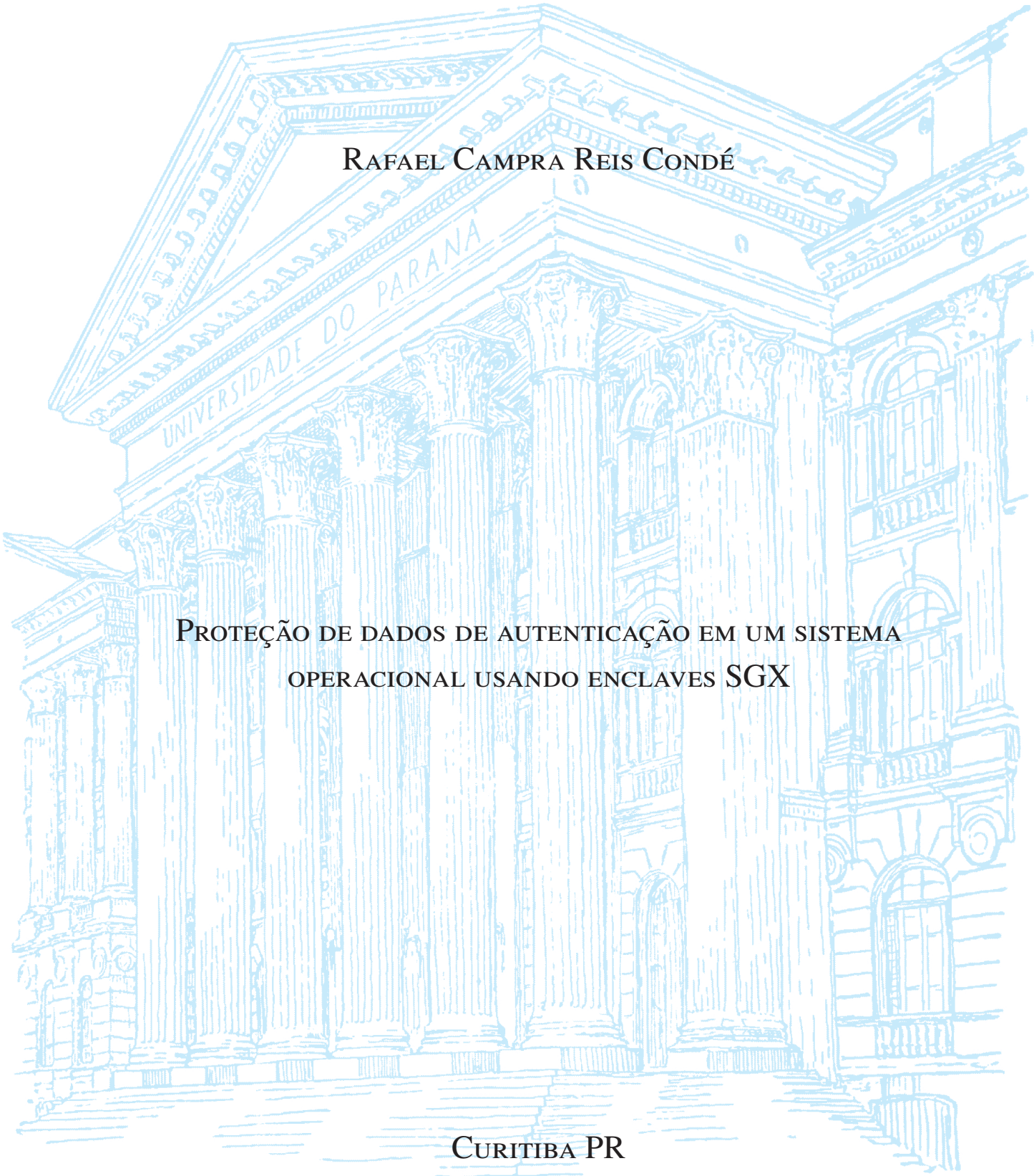
Universidade Federal do Paraná

Rafael Campra Reis Condé



Proteção de dados de autenticação em um sistema operacional usando enclaves SGX

Curitiba PR
2017

Rafael Campra Reis Condé

Proteção de dados de autenticação em um sistema operacional usando enclaves SGX

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre 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
2017

C745p Condé, Rafael Campra Reis
Proteção de dados de autenticação em um sistema operacional usando enclaves SGX / Rafael Campra Reis Condé. – Curitiba, 2017.
100 f. : il. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2017.

Orientador: Carlos Alberto Maziero .
Bibliografia: p. 91-95.

1. Proteção de dados. 2. Autenticação. 3. Redes de Computadores – Medidas de segurança. 4. Criptografia de dados (Computação) . I. Universidade Federal do Paraná. II.Maziero, Carlos Alberto. III. Título.

CDD: 005.8

MINISTÉRIO DA EDUCAÇÃO
SETOR CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA

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 Dissertação de Mestrado de **RAFAEL CAMPRA REIS CONDÉ** intitulada: **Proteção de dados de autenticação em um sistema operacional usando enclaves SGX**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação no rito de defesa.

A outorga do título de mestre 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, 11 de Dezembro de 2017.

CARLOS ALBERTO MAZIERO
Presidente da Banca Examinadora (UFPR)

MARCELO DE OLIVEIRA ROSA
Avaliador Externo (UTFPR)

ANDRÉ RICARDO ABED GRÉGIO
Avaliador Interno (UFPR)

DIEGO DE FREITAS ARANHA
Avaliador Externo (UNICAMP)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

À minha esposa Flávia e aos meus pais Altair e Nilcéa.

Agradecimentos

No decorrer desta pós-graduação tive a oportunidade de conhecer e conviver com pessoas de alta capacidade técnica e profissional, o que me possibilitou, além novas amizades, adquirir novos conhecimentos e acelerar o crescimento técnico, profissional e pessoal. Ao término desta jornada tenho o dever de agradecer à minha esposa, Flávia, que me deu suporte em todos os momentos no decorrer deste trabalho. Também não posso deixar de agradecer aos meus pais Altair e Nilcéa que sempre me deram apoio incondicional para todas as minhas realizações pessoais e profissionais.

Mas não foi somente a família que tornou possível o desenvolvimento deste trabalho. Devo também agradecimento aos amigos do 11º Centro de Telemática, em especial ao Cel Flávio, chefe do Centro, que de fato tornou possível o desenvolvimento do trabalho através da autorização e suporte necessário. Também aos amigos Cap Mateus, Ten Diego e Ten Prioli, colegas na caserna e também na pós-graduação na UFPR que, por diversas vezes, interromperam seu trabalho para poder ajudar.

Agradeço ao orientador Prof. Dr. Carlos Maziero que mostrou grande capacidade técnica, excelente didática e sempre se colocou à disposição para ajudar, sendo elemento fundamental nos momentos críticos do trabalho. Também ao Prof. Dr. Marcelo Rosa, de quem muito aprendi e sempre se fez acessível para ajudar nas dificuldades enfrentadas. Aos Prof. Dr. Diego Aranha e Prof. Dr. André Grégio, que através de suas análises e comentários contribuíram de forma inestimável para o grau de qualidade do resultado final do trabalho e aos demais professores que participaram da formação através das disciplinas ministradas que contribuíram para que o objetivo fosse atingido. Por fim, agradeço aos amigos que fiz nesta jornada.

Resumo

Quando se trata sobre segurança computacional alguns aspectos devem ser observados. Um princípio importante é manter a base de computação confiável (TCB) a menor possível. Desta forma a superfície de ataque fica reduzida, o que restringe as possibilidades de ataque do adversário e aumenta a segurança do sistema. Mesmo mantendo uma TCB pequena, não existe técnica de segurança que ofereça garantia de proteção contra qualquer tipo de adversário. Por este motivo, aplicações que requerem alto nível de sigilo devem dispor de um esquema de proteção projetado em diversas de camadas de segurança, de forma que o conteúdo sensível não seja comprometido mesmo considerando o caso do adversário explorar vulnerabilidades e violar uma das camadas de proteção. Em 2015 a Intel lançou em seus processadores a tecnologia *Software Guard Extensions* (SGX), que introduz um novo conjunto de instruções de CPU que permite que a aplicação utilize uma região encriptada de memória, denominada *enclave*, que é protegida inclusive de códigos com alto nível de privilégio como do *kernel* e do sistema operacional. Junto ao SGX, a Intel provê, ainda, um mecanismo de criptografia de dados, denominado *selagem*, onde operações de cifragem ou decifragem são possíveis somente dentro dos enclaves. Em geral aplicações comuns têm em sua TCB além do próprio código, partes do sistema operacional, *kernel* e/ou hipervisor. O SGX proporciona a possibilidade de reduzir a TCB apenas à CPU e à fração sensível da aplicação, que será colocada dentro da região segura do enclave. Ainda, o SGX torna possível, através do processo de selagem, a adição de uma camada extra de segurança no armazenamento de informações sensíveis. Este trabalho propõe uma arquitetura de proteção de arquivos de senha valendo-se das possibilidades trazidas pela tecnologia Intel SGX e implementa, na forma de uma prova de conceito, um módulo de autenticação para o *framework* PAM do sistema operacional Linux, baseado no módulo `pam_unix.so`, que utiliza como referência de autenticação um arquivo de credenciais protegido. Neste esquema o arquivo de senhas é armazenado selado através do SGX e todo o processo de validação de usuário e senha para autenticação é realizado seguramente dentro do enclave. É senso comum que segurança e desempenho em geral caminham em direções opostas, assim já era esperado custo para se trabalhar com regiões de memória e arquivos encriptados. No entanto, o hardware SGX foi projetado para realizar operações criptográficas de maneira otimizada de forma a reduzir os impactos de desempenho. Usando um arquivo de credenciais de 500KB o tempo esperado para realização da autenticação aumentou de 1.4ms, no módulo original, para 27.1ms no protótipo implementado. O impacto no desempenho não é suficientemente grande para ser percebido pelo usuário, o que mostra que a solução é adequada para a finalidade proposta. A arquitetura apresentada neste trabalho também pode ser aplicada em outros sistemas de autenticação.

Palavras-chave: Autenticação, Pluggable Authentication Module, PAM, Software Guard Extensions, SGX, Enclave, Criptografia, Proteção de credenciais.

Abstract

When it comes to computational security, some aspects must be observed. An important principle is to maintain the trusted computing base (TCB) as small as possible. With a reduced attack surface, attack possibilities are restricted and an improvement in the system security is achieved. Even keeping the TCB small, there is no security technique that guarantees protection against any type of adversary. For this reason, when it comes to applications that require high level of security, it is desirable that the protection scheme be designed with several layers of security. Thus, sensitive content will not be compromised, even considering that an adversary might find a vulnerability and violate some of the protection layers. In 2015, Intel released the Software Guard Extensions (SGX) technology in its processors, which introduces of a new set of CPU instructions that allows the application to allocate a private memory region, called *enclave*, that is protected even from high-level privileged kernel and operating system processes. Along with SGX, Intel also provided a mechanism of data encryption, called *sealing*, where encryption and decryption operations are possible only within enclaves. Usually, non-SGX applications include in its TCB not only the application code itself, but and also parts of the operating system, kernel and hypervisor. Thus SGX provides the possibility of reducing the TCB to just the CPU and to the sensitive fraction of the application, which is safely placed inside the enclave. In addition, SGX makes it possible to add an extra layer of security to sensitive information storage through the sealing process. This document presents a novel passwords file protection architecture leveraging Intel® SGX deliveries and implements, as a proof of concept, an authentication module for the Linux PAM framework based on `pam_unix.so` module. In this scheme, an extra layer of security is applied to the password file, by storing it sealed by the SGX mechanism. All user and password checking for authentication is securely done within the enclave. It is common sense that safety and performance in general walk in opposite directions. Obviously a cost was expected to work with memory regions and encrypted files. SGX hardware is designed to perform cryptographic operations efficiently to reduce performance impacts. Using a 500KB credential file the expected time to perform the authentication increased from 1.4ms in the original module to 27.1ms in the prototype. Despite the overhead the result is perfectly suitable for the proposed application. Finally, performance impacts were quantified due to the use of SGX. The architecture presented in this scheme is also worthy to many other authentication system.

Keywords: Authentication, Pluggable Authentication Module, PAM, Software Guard Extensions, SGX, Enclave, Credencial file protection.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

API	Application Programming Interface
ASIC	Application Specific Integrated Circuits
CPU	Central Processing Unit
DES	Data Encryption Standard
DMA	Direct Memory Access
EPC	Enclave Page Cache
EPCM	Enclave Page Cache Map
EPID	Ehanced Privacy ID
FBI	Federal Bureau of Investigation
FPGA	Field Programmable Gate Array
GPP	General Propose Processor
GPU	Graphics Processing Unit
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IoT	Internet of Things
ISV	Independent Software Vendor
LPC	Low Pin Count
MAC	Message Authentication Code
ME	Management Engine
Nfc	Near Field Communication
NSS	Network Security Service
OTP	One Time Password
PAM	Pluggable Authentication Module
PEM	Privacy-enhanced Electronic Mail
PRM	Processor Reserved Memory
QE	Quoting Enclave
RAM	Random Access Memory
RFID	Radio-Frequency IDentification
SDK	Software Development Kit
SMM	System Management Mode
SMS	Short Message Service
SGX	Software Guard Extensions
SVN	Security Version Number
TCB	Trusted Computing Base
TPM	Trusted Plataform Module
VMM	Virtual Machine Monitor

Lista de Símbolos

σ	desvio padrão
$\bar{X}$	média aritmética

Capítulo 1

Introdução

Este capítulo apresenta a contextualização que motivou o presente trabalho. As Seções 1.1 e 1.2 abordam o desafio que o cenário atual representa para o armazenamento seguro das credenciais de autenticação e a motivação que levou à proposta apresentada brevemente na Seção 1.3. Por fim, as Seções 1.4 e 1.5 apresentam, respectivamente, um resumo das contribuições deste trabalho e a organização deste documento.

