comparar o desempenho obtido entre a alocação de um enclave em memória, pela própria aplicação requisitante, e a requisição deste mesmo enclave através do provedor de enclaves. Para efetuar tais medições, foi utilizada a instrução `RDTSCP` (*Read Time-Stamp Counter and Processor ID*), a qual oferece uma alta resolução, contabilizando o número de ciclos de processamento entre as medições (Paoloni, 2010). Para todos os dados apresentados nessa seção, a largura do intervalo de confiança em 95% não ultrapassa 0,67% da média.

Os dados apresentados no gráfico da Figura 7.6 representam os resultados obtidos após a execução de 10.000 requisições sequenciais, em cada um dos cenários, tomando o tempo médio de resposta para cada requisição. Nota-se que o tempo médio de resposta das requisições efetuadas pela aplicação sem a utilização do provedor de enclaves, ou seja, através da instanciação do próprio enclave, aumentam conforme o tamanho da *stack* do enclave aumenta, devido à necessidade de alocar estaticamente a memória necessária, no momento de inicialização do enclave.

Outro ponto a destacar é que não há diferença significativa no tempo de resposta no uso de um enclave compartilhado ou em um *pool* de enclaves, e que os mesmos permanecem constantes, independentemente do tamanho da *stack* do enclave. Além disso, a utilização do provedor de enclaves acarreta em um impacto de desempenho 30% menor na requisição do enclave, quando comparado com a utilização direta de um enclave configurado com 8 KB de

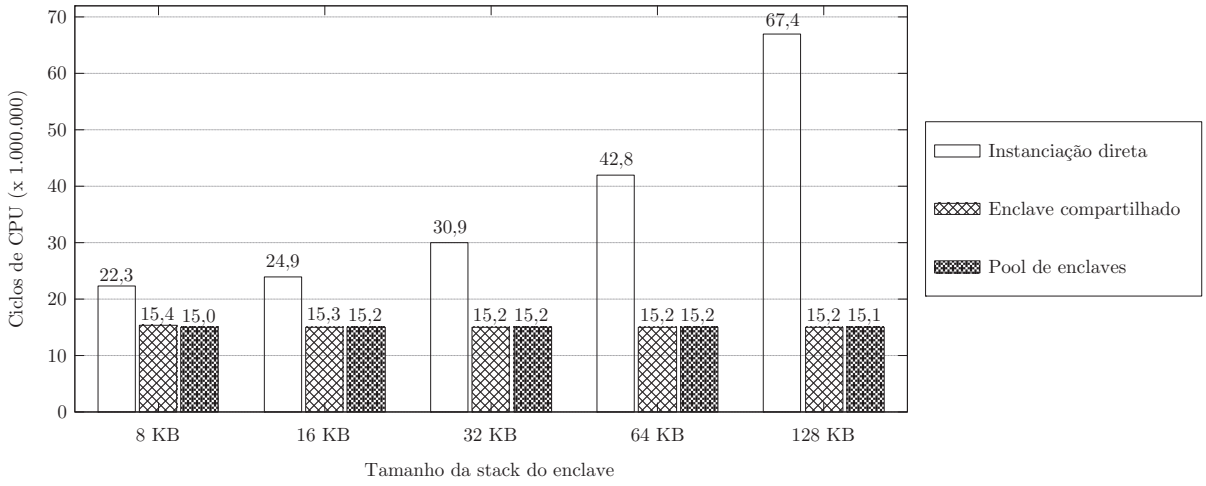


Figura 7.6: Tempo de resposta para a requisição de um enclave.

stack. Por não haver variação no tempo de resposta em diferentes tamanhos de enclave, tal diferença se torna maior com enclaves maiores.

O segundo quesito avaliado é o tempo decorrido para a execução de uma chamada ECALL. Para isso, utilizou-se uma ECALL vazia, ou seja, uma função que não executa qualquer instrução, com o intuito de mensurar apenas o *overhead* imposto pelas trocas de contexto entre aplicação e enclave, e pelas rotinas de validação executadas pelo provedor de enclaves. Os resultados de tais testes são apresentados no gráfico da Figura 7.7, novamente tomando a média de 10.000 execuções.

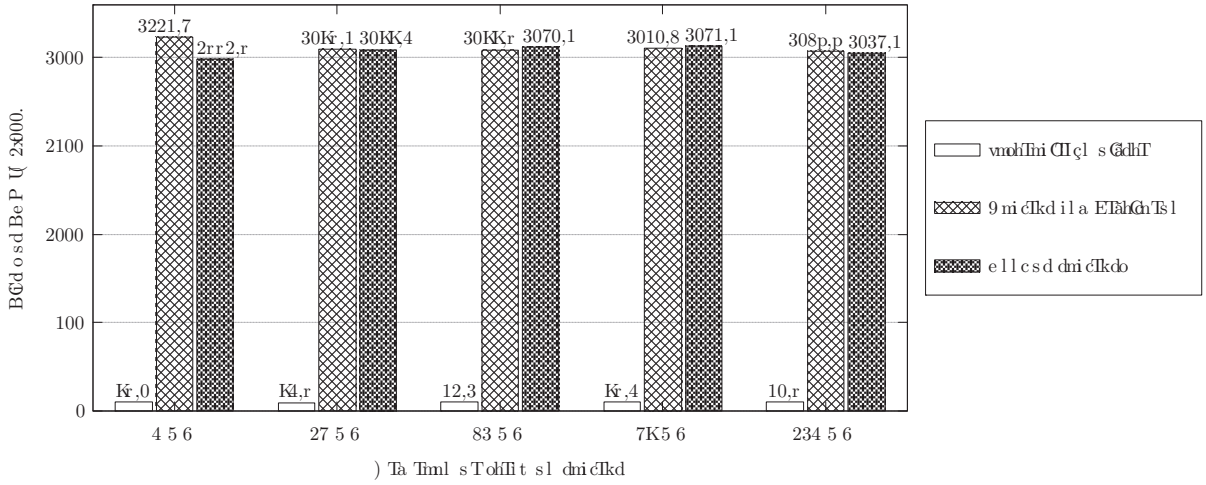


Figura 7.7: Tempo de resposta para a requisição de uma ECALL.

É possível perceber que não há variação significativa no tempo de execução conforme aumenta o tamanho da *stack* do enclave, e que há um grande impacto de desempenho na execução de chamadas ECALL quando se utiliza o provedor de enclaves. Tal impacto se deve ao fato de haver mais trocas de contexto (6 trocas, ao invés de 2 quando se utiliza o enclave diretamente) devido ao acesso do enclave gerente para a verificação de permissões de acesso e localização do enclave requisitado. Além disso, há as tarefas de decifragem da requisição e cifragem da resposta, e também as operações de cifragem e decifragem que ocorrem na aplicação requisitante, bem como o tempo decorrido no tráfego da mensagem pelo IPC. Destaca-se porém que, quando

somados os tempos decorridos na requisição do enclave e na execução da ECALL, ainda obtém-se uma melhora significativa de desempenho quando se utiliza o provedor de enclaves, principalmente em enclaves com tamanho de *stack* maiores.

Por fim, buscou-se mensurar o desempenho do provedor de enclaves em uma aplicação real, sendo escolhido o serviço de autenticação do Linux, o *Pluggable Authentication Module* (PAM), como caso de teste. O desempenho do provedor de enclaves foi confrontado com outras três soluções:

- O módulo PAM original, sem qualquer alteração;
- A solução proposta por Condé *et al.* (2018), chamada *UniSGX*, que altera o módulo de autenticação do PAM, incluindo a selagem do arquivo `etc/shadow` e a verificação das credenciais dentro de um enclave;
- A solução proposta por Will e Maziero (2020), que altera o UniSGX e aplica uma arquitetura cliente-servidor, mantendo o enclave instanciado no servidor para que os clientes possam requisitar a validação das credenciais providas pelo usuário. Tal abordagem utiliza o conceito de enclave compartilhado.

O provedor de enclaves também foi avaliado em duas condições: sem cifrar o *payload* e cifrando o *payload*. A cifragem do *payload* adiciona uma camada de segurança fim-a-fim, visto que a aplicação requisitante cifra os dados e estes serão decifrados apenas pelo enclave requisitado. Para tal, antes de efetuar a chamada para a função ECALL desejada (neste caso, a verificação das credenciais do usuário), é aplicado um protocolo de acordo de chaves entre a aplicação e o enclave requisitado, conforme descrito na Figura 6.2. Para efetuar o acordo de chaves utilizou-se o algoritmo Diffie-Hellman de curva elíptica X25519, gerando uma chave de 128 *bits* que é utilizada para cifrar o *payload* com um algoritmo AES-CTR. Os testes foram realizados utilizando um arquivo *shadow* de 5 KB, mensurando o tempo total decorrido para a autenticação do usuário. Os resultados desta etapa são apresentados na Figura 7.8, tomando a média entre 10.000 execuções.

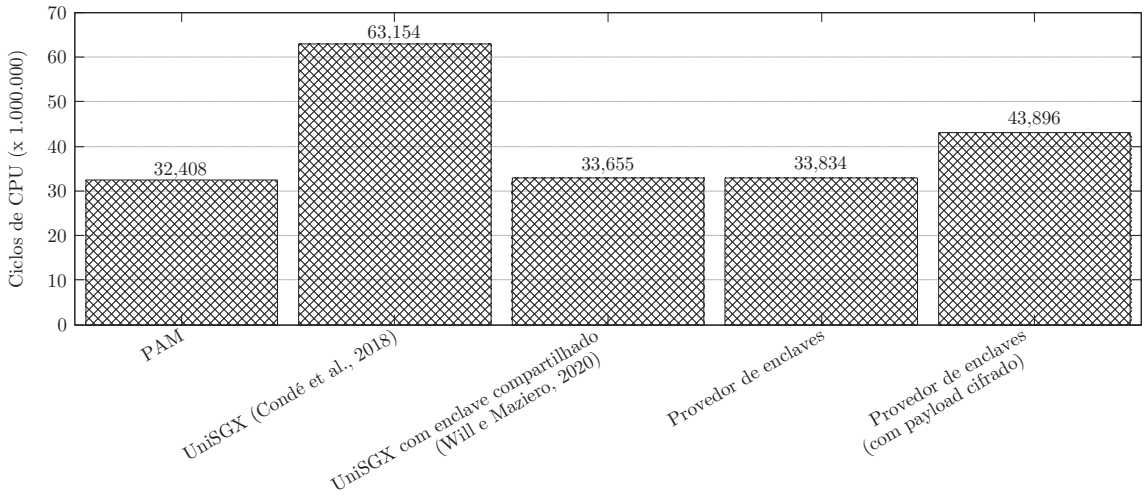


Figura 7.8: Tempo de resposta para a autenticação de usuário utilizando o PAM e as alternativas propostas com SGX.

Nota-se que a instanciação direta de um enclave para validar as credenciais do usuário (solução UniSGX, proposta por Condé *et al.* (2018)) induz um elevado custo de desempenho, e que a utilização de uma abordagem de compartilhamento de enclaves, como a proposta por

Will e Maziero (2020), garante as propriedades de segurança do SGX ao manipular o arquivo de credenciais dentro de um enclave, enquanto apresenta um impacto de desempenho praticamente zero quando comparado com o uso do módulo padrão de autenticação do PAM. A utilização do provedor de enclaves também não apresenta impacto significativo de desempenho, e mantém as mesmas premissas de confidencialidade dos dados trafegados no barramento de comunicação, já que estes são totalmente cifrados. Por fim, por se tratar de dados estritamente sensíveis (credenciais de usuário), a cifragem do *payload* garante que nem mesmo o provedor de enclaves tenha conhecimento acerca do que está sendo trafegado entre a aplicação cliente e o enclave requisitado. Tal operação adiciona um pequeno custo de desempenho, aproximadamente 30%, em decorrência das operações de cifragem e decifragem dos dados, além da operação adicional de acordo de chaves.

7.2.1.4 Taxa de Resposta para Múltiplas Requisições

Além do tempo de resposta para requisições sequenciais, buscou-se também avaliar o comportamento do provedor de enclaves para responder múltiplas requisições simultâneas. Para tal, foram avaliados o tempo de resposta para cada requisição individual, considerando a requisição do enclave e a execução da chamada ECALL, e o número de requisições atendidas por segundo. Para os testes, a *stack* do enclave foi configurada para 8 KB, visando avaliar o menor tempo de resposta, e foram considerados três cenários distintos:

- **Cenário 1:** São lançadas 20 instâncias simultâneas da aplicação cliente, cada uma efetuando 500 requisições sequenciais ao enclave. O *pool* de enclaves está configurado com 20 instâncias inicializadas;
- **Cenário 2:** São lançadas 50 instâncias simultâneas da aplicação cliente, cada uma efetuando 200 requisições sequenciais ao enclave. O *pool* de enclaves está configurado com 20 instâncias inicializadas;
- **Cenário 3:** São lançadas 50 instâncias simultâneas da aplicação cliente, cada uma efetuando 200 requisições sequenciais ao enclave. O *pool* de enclaves está configurado com 50 instâncias inicializadas.

A Figura 7.9 apresenta os resultados do tempo de resposta de cada uma das requisições. Nota-se que no Cenário 1 a utilização de um enclave compartilhado e de um *pool* de enclaves garante um tempo de resposta consideravelmente inferior quando comparado com a instanciação direta do enclave pela aplicação.

Já o Cenário 2 apresenta um aumento de 13% no tempo de resposta para o enclave compartilhado, quando compara-se com a instanciação direta. Tal fato se deve à serialização das requisições imposta pelo provedor de enclaves para evitar condições de corrida no acesso às suas estruturas de controle. A utilização do *pool* de enclaves também acarretou em um tempo de resposta individual bem superior à instanciação direta (próximo de 33% de acréscimo) devido ao subdimensionamento do *pool*, já que esse estava configurado com apenas 20 instâncias de enclaves para responder a 50 requisições simultâneas. Nesse cenário o provedor instancia novos enclaves para responder às requisições que estão na fila, aumentando consideravelmente o tempo de resposta para a requisição de um novo enclave. Por fim, no Cenário 3, com o tamanho do *pool* dimensionado adequadamente, o tempo de resposta para cada requisição é muito semelhante ao apresentado quando a aplicação instancia o enclave diretamente.

Além do tempo de resposta para cada requisição, avaliou-se também a média de requisições atendidas por segundo em cada um dos cenários, com os resultados sendo apresentados

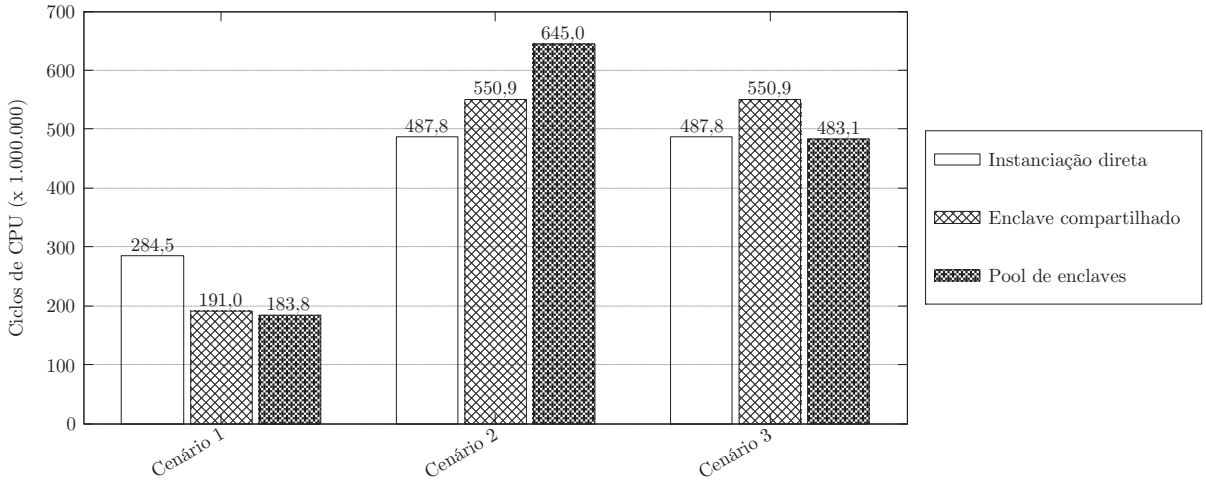


Figura 7.9: Tempo médio de resposta para cada requisição (somados requisição do enclave e execução de ECALL).

na Figura 7.10. Nota-se que, no Cenário 1, tanto o uso de enclave compartilhado quanto o *pool* de enclaves apresentaram desempenho superior à instanciação direta de enclaves. No Cenário 2 a abordagem com compartilhamento de enclave manteve o bom desempenho, já o *pool* de enclaves ficou com um desempenho bem abaixo, visto que o tamanho do *pool* estava subdimensionado. Com o dimensionamento correto do *pool* (Cenário 3) o provedor de enclaves apresenta uma alta taxa de resposta, novamente superior à instanciação direta.