1.1 Desafio

Com a expansão do uso de dispositivos digitais, pessoas e empresas têm ficado cada vez mais dependentes de recursos computacionais e da Internet para exercer desde atividades simples do dia a dia, até complexas atividades industriais. A McAfee Labs estima que devido à convergência da Internet das Coisas (IoT) 200 bilhões de dispositivos estejam conectados à internet em 2020 [Weafer, 2016]. Consequentemente, é crescente a quantidade de dados e aplicações sensíveis sendo armazenados e executados em sistemas computacionais, sejam locais ou na nuvem. Ainda segundo a McAfee Labs, em 2006 empresas enfrentavam, em média, 25 novas ameaças por dia, enquanto hoje 500.000 ameaças são vistas diariamente. O FBI estima que crimes cibernéticos geram U$67.2 bilhões de prejuízo por ano [FBI, 2005]. Desta forma, não causa estranheza que os ataques a recursos computacionais estão ficando cada vez mais sofisticados e difíceis de se detectar e de se defender.

Sistemas de autenticação normalmente constituem parte fundamental da proteção de dados e aplicações sensíveis e, portanto, frequentemente são alvos de ataques. Em linhas gerais, verifica-se três categorias de sistemas de autenticação: baseados em conhecimento (ex: senhas); baseados em fatores inerentes ao indivíduo (ex: biometria); e baseados em propriedade (ex: chaves privadas) [Ju et al., 2014]. O método mais tradicional, que ainda é largamente utilizado, é a autenticação através de usuários e senhas. Em muitos casos, como no sistema operacional Linux, a lista dos usuários e senhas são armazenados em um arquivo ou banco de dados protegido. Situação similar ocorre nos sistemas de autenticação baseados na propriedade e em fatores inerentes ao indivíduo, onde faz-se necessário o armazenamento das senhas, traços biométricos ou chaves para servir como referência de comparação no ato da autenticação. Estas informações são armazenados em arquivos ou bancos de dados e constituem objetos extremamente sensíveis que, nas mãos de adversários, podem causar efeitos catastróficos. Por este motivo dados sensíveis devem ser armazenados sob algumas camadas de proteção.

1.2 Motivação

A história nos mostra a importância da proteção dos arquivos que servem como referência para realização de autenticação seja por senhas, traços biométricos ou outros métodos. Uma rápida pesquisa no Google é suficiente para encontrar uma série de eventos onde senhas e dados de usuários foram capturados por agentes mal intencionados. O exemplo famoso mais recente é o caso da Apple que, supostamente, teve 300 milhões de contas de e-mail de usuários do *iCloud* que não usam autenticação de dois fatores[1] capturadas por uma equipe de *hackers* [Cox, 2017].

Na revisão bibliográfica apresentada no Capítulo 3, constata-se que existe um grande esforço para aumentar a dificuldade computacional para a quebra de *hashes*[2] e arquivos encriptados de modo a melhorar a proteção de dados sensíveis. No entanto, ainda assim, com a melhoria da capacidade de hardware, quebrar senhas tem ficado cada vez mais barato e, assim, de posse do arquivo de senhas, mesmo que cifrado, a única limitação de um adversário motivado é a quantidade de recursos que está disposto a investir para montar um ataque *offline* e descobrir as senhas desejadas. Isto sem considerar que estudos empíricos, como os realizados por [Doel, 2013] e [Slain, 2016], mostram que usuários tendem a escolher senhas de baixa entropia que são bem mais fáceis de se quebrar através de, por exemplo, ataques por dicionário ou por *rainbow tables*. Além disso o armazenamento seguro das chaves criptográficas constitui um grande desafio.

Métodos alternativos de autenticação, como por exemplo biometria, também estão sendo bastante abordados por trabalhos recentes. Autenticação por biometria é realizado através de características individuais biológicas inalteráveis e universais como impressão digital, íris, reconhecimento facial e reconhecimento vocal. Este tipo de autenticação tem se mostrado bastante eficiente para eliminar problemas como senhas fracas, extravio de senhas anotadas por usuários e a transferência não autorizada de senha entre usuários. No entanto, além da dificuldade causada pela necessidade de hardware especial para colher as informações biométricas, o armazenamento destas informações, assim como no caso do armazenamento das senhas tradicionais, deve ser alvo de preocupações especiais. Por se tratar de características inerentes ao indivíduo, muitas vezes, a revogabilidade de amostras biométricas pode não ser tão simples quanto no caso de senhas tradicionais. O extravio dessas amostras pode, portanto, gerar grandes problemas como roubo de identidade e a impossibilidade de utilizar os mesmos dados biométricos para se autenticar com segurança no futuro.

Ao final de 2015 a Intel lançou uma tecnologia com potencial de melhorar a segurança de aplicações sensíveis, dentre elas, dos sistemas de autenticação: o Intel SGX, que foi projetado com o objetivo de proteger dados e código de acessos e modificações não autorizadas. Esta proteção é garantida por um novo conjunto de instruções de CPU que possibilitam a criação de regiões privadas de memória, chamadas de *enclaves*[3], cujo acesso é restrito até mesmo para códigos com alto nível de privilégio como do *kernel* e do sistema operacional. Junto ao SGX foi entregue um mecanismo de criptografia de dados, denominado *selagem*. A chave criptográfica utilizada neste processo é gerada especificamente para um determinado enclave e não necessita ser armazenada em memória. Operações de cifragem ou decifragem só são possíveis dentro dos enclaves.

[1] Autenticação de dois fatores é uma camada adicional de segurança utilizada no processo de autenticação onde um código de verificação é enviado ao usuário através de um meio alternativo como SMS por exemplo.

[2] "Quebra de *hashes*" ou "quebra de senhas" refere-se ao ato de se obter credenciais válidas de autenticação em tempo razoável sem conhecimento prévio das mesmas.

[3] A memória dos enclaves é criptografada utilizando chaves geradas pelo processador SGX. Estas chaves estão restritas ao processador e são alteradas a cada ciclo de energia.

1.3 Proposta

Neste contexto, este trabalho aborda o problema da proteção de arquivos contendo credenciais de autenticação. Mais precisamente a melhoria da proteção do arquivo de senhas usado pelo sistema de autenticação do Linux, o `/etc/shadow`. No entanto, a arquitetura proposta é aplicável aos diversos sistemas de autenticação que utilizam arquivos ou base de dados para armazenar as credenciais de autenticação.

No sistema operacional GNU/Linux a autenticação é feita, por padrão, através de uma plataforma (*framework*) chamada PAM. O PAM utiliza módulos conectáveis que estabelecem os critérios de autenticação. Também por padrão, a lista de usuários e senhas de autenticação é armazenada nos arquivos `passwd` ou `shadow`.

Especificamente no caso do arquivo `shadow`, pode-se verificar basicamente duas camadas de segurança: a proteção dos níveis de acesso aplicados pelo sistema operacional onde o acesso é permitido ao administrador mas bloqueado para os demais usuários; e a proteção das senhas que são gravadas no arquivo em forma de *hash*. A função de *hash* utilizada pode ser escolhida pelo administrador do sistema, mas os diversos algoritmos disponíveis para esta criptografia são de conhecimento disseminado. Desta forma, esta camada de proteção é vulnerável à ataques *offline* de adversários motivados e com recursos computacionais suficientes. No entanto, para isto o adversário ainda precisa ter acesso ao arquivo `shadow`, que é protegido pelo sistema operacional. Mas esta proteção não considera situações onde sistema operacional e o *kernel* não são confiáveis e podem ser maliciosos como no caso dos *Iago Attacks* [Checkoway e Shacham, 2013]. Neste caso o atacante poderia utilizar-se do alto nível de privilégio para capturar o arquivo de senhas e montar um ataque *offline* para quebrá-las. Vale ressaltar que problemas muito similares a este podem ser encontrados nos diversos métodos de autenticação.

Este trabalho propõe uma arquitetura de proteção de arquivos de senhas valendo-se da tecnologia Intel SGX, de forma a aumentar a segurança de sistemas de autenticação. Esta arquitetura consiste em selar o arquivo de senhas e realizar o processo de autenticação dentro dos enclaves SGX. Desta forma, as chances do adversário obter acesso às informações sensíveis de autenticação ficam bastante reduzidas. Mesmo que o atacante consiga capturar o arquivo selado (por exemplo através de privilégios obtidos no caso de um sistema operacional malicioso), montar um ataque *offline* para quebrar as senhas será uma tarefa computacionalmente difícil.

1.4 Contribuição

Foi proposta uma arquitetura para a proteção de arquivos contendo credenciais de autenticação utilizando-se a tecnologia Intel SGX para aumentar o nível de segurança. Também foi realizada uma análise de segurança, considerando as vulnerabilidades e questões em aberto do SGX e da solução proposta. Em seguida foi implementada uma prova de conceito da arquitetura proposta na forma de um módulo de autenticação seguro para o *framework* de autenticação PAM baseado no `pam_unix.so`. Por fim foram realizados testes para avaliar o desempenho do módulo implementado perante o módulo original e perante cenários que permitem inferir como os processos do SGX influenciam o desempenho da solução como um todo.

1.5 Organização do documento

Este trabalho está organizado da seguinte forma: no Capítulo 2, fundamentação teórica, são apresentados uma visão geral sobre sistemas de autenticação, PAM Linux e seu funcionamento

e sobre o Intel SGX e suas possibilidades. O Capítulo 3 apresenta uma revisão bibliográfica contendo os principais trabalhos na área de autenticação; proteção de dados; casos de uso e vulnerabilidades do SGX; e sistemas de segurança computacional de funcionamento similar ao SGX.

O Capítulo 4 apresenta o contexto de aplicação a justificativa para sua realização, a arquitetura de autenticação proposta, uma análise de segurança, limitações da arquitetura e a relação com outros trabalhos. O Capítulo 5 discorre sobre a implementação, os cenários e as métricas utilizados para os testes e seus respectivos resultados. Por fim, o Capítulo 6 resume os resultados obtidos, as contribuições e limitações do trabalho e discute possíveis trabalhos futuros que complementam ou melhoram a solução proposta.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta o embasamento teórico para a concepção da solução ora apresentada. A primeira seção traz conceitos sobre sistemas de autenticação, apresenta os mecanismos mais comuns de sistemas de autenticação e aponta os principais ataques contra estes sistemas. A segunda seção aborda especificamente a autenticação em sistema operacional Linux, explica como é realizado o armazenamento de suas credenciais e discorre sobre o principal *framework* de autenticação do Linux, o PAM. Por fim, a terceira seção trata sobre o SGX, tecnologia presente em uma família de processadores da Intel que, através da introdução de uma nova arquitetura de proteção centrada em regiões privadas de memória, abre possibilidades para as melhorias propostas para sistemas de autenticação.

2.1 Sistemas de autenticação

Três processos principais devem ser implantados para prover acesso controlado a recursos computacionais: *Autenticação*; *Autorização* e; *Responsabilização* (*Accounting*) [Todorov, 2007]. A autenticação trata da identificação e validação do usuário. A autorização se refere à determinação de quais recursos computacionais o usuário autenticado pode ter acesso. Por fim, a responsabilização tem por objetivo o registro das ações realizadas pelo usuário durante o acesso aos recursos computacionais. Para conseguir realizar estas tarefas os sistemas computacionais normalmente usam um objeto abstrato chamado *conta de usuário*, que contém uma série de atributos como o ID de usuário, sua credencial[1], seu nível de privilégio no sistema, dentre outros. Esta seção discorrerá em maiores detalhes o primeiro processo, a autenticação.

Frequentemente, a autenticação é dividida nas fases de *identificação* e de *autenticação*. Na fase de identificação o usuário informa ao sistema a sua identidade, em geral na forma de um ID de usuário. Normalmente nesta fase, o sistema de segurança percorrerá as contas de usuário para identificar o usuário em questão e determinar os privilégios aplicados a ele. A fase de autenticação é responsável por validar a identidade do usuário, já que a mera afirmação do usuário de que ele representa um determinado objeto abstrato do sistema não é necessariamente verdade. Para provar sua identidade o usuário deve fornecer sua credencial. Desta forma, pode-se definir autenticação como o processo de garantir a veracidade da identidade do usuário através de uma ou mais evidências fornecidas.

De modo simplificado, um sistema de autenticação é composto de três componentes: O elemento suplicante ou cliente, que é a parte requerendo autenticação que irá fornecer sua

[1]Credencial é definida por Todorov como a evidência fornecida pelo usuário no processo de autenticação [Todorov, 2007]. O tipo de evidência mais comum é a senha, que é um segredo conhecido somente pelo usuário e pelo sistema de autenticação.

identidade e a evidência que a comprova; o autenticador ou servidor de autenticação é a parte do sistema que irá permitir o acesso do cliente aos recursos requisitados após garantir que ele está devidamente autenticado e autorizado; por fim a autoridade de segurança e banco de dados que irá checar as credenciais do cliente, podendo ser desde a simples consulta à um arquivo até um conjunto de servidores de autenticação distribuídos na rede.

2.1.1 Métodos de autenticação

O trabalho de [Ju et al., 2014] estabelece três métodos básicos para identificar e autenticar os indivíduos por sistemas:

1. **Métodos baseados em conhecimento**: informações que somente o usuário sabe. Este é o caso das senhas. Este método tem dominado a autenticação humano-computador desde 1960 [Bonneau, 2012]. Apesar de ser um método de autenticação amplamente difundido, possui uma série de limitações e problemas. O trabalho de [Herley et al., 2009] enumera alguns problemas como a tendência dos usuários a escolher senhas fracas[1] facilitando ataques por força bruta ou por dicionário; usuários frequentemente se esquecem das senhas, tornando necessários serviços de suporte e formas alternativas de autenticação (como "perguntas desafio") que frequentemente são formas de autenticação ainda mais fracas; usuários tendem a escolher a mesma senha para múltiplos sistemas, desta forma se o adversário conseguir comprometer um sistema ele terá acesso à diversos sistemas e; senhas podem ser roubadas por diversos métodos como *phishing*, engenharia social, *man-in-the-middle* e ataques por *keylogging*.