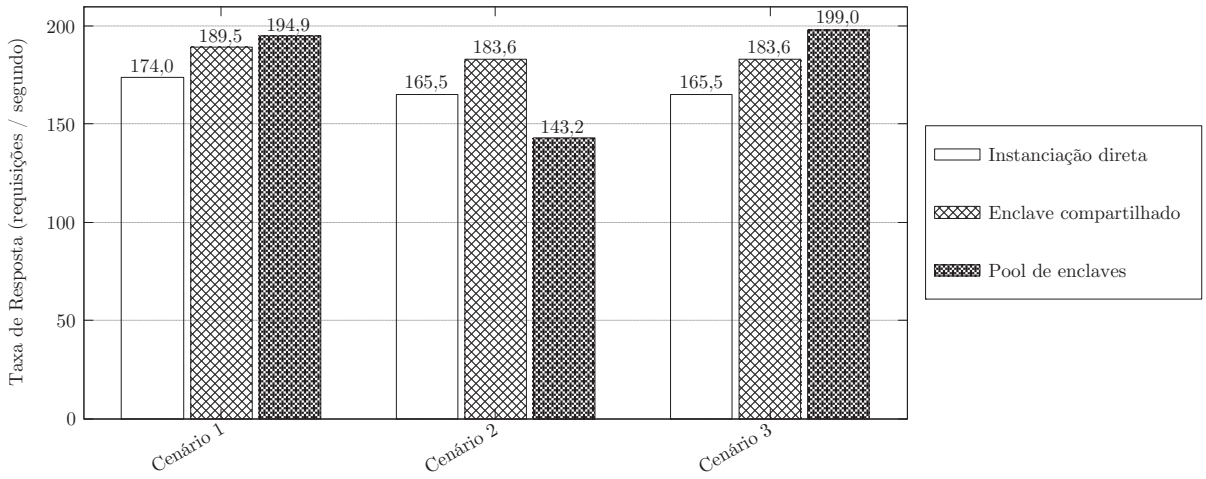


Figura 7.10: Taxa de requisições de enclaves atendidas por segundo.

O comportamento do provedor de enclaves na existência de múltiplas requisições também foi avaliado em uma aplicação real. Novamente utilizou-se o PAM e as soluções propostas por Condé *et al.* (2018) e Will e Maziero (2020) como parâmetros de análise. Foram considerados dois cenários:

- **Cenário 1:** São lançadas 20 instâncias simultâneas, cada instância efetua 500 requisições para autenticação de usuário;
- **Cenário 2:** São lançadas 50 instâncias simultâneas, cada instância efetua 200 requisições para autenticação de usuário.

O tempo médio de resposta para cada requisição é apresentado na Figura 7.11. As soluções baseadas no provedor de enclaves demonstraram um tempo de resposta superior ao PAM e à solução de compartilhamento de enclaves proposta por Will e Maziero (2020), mas ainda inferior à utilização de um enclave instanciado diretamente pelo módulo de autenticação do PAM, conforme proposto por Condé *et al.* (2018). Nota-se também que a cifragem do *payload* não acrescenta grande impacto no desempenho do provedor de enclaves.

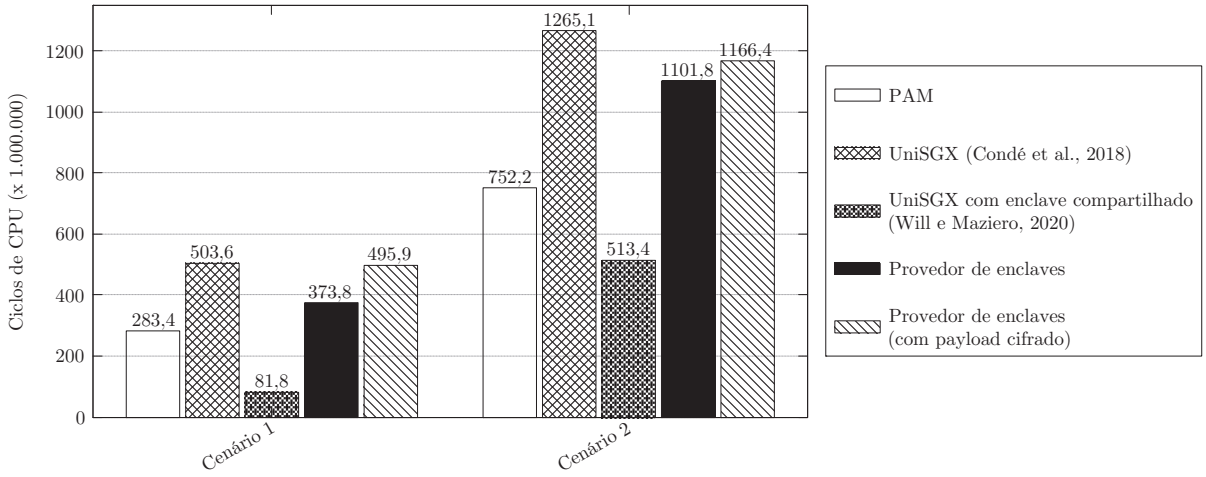


Figura 7.11: Tempo médio de resposta para cada processo de autenticação.

Já ao analisar a média de requisições atendidas por segundo o provedor de enclaves demonstra bom desempenho, apresentando, no Cenário 1, taxas superiores às demais soluções baseadas em enclaves. No Cenário 2, o provedor de enclaves alcançou uma taxa de resposta equivalente à solução de compartilhamento de enclaves descrita em Will e Maziero (2020) e, quando adicionada a cifragem do *payload*, demonstrou uma taxa de resposta equivalente à instanciação direta do enclave, conforme proposto por Condé *et al.* (2018). Tais resultados demonstram a robustez do provedor de enclaves na demanda de várias requisições simultâneas.

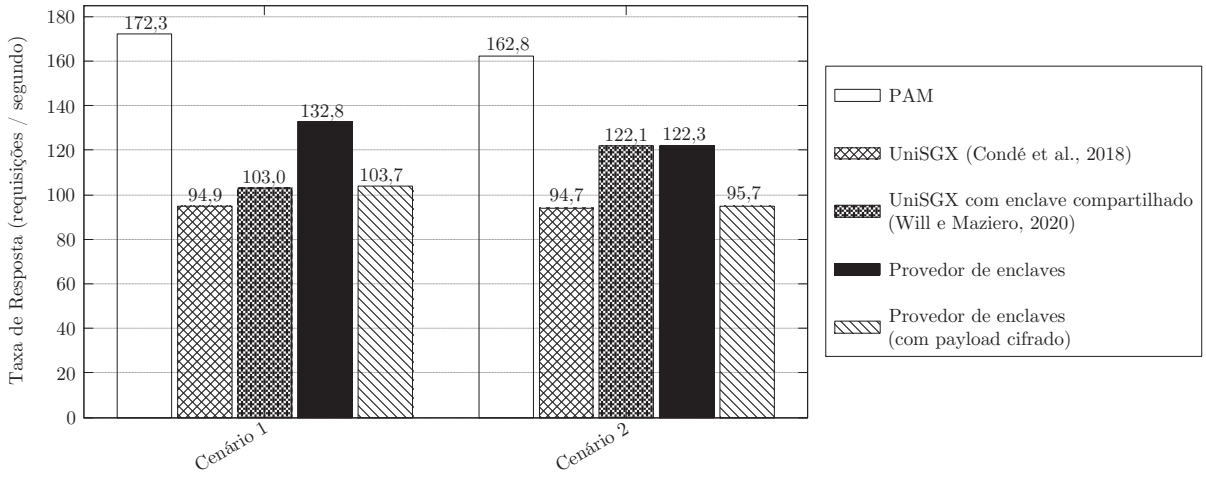


Figura 7.12: Taxa de requisições atendidas por segundo para a autenticação de usuário utilizando o PAM.

7.2.2 Análise de Segurança

O principal objetivo da arquitetura Intel SGX é garantir a confidencialidade e integridade dos dados manipulados pelos enclaves, provendo, assim, uma forma das aplicações manipularem dados sensíveis de maneira segura e confiável. Tais aspectos de segurança foram considerados na elaboração da proposta e na construção do protótipo, visando manter tais garantias enquanto oferece um ganho de desempenho e na otimização de recursos.

Todas as estruturas de dados sensíveis para a execução do provedor de enclaves são mantidas e manipuladas por um enclave gerente utilizando, assim, os mecanismos providos pela arquitetura Intel SGX para manter a confidencialidade e integridade de tais dados. O enclave gerente também é responsável por receber e distribuir as requisições oriundas das aplicações clientes, mantendo os registros de quais enclaves estão instanciados, quais aplicações estão efetuando as solicitações, e quais instâncias de enclaves estão associados a essas aplicações.

Os enclaves manipulados pelo provedor são previamente registrados, conforme descrito nas Seções 6.4.3 e 7.1.1. Tais registros são mantidos e atualizados pelo enclave gerente, sendo armazenados em memória secundária de forma criptografada, utilizando o recurso de selagem do SGX, que foi descrito na Seção 3.5. Desta forma, os registros somente podem ser lidos pelo próprio enclave gerente, e apenas na própria plataforma utilizada para efetuar a selagem dos dados. Além disso, qualquer alteração nos dados cifrados é identificada pelos mecanismos de autenticação contidos no próprio procedimento de selagem.