2. **Métodos baseados em fatores inerentes ao que o indivíduo é ou faz**: este é o caso da biometria. Um identificador biométrico é baseado em características fisiológicas ou comportamentais únicas, inalteráveis e universais que podem ser medidas, como por exemplo impressão digital, características faciais, íris, palma, retina, voz, assinatura manuscrita etc. Devido às suas características este método está se tornando mais popular que os métodos tradicionais como as senhas. O cadastro em sistemas de biometria é baseado em três passos: primeiro uma amostra biométrica é tomada do indivíduo através de, por exemplo, um escaneamento de íris; em seguida esta amostra, ou dados extraídos dela, poderão ser apresentados como referência biométrica para proceder a autenticação; por fim a amostra biométrica, seu extrato, ou ambos são armazenados em um banco de dados (local ou remoto). Uma vez cadastrado, para proceder a autenticação o usuário apresenta sua amostra biométrica que será comparada à referência armazenada. Sistemas biométricos devem tomar cuidados especiais em relação ao segredo de dados pessoais e roubos de identidade. Existem basicamente quatro tipos de ataques contra sistemas de biometria [Murillo-Escobar et al., 2015]: ataques contra o sensor; ataques entre módulos; ataques contra o software e; ataques contra as amostras armazenadas. Ressalta, ainda, que este último tipo de ataque é particularmente perigoso, pois o arquivo contendo as referências biométricas poderia ser substituído por um falso, comprometendo a segurança do sistema. Ainda, de posse deste arquivo, o adversário poderá roubar traços biométricos dos usuários, que não poderão mais utilizá-los com segurança para autenticação futura. Assim, algoritmos visando proteger as referências biométricas devem cumprir quatro requisitos principais: *revogabilidade*, que é a capacidade de cancelar referências biométricas comprometidas

[1] Estudo realizado pela SplashData baseado em mais de 5 milhões de senhas colocadas a venda na internet mostra que 4% das senhas eram 123456 e que 10% das pessoas usam uma entre as 25 piores senhas possíveis [Slain, 2016].

e gerar uma nova a partir da mesma amostra biométrica; *diversidade*, uma vez que, visando proteger a privacidade dos usuários, não deve ser possível identificar o usuário cruzando bancos de dados de sistemas distintos; *segurança*, reconstituir a amostra biométrica através da referência biométrica deve ser computacionalmente difícil; e *desempenho*, visto que proteger o algoritmo não deve alterar as taxas de falsos positivos e falsos negativos do sistema.

3. **Métodos baseados na propriedade**: A credencial fornecida pelo cliente neste caso é baseada na propriedade de algum objeto de identificação. Pode-se citar como exemplo os cartões de identificação, *tags* de RFID ou NFC. O maior problema deste método é que estes objetos podem facilmente ser extraviados, comprometendo a segurança do sistema.

2.1.2 Ataques a sistemas de autenticação

Por ser o elemento regulador do acesso à recursos computacionais, os sistemas de autenticação são frequentemente alvos de ataques. O trabalho de [Todorov, 2007] enumera uma série de ataques a sistemas de autenticação a serem expostos a seguir.

1. **Força bruta**: Descobrir o ID de usuário normalmente é uma tarefa fácil. Frequentemente corresponde à primeira parte do *e-mail* do usuário. Ataques de força bruta tentam descobrir as senhas testando-as por exaustão de possibilidades. Este método pode ser aplicado diretamente no sistema em que o atacante deseja se autenticar ou também para quebrar senhas protegidas por algoritmos conhecidos obtidas de algum arquivo capturado. Este processo consome muito tempo e recursos computacionais.

2. **Ataques por dicionário**: Muitas vezes as pessoas se baseiam em palavras para definir senhas mais fáceis de se recordar. Sabendo disso, ataques por dicionário testam uma lista de palavras na expectativa de descobrir a senha. É um método mais eficiente que a força bruta, porém menos eficaz.

3. **Uso de senhas padrão**: Dentro da mesma idéia do ataque por dicionário, os atacantes poderão testar primeiramente *logins* e senhas padrão pré-configuradas pelos fabricantes das soluções. É comum que ao implementar uma solução os usuários se preocuparem demais com as funcionalidades implementadas mas se esquecerem de trocar estas senhas, gerando uma séria vulnerabilidade.

4. **Ataques por *rainbow tables***: Para melhorar a eficiência e economizar recursos de processamento de ataques por dicionário para quebrar senhas protegidas, por exemplo na forma de *hash*, o atacante poderá usar tabelas com valores pré-calculados. A eficiência deste tipo de ataque é bastante reduzida com a adição de fatores aleatórios às senhas (Ex. *salt*).

5. **Ataques *offline***: Muitas vezes o atacante consegue informações ou arquivos através de algum método de ataque e os utiliza para montar ataques *offline*, em ambientes seguros e com alta capacidade de processamento para, por exemplo, analisar dados capturados por *keyloggers*, ou mesmo para quebrar senhas protegidas por funções de *hash*.

6. ***Sniffers* na rede**: Sistemas e aplicações antigas ou mal configuradas podem transmitir na rede senhas sem a proteção adequada que podem ser capturadas por atacantes.

7. **Repetir a autenticação**: Neste mecanismo, o atacante com acesso ao tráfego da rede poderá capturar a *string* criptografada que o cliente usa para se autenticar no servidor e enviar a mesma *string* criptografada para se autenticar. Este ataque é prevenido através de mecanismos de perguntas desafio geradas aleatoriamente pelo servidor. Ao receber o desafio o cliente manipula o desafio recebido, normalmente utilizando a senha correta, e retorna para o servidor. Como o atacante não tem acesso à senha em claro ele não será capaz de responder corretamente ao desafio. A segurança deste mecanismo está intimamente ligada à capacidade de gerar sequências aleatórias para o desafio, pois caso a sequência se repita o atacante poderá já ter capturado e armazenado a resposta correta do usuário.

8. **Ataques de homem-no-meio (*man-in-the-middle*)**: Se o atacante consegue interceptar e enviar mensagens para o cliente e para o servidor ele poderá fazer o papel do cliente para o servidor, e o papel do servidor para o cliente, decidindo se transmite as mensagens corretamente para a outra parte ou alterá-las antes da transmissão. Para evitar ataques *man-in-the-middle* deve-se estabelecer um canal seguro e checar a integridade dos dados recebidos.

9. **Sequestro de sessão**: Quando o atacante tem o controle do meio de comunicação, poderá aguardar o usuário se autenticar no servidor e aplicar esta variante do *man-in-the-middle* assumindo a identidade de uma das partes, selecionando ou bloqueando as mensagens da outra parte.

10. **Ataque de degeneração (*downgrade*)**: Se um cliente e um servidor dispõem de mais de um mecanismo de autenticação eles irão negociar para definir qual o melhor. Um atacante na rede poderá interceptar esta negociação (através de *man-in-the-middle*) para selecionar o mecanismo de autenticação mais fraco disponível. Para prevenir este tipo de ataque um canal seguro deve ser estabelecido antes de realizar este tipo de negociação.

11. **Escalar privilégio**: Este ataque tem por objetivo adicionar direitos à um usuário que inicialmente não os possui. Isto é feito através de vulnerabilidades na segurança da aplicação ou do sistema operacional. Uma forma comum desse tipo de ataque é o *buffer overflow* que é executado injetando código através da entrada de parâmetros reservada ao usuário. Isto pode forçar a aplicação a executar este código no mesmo nível de privilégio. Este tipo de ataque pode ser prevenido com checagem adequada dos parâmetros de entrada.

12. **Obter acesso físico**: Uma vez que o atacante obtenha acesso físico à máquina ele poderá conseguir acesso aos recursos desejados muitas vezes sem necessidade de autenticação. Ele poderá, por exemplo, dar o *boot* na máquina com o sistema operacional de sua preferência e conseguir o arquivo de senhas para montar um ataque *offline* para quebrar usuários e senhas. Ele poderá, ainda, trocar o arquivo de senhas por outro contendo usuários e senhas de seu interesse usando o mesmo algoritmo de *hash* do arquivo de senhas original.

13. **Contornar o sistema de autenticação**: Se o atacante conseguir, por algum meio, privilégios de administrador no sistema ele normalmente terá acesso à todos arquivos e processos, podendo depurar a aplicação para burlar seu esquema de autenticação, por exemplo, trocando o comando que compara a senha do usuário por um comando que retorna sucesso na autenticação.

14. **Ataques por DMA**: A arquitetura x86 permite que diversos *hardwares* façam acesso direto à memória principal sem utilizar a CPU. O DMA foi concebido para melhorar o desempenho de dispositivos gráficos, de rede, do disco etc. No entanto, um atacante pode utilizar esta técnica para extrair segredos diretamente da memória ou modificar a execução de aplicações.

15. ***Keyloggers*, *trojans* e vírus**: O atacante poderá instalar uma aplicação *keylogger* na máquina que salva todas as teclas digitadas em um arquivo de *log*. Ele poderá ainda usar um *trojan* ou vírus para interceptar a comunicação entre cliente e servidor salvando os *logs* num banco de dados para análise posterior.

16. **Servidores falsos**: O atacante oferece um servidor falso para que o usuário forneça seus dados em claro na tentativa de autenticação. Este ataque é prevenido através de esquemas de autenticação mútua, onde não só o cliente se autentica no servidor, mas o servidor também se autentica no cliente.

17. **Engenharia social**: O atacante irá tentar enganar o usuário para voluntariamente fornecer a senha ou alterá-la de modo que o atacante saiba ou consiga capturar a nova senha. Este método é muito poderoso e só pode ser evitado através do treinamento e conscientização dos usuários.

18. ***Shoulder surfing***: Muitos ataques não envolvem, necessariamente conhecimentos em informática. Um dos ataques mais tolos é simplesmente olhar enquanto o usuário digita sua senha.

19. **Roubo de identidade e *Dumpster diving***: Muitas vezes o usuário pode se descuidar e deixar exposto ou mesmo descartar de modo impróprio documentos e outros itens como cartões bancários que contém informações que nas mãos de um atacante serão úteis para montar ataques direcionados e até mesmo roubar sua identidade

2.2 Autenticação Linux e o PAM

A autenticação UNIX é, historicamente, realizada pela comparação da senha do usuário com a senha contida no arquivo de senhas (`/etc/passwd` ou `/etc/shadow`). Inicialmente, cada aplicação tinha codificada o seu mecanismo de autenticação de usuários. Desta forma, se várias aplicações requerem autenticação, o administrador possivelmente teria que manter vários bancos de dados além do `/etc/passwd` e `/etc/shadow`. Ainda, se o administrador desejasse trocar o método de autenticação ou colocar restrições adicionais era necessário reescrever parte da aplicação. Para evitar este tipo de problema e reduzir a complexidade da autenticação, o PAM[1] foi concebido como um mecanismo genérico para separar a autenticação do usuário do restante da aplicação. Assim a aplicação desenvolvida com suporte ao PAM delega a ele a responsabilidade de autenticar seus usuários, restando ao administrador realizar a configuração do PAM para determinar as condições de autenticação. O projeto Linux-PAM se iniciou em 1997 e desde então muitos sistemas UNIX adotaram o conceito do PAM. Hoje a maioria das distribuições GNU/Linux usam o *framework* PAM.

[1] O PAM foi desenvolvido pela SunSoft (Divisão de *software* da Sun Microsystems) no início dos anos 90 e publicado na *Open Software Foundation RFC 86*.

2.2.1 Armazenamento de credenciais

Inicialmente o UNIX armazenava as informações do usuário em um único arquivo `/etc/passwd`. Normalmente este arquivo tinha permissão de leitura para todos usuários do sistema e permissão de escrita somente para o administrador. Cada linha continha os atributos relativos a um usuário como como *username* e senhas protegidas por funções de *hash*. Isto representava um problema de segurança, pois qualquer usuário poderia listar todos os usuários do sistema e, além disso, um adversário poderia usar as senhas protegidas para lançar um ataque de dicionário ou força-bruta. Então, posteriormente o arquivo `/etc/shadow` foi criado para separar as informações de usuário. O `/etc/shadow` tem permissão de leitura e escrita somente para o administrador (*root*) do sistema e assim constitui um ambiente mais seguro para armazenar as senhas criptografadas e outras informações dos usuários. Quando o `/etc/shadow` é utilizado no campo da senha do `/etc/passwd` consta o caractere 'x'.

Sistemas UNIX armazenam as senhas dos usuários protegidas por funções de *hash*. A função `crypt()` é utilizada para criptografar as senhas do usuário. Esta função recebe como parâmetro uma chave (senha do usuário) e um *salt* (caracteres gerados randomicamente para melhorar a eficiência do algoritmo de criptografia). Inicialmente o algoritmo utilizado nesta criptografia era o DES. Neste algoritmo somente os 8 primeiros caracteres da senha do usuário eram considerados. Os primeiros caracteres do *hash* da senha gerada correspondem ao *salt*, de modo que ele fique disponível posteriormente para gerar a novamente o *hash* a partir da senha digitada pelo usuário no ato da autenticação. Se a aleatoriedade do *salt* for boa o suficiente, seu uso garante que mesmo que duas senhas sejam idênticas o resultado da função `crypt()` será diferente. Este mecanismo visa dificultar a quebra das senhas em ataques por dicionário, por exemplo.

A maior parte das implementações Linux atuais suportam também outros algoritmos para gerar o resumo criptográfico das senhas. As escolhas mais populares são o MD5, SHA256 e o SHA512[1], que consistem na aplicação de transformadas matematicamente irreversíveis, fazendo com que a senha não seja facilmente recuperável. Ao contrário do DES, nestes algoritmos não há a limitação de somente 8 caracteres considerados.