Cada enclave registrado no provedor é acompanhado de um resumo criptográfico SHA-256, que tem por objetivo validar o conteúdo do enclave antes do seu carregamento. Assim, o provedor de enclaves confronta o resumo criptográfico obtido do conteúdo do enclave (incluídos código e dados) com aquele informado nos dados do registro, garantindo que o código ou dados do enclave não foram alterados ou substituídos. Essa verificação é efetuada tanto no ato do registro do enclave quanto na criação de cada uma de suas instâncias, o que permite identificar alterações efetuadas no enclave após o seu registro. Todo esse processo de verificação é efetuado dentro dos limites do enclave gerente, evitando a manipulação de tais dados por entidades não autorizadas. Após criadas as instâncias, os mecanismos presentes na arquitetura Intel SGX garante a integridade dos dados e código do enclave.

Já a comunicação entre a aplicação cliente e o provedor de enclaves é totalmente cifrada, com o enclave gerente sendo responsável por manipular as chaves de sessão e decifrar as requisições recebidas. As chaves de sessão são acordadas entre a aplicação cliente e o enclave gerente através de um protocolo de acordo de chaves Diffie-Hellman de curva elíptica (ECDH), utilizando a curva X25519 proposta por Bernstein (2006), conforme já descrito na Seção 7.1.3. A utilização do protocolo Diffie-Hellman permite a definição de uma chave de criptografia entre as duas partes através de um canal inseguro, prevenindo que um atacante obtenha as chaves acordadas entre as aplicações.

As requisições enviadas para o provedor de enclaves são cifradas utilizando um algoritmo de criptografia simétrica AES-CTR, utilizando a chave de 128 *bits* acordada entre as partes. Tal chave é utilizada para cifrar toda a comunicação efetuada entre a aplicação e o provedor de enclaves, incluindo também as respostas enviadas pelo provedor. A aplicação pode redefinir a chave de criptografia a qualquer momento, iniciando um novo acordo de chaves com o provedor de enclaves. A criptografia das mensagens trocadas entre as partes garante que, caso algum atacante esteja conectado no barramento de mensagens e coletando as informações trocadas, ele não terá um acesso direto às mesmas, sendo necessária a aplicação de técnicas para obter a chave utilizada na cifragem dos dados. Mesmo utilizando um ambiente com alto poder computacional, o processo de obtenção da chave é extremamente custoso, tornando-o inviável.

Destaca-se também que os procedimentos para decifrar a requisição e cifrar a resposta a ser enviada à aplicação ocorrem dentro do enclave gerente, o qual manipula as chaves de sessão. Dessa forma, as chaves de criptografia são mantidas sempre protegidas pelo provedor de enclaves, garantindo que elas não sejam acessadas por entidades não autorizadas.

A aplicação cliente pode adicionar uma camada extra de segurança ao efetuar um acordo de chaves diretamente com o enclave requisitado, possibilitando assim a criptografia fim-a-fim dos dados enviados ao enclave. Essa alternativa torna os dados trocados entre a aplicação e o enclave totalmente opacos ao provedor de enclaves, além de acrescentar um obstáculo extra a um atacante que possa estar coletando os dados trafegados pelo barramento.

A arquitetura cliente-servidor também implica em um único ponto de falha: a aplicação servidora. Assim, o provedor de enclaves deve ser resiliente e capaz de se recuperar de falhas, para evitar ataques de negação de serviço, que impedem aplicações clientes legítimas de acessarem os enclaves requisitados.

Por fim, além das questões de segurança que devem ser consideradas ao provedor de enclaves de maneira geral, deve-se analisar também as implicações de cada modelo de gerenciamento de enclaves proposto. Ressalta-se, porém, que o modo de gerenciamento é uma escolha tomada pelo desenvolvedor do enclave que será adicionado ao provedor, e deve levar em conta o equilíbrio entre o desempenho e os níveis de segurança esperados.

7.2.2.1 Compartilhamento de Enclaves

O compartilhamento de enclaves oferece uma otimização de recursos e uma melhora de desempenho consideráveis, mas em troca de um nível de isolamento entre as aplicações mais permissivo. Primeiramente, o compartilhamento não requer extensas alterações no código fonte do enclave, necessitando apenas acrescentar as estruturas de dados e rotinas necessárias para gerenciar as múltiplas sessões com as aplicações clientes. Outro fator a ser considerado é o aumento da *heap* do enclave, que deverá ser definida de modo a comportar todo o conjunto de dados necessário para atender as aplicações clientes.

Por outro lado, o isolamento entre os dados das aplicações clientes que utilizam o enclave é efetuado exclusivamente via software, já que todos os dados estarão nos domínios de um mesmo enclave. A principal consequência dessa decisão de projeto é que, ao explorar uma vulnerabilidade de segurança em uma instância de enclave compartilhado, todos os clientes que estão conectados a ele serão potencialmente afetados. Assim, considera-se a utilização de enclaves compartilhados apenas quando as aplicações não dependem do estado do enclave.

7.2.2.2 Pool de Enclaves

O principal objetivo do *pool* de enclaves é reduzir o tempo de resposta da aplicação na requisição de um enclave. Assim, todas as propriedades de isolamento entre as aplicações são mantidas, já que um enclave é responsável por atender a uma única aplicação. Apesar de depender de uma comunicação com o provedor de enclaves, a utilização de criptografia fim-a-fim garante, além da opacidade do *payload* da requisição para o provedor, que somente o enclave que detém a chave de sessão previamente acordada será apto a ler os dados recebidos, garantindo que um erro no roteamento das mensagens não comprometa a confidencialidade dos dados.

É importante salientar que, para garantir o total isolamento entre as aplicações que utilizam instâncias de enclaves que fazem parte do *pool*, as aplicações clientes devem eliminar todos os dados confidenciais que estão na memória do enclave antes de liberá-lo. Para tanto, o enclave deve fornecer mecanismos, através de uma função ECALL por exemplo, para que esta tarefa seja executada. Assim, evita-se que a instância seja alocada para outra aplicação

contendo ainda dados sensíveis que possam comprometer a operação das aplicações que utilizam o provedor de enclaves.

7.3 CONSIDERAÇÕES FINAIS

O presente capítulo apresentou os detalhes técnicos do provedor de enclaves e a implementação de uma prova de conceito, com o objetivo de validar a proposta apresentada no capítulo anterior, detalhando os protocolos e mecanismos de comunicação utilizados na implementação. Por fim, foram analisados os aspectos de desempenho e segurança da solução.

Através da análise de desempenho pode-se concluir que a utilização do provedor de enclaves atinge os objetivos propostos, reduzindo consideravelmente o tempo necessário para requisitar um enclave e executar uma chamada ECALL, tanto na utilização de enclaves compartilhados quanto em um *pool* de enclaves. Ao efetuar múltiplas requisições simultâneas ao provedor de enclave, é perceptível uma queda de desempenho no tempo de resposta de cada requisição individual, tanto no *pool* quanto em compartilhamento de enclaves. Mostrou-se, porém, que um correto dimensionamento do *pool* elimina esse entrave, e refinamentos nas estruturas de controle do provedor de enclaves pode trazer melhoras consideráveis nesse aspecto. Em contrapartida, os resultados demonstram uma alta taxa de resposta do provedor de enclaves para requisições simultâneas, superando os números obtidos pela instanciação direta de enclaves. Além disso, ressalta-se que a utilização de enclaves compartilhados tem como objetivo primário a redução do uso da PRM, ao manter apenas um enclave ativo para atender múltiplas requisições simultâneas.

A análise de segurança demonstra que os ganhos de desempenho obtidos não afetam as propriedades de confidencialidade e integridade dos dados manipulados pelos enclaves, além de garantir uma comunicação segura entre a aplicação cliente e o enclave que esta requisita. A utilização de um enclave gerente pelo provedor garante a manipulação e roteamento seguro das requisições recebidas, as quais são transmitidas totalmente cifradas. Também acrescenta-se a possibilidade da aplicação cifrar os dados enviados ao enclave requisitando, mantendo estes opacos até mesmo ao provedor, e garantindo uma confidencialidade fim-a-fim. Por fim, um isolamento completo entre as aplicações é alcançado com o uso de um *pool* de enclaves, garantido o acesso de uma única aplicação a determinado enclave.

8 CONCLUSÃO

Atualmente, a confidencialidade e segurança nos dados sensíveis manipulados pelas aplicações e serviços têm ganhado cada vez mais atenção, tanto das empresas provedoras desses serviços ou desenvolvedoras de aplicações, quanto dos governos, os quais já estão criando normas regulatórias a respeito do assunto.

Por parte das empresas, as desenvolvedoras de software e provedoras de serviços buscam fornecer mais garantias no que diz respeito à confidencialidade dos dados dos usuários que utilizam suas aplicações e serviços. No que tange ao desenvolvimento de hardware, o desenvolvimento de mecanismos para fornecer suporte à segurança de aplicações também tem tido uma atenção especial.

A busca por maiores garantias de segurança e confidencialidade dos dados invariavelmente impacta em uma sensível queda de desempenho da aplicação, em virtude das diversas verificações e algoritmos inerentes ao processo. A utilização de mecanismos de hardware para efetuar essas verificações acaba por reduzir esse impacto, mas não o elimina. Um dos objetivos é fornecer tais garantias com um impacto mínimo no desempenho da aplicação, e a proposta apresentada neste trabalho busca reduzir o impacto de desempenho causado pela utilização da arquitetura Intel SGX através de técnicas e modelos de programação que visam garantir uma utilização eficiente dos recursos necessários para a utilização dos enclaves.

8.1 CONTRIBUIÇÕES OBTIDAS

O presente trabalho objetivou a execução segura e eficiente de rotinas sensíveis de aplicações, utilizando modelos de programação para garantir a otimização de recursos e redução do impacto de desempenho.

Para tal, foi apresentado um estudo dos mecanismos de segurança baseados em hardware, que visam garantir a confidencialidade de dados sensíveis, com atenção particular à arquitetura Intel SGX, que foi objeto de estudo. Obteve-se, assim, um completo estudo sobre tal arquitetura, avaliando suas características, objetivos, limitações e vulnerabilidades, fornecendo a fundamentação sólida para o decorrer do texto.

Foi estudada uma ampla gama de aplicações da arquitetura SGX disponível na literatura, buscando abranger os mais diversos contextos de desenvolvimento e formas de aplicação dos recursos providos pelo SGX. Tal estudo possibilitou analisar os diversos modelos e técnicas de programação aplicados a essas soluções, embasando, assim, a proposta deste trabalho.

Através de tal análise, foram descritos os conceitos para dois modelos distintos de gerenciamento de enclaves: compartilhamento e *pool*. Cada modelo apresenta suas próprias características e pode ser utilizado em diferentes contextos. Através desses modelos, foi proposta uma arquitetura para um provedor de enclaves, o qual trata cada enclave como um recurso de software distinto, e o disponibiliza às aplicações clientes na forma de serviço, desacoplando o enclave da aplicação que o utiliza e permitindo um gerenciamento centralizado e eficiente das instâncias de enclaves e dos recursos utilizados por tais instâncias.

Através da especificação do provedor de enclaves, desenvolveu-se um protótipo para validar a proposta, o qual foi avaliado em termos de desempenho e segurança. Os testes de desempenho mostraram resultados satisfatórios, alcançando os objetivos propostos de reduzir o tempo de resposta para a requisição de enclaves, enquanto demonstra que a solução pode ser

utilizada em um ambiente com alta demanda. Por fim, a análise de segurança demonstrou que os ganhos de desempenho obtidos não afetam as propriedades garantidas pelos enclaves.

8.2 PUBLICAÇÕES GERADAS

O desenvolvimento do presente trabalho resultou na publicação de um capítulo de livro e três artigos em conferências, de abrangência nacional e internacional, conforme listado:

- **Capítulo de Livro:** Mecanismos de Segurança Baseados em Hardware: Uma Introdução à Arquitetura Intel SGX. "Livro de Minicursos do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)" (Will *et al.*, 2017).
- **Artigo em Conferência:** Using Intel SGX to Protect Authentication Credentials in an Untrusted Operating System. "IEEE Symposium on Computers and Communications (ISCC)". (Condé *et al.*, 2018)
- **Artigo em Conferência:** Using a Shared SGX Enclave in the UNIX PAM Authentication Service. "Annual IEEE International Systems Conference (SysCon)" (Will e Maziero, 2020).
- **Artigo em Conferência:** A Trusted Message Bus Built on Top of D-Bus. "Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)" (Will *et al.*, 2020).

Além dos trabalhos já publicados, destaca-se também a submissão de um quarto artigo para conferência, intitulado *Trusted Inter-Process Communication Using Hardware Enclaves*, ainda aguardando o resultado do processo de revisão por pares. Também está em desenvolvimento dois *surveys* na temática do SGX, a serem submetidos para publicação em periódicos da área. Por fim, outro artigo será escrito com base nos resultados obtidos da tese, com o objetivo de publicação em um periódico da área.

8.3 LIMITAÇÕES DA SOLUÇÃO PROPOSTA

Além de destacar as contribuições e publicações geradas neste trabalho, é importante notar que a solução descrita apresenta algumas limitações, seja em decorrência do seu escopo, projeto, ou na implementação da prova de conceito.

A primeira limitação é em decorrência do padrão arquitetural adotado, cliente/servidor, que promove a divisão entre aplicações (clientes) e o provedor de enclaves (servidor). Tal arquitetura resulta em um ponto único de falha, centrado no próprio provedor de enclaves, que pode ser alvo de um ataque coordenado por processos maliciosos no computador, visando resultar em uma situação de negação de serviço e afetando as aplicações que dependem de uma resposta do provedor. Além disso, um atacante com privilégios para controlar o sistema pode deliberadamente encerrar o processo do provedor de enclaves, causando o mesmo dano às aplicações requisitantes. Tal limitação pode ser contornada garantindo a resiliência do provedor de enclaves, de forma que o mesmo seja reinicializado quando for inundado com requisições suspeitas ou deliberadamente encerrado. Além disso, pode-se utilizar um *pool* de provedores, de forma a descentralizar o serviço.

Outra questão importante é o fato de a parte não confiável do provedor de enclaves estar sujeita a manipulações por parte de um atacante que tenha acesso privilegiado aos recursos do sistema operacional, podendo sobrescrever os dados contidos na memória do processo. Tal

operação pode permitir que o atacante substitua os dados enviados e recebidos dos enclaves gerenciados, bem como possibilitar que tais dados sejam enviados a outro enclave, e não aquele indicado pelo enclave gerente. A utilização de técnicas de ofuscação de memória, como ORAM (Chang *et al.*, 2016), pode mitigar tal problema.

Por fim, o provedor de enclaves carece de algum mecanismo que permita às aplicações clientes atestarem a sua identidade, o que abre a possibilidade de um atacante criar um processo malicioso que personifique o próprio provedor de enclaves. Assim, propõe-se a adição de mecanismos de autenticação que permitam que o provedor de enclaves prove a sua identidade para as aplicações clientes, e que também permita que as próprias aplicações clientes provem a sua identidade para o provedor de enclaves, além de adicionar mecanismos de controle de acesso que permitam identificar quais aplicações podem acessar determinados enclaves registrados no provedor.

8.4 TRABALHOS FUTUROS

Além de implementar mecanismos para mitigar as limitações previamente apresentadas, há uma gama de encaminhamentos que podem ser tomados para a continuidade da pesquisa e no desenvolvimento de soluções derivadas.

Como possibilidade de melhoria e extensão da solução proposta, pode-se atacar o problema de subdimensionamento do *pool* de enclaves através do uso de políticas mais eficientes na liberação de enclaves. Atualmente, o provedor elimina os enclaves excedentes no *pool* (quando o número de instâncias está acima do especificado no momento do registro do enclave) assim que eles são liberados pelas aplicações que os detém. Assim, quando outra aplicação requisita uma instância do enclave, caso todo o *pool* esteja alocado, é necessário criar uma nova instância para atender tal requisição. Pode-se aplicar uma política de manter ativas as instâncias que são liberadas pelas aplicações quando há alta demanda, mesmo que o número de instâncias esteja acima do especificado para o *pool*, evitando assim repetidas instanciações de enclaves. Outra possibilidade é determinar um número ou percentual mínimo de instâncias livres no *pool*, fazendo com que o provedor de enclaves monitore a utilização do *pool* e crie novas instâncias em segundo plano assim que seja atingido determinada taxa de utilização.

Também é possível acrescentar novos modelos de gerenciamento de enclaves ao provedor, garantindo maior diversividade de modelos e atendendo outros tipos de demanda. Um exemplo pode ser a combinação dos modelos de compartilhamento e *pool*, criando um *pool* de enclaves compartilhados, a fim de evitar gargalos na utilização do compartilhamento de enclaves. Com essa combinação, o provedor pode balancear o número de clientes entre todas as instâncias do *pool*, evitando que uma instância fique sobrecarregada com alto número de requisições, e mantendo uma alta taxa de resposta para as aplicações clientes.

Além disso, a solução também pode ser expandida para outros ambientes de uso, não ficando restrita somente para o escopo de requisições locais. Uma proposta é utilizar o provedor de enclaves para atender requisições de dispositivos IoT espalhados em uma rede local. É notório que tais dispositivos dispõem de capacidades de memória e processamento bem limitadas, o que pode dificultar a execução de operações complexas sobre os dados que são coletados por estes. Com a utilização de um provedor de enclaves na rede, tais dados podem ser processados de forma segura, usufruindo de todas as garantias obtidas pelo uso de enclaves.

Por fim, o provedor de enclaves pode se utilizado também para atender requisições em um ambiente de nuvem ou distribuído, principalmente para operações em que não há dependência do estado do enclave, ou que o estado do enclave não dependa de requisições anteriores. Operações que dependem do estado do enclave também podem ser realizadas em ambiente distribuído

através do uso de técnicas de migração de enclaves, como as apresentadas por Alder *et al.* (2018) e Park *et al.* (2019).