2.2.2 *Pluggable Authentication Module* - PAM

O PAM é um conjunto de bibliotecas capaz de realizar a autenticação de modo flexível, configurável e escalável para o sistema operacional e aplicações diversas. Como uma plataforma genérica, o PAM faz uso de módulos carregados dinamicamente (arquivos SO). Um módulo PAM implementa mecanismos para autenticar um usuário a partir das informações armazenadas em arquivos ou servidores ou preparar o ambiente de trabalho para este usuário. Cada módulo pode implementar até quatro atividades relacionadas à autenticação e à autorização dos usuários. Essas atividades são referidas como os serviços ou grupos de gerenciamento PAM [Geisshirt, 2007], que são:

1. **Serviço de autenticação - *auth***: sua atividade é provar a identidade do usuário através da validação de credenciais. No modo tradicional o usuário fornece um nome relacionado à sua identidade e uma credencial (que pode ser uma senha) que prova a identidade indicada. Normalmente sistemas UNIX armazenam o nome de usuário, senha e participação em grupos nos arquivos `/etc/passwd`, `/etc/shadow`, e

[1] A função `crypt()` GNU atual usa um identificador fornecido junto ao *salt*, nos primeiros dígitos do *hash*, para determinar qual algoritmo de resumo criptográfico utilizar. O identificador `$1$` corresponde ao algorítmo MD5, `$5$` ao SHA256 e `$6$` ao SHA512.

`/etc/group`, no entanto, existem implementações PAM que utilizam LDAP e bancos de dados relacionais. Além da autenticação por senhas, módulos PAM podem ser utilizados para fazer este serviço de diversas formas, como através de *tokens* físicos ou biometria. Dentre os módulos comuns que implementam este serviço pode-se citar o `pam_unix.so`, o `pam_rootok.so`, o `pam_ldap.so` e o `pam_krb5.so`.

2. **Serviço de gerenciamento de conta - *account***: sabendo com qual usuário está se tratando, tem-se que decidir se ele poderá acessar o sistema. Para isso, o serviço *account* verifica os dados da conta do usuário, se está vencida, se existem restrições de horário, se o usuário já está assessando o sistema por outro meio, etc. `pam_permit.so` e `pam_denny.so` são exemplos de módulo que implementam este serviço.

3. **Serviço de gerenciamento de sessão - *session***: Cria o ambiente da sessão de uma dada aplicação, aloca os recursos que o usuário pode precisar durante sua sessão, por exemplo, montando o seu diretório *home*, estabelecendo os limites de uso dos recursos, e salvando os *logs* de acesso do usuário. Quando o serviço deixar de ser utilizado, o *session* irá finalizar o ambiente. O próprio `pam_unix.so` é um exemplo de módulo que implementa o serviço *session*.

4. **Serviço de gerenciamento de senhas - *passwd***: Permite a atualização de credenciais para que o usuário possa trocar sua senha ou para que o administrador possa recuperá-la. Novamente, pode-se citar o exemplo do `pam_unix.so` além de outros como `pam_cracklib.so`

Pode-se perceber através dos exemplos citados acima que os módulos PAM podem implementar um ou mais destes serviços. A plataforma PAM permite que o administrador empilhe uma sequência de módulos para tentar mais de uma técnica de validação durante uma única tentativa de *login*. É possível, ainda, estabelecer através de *flags* se determinado módulo é requisito, suficiente, requerido ou opcional. Dessa forma, o administrador pode personalizar o processo de autenticação e autorização com granularidade suficiente para atender suas necessidades.

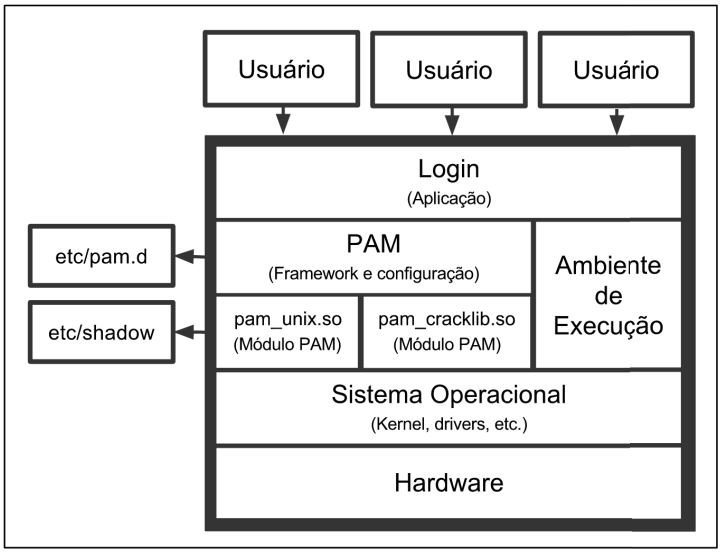


Figura 2.1: Contexto de utilização da plataforma PAM. Adaptado de [Geisshirt, 2007]

A Figura 2.1 ilustra o contexto de utilização da plataforma PAM. O usuário faz a requisição para acessar a aplicação (no caso *login*). A aplicação que faz interface com a biblioteca

PAM não tem nenhum conhecimento sobre o método de autenticação configurado. A biblioteca PAM consulta o conteúdo dos arquivos de configuração para carregar os módulos apropriados para atender a requisição do usuário de acesso à determinada aplicação. Esses módulos se enquadram em um dos quatro grupos de gerenciamento e são empilhados na ordem em que aparecem no seu arquivo de configuração. Esses módulos, no caso o `pam_unix.so` e `pam_cracklib.so`, quando chamados pelo PAM executarão as diversas tarefas necessárias para autenticação. Se os serviços requisitados forem *auth* e/ou *session*, apenas o módulo `pam_unix.so` será utilizado, mas se o serviço requisitado for *passwd* ambos os módulos serão utilizados (maiores detalhes sobre os mecanismos implementados pelos principais módulos serão tratados na Seção 2.2.3).

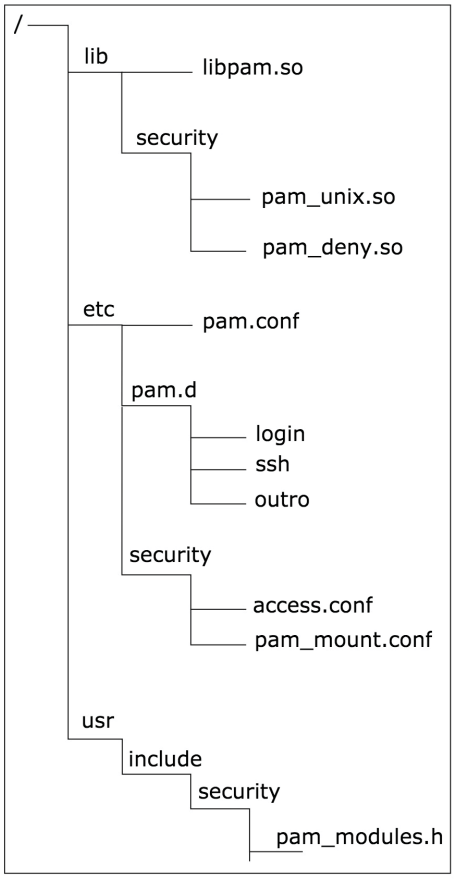


Figura 2.2: Estrutura de arquivos do PAM. Adaptado de [Geisshirt, 2007]

Os arquivos de configuração utilizados são armazenados em uma estrutura de arquivos normalmente organizados conforme mostra a Figura 2.2. Aplicações que utilizam o PAM são ligadas à biblioteca PAM que normalmente se encontra na pasta `/lib`. Os módulos PAM são objetos compartilhados (arquivos SO) que são carregados dinamicamente conforme especificado nos arquivos de configuração do PAM. Normalmente eles são armazenados na pasta `/lib/security`. Em algumas implementações a configuração do PAM é feita através de um arquivo único `/etc/pam.conf`, mas em suas versões mais recentes o PAM permite o uso de arquivos na pasta `/etc/pam.d/` dedicados para cada serviço ou aplicação que necessite de configurações específicas. Arquivos de configuração específicas para alguns módulos PAM são armazenados na pasta `/etc/security`. Por fim, quando desenvolvedores desejarem utilizar o PAM em suas aplicações deverão incluir o arquivo *header* `pam_modules.h` que se encontra na pasta `/usr/include/security`.

2.2.3 Módulos PAM

O PAM depende de seus módulos que são bibliotecas compartilhadas carregadas dinamicamente e que implementam um ou mais serviços realizados pelo PAM. Esta seção irá listar alguns dos módulos PAM de uso mais comum e um resumo de suas funcionalidades e aplicações.

- **`pam_permit.so`**: Este módulo sempre retorna sucesso. Simplesmente diz "sim" para todas requisições.

- **`pam_deny.so`**: Este módulo sempre retorna falha de autenticação. É uma alternativa para utilizar como padrão para a entradas não especificadas.

- **`pam_cracklib.so`**: Este é um módulo padrão da autenticação Unix que pode ser acrescentado à pilha do gerenciamento de senhas (*passwd*) para checar a força da senha escolhida pelo usuário contra ataques de dicionário e configurar um conjunto de regras para identificar escolhas de senhas fracas. Primeiramente ele pede que o usuário digite uma senha e então checa se ela é considerada forte, então pede para o usuário entrar a senha novamente para evitar erros de digitação. Se a senha for forte o suficiente e ambas as digitações forem idênticas, a execução é passada para os módulos subsequentes. A verificação da força da senha envolve constatar se a nova senha é um palíndromo, se houve apenas mudança de letras maiúsculas/minúsculas, se é muito similar à antiga, se é uma rotação da antiga, se tem caracteres repetidos consecutivos, se contém o nome de usuário, etc.

- **`pam_limits.so`**: este módulo permite estabelecer limitações do acesso aos recursos do sistema que podem ser obtidos em uma sessão de usuário. Por padrão as limitações são obtidas do arquivo `/etc/security/limits.conf`. Aqui podem ser estabelecidos limites para número máximo de processos, tamanho máximo de arquivos etc.

- **`pam_rootok.so`**: é um módulo que autentica o usuário se o seu UID é 0, ou seja, se é o *root*. Pode ser usado em conjunto com outros módulos para liberar acesso ao *root* de modo direto e exigir autenticação dos demais usuários.

- **`pam_time.so`**: este módulo não autentica o usuário, mas restringe o seu acesso ao sistema ou à aplicações específicas em determinados horários do dia, dias da semana. Por padrão as regras para acesso baseado no tempo estão no arquivo de configuração */etc/security/time.conf*.

- **`pam_unix.so`**: é o módulo padrão de autenticação do Unix. Ele usa chamadas padrão das bibliotecas do sistema para retornar e estabelecer informações de conta (*account*), de autenticação (*auth*) de senha (*passwd*) e de sessão (*session*). Normalmente isto é obtido do arquivo `/etc/passwd` ou do `/etc/shadow` quando o *shadow* está habilitado. O componente ***account*** tem a tarefa de estabelecer o status da conta do usuário e da sua senha baseado em informações como data de expiração, última modificação, modificação máxima e mínima, etc. Se for o caso, pode aconselhar ao usuário para que troque sua senha, ou atrasar a entrega do serviço até que o usuário crie uma nova senha. O componente ***auth*** é responsável pela verificação das credenciais (senha) do usuário. Por padrão este módulo não permite acesso a um serviço se sua senha oficial é em branco. Um binário auxiliar, `unix_chkpwd(8)`, é fornecido para verificar a senha do usuário quando a mesma é armazenada em um banco de dados protegido para

leitura. Este binário é chamado de forma transparente pelo componente *auth* do módulo, o que permite que aplicações como `xlock(1)` funcionem sem terem UID de root. O componente ***passwd*** deste módulo cumpre a tarefa de atualizar as senhas de usuário. O método de criptografia padrão utilizado é obtido da variável `ENCRYPT_METHOD` do arquivo `/etc/login.defs`. Por fim, o componente ***session*** registra quando o usuário faz *login* e *logout* do sistema.

- **`pam_ldap.so`**: existem duas formas de configurar o PAM para usar um servidor LDAP. Na primeira, através do módulo `pam_unix.so`, é seguido o modelo tradicional da autenticação Unix. Isto significa que a senha criptografada do usuário é transferida do diretório do servidor para a máquina local onde usuário digita sua senha que é criptografada e, finalmente, ambas as senhas criptografadas são comparadas para decidir se o usuário deve ser autenticado ou não. Neste caso a senha deve ser armazenada em um atributo `userPassword`. A segunda maneira é utilizar uma das diversas implementações de módulos `pam_ldap.so` que podem ser compilados para qualquer sistema Unix. Este módulo autentica os usuários diretamente no servidor de diretório. Desta forma os clientes não necessitam de acesso de leitura do atributo da senha, e esta tampouco necessita ser armazenada na máquina do cliente, o que reduz significamente sua exposição. O módulo `pam_ldap.so` ainda pode ser usado para troca de senhas quando o LDAP[1] estiver configurado para utilização do `pam_unix.so`.

- **`pam_krb5.so`** é um módulo desenvolvido para permitir a integração da verificação de senha do Kerberos 5 para aplicações que usam o PAM. Ele cria arquivos de *cache* de credenciais específicas da sessão. Quando um usuário efetua o *login*, a função de autenticação do módulo executa uma verificação de senha simples e, se possível, obtém as credenciais do Kerberos 5, colocando-as em *cache* para uso posterior. Estes arquivos são criados e excluídos no início e término da sessão. Este módulo implementa os quatro grupos de gerenciamento do PAM. Maiores informações podem ser acessadas em `https://linux.die.net/man/8/pam_krb5`.

- **`pam_winbind.so`** faz parte da distribuição do Samba e faz a mediação das requisições de autenticação. Com este módulo o PAM pode usar o *Winbind* como *back end*, sendo um método para integrar qualquer um sistema Unix com o *Microsoft Active Directory* (AD). Esta integração passa pela configuração do *Winbind*, pela junção do computador Unix ao diretório ou domínio e por fim pela configuração do módulo PAM. Este é um componente fundamental para integrar sistemas Unix à infraestrutura *Windows* existente mantendo um banco de dados de contas comum entre ambos. Maiores detalhes sobre o *Winbind* podem ser encontrados em [Geisshirt, 2007].

- **`pam_mysql.so`** MySQL é um mecanismo de banco de dados relacional largamente utilizado. O `pam_mysql.so` pode ser utilizado para autenticar usuários cujas credenciais estão armazenados em bancos de dados. Normalmente este módulo não vem nas distribuições do PAM, mas pode ser compilado para a maioria dos sistemas Unix.