REFERÊNCIAS

Alder, F., Asokan, N., Kurnikov, A., Paverd, A. e Steiner, M. (2019). S-FaaS: Trustworthy and accountable function-as-a-service using Intel SGX. Em *Proceedings of the Conference on Cloud Computing Security Workshop*, páginas 185–199, Londres, Reino Unido. ACM.

Alder, F., Kurnikov, A., Paverd, A. e Asokan, N. (2018). Migrating SGX enclaves with persistent state. Em *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, páginas 195–206, Cidade de Luxemburgo, Luxemburgo. IEEE.

Amin, M., Khan, S., Ali, T. e Gul, S. (2008). Trends and directions in trusted computing: Models, architectures and technologies. Em *Proceedings of the International Multiconference Of Engineers and Computer Scientist*, páginas 19–21, Hong Kong, China. International Association of Engineers.

Anati, I., Gueron, S., Johnson, S. P. e Scarlata, V. R. (2013). Innovative technology for CPU based attestation and sealing. Em *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, Tel-Aviv, Israel. ACM.

Anderson, R., Bond, M., Clulow, J. e Skorobogatov, S. (2006). Cryptographic processors: A survey. *Proceedings of the IEEE*, 94(2):357–369.

Apple (2015). *iOS Security*. Apple Inc.

ARM (2009). Security technology: Building a secure system using TrustZone technology (white paper). *ARM Limited*.

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M. *et al.* (2016). SCONE: Secure Linux containers with Intel SGX. Em *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, páginas 689–703, Savannah, GA, EUA. USENIX Association.

Arthur, W. e Challener, D. (2015). *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, Berkeley, CA, EUA, 1st edition.

Aublin, P.-L., Kelbert, F., O'Keeffe, D., Muthukumaran, D., Priebe, C., Lind, J., Krahn, R., Fetzer, C., Eyers, D. e Pietzuch, P. (2018). LibSEAL: Revealing service integrity violations using trusted execution. Em *Proceedings of the 13th EuroSys Conference*, páginas 24:1–24:15, Porto, Portugal. ACM.

Bailleu, M., Thalheim, J., Bhatotia, P., Fetzer, C., Honda, M. e Vaswani, K. (2019). SPEICHER: Securing LSM-based key-value stores using shielded execution. Em *Proccedings of the 17th USENIX Conference on File and Storage Technologies*, páginas 173–190, Boston, MA, EUA. USENIX Association.

Bauman, E. e Lin, Z. (2016). A case for protecting computer games with SGX. Em *Proceedings of the 1st Workshop on System Software for Trusted Execution*, páginas 4:1–4:6, Trento, Itália. ACM.

Baumann, A., Peinado, M. e Hunt, G. (2015). Shielding applications from an untrusted cloud with Haven. *ACM Transactions on Compututer Systems*, 33(3):8:1–8:26.

Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman speed records. Em *Proceedings of the International Workshop on Public Key Cryptography*, páginas 207–228, Nova Iorque, NY, EUA. Springer.

Bossuet, L., Grand, M., Gaspar, L., Fischer, V. e Gogniat, G. (2013). Architectures of flexible symmetric key crypto engines: A survey: From hardware coprocessor to multi-crypto-processor system on chip. *ACM Compututer Survey*, 45(4):41:1–41:32.

Brasil (2018). Lei nº 13.709, de 14 de agosto de 2018. *Dispõe sobre a proteção de dados pessoais e altera a Lei nº 12.965, de 23 de abril de 2014 (Marco Civil da Internet)*.

Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S. e Sadeghi, A.-R. (2017). Software Grand Exposure: SGX cache attacks are practical. Em *Proceedings of the 11th USENIX Workshop on Offensive Technologies*, Vancouver, BC, Canadá. USENIX Association.

Brekalo, H., Strackx, R. e Piessens, F. (2016). Mitigating password database breaches with Intel SGX. Em *Proceedings of the 1st Workshop on System Software for Trusted Execution*, páginas 1:1–1:6, Trento, Itália. ACM.

Brenner, S., Behlendorf, M. e Kapitza, R. (2018). Trusted execution, and the impact of security on performance. Em *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, páginas 28–33, Toronto, Canadá. ACM.

Brenner, S., Hundt, T., Mazzeo, G. e Kapitza, R. (2017). Secure cloud micro services using Intel SGX. Em *Proceedings of the Distributed Applications and Interoperable Systems*, páginas 177–191, Neuchâtel, Suíça. Springer.

Brenner, S., Wulf, C., Goltzsche, D., Weichbrodt, N., Lorenz, M., Fetzer, C., Pietzuch, P. e Kapitza, R. (2016). SecureKeeper: Confidential ZooKeeper using Intel SGX. Em *Proceedings of the 17th International Middleware Conference*, páginas 14:1–14:13, Trento, Itália. ACM.

Buhren, R., Werling, C. e Seifert, J.-P. (2019). Insecure until proven updated: Analyzing AMD SEV's remote attestation. Em *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, páginas 1087–1099, Londres, Reino Unido. ACM.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. e Stal, M. (2013). *Pattern-Oriented Software Architecture, A System of Patterns*, volume 1 de *Pattern-Oriented Software Architecture*. Wiley.

Butterworth, J., Kallenberg, C., Kovah, X. e Herzog, A. (2013). Problems with the Static Root of Trust for Measurement. Em *Proceedings of the Black Hat*, Las Vegas, NV, EUA.

Chang, Z., Xie, D. e Li, F. (2016). Oblivious RAM: A dissection and experimental evaluation. *Proceedings of the VLDB Endowment*, 9(12):1113–1124.

Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z. e Lai, T. H. (2019). SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution. Em *Proceedings of the 4th IEEE European Symposium on Security and Privacy*, páginas 142–157, Estocolmo, Suécia. IEEE.

Chiesa, M., di Lallo, R., Lospoto, G., Mostafaei, H., Rimondini, M. e Di Battista, G. (2017). PrIXP: Preserving the privacy of routing policies at Internet eXchange Points. Em *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management*, páginas 435–441, Lisboa, Portugal. IEEE.

Condé, R. C. R., Maziero, C. A. e Will, N. C. (2018). Using Intel SGX to protect authentication credentials in an untrusted operating system. Em *Proceedings of the IEEE Symposium on Computers and Communications*, páginas 158–163, Natal, RN, Brasil. IEEE.

Conti, M., Dushku, E., Mancini, L. V., Rabbani, M. M. e Ranise, S. (2019). Remote attestation as a service for IoT. Em *Proceedings of the 6th International Conference on Internet of Things: Systems, Management and Security*, páginas 320–325, Granada, Espanha. IEEE.

Coplien, J. O. (2003). Software design patterns. Em *Encyclopedia of Computer Science*, páginas 1604–1606. John Wiley and Sons Ltd., Chichester, UK.

Costa, R. d. S., Pigatto, D. F., Fonseca, K. V. O. e Rosa, M. d. O. (2018). Securing video on demand content with SGX: A decryption performance evaluation in client-side. Em *Anais do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, páginas 127–140, Natal, RN, Brasil. SBC.

Costan, V. e Devadas, S. (2016). Intel SGX explained. Cryptology ePrint Archive, Report 2016/086.

Coughlin, M., Keller, E. e Wustrow, E. (2017). Trusted Click: Overcoming security issues of NFV in the cloud. Em *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, Scottsdale, AZ, EUA. ACM.

Davenport, S. e Ford, R. (2014). SGX: the good, the bad and the downright ugly. *Virus Bulletin*.

Duan, H., Wang, C., Yuan, X., Zhou, Y., Wang, Q. e Ren, K. (2019). LightBox: Full-stack protected stateful middlebox at lightning speed. Em *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, páginas 2351–2367, Londres, Reino Unido. ACM.

Duan, Y., Cao, Y. e Sun, X. (2015a). Various "aaS" of Everything as a Service. Em *Proceedings of the 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, páginas 1–6, Takamatsu, Japão. IEEE.

Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. C. e Hu, B. (2015b). Everything as a Service (XaaS) on the cloud: Origins, current and future trends. Em *Proceedings of the 8th International Conference on Cloud Computing*, páginas 621–628, Nova Iorque, NY, EUA. IEEE.

Duan, Y., Sun, X., Longo, A., Lin, Z. e Wan, S. (2016). Sorting terms of "aas" of everything as a service. *International Journal of Networked and Distributed Computing*, 4:32–44.

Fetzer, C. (2016). Building critical applications using microservices. *IEEE Security Privacy*, 14(6):86–89.

Fisch, B. A., Vinayagamurthy, D., Boneh, D. e Gorbunov, S. (2017). Iron: Functional encryption using Intel SGX. Em *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, páginas 1–37, Dallas, TX, EUA. ACM.

freedesktop.org (2018). D-Bus.

Fuhry, B., Bahmani, R., Brasser, F., Hahn, F., Kerschbaum, F. e Sadeghi, A.-R. (2017). HardIDX: Practical and secure index with SGX. Em *Proceedings of the XXXI Data and Applications Security and Privacy*, páginas 386–408, Filadélfia, PA, EUA. Springer.