[1] O LDAP é, provavelmente, o banco de dados distribuído de usuários mais utilizado atualmente. O Microsoft Active Directory (AD) e Novell eDirectories são ambos (praticamente) exemplos de implementações de LDAP. O AD armazena diversos dados do usuário, mas as senhas são verificadas através do Kerberos. Além destas existem diversas implementações puras do LDAP, incluindo o OpenLDAP [Geisshirt, 2007].

2.2.4 Desenvolvimento com PAM

A arquitetura da plataforma PAM está esquematizada na Figura 2.3. A plataforma PAM fornece uma interface (API) para as aplicações muito bem definida. As aplicações que irão delegar a autenticação dos seus usuários ao PAM devem aderir a esta interface de comunicação com a plataforma. A biblioteca PAM acessa os arquivos de configuração específicos dos serviços ou aplicações para determinar os módulos a serem utilizados bem como as suas políticas de uso. Por fim, os módulos que se ligam à biblioteca PAM e fazem interface com seus respectivos *back end* são carregados para realizar as suas respectivas tarefas relacionadas à autenticação.

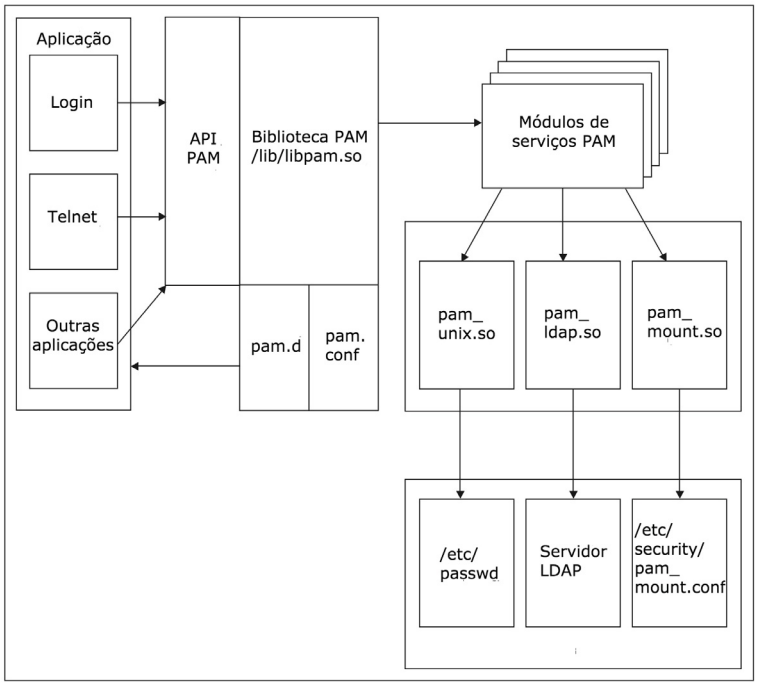


Figura 2.3: Framework do PAM. Adaptado de [Geisshirt, 2007]

A Figura 2.4 mostra o fluxo de execução de uma aplicação típica compilada com suporte ao PAM para autenticação. A maior parte do uso é bastante direto. A aplicação chama uma série de funções bem definidas que criam, operam e destroem as estruturas de dados relacionadas ao PAM. Ao se iniciar uma sessão PAM é necessário criar uma estrutura de dados que irá conter todos os dados relevantes à esta sessão PAM. Esta estrutura é chamada de `pam_handle_t`. A criação desta estrutura é equivalente a abrir uma sessão PAM, e ela é inicializada pela função `pam_start` da biblioteca `pam_appl.h`. Esta estrutura não pode ser manipulada diretamente, para isto a API fornece o método `pam_set_item`.

A função `pam_start` recebe como parâmetros o nome do serviço, o nome de usuário (que pode ser obtido da função do C padrão `getlogin`), um ponteiro para a função de conversação e um ponteiro para a estrutura de dados `pam_handle`. As funções de conversação são usadas pelos módulos para obter da aplicação informações relevantes à autenticação do usuário, como por exemplo a sua senha. Desta forma o programador deve implementar uma função convencionada capaz de manipular estas chamadas dos módulos. O PAM Linux implementa uma função de conversação genérica `conv` da biblioteca `pam_misc.h`. A autenticação em si é realizada por uma simples chamada à função `pam_authenticate` que recebe como parâmetro a estrutura `pam_handle` e uma *flag* opcional. Por fim, a função `pam_end` que destrói a estrutura de dados quando não for mais nececessária.

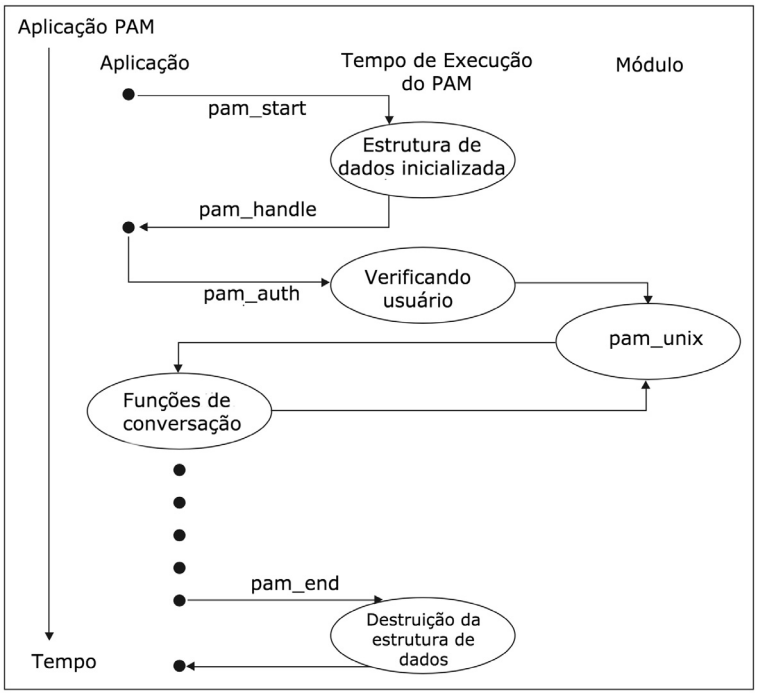


Figura 2.4: Fluxo de uma aplicação que usa autenticação PAM. Adaptado de [Geisshirt, 2007]

Tratando agora do desenvolvimento dos módulos PAM, um conjunto de funções são esperadas pelo ambiente de execução do PAM, em especial aquelas relacionadas aos grupos de gerenciamento. Um módulo pode suportar um ou mais grupos de gerenciamento que são implementados por uma ou mais funções no módulo. A declaração das funções do módulo são da seguinte forma:

```
PAM_EXTERN int FUNC(pam_handle_t *pamh, int flags, int argc,
const char **argv)
```

As funções operam a estrutura de dados `pam_handle_t pamh` criada pela função `pam_start` que contém todos os dados da sessão PAM corrente. *argc* e *argv* representam argumentos para função em particular. O termo FUNC representa uma das seis alternativas abaixo:

1. **`pam_sm_authenticate`** (serviço *auth*) que faz a autenticação do usuário e retorna `PAM_SUCCESS`, `PAM_USER_UNKNOWN` ou `PAM_AUTH_ERR`.

2. **`pam_sm_setcred`** (serviço *auth*) que atribui as credenciais e retorna `PAM_SUCCESS`, `PAM_USER_UNKNOWN` ou `PAM_AUTH_ERR`.

3. **`pam_sm_acct_mgmt`** (serviço *account*) que faz a validação do estado da conta e retorna `PAM_SUCCESS`, `PAM_USER_UNKNOWN`, `PAM_AUTH_ERR` ou `PAM_ACCT_EXPIRED`.

4. **`pam_sm_chauthtok`** (serviço *passwd*) que faz a manipulação das senhas e `PAM_SUCCESS` ou `PAM_USER_UNKNOWN`.

5. **`pam_sm_open_session`** (serviço *session*) que abre uma nova sessão e retorna `PAM_SUCCESS` ou `PAM_SESSION_ERR`.

6. **`pam_sm_close_session`** (serviço *session*) limpa os dados ao finalizar a sessão e retorna `PAM_SUCCESS` ou `PAM_SESSION_ERR`.

Para sua utilização, deve ser incluído o arquivo `security/pam_modules.h`. Além disso, uma macro deve ser definida para cada grupo de gerenciamento no seguinte formato: `PAM_SM_<grupo de gerenciamento>`. Por exemplo para um módulo de autenticação a seguinte definição deve constar no topo do código fonte:

```
#define PAM_SM_AUTH
```

Por fim, a biblioteca PAM traz uma série de funções de suporte à autenticação, como por exemplo a `pam_get_user`, da biblioteca *security/pam_modules.h*, usada para obter o nome de usuário, que é fornecido quando o usuário faz o *login*. Quando esta função é chamada pode ser que o PAM já tenha obtido esta informação de um módulo anterior. Se o nome do usuário não for conhecido pelo PAM ele irá automaticamente iniciar uma função de conversação com a aplicação para obtê-lo. A decisão de chamar ou não a função de conversação é inteiramente da plataforma PAM, e não do desenvolvedor do módulo. Outra função importante é a `pam_get_authtok`, da biblioteca `security/pam_ext.h`, que retorna o *token* de autenticação se o PAM já tiver esta informação ou solicita o seu fornecimento para o usuário. Além disso, logicamente, os desenvolvedores podem utilizar quaisquer funções dentro da linguagem C.

Este texto não pretende apresentar uma referência exaustiva das configurações, *flags*, métodos, parâmetros e do uso dos diversos módulos PAM, mas sim apresentar, em linhas gerais, o seu funcionamento e principais funcionalidades. Uma referência completa para administradores e desenvolvedores sobre configuração, módulos e desenvolvimento de módulos e aplicações pode ser acessado em `http://www.linux-pam.org/Linux-PAM-html/` e no livro [Geisshirt, 2007], que foram as principais referências para a escrita desta Seção.

2.3 Intel *Software Guard Extensions* (SGX)

A tecnologia *Software Guard Extensions*, ou Intel SGX, foi lançada pela Intel no final de 2015 junto à família de processadores *Skylake* com o objetivo de proteger código e dados sensíveis de acessos e modificações não autorizados. O SGX implementa um novo conjunto de instruções de CPU que permite que a aplicação aloque uma região privada de memória, denominada *enclave*, que é protegida inclusive de códigos com alto nível de privilégio, como da BIOS, *kernel* e sistema operacional. A arquitetura dos processadores com esta tecnologia inclui 18 novas instruções que permitem que o desenvolvedor utilize áreas de execução protegidas pelo hardware da CPU, os enclaves, e assim aumente a segurança da aplicação mesmo quando o ambiente de execução é malicioso. Para cumprir essa promessa, parte da memória física (RAM) é destinada aos enclaves, trata-se da *Enclave Page Cache*, ou EPC. A memória do enclave é encriptada para impedir ataques por hardware. A chave criptográfica é armazenada seguramente dentro da CPU e muda aleatoriamente a cada ciclo de energia.

A tecnologia SGX entrega, ainda, dois mecanismos para suportar sistemas que demandam alto nível de segurança. O primeiro mecanismo é a *selagem* de dados, permite criptografar dados que necessitem ser salvos fora da região segura do enclave. Dados selados só podem ser decifrados dentro dos enclaves. O SGX oferece também um segundo mecanismo chamado de *atestação* que permite que o um enclave prove a sua identidade e integridade à outro enclave. Dois tipos de atestação são suportados: *atestação local*, que é realizada entre enclaves rodando em uma mesma plataforma, e *atestação remota*, realizada por entidade externa. Estes mecanismos serão tratados nas subseções posteriores.

A arquitetura SGX exclui da parcela confiável os elementos com maior superfície de ataque, como o sistema operacional, hipervisor, *drivers* e BIOS, reduzindo ao mínimo a base de

computação confiável (TCB). A Intel recomenda que o enclave se limite à operação dos dados sensíveis, e com a menor dependência de funções externas possível, minimizando a superfície de ataque e elevando o nível de segurança da aplicação. A Figura 2.5 mostra redução da superfície de ataque em aplicações utilizando o SGX.

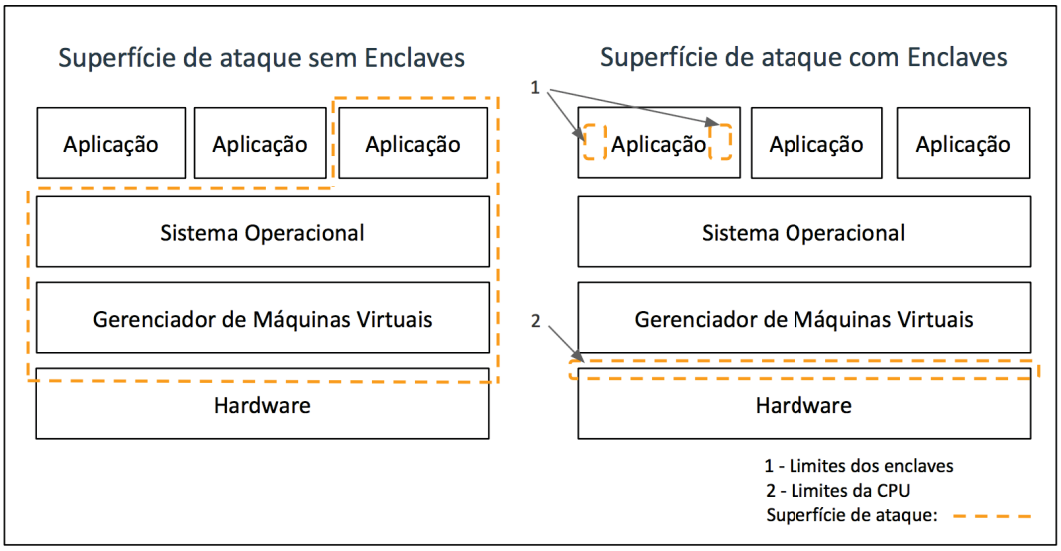


Figura 2.5: Superfície de ataque em aplicações com e sem enclaves SGX. Adaptado de [Intel®, 2017]

2.3.1 Organização da memória

Um processador que suporta a arquitetura SGX dará suporte para a BIOS reservar um intervalo de memória chamado *Processor Reserved Memory* (PRM) que é um subconjunto da memória principal que não pode ser acessado diretamente por outras aplicações. O controlador de memória da CPU também rejeita transferências via DMA para o PRM, de forma a protegê-lo do acesso a dispositivos periféricos. O PRM é uma área contígua de memória, tendo o seu tamanho e seu endereço de início como um valor inteiro e potência de dois, o que permite que seus endereços possam ser verificados por um *hardware* simples e barato.

O código e os dados dos enclaves são armazenados em um subconjunto do PRM, chamado de *Enclave Page Cache* (EPC). O EPC, por sua vez, é subdividido em páginas de 4KB, que podem ser atribuídas a diferentes enclaves, o que permite que se tenha múltiplos enclaves em execução no sistema ao mesmo tempo, o que é uma necessidade para ambientes multiprogramados. O armazenamento do EPC na memória principal é protegido encriptando 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. O MEE usa um Carter-Wegman MAC (*Message Authentication Code*) dedicado de alta velocidade combinado com um AES-CTR [Aumasson e Merino, 2016][1]. O EPC é gerenciado pelo mesmo *software* que gerencia o restante da memória física do computador, que pode ser um hipervisor ou o *kernel* do sistema operacional. O gerenciador de memória física utiliza as instruções do SGX para alocar páginas não utilizadas por enclaves e para liberar páginas previamente alocadas [Costan e Devadas, 2016].

[1]Uma descrição detalhada do funcionamento do MEE do SGX é encontrada em [Gueron, 2016].

Softwares que não utilizam enclaves não podem ter acesso ao PRM e consequentemente ao EPC. Isto garante um isolamento do enclave, mas também cria alguns obstáculos para um software carregar trechos de código e dados iniciais para um enclave recém criado. Estes obstáculos são contornados pelo uso de instruções do próprio SGX que permitem carregar conteúdo de outras páginas de memória para as páginas do EPC. Assim, quem é responsável por carregar as páginas de memória para o EPC é o *software* não confiável e, por este motivo, este processo é validado pelo SGX, que pode recusar qualquer operação que comprometa as garantias de segurança dos enclaves, como, por exemplo, a tentativa de atribuir a mesma página no EPC para dois enclaves diferentes.

O processador bloqueia qualquer acesso ao PRM vindo de agentes externos tratando esses acessos como uma referência não existente na memória. Já o acesso a páginas de memória dentro de um enclave utilizando instruções de memória, como MOV, são conferidas pelo hardware para verificar 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, 2014a, Costan e Devadas, 2016].

Para realizar suas verificações de segurança, o SGX grava algumas informações sobre as decisões de alocação para cada página EPC no EPCM *(Enclave Page Cache Map*), que é uma matriz com uma entrada para cada página EPC, o que permite que a computação do endereço da entrada EPCM de uma página seja efetuada com apenas uma operação de troca *bit* a *bit* e uma adição. O conteúdo do EPCM é utilizado apenas pelas verificações de segurança da SGX, sendo que, em condições normais de funcionamento, o EPCM não gera qualquer comportamento visível ao *software* e a sua utilização é transparente para os desenvolvedores.

O EPCM basicamente armazena três informações para identificar o proprietário de cada página EPC: um campo com a indicação se a página EPC foi alocada ou está livre, o que evita que dados sejam sobrescritos em páginas EPC já alocadas; um campo que identifica a qual enclave a página EPC pertence, de forma a impedir que um enclave acesse informações confidenciais de outro enclave, o que impossibilita que os enclaves se comuniquem via memória compartilhada utilizando páginas EPC; e um campo com o de tipo de página da entrada EPCM correspondente, que é registrado conforme instrução utilizada para alocar a página EPC que também determina o seu uso pretendido. As páginas que armazenam o código e dados de um enclave são consideradas de tipo regular, enquanto as páginas dedicadas ao armazenamento das estruturas de dados de suporte da arquitetura SGX são marcadas com tipos especiais [Costan e Devadas, 2016].

Um exemplo de página EPC com um tipo especial é o *SGX Enclave Control Structure* (SECS), que armazena metadados do enclave. As páginas deste tipo não são mapeadas no espaço de endereçamento dos enclaves, e são utilizadas exclusivamente pela CPU. O SECS pode ser considerado como um sinônimo da identidade de um enclave. O primeiro passo para a criação de um enclave é alocar uma página no EPC para armazenar seu SECS, e a última etapa para a sua destruição é desalocar esta página. A entrada no EPCM que identifica um enclave possui uma página EPC que aponta para o SECS do enclave, sendo que o endereço virtual do SECS é utilizado para identificar o enclave quando este invoca uma instrução SGX.

Como as instruções do SGX utilizam endereços SECS para identificar os enclaves, o *software* não confiável deve criar entradas na sua tabela de páginas apontando para os SECS dos enclaves que gerencia. Apesar disso, o *software* não confiável não pode acessar as páginas SECS, já que estas são armazenadas no PRM. Esta limitação de acesso às páginas SECS existe para que a arquitetura SGX possa armazenar informações confidenciais no SECS, como por exemplo a

sua medição (a ser abordada na Seção 2.3.3), de modo a garantir que *software* mal-intencionado não irá acessar essas informações. Se *software* não-confiável conseguisse realizar alterações na medição do enclave toda a segurança da arquitetura SGX ficaria comprometida.

2.3.2 Ciclo de vida do enclave

Como pode-se perceber através da Figura 2.6, aplicações SGX, em termos gerais, são divididas em duas partes: código não-confiável e o enclave confiável. Ao rodar, a parte não-confiável da aplicação cria uma instância do enclave que é alocado em memória confiável (1). 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* adiciona a medição criptográfica do conteúdo do enclave. A instrução *EINIT* completa o processo de criação do enclave e cria a sua identidade.

A parte não-confiável da aplicação chama as funções seguras dentro do enclave (2) através de uma interface pré-definida que é bastante restrita (3). Quando uma função confiável é chamada, a execução migra para dentro do enclave (4). 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.

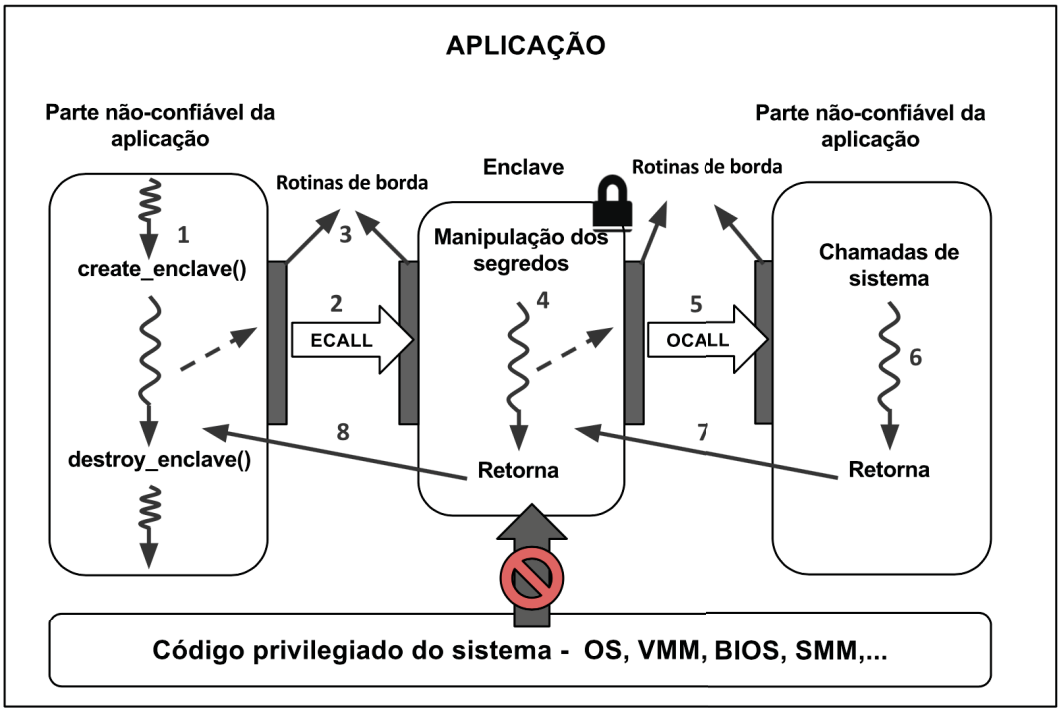


Figura 2.6: Aplicação em tempo de execução. Adaptado de [Intel®, 2017]

O código do enclave e os dados dentro do perímetro da CPU rodam em claro, mas acessos externos aos dados do enclave são bloqueados. Somente o enclave pode acessar e modificar a memória reservada para o enclave. Isso permite que a execução do código dentro do enclave ocorra de modo seguro mas com impactos mínimos no desempenho.

Em um ambiente sem SGX, operações envolvendo memória realizam uma consulta à tabela de páginas, obtêm uma tradução do endereçamento e, se tudo correr bem, o acesso é

realizado para leitura ou escrita. Num ambiente com SGX, após este processo, o hardware ainda realizará uma checagem adicional. Esta checagem envolve garantir que o acesso é realizado por código executando de dentro do enclave e ninguém mais; verificar se o endereço linear em que o enclave foi construído é o mesmo que está sendo referenciado, evitando assim que o sistema operacional tente remapear as páginas por baixo da aplicação; e verificar se a aplicação permite este acesso de leitura ou escrita. O *Enclave Page Cache Map*, EPCM, contém os metadados utilizados pelo hardware para realizar esta checagem.

Uma vez iniciada a execução segura, uma função OCALL (5) deve ser chamada se for necessário algum processamento fora do enclave, para realizar chamadas de sistema por exemplo (6). A OCALL retorna para a execução dentro do enclave (7), que por sua vez irá retornar para a execução não-confiável da aplicação (8). Ao sair do enclave deve-se limpar os valores em *cache* que se referem a endereços de memória protegidos pelo enclave, de forma a evitar acessos indevidos. Conforme apresentado no Anexo A, o SGX oferece as instruções *EENTER* e *EEXIT* para, respectivamente, entrar e sair do enclave. 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*.

Quando o enclave não for mais necessário ele será destruído através da instrução *EREMOVE*, que irá liberar todas as páginas EPC utilizadas pelo enclave, garantindo que nenhum processador lógico está executando instruções dentro das páginas EPC a serem removidas. O enclave é completamente destruído quando a página EPC que contém a sua estrutura *SECS* é liberada [McKeen et al., 2013, Intel, 2014a, Costan e Devadas, 2016].

2.3.3 Medição e assinatura do enclave

Assinar um enclave é um processo que envolve a criação de uma estrutura de assinatura (SIGSTRUCT) que contém as seguintes propriedades do enclave: o seu *Product ID*; o *Security Version Number* (SVN); a chave pública do autor e; um *hash* de 256 bits que identifica código, dados iniciais e suas posições relativas dentro do enclave (esta *hash* é conhecida como *Enclave Measurement* ou **medição do enclave**). Estas propriedades permitem à arquitetura SGX detectar quaisquer adulterações do enclave que, desta forma, pode provar que é legítimo e que foi carregado corretamente. O algorítmo RSA-3072 PKCS#1 v1.5 com SHA-256 é utilizado para computar a assinatura do enclave [Aumasson e Merino, 2016].

Para cada enclave instanciado a CPU fornece dois registradores. O MRENCLAVE armazena a medição do enclave e o MRSIGNER armazena um *hash* da chave pública do autor. Ao ser carregado em memória confiável, a CPU calcula a medição do enclave, o salva no registrador MRENCLAVE e então compara ao valor da estrutura de assinatura. Se forem iguais o enclave poderá ser iniciado. 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.

O *Software Development Kit* (SDK) fornecido pela Intel inclui uma ferramenta, chamada *Enclave Signing Tool*, que realiza o processo de assinatura. Esta ferramenta também avalia a imagem do enclave em busca de potenciais erros e problemas de segurança. Ao ser carregado a assinatura é conferida para confirmar a integridade do enclave.

São admitidos dois métodos de assinatura: passo único usando chave privada do usuário ou em dois passos usando uma ferramenta de assinatura externa. O método de dois passos protege a chave de assinatura em um ambiente separado e deve ser o método utilizado nas versões

de produção da aplicação. Quando um projeto de enclave é criado pela primeira vez o usuário deve escolher entre usar uma chave de assinatura existente ou gerar uma chave automaticamente. Quando for optado por utilizar uma chave existente, ela deve estar no formato PEM e não deve estar encriptada.

2.3.4 Interface do enclave: ECALL e OCALL

A transição entre o enclave e a aplicação não-confiável é realizada através de uma interface restrita e pré-definida. Quando a aplicação precisa que alguma operação seja realizada de modo seguro ela faz a chamada de funções que executam dentro do enclave. Estas funções são chamadas de ECALLs. Por outro lado, quando o enclave precisa que a execução de alguma operação ocorra em ambiente não-confiável, como acessar recursos do sistema operacional por exemplo, ele faz uso de funções especificamente desenvolvidas para este objetivo chamadas de OCALLs.

Como consequência dos processos de medição e assinatura do enclave, a arquitetura SGX requer que toda a funcionalidade dentro de um enclave seja ligada estaticamente, em tempo de compilação. Ao usar a funcionalidade de biblioteca estática, os desenvolvedores têm que optar entre executar determinadas funções fora do enclave através de OCALLs, o que pode adicionar uma sobrecarga de desempenho, ou incluir a implementação da biblioteca como parte do enclave, o que aumenta o tamanho da TCB. A Intel recomenda que as operações realizadas dentro do enclave sejam as mínimas possíveis para garantir a segurança da aplicação. Ainda,o desenvolvedor deve estar ciente que as ECALL expõem a interface do enclave que o aplicativo não confiável pode utilizar e, por isso, deve-se limitar o número de funções ECALLs para reduzir a superfície de ataque ao enclave [Intel, 2016a].