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (2004). *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*. Bookman.

Gaspar, L., Fischer, V., Bernard, F., Bossuet, L. e Cotret, P. (2010). Hcrypt: A novel concept of crypto-processor with secured key management. Em *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, páginas 280–285, Cancún, México. IEEE.

Gjerdrum, A. T., Pettersen, R., Johansen, H. D. e Johansen, D. (2017). Performance of trusted computing in cloud infrastructures with Intel SGX. Em *Proceedings of the 7th International Conference on Cloud Computing and Services Science*, páginas 668–675, Porto, Portugal. SCITEPRESS.

Goltzsche, D., Wulf, C., Muthukumaran, D., Rieck, K., Pietzuch, P. e Kapitza, R. (2017). TrustJS: Trusted client-side execution of JavaScript. Em *Proceedings of the 10th European Workshop on Systems Security*, Belgrado, Sérvia. ACM.

Greene, J. (2012). Intel Trusted Execution Technology (white paper). *Online: http://www.intel.com/txt*.

Göttel, C., Pires, R., Rocha, I., Vaucher, S., Felber, P., Pasin, M. e Schiavoni, V. (2018). Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms. Em *Proceedings of the 37th Symposium on Reliable Distributed Systems*, páginas 133–142, Salvador, BA, Brasil. IEEE.

Hoekstra, M., Lal, R., Pappachan, P., Phegade, V. e Del Cuvillo, J. (2013). Using innovative instructions to create trustworthy software solutions. Em *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, páginas 11:1–11:1, Tel-Aviv, Israel. ACM.

Hou, F., Wang, Z., Tang, Y. e Liu, Z. (2004). Protecting integrity and confidentiality for data communication. Em *Proceedings of the 9th International Symposium on Computers And Communications*, páginas 357–362, Alexandria, Egito. IEEE.

Intel (2013). Graphics - Blu-Ray Disc playback with the Intel(R) Graphics (FAQ). `http://www.intel.com/support/graphics/sb/CS-029871.htm#whatis`. Acessado em 30/08/2018.

Intel (2014). *Intel Software Guard Extensions Programming Reference*. Intel Corporation.

Intel (2016a). *Intel Software Guard Extensions Developer Guide*. Intel Corporation.

Intel (2016b). *Intel Software Guard Extensions SDK for Linux OS Developer Reference*. Intel Corporation.

Intel (2018). Intel Software Guard Extensions (SGX) software development guidance for potential bounds check bypass (CVE-2017-5753) side channel exploits (white paper). *Intel Corporation*.

Jain, P., Desai, S., Kim, S., Shih, M.-W., Lee, J., Choi, C., Shin, Y., Kim, T., Kang, B. B. e Han, D. (2016). OpenSGX: An open platform for SGX research. Em *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, EUA. Internet Society.

Jang, Y., Lee, J., Lee, S. e Kim, T. (2017). SGX-Bomb: Locking down the processor via rowhammer attack. Em *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, páginas 5:1–5:6, Shanghai, China. ACM.

Jia, Y., Tople, S., Moataz, T., Gong, D., Saxena, P. e Liang, Z. (2017). Robust synchronous P2P primitives using SGX enclaves. *IACR Cryptology ePrint Archive*.

Karande, V., Bauman, E., Lin, Z. e Khan, L. (2017). SGX-Log: Securing system logs with SGX. Em *Proceedings of the Asia Conference on Computer and Communications Security*, páginas 19–30, Abu Dhabi, Emirados Árabes Unidos. ACM.

Katz, J. e Lindell, Y. (2014). *Introduction to modern cryptography*. CRC Press.

Kim, S., Han, J., Ha, J., Kim, T. e Han, D. (2017). Enhancing security and privacy of tor's ecosystem by using trusted execution environments. Em *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, Boston, MA, EUA. USENIX Association.

Kim, T., Park, J., Woo, J., Jeon, S. e Huh, J. (2019). ShieldStore: Shielded in-memory key-value storage with SGX. Em *Proceedings of the 14th EuroSys Conference 2019*, páginas 14:1–14:15, Dresden, Alemanha. ACM.

Kircher, M. e Jain, P. (2002). Pooling pattern. Em *Proceedings of the 7th European Conference on Pattern Languages of Programs*, Irsee, Alemanha.

Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M. e Yarom, Y. (2019). Spectre attacks: Exploiting speculative execution. Em *Proceedings of the IEEE Symposium on Security and Privacy*, páginas 19–37, San Fransisco, CA, USA. IEEE.

Krawiecka, K., Kurnikov, A., Paverd, A., Mannan, M. e Asokan, N. (2018). SafeKeeper: Protecting web passwords using trusted execution environments. Em *Proceedings of the World Wide Web Conference*, páginas 349–358, Lyon, França. International World Wide Web Conferences Steering Committee.

Le Quoc, D., Gregor, F., Singh, J. e Fetzer, C. (2019). SGX-PySpark: Secure distributed data analytics. Em *Proceedings of the World Wide Web Conference*, páginas 3564–3563, São Francisco, CA, EUA. International World Wide Web Conferences Steering Committee.

Leach, K., Zhang, F. e Weimer, W. (2017). Scotch: Combining Software Guard Extensions and system management mode to monitor cloud resource usage. Em *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*, páginas 403–424, Atlanta, GA, EUA. Springer.

Lesjak, C., Hein, D. e Winter, J. (2015). Hardware-security technologies for industrial IoT: TrustZone and security controller. Em *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society*, páginas 2589–2595, Yokohama, Japão. IEEE.

Li, D., Lin, R., Tang, L., Liu, H. e Tang, Y. (2019). SGXPool: Improving the performance of enclave creation in the cloud. *Transactions on Emerging Telecommunications Technologies*.

Li, G. e Wei, M. (2014). Everything-as-a-service platform for on-demand virtual enterprises. *Information Systems Frontiers*, 16(3):435–452.

Li, H., Lin, J., Li, B. e Cheng, W. (2018). PoS: Constructing practical and efficient public key cryptosystems based on symmetric cryptography with SGX. Em *Proceedings of the International Conference on Information and Communications Security*, páginas 767–777, Lille, França. Springer.

Liang, X., Shetty, S., Zhang, L., Kamhoua, C. e Kwiat, K. (2017). Man in the Cloud (MITC) Defender: SGX-Based user credential protection for synchronization applications in cloud computing platform. Em *Proceedings of the 10th International Conference on Cloud Computing*, páginas 302–309, Honolulu, HI, EUA. IEEE.

Lind, J., Naor, O., Eyal, I., Kelbert, F., Pietzuch, P. e Sirer, E. G. (2019). Teechain: A secure payment network with asynchronous blockchain access. Em *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, Huntsville, ON, Canadá. ACM.

McKeen, F., Alexandrovich, I., Anati, I., Caspi, D., Johnson, S., Leslie-Hurd, R. e Rozas, C. (2016). Intel Software Guard Extensions (Intel SGX) support for dynamic memory management inside an enclave. Em *Proceedings of the 5th International Workshop on Hardware and Architectural Support for Security and Privacy*, páginas 10:1–10:9, Seul, Coréia do Sul. ACM.

McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V. e Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. Em *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, páginas 10:1–10:1, Tel-Aviv, Israel. ACM.

Microsoft (2017). Event-based asynchronous pattern overview. `https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/event-based-asynchronous-pattern-overview`. Acessado em 22/05/2019.

Milutinovic, M., He, W., Wu, H. e Kanwal, M. (2016). Proof of luck: An efficient blockchain consensus protocol. Em *Proceedings of the 1st Workshop on System Software for Trusted Execution*, páginas 2:1–2:6, Trento, Itália. ACM.

Mofrad, M. H., Lee, A. e Gray, S. L. (2017). Leveraging Intel SGX to create a nondisclosure cryptographic library. *arXiv preprint arXiv:1705.04706*.

Mofrad, S., Zhang, F., Lu, S. e Shi, W. (2018). A comparison study of Intel SGX and AMD memory encryption technology. Em *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, Los Angeles, CA, EUA. ACM.

Moghimi, A., Irazoqui, G. e Eisenbarth, T. (2017). CacheZoom: How SGX amplifies the power of cache attacks. Em *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*, páginas 69–90, Taipei, Taiwan. Springer.

Nguyen, H., Acharya, B., Ivanov, R., Haeberlen, A., Phan, L. T. X., Sokolsky, O., Walker, J., Weimer, J., Hanson, W. e Lee, I. (2016). Cloud-based secure logger for medical devices. Em *Proceedings of IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies*, páginas 89–94, Washington, DC, EUA.

Orenbach, M., Lifshits, P., Minkin, M. e Silberstein, M. (2017). Eleos: Exitless OS services for SGX enclaves. Em *Proceedings of the 12th European Conference on Computer Systems*, páginas 238–253, Belgrado, Sérvia. ACM.

Paoloni, G. (2010). How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures. *Intel Corporation*. `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf`.