As ECALLs e OCALLs devem ser pré-definidas e declaradas em um arquivo EDL (*Enclave Definition Language*). Ponteiros e passagens de parâmetros por referência são tratados com um cuidado especial para evitar, por exemplo, que aplicativo não-confiável passe um ponteiro referenciando um endereço de memória dentro do limite do enclave. Isto poderia fazer com que o enclave sobrescreva seus dados ou código de forma indevida, podendo até mesmo ocasionar o vazamento de informações confidenciais do enclave. Desta forma, na declaração das funções são utilizados atributos especiais para a passagem de parâmetros e manipulação de ponteiros. Estes atributos permitem que rotinas de borda façam a manipulação segura dos destes parâmetros através de *buffers* utilizados na transição de dados e contexto entre o enclave e o ambiente não confiável. O arquivo EDL permite ainda importar arquivos *header*, o que pode ser útil na migração de aplicações existentes para a tecnologia SGX.

O SDK da Intel acompanha uma ferramenta responsável por gerar rotinas de borda, mostradas na Figura 2.7, a partir da leitura de um arquivo EDL. Tais rotinas têm como objetivo prover a interface entre o universo confiável (enclave) e não-confiável (aplicação). Esta ferramenta é chamada de *Edger8r Tool*. A partir de um arquivo EDL são gerados, por padrão, quatro arquivos contendo as declarações e definições das rotinas de borda confiáveis e não-confiáveis. Nestes arquivos são criadas as *wrapper functions* em linguagem C para as exportações (usadas pelas ECALL) e para as importações (usadas pelas OCALL).

2.3.5 Selagem de dados

Quando um enclave é instanciado, a confidencialidade e integridade dos dados são garantidas mantendo os mesmos dentro dos limites do enclave. De modo geral, quando o enclave é destruído os segredos são perdidos. Caso seja necessário preservar estes dados, eles devem ser

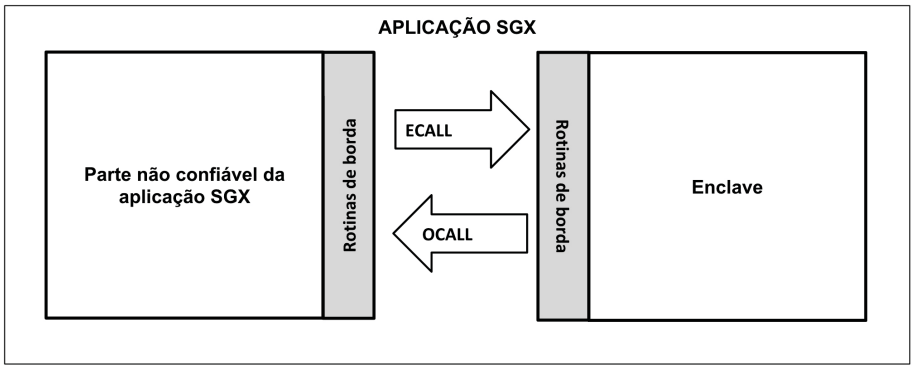


Figura 2.7: Transição da execução entre a aplicação não confiável e o enclave: as rotinas de borda geradas pelo *Edger8r Tool* são responsáveis por chamar as ECALL e OCALL. Adaptado de [Intel, 2016a].

armazenados fora do enclave. Aqui entra em cena um mecanismo que permite ao enclave, através de instruções de CPU, retornar uma chave única para aquele enclave rodando em determinada plataforma, a *Sealing Key*. Conforme esquematizado na Figura 2.8, esta chave é usada para cifrar e decifrar, ou selar e de-selar, os dados. Para aumentar a segurança é gerada uma chave única para cada *data blob*, e um identificador desta chave é armazenada em claro no *data blob* cifrado. Este identificador é usado para gerar a chave que decifra o *data blob* [Intel, 2016a].

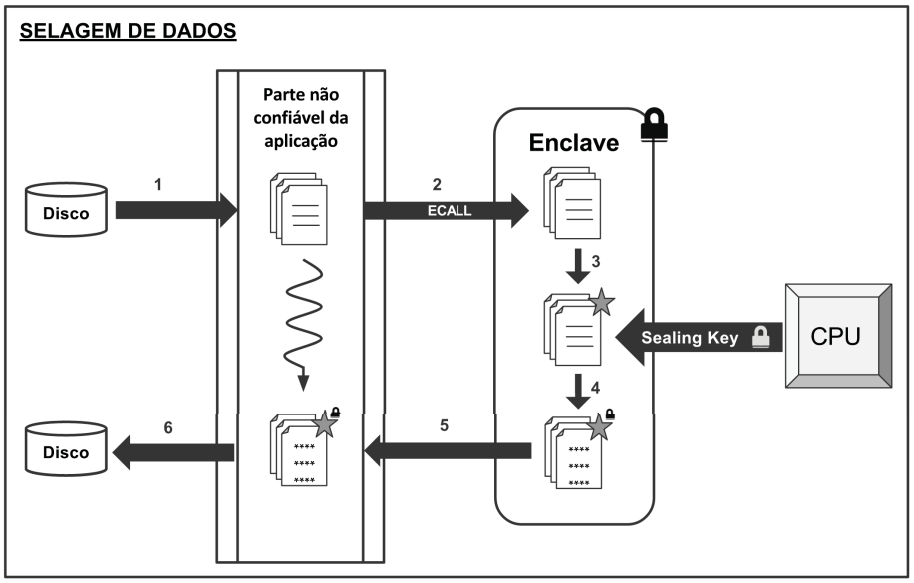


Figura 2.8: Selagem de dados: mecanismo de proteção dos dados sensíveis fora do enclave.

Ao selar os dados o enclave deve estabelecer uma dentre duas situações de de-selagem. Na primeira somente aquele enclave será capaz de de-selar os dados. Para isto a chave usada na operação de selagem é ligada ao valor contido no registrador MRENCLAVE, que contém identificação daquele enclave específico. Na segunda situação qualquer enclave do mesmo autor e mesmo identificador de produto poderão de-selar os dados. Neste caso, a chave usada na operação de selagem é fundida ao valor contido no registrador MRSIGNER, que contém a identificação do autor do enclave. Não existe a necessidade de armazenamento de chave para recuperar os dados selados. Esta chave é gerada somente pela CPU que selou os dados a partir de informações do enclave e desta CPU em particular. O algoritmo AES-GCM é utilizado no

processo de selagem de dados e o algoritmo AES-CMAC é utilizado na derivação das chaves [Aumasson e Merino, 2016].

Os desenvolvedores também podem definir o *Security Version Number* (SVN) quando codificam o enclave, que também é armazenado na CPU quando o enclave é instanciado. Um enclave deve fornecer o SVN em sua solicitação para obter a chave de criptografia da CPU, não sendo possível especificar um SVN superior ao especificado na assinatura do enclave (ISVSVN). No entanto, o enclave pode especificar um SVN anterior ao ISVSVN do enclave, dando-lhe a capacidade de acessar dados selados por uma versão anterior do enclave, o que facilitaria as atualizações de *software*, por exemplo.

2.3.6 Atestação

Enclaves são construídos inicialmente sem nenhum segredo. Não convém colocar segredos no enclave antes de ser instanciado, mesmo criptografados, uma vez que código e dados estão em claro e não existe lugar seguro para armazenar as chaves. A atestação permite ao servidor da aplicação passar os segredos aos enclaves de modo seguro e é iniciada durante a instalação da aplicação ou ao registrar usuários.

A atestação é um mecanismo para provar a terceiros que um determinado enclave é legítimo, não foi adulterado e foi carregado corretamente em uma plataforma SGX. 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. A arquitetura Intel SGX fornece os mecanismos para suportar duas formas de atestação que podem ser utilizados dependendo do tipo de necessidade: a *atestação local* e a *atestação remota*.

Atestação local

O desenvolvedor pode codificar uma aplicação onde vários enclaves devem trabalhar em conjunto. Neste caso, o enclave precisa provar sua identidade e autenticidade para outro enclave rodando na mesma plataforma através da atestação local. Este mecanismo de atestação utiliza um sistema de chaves simétricas, onde apenas o enclave que verifica a estrutura de relatório e o *hardware* de enclave que cria o relatório conhecem a chave. Em linhas gerais, a atestação local entre dois enclaves A e B ocorre conforme esquematizado na Figura 2.9:

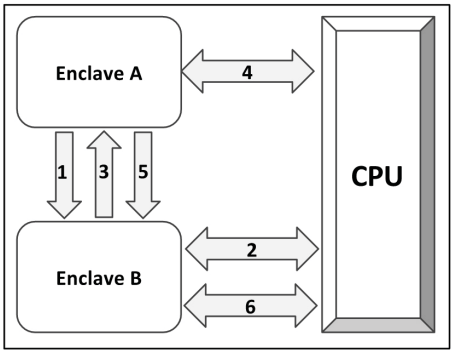


Figura 2.9: Atestação local. Adaptado de [Intel, 2016a].

1. O enclave A envia seu MRENCLAVE para o enclave B;
2. Por sua vez, o enclave B pede à CPU (através da instrução EREPORT) para produzir uma credencial chamada *report* usando o MRENCLAVE recebido de A;

3. Enclave B transmite o *report* ao Enclave A;

4. O enclave A pergunta à CPU se B está na mesma plataforma e é um enclave legítimo;

5. Enclave A cria seu próprio *report* para o Enclave B usando o MRENCLAVE do *report* que recebeu e transmite ao Enclave B; e

6. O Enclave B verifica o *report* para afirmar que o Enclave A esta na mesma plataforma que o Enclave B.

Atestação Remota

O mecanismo de atestação remota deve ser utilizado quando o enclave precisa de serviços providos por servidor fora de sua plataforma e precisa provar sua identidade e autenticidade para esse servidor. A atestação remota utiliza uma estrutura chamada *quote*, que é gerado na quarta etapa do fluxograma da Figura 2.10. O *quote* é uma credencial produzida pela CPU que reflete o estado da plataforma e do enclave e contém o *Enclave Measurement*; um *hash* da chave pública do autor do enclave; o *product ID*; o *Security Version Number*; atributos do enclave (como por exemplo se esta no modo debug) e; dados de suporte para estabelecer um canal seguro do enclave com o servidor. Estes itens são assinados com uma chave de grupo da Intel chamada de *Enhanced Privacy ID*, ou EPID *key*. Esta assinatura é realizada por um enclave especial provido pela Intel chamado de *Quoting Enclave* (QE).

O *Enhanced Privacy ID* é um esquema de assinatura em grupo, assimétrica, que permite à CPU assinar objetos criptograficamente sem expor a identidade do assinante. Em um grupo cada assinante tem sua própria chave EPID privada, mas os validadores usam uma mesma chave EPID pública para verificar as assinaturas individuais dos elementos do grupo, não sendo possível identificar especificamente qual elemento do grupo assinou o objeto. Se, durante o processo de instalação, a plataforma do SGX identificar a falta da EPID, uma requisição será enviada ao serviço de provisionamento da Intel. Se o protocolo de comunicação entre a plataforma SGX e o serviço de provisionamento obtiver sucesso, o que requer um processador SGX autêntico, um EPID será fornecido.

A Figura 2.10 mostra o fluxograma para atestação remota sugerida pela Intel:

1. A aplicação estabelece comunicação com o ISV (*Independent Software Vendor*) que propõe um desafio contendo um *nonce*[1].

2. A aplicação requer um *report* do EA e passa o *nonce* recebido.

3. O EA gera a estrutura do *report* e repassa à aplicação.

4. O *report* é passado ao QE que o autentica e o converte em um *quote*.

5. QE retorna o *quote* à aplicação.

6. A aplicação envia o *quote* ao ISV.

7. ISV usa a EPID pública para validar a assinatura do *quote* ou, opcionalmente, envia ao serviço de verificação de atestação da Intel.

[1]O *nonce* é um número arbitrário que só pode ser usado uma vez. O nome é a combinação de 'N' = number (número) e 'once' = (uma vez, em inglês).

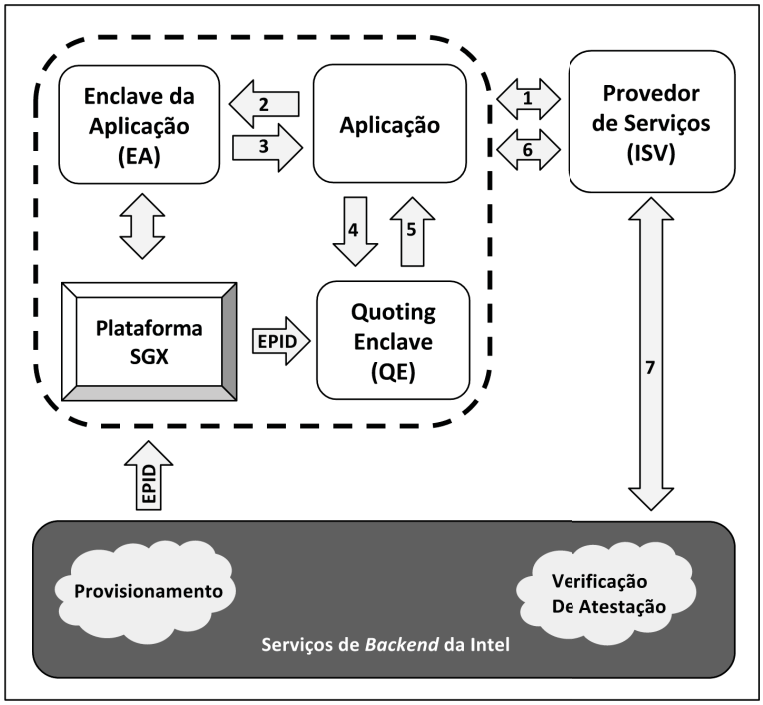


Figura 2.10: Atestação remota. Adaptado de [Intel, 2016a].

8. O ISV compara as informações recebidas à informação de configuração desejada para definir se o Enclave é seguro.

Se todas as verificações obtiverem sucesso o servidor saberá que pode confiar no enclave. Normalmente, juntamente ao *quote* é enviada uma chave criptográfica usada para estabelecer um canal de comunicação seguro entre o enclave e o servidor da aplicação depois da atestação.