Park, J., Park, S., Kang, B. B. e Kim, K. (2019). eMotion: An SGX extension for migrating enclaves. *Computers & Security*, 80:173–185.

Priebe, C., Vaswani, K. e Costa, M. (2018). EnclaveDB: A secure database using SGX. Em *Proceedings of the IEEE Symposium on Security and Privacy*, páginas 264–278, San Francisco, CA, EUA. IEEE.

Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor.

Richter, L., Götzfried, J. e Müller, T. (2016). Isolating operating system components with Intel SGX. Em *Proceedings of the 1st Workshop on System Software for Trusted Execution*, páginas 8:1–8:6, Trento, Itália. ACM.

Robison, S. (2008). The next wave: Everything as a service. *Executive Viewpoint*.

Sartakov, V. A., Brenner, S., Ben Mokhtar, S., Bouchenak, S., Thomas, G. e Kapitza, R. (2018). EActors: Fast and flexible trusted computing using SGX. Em *Proceedings of the 19th International Middleware Conference*, páginas 187–200, Rennes, França. ACM.

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Em *Proceedings of the 1st International Conference on Peer-to-Peer Computing*, páginas 101–102, Linköping, Suécia. ACM.

Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G. e Russinovich, M. (2015). VC3: Trustworthy data analytics in the cloud using SGX. Em *Proceedings of the IEEE Symposium on Security and Privacy*, páginas 38–54, San Jose, CA, EUA. IEEE.

Schwarz, M., Weiser, S., Gruss, D., Maurice, C. e Mangard, S. (2017). Malware Guard Extension: Using SGX to conceal cache attacks. Em *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Bonn, Alemanha. Springer.

Shaon, F., Kantarcioglu, M., Lin, Z. e Khan, L. (2017). SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors. Em *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, páginas 1211–1228, Dallas, TX, EUA. ACM.

Shen, Y., Chen, Y., Chen, K., Tian, H. e Yan, S. (2018). To isolate, or to share?: That is a question for Intel SGX. Em *Proceedings of the 9th Asia-Pacific Workshop on Systems*, páginas 4:1–4:8, Jeju Island, Coréia do Sul. ACM.

Shih, M.-W., Kumar, M., Kim, T. e Gavrilovska, A. (2016). S-NFV: Securing NFV states by using SGX. Em *Proceedings of the 1st ACM International Workshop on Security in SDN and NFV*, páginas 45–48, New Orleans, LA, EUA. ACM.

Shih, M.-W., Lee, S., Kim, T. e Peinado, M. (2017). T-SGX: Eradicating controlled-channel attacks against enclave programs. Em *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, EUA. Internet Society.

Shinde, S., Le Tien, D., Tople, S. e Saxena, P. (2017). Panoply: Low-TCB Linux applications with SGX enclaves. Em *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, EUA. Internet Society.

Silva, R., Barbosa, P. e Brito, A. (2017). DynSGX: A privacy preserving toolset for dinamically loading functions into Intel SGX enclaves. Em *International Conference on Cloud Computing Technology and Science*, páginas 314–321, Hong Kong, China. IEEE.

Sinha, R. e Christodorescu, M. (2018). VeritasDB: High throughput key-value store with integrity. *IACR Cryptology ePrint Archive*, 2018:251.

Sobchuk, J., O'Melia, S., Utin, D. e Khazan, R. (2018). Leveraging Intel SGX technology to protect security-sensitive applications. Em *Proceedings of the 17th International Symposium on Network Computing and Applications*, páginas 1–5, Cambridge, MA, EUA. IEEE.

Sun, W., Zhang, R., Lou, W. e Thomas Hou, Y. (2018). Rearguard: Secure keyword search using trusted hardware. Em *Proceedings of the IEEE Conference on Computer Communications*, páginas 801–809, Honolulu, HI, EUA. IEEE.

Taassori, M., Shafiee, A. e Balasubramonian, R. (2018). VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. Em *International Conference on Architectural Support for Programming Languages and Operating Systems*, páginas 665–678, Williamsburg, VA, EUA. ACM.

TCG (2016). Trusted Platform Module library - part 1: Architecture. `https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf`. Acessado em 02/04/2017.

Tian, H., Zhang, Q., Yan, S., Rudnitsky, A., Shacham, L., Yariv, R. e Milshten, N. (2018). Switchless calls made practical in Intel SGX. Em *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, páginas 22–27, Toronto, Canadá. ACM.

Tian, H., Zhang, Y., Xing, C. e Yan, S. (2017). SGXKernel: A library operating system optimized for Intel SGX. Em *Proceedings of the Computing Frontiers Conference*, páginas 35–44, Siena, Itália. ACM.

Tikkinen-Piri, C., Rohunen, A. e Markkula, J. (2018). Eu general data protection regulation: Changes and implications for personal data collecting companies. *Computer Law & Security Review*, 34(1):134–153.

Trach, B., Krohmer, A., Gregor, F., Arnautov, S., Bhatotia, P. e Fetzer, C. (2018). ShieldBox: Secure middleboxes using shielded execution. Em *Proceedings of the Symposium on SDN Research*, páginas 2:1–2:14, Los Angeles, CA, EUA. ACM.

Tran, M., Luu, L., Kang, M. S., Bentov, I. e Saxena, P. (2018). Obscuro: A bitcoin mixer using trusted execution environments. Em *Proceedings of the 34th Annual Computer Security Applications Conference*, páginas 692–701, San Juan, PR, EUA. ACM.

Tsai, C. C., Arora, K. S., Bandi, N., Jain, B., Jannen, W., John, J., Kalodner, H. A., Kulkarni, V., Oliveira, D. e Porter, D. E. (2014). Cooperation and security isolation of library OSes for multi-process applications. Em *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, páginas 9:1–9:14, Amsterdam, Países Baixos. ACM.

Tsai, C. C., Porter, D. E. e Vij, M. (2017). Graphene-SGX: A practical library OS for unmodified applications on SGX. Em *Proceedings of the USENIX Annual Technical Conference*, páginas 645–658, Santa Clara, CA, EUA. USENIX Association.

Vacca, J. R. (2016). *Cloud Computing Security: Foundations and Challenges*. CRC Press.

Wang, J., Fan, C., Wang, J., Cheng, Y., Zhang, Y., Zhang, W., Liu, P. e Hu, H. (2019a). SvTPM: A secure and efficient vTPM in the cloud. *arXiv preprint arXiv:1905.08493*.

Wang, W., Jiang, Y., Shen, Q., Huang, W., Chen, H., Wang, S., Wang, X., Tang, H., Chen, K., Lauter, K. E. e Lin, D. (2019b). Toward scalable fully homomorphic encryption through light trusted computing assistance. *arXiv preprint arXiv:1905.07766*.

Weichbrodt, N., Kurmus, A., Pietzuch, P. e Kapitza, R. (2016). AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. Em *Proceedings of the European Symposium on Research in Computer Security*, páginas 440–457, Heraklion, Grécia. Springer.

Weis, S. (2014). Protecting data in-use from firmware and physical attacks. Em *Proceedings of the Black Hat*.

Weisse, O., Bertacco, V. e Austin, T. (2017). Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves. Em *Proceedings of the 44th Annual International Symposium on Computer Architecture*, páginas 81–93, Toronto, ON, Canadá. ACM.

Will, N. C., Condé, R. C. R. e Maziero, C. A. (2017). Mecanismos de Segurança Baseados em Hardware: Uma Introdução à Arquitetura Intel SGX. Em NUNES, R. C., CANEDO, E. D. e SOUSA JÚNIOR, R. T., editores, *Minicursos do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, capítulo: 2, páginas 49–98. Sociedade Brasileira de Computação, Brasilia, DF, Brasil.

Will, N. C., Heinrich, T., Viescinski, A. B. e Maziero, C. A. (2020). A trusted message bus built on top of D-Bus. Em *Proceedings of the XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Petrópolis, RJ, Brasil. SBC.

Will, N. C. e Maziero, C. A. (2020). Using a shared SGX enclave in the UNIX PAM authentication service. Em *Proceedings of the 14th IEEE International Systems Conference*, Montreal, QC, Canadá. IEEE.

Wojtczuk, R. e Rutkowska, J. (2009). Attacking Intel Trusted Execution Technology. Em *Proceedings of the Black Hat*.

Zhang, F., Cecchetti, E., Croman, K., Juels, A. e Shi, E. (2016). Town Crier: An authenticated data feed for smart contracts. Em *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, páginas 270–282, Vienna, Áustria. ACM.

Zhao, C., Saifuding, D., Tian, H., Zhang, Y. e Xing, C. (2016). On the Performance of Intel SGX. Em *Proceedings of the 13th Web Information Systems and Applications Conference*, páginas 184–187, Wuhan, China. IEEE.

Zheng, W., Dave, A., Beekman, J. G., Popa, R. A., Gonzalez, J. E. e Stoica, I. (2017). Opaque: An oblivious and encrypted distributed analytics platform. Em *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, páginas 283–298, Boston, MA, EUA. USENIX Association.