2.4 Considerações finais

Neste capítulo foram apresentados conceitos sobre sistemas de autenticação envolvendo os principais métodos utilizados, os tipos de ataque a estes sistemas e a autenticação em sistemas operacionais Linux. A Seção 2.1 mostrou que a importância da proteção do arquivo de armazenamento de credenciais permeia os diversos métodos de autenticação (baseados em conhecimento, em fatores inerentes ao indivíduo ou na propriedade), uma vez que constitui ponto de vulnerabilidade explorada por diversos tipos de ataques.

A Seção 2.2 trouxe um breve relato sobre a autenticação em sistemas Linux e os fatores que levaram ao desenvolvimento da plataforma PAM, que proporcionou grande flexibilidade para autenticação Linux. O PAM permite que diferentes métodos de autenticação sejam implementados numa mesma plataforma sem a necessidade de modificações na plataforma ou nas aplicações que, por sua vez, simplesmente delegam ao PAM a responsabilidade de autenticar os usuários.

Finalmente, a Seção 2.3 apresenta uma visão geral do Intel SGX, que introduz um novo conjunto de instruções de CPU que possibilitam a criação de regiões encriptadas de memória, enclaves, cujo acesso é permitido somente através de uma interface pré-definida bastante restrita. Estas instruções permitem ainda a implementação dos pilares de um *Trusted Platform Module*, TPM, que são a verificação da integridade (*measurement*), o armazenamento seguro de dados (*sealing*) e a possibilidade de provar sua identidade a terceiros (*attestation*), sem a necessidade de *hardware* adicional além da CPU. Desta forma, o Intel SGX abre uma

possibilidade interessante para aumentar a segurança de sistemas de autenticação, em especial neste trabalho, na autenticação em sistemas operacionais.

Capítulo 3

Trabalhos relacionados

Este Capítulo contém uma breve revisão bibliográfica acerca dos trabalhos mais relevantes relacionados de alguma forma a este trabalho. A Seção 3.1 apresenta alguns trabalhos recentes sobre mecanismos de autenticação em geral. A Seção 3.2 trata de trabalhos voltados à proteção de credenciais, em sua maioria baseados em criptografia. A Seção 3.3 apresenta tecnologias e conceitos relativos à proteção de aplicações baseadas em *hardware*. A Seção 3.4 apresenta os trabalhos mais relevantes que buscam a proteção baseadas em virtualização. A Seção 3.5 apresenta alguns trabalhos relacionados ao SGX, tecnologia a ser empregada na proposta. Por fim, a Seção 3.6 traz um resumo das principais questões em aberto, limitações e vulnerabilidades encontradas no SGX até o presente momento.

3.1 Sistemas de autenticação

O trabalho [Jeong et al., 2015] propôs um sistema de autenticação baseado em multi-fatores, combinando o tradicional conjunto ID/senha com a autenticação através de biometria, visando melhorar a segurança no acesso a serviços de nuvem partindo de dispositivos móveis devido às suas peculiaridades (exposição a risco de roubo e perda, limitação de memória e desempenho). O reforço de segurança é provido adicionando os seguintes fatores de autenticação: ID/Senha; reconhecimento de voz; reconhecimento facial; IMEI (*International Mobile Equipment Identity*); e IMSI (*International Mobile Subscriber Identity*). Cada fator de autenticação é avaliado por uma máquina virtual do provedor do serviço de nuvem, de modo que mesmo adicionando fatores não há aumento significativo no tempo total do processo de autenticação. Os dados para autenticação são armazenados em *storages* separados e as entradas de dados para autenticação são armazenados em servidores de *log*. Como as entradas de voz e de face são sempre diferentes, os *logs* são utilizados para bloquear tentativas de autenticação de entradas repetidas, para evitar autenticação utilizada por gravações ou fotos.

O trabalho [Shin et al., 2015] propôs um sistema de autenticação para dispositivos móveis considerando a limitação de bateria, restrições de processamento e armazenamento e o custo de comunicação. Os objetivos desta solução são garantir a privacidade e o anonimato do usuário, proporcionar autenticação mútua, proteger contra ataques de repetição, ataques por adivinhação e roubo de senha e garantir que o comprometimento de uma chave de sessão não afete chaves de sessão anteriores. O esquema proposto se baseia no estabelecimento de uma sessão segura por chave simétrica e no uso de funções *hash*, XOR e concatenação para garantir a segurança do sistema. No entanto, não considera a hipótese de que o ambiente da autoridade autenticadora pode estar comprometida.

3.2 Proteção de credenciais

Os trabalhos relacionados nesta seção sintetizam algumas estratégias na utilizadas na proteção de credenciais. Percebe-se que as principais abordagens consistem na aplicação de funções de *hash* ou algoritmos criptográficos individualmente sobre cada credencial. A Seção 4.7 apresenta a relação da abordagem dos trabalhos abaixo com a do presente trabalho.

O trabalho de [Bellovin e Merritt, 1993] propõe uma melhoria do protocolo *Encrypted Key Exchange* (EKE) que aplica funções *hash* unidirecionais nas senhas para evitar o tráfego e armazenamento de senhas em claro. A proposta *Augmented Encrypted Key Exchange* (A-EKE) aprimora esse protocolo considerando que uma função *hash* pode ser comprometida. Neste esquema os *hashes* das senhas de usuários locais são protegidos pelo sistema de proteção de arquivos do sistema operacional, o protocolo proposto A-EKE cuida somente da autenticação de usuários remotos.

Um sistema de autenticação por impressão digital embarcado em um microcontrolador foi proposto no trabalho [Murillo-Escobar et al., 2015]. As referências biométricas são protegidas através de algoritmos criptográficos baseados no caos. O algoritmo utilizado é baseado num esquema dos próprios autores que consiste em um algoritmo simétrico com chave de 128 bits que usa *permutation-diffusion rounds*, e sequencias caóticas otimizadas de mapa logístico unidimensional. O cadastro do usuário autorizado é realizado retirando três amostras do usuário através do módulo de impressão digital para gerar a referência biométrica que é transmitida ao microcontrolador. Ao receber a referência biométrica o controlador aplica a criptografia proposta e armazena o resultado na memória *flash* do controlador. O processo de autenticação se dá através de requisição da referência biométrica criptografada, que é decifrada usando a mesma chave secreta e enviada para o módulo de autenticação para realizar a comparação e autenticar o usuário.

Um mecanismo de *key-stretching* aleatório foi utilizado para minimizar a parcela de senhas que podem ser quebradas por um atacante *offline* sem aumentar os custos para o servidor de autenticação [Blocki e Datta, 2016]. Segundo os autores, nos esquemas de autenticação tradicionais o custo de aceitação e de rejeição de uma senha são iguais. O esquema CASH (*Cost Assymmetric Secure Hash*) se aproveita do fato que num processo de autenticação a maioria das senhas digitadas são corretas, enquanto que num ataque a maioria das senhas digitadas são incorretas para desenvolver um sistema de autenticação com custos de aceitação e rejeição assimétricos, reduzindo o custo para aceitar uma senha correta e aumentando o custo para rejeitar senhas incorretas.

Um trabalho relevante que utiliza estratégia baseada na criação de funções *hash* é o Argon2 [Biryukov et al., 2016]. Segundo o autor, atacantes que visam quebrar senhas têm aplicado novas arquiteturas, como FPGA, GPUs com múltiplos núcleos, e módulos ASIC que amortizadom o custo para computar funções *hash* com múltiplas iterações. Estas arquiteturas, no entanto, são eficientes quando a computação demanda pouca memória. Para se defender deste tipo de atacante foram criadas *memory-hard functions*, que requerem grande quantidade de memória para serem computadas e impõe penalidade computacional se menos memória for usada. Argon2 aplica *memory-hard functions* para criação de funções *hash* que oferecem melhor proteção contra atacantes. Esta solução foi a vencedora da competição *Password Hashing Competition* (PHC).

3.3 Mecanismos de proteção por *hardware*

Mecanismos de proteção baseados em *hardware* têm ganhado grande relevância no provimento de infraestrutura para o desenvolvimento de aplicações seguras. Esta seção aborda os principais mecanismos de segurança em *hardware*, começando pelos cripto-processadores, que são implementações utilizadas há muito tempo e encontram-se presentes em muitos mecanismos de segurança em *hardware* modernos. Em seguida o *Trusted Platform Module* (TPM) será abordado. Além de ser utilizado em outras implementações como o Intel *Trusted Execution Technology*, alguns de seus conceitos e especificações inspiraram outras implementações como o ARM *TrustZone* e o AMD Secure Processor.

3.3.1 Cripto-processadores

Um cripto-processador típico é um processador embarcado dedicado que executa um conjunto pré-definido de operações criptográficas usando chaves internas, protegidas do ambiente externo, que se comunicam com o PC. Os cripto-processadores devem ser resistentes à ataques físicos e por software. Operações criptográficas são naturalmente custosas, o uso de hardware dedicado para este fim contribui para reduzir os impactos negativos no desempenho. Processadores deste tipo são utilizados há muito tempo em aplicações específicas como em sistemas bancários e aplicações militares. No entanto, com a crescente necessidade da proteção de dados sensíveis nas mais diversas aplicações, cripto-processadores tem ganhado maior relevância e diferentes implementações têm sido empregadas [Anderson et al., 2006] .

Dentre estas implementações destacam-se os seguintes mecanismos [Bossuet et al., 2013]:

1. **GPP Customizado**: Utiliza processadores de propósito geral customizados para implementação de algoritmos criptográficos eficientes. Eles absorvem operações criptográficas específicas como o *Data Encryption Standard* (DES) ou o *Advanced Encryption Standard* (AES). Estas soluções melhoram o desempenho, mas não a segurança, uma vez que as chaves criptográficas são armazenados em memória e tratadas como dados ordinários da aplicação, podendo sofrer ataques por software. Outra dificuldade é a seleção do conjunto de instruções, visto que algoritmos de chave simétrica e assimétrica têm necessidades distintas.

2. **Cripto-coprocessador**: Implementação em *hardware* customizado altamente eficiente para funções criptográficas específicas. Eles podem conter um ou mais núcleos de processamento, mas não são programáveis e sim controlados, configurados ou parametrizados usando o processador a que estão acoplados. São usados para acelerar cálculos criptográficos. Seu uso é indicado para aplicações com necessidade de alta vazão.

3. **Cripto-processador**: É um módulo de *hardware* que se difere do coprocessador por ser programável, com um conjunto de instruções dedicadas para obter eficiência em funções criptográficas. Cripto-processadores contam com uma ou mais unidades lógico-aritméticas (ULA) desenhadas especificamente para cálculos criptográficos e portanto, para realizar operações convencionais é necessário o uso de um processador de propósito geral. Esta arquitetura é mais enxuta que as duas anteriores, podendo ser considerada relativamente flexível, uma vez que as ULAs podem ser reconfiguradas. Apesar de executar somente cálculos criptográficos, cripto-processadores são autônomos e

representam um ótimo custo-benefício entre desempenho e capacidade de processamento. São boas opções para serem incluídas em *Multi-Processor-System-on-a-Chip (MP-SoC)*.

4. **Cripto array**: É um acelerador criptográfico que, assim como os cripto-processadores, precisam ser acoplados a processadores de propósito geral. Esta arquitetura usa vetores de núcleos de processamento para realizar as operações criptográficas. Normalmente os núcleos são reconfiguráveis para obtenção de uma maior flexibilidade.

3.3.2 *Trusted Platform Module* (TPM)

O *Trusted Computing Group* [TCG, 2008] é uma organização internacional de normas que conta com cerca de 140 empresas, que participou da especificação de uma série de padrões de segurança, tais como IPSEC (*Internet Protocol Security Protocol*), IKE (*Internet Key Exchange*), VPN (*Virtual Private Network*), PKI (*Public Key Infrastructure*), S/MIME *(Secure Multi-purpose Internet Mail Extensions*), SSL (*Secure Socket Layer*), SET (*Secure Electronic Transaction*), dentre outros [Amin et al., 2008]. Uma das frentes de trabalho atuais do TCG é a especificação dos padrões para o *Trusted Computing Module*.

O TPM (*Trusted Platform Module*) é um *chipset* capaz de armazenar artefatos usados para autenticar a plataforma, o que inclui senhas, certificados e chaves criptográficas. Um TPM também pode ser utilizado para armazenar as medidas da plataforma, para garantir que ela continua confiável. As especificações destacam ainda que os dois elementos fundamentais para o estabelecimento de um ambiente seguro são: garantir que a plataforma pode provar que é quem ela diz ser (autenticação); e provar a entidades externas que a plataforma é confiável e não foi adulterada (atestação) [TCG, 2008]. O conceito de TPM é aplicável a outros equipamentos além de PCs, tais como *smartphohes* ou equipamentos de rede.

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 interface definida nas suas especificações. 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 LPC. A Figura 3.1 mostra a composição básica de um TPM. 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 [TCG, 2016].

Outra implementação possível é executar o código seguro no processador do hospedeiro enquanto o mesmo se encontra em um modo de execução especial [TCG, 2016]. Nesse caso a memória do sistema é particionada pelo *hardware* de modo que a parte dedicada ao TPM fique inacessível, exceto quando o processador se encontra no modo especial. Quando o processador alterna entre os modos, ele sempre inicia a execução em pontos de entrada específicos. Existem diversos esquemas de implementação possíveis para isto, tais como *System Management Mode*, *TrustZone* e virtualização.

Dentro da construção de uma plataforma confiável existem alguns elementos cujas falhas não podem ser detectadas, e portanto precisam ser confiados. Assim, estes elementos devem ser projetados de modo a garantir que seu comportamento será sempre aquele desejado. Estes elementos são definidos como *Root of Trust*. Este conjunto de elementos deve ser o mínimo para prover as funcionalidades necessárias a uma plataforma confiável. Foram estabelecidos três *Roots of Trust* necessários ao TPM:

- ***Root of Trust for Measurement* (RTM):** Responsável por gerar informações relevantes à integridade da plataforma (suas medidas) e armazenar no RTS (próximo elemento).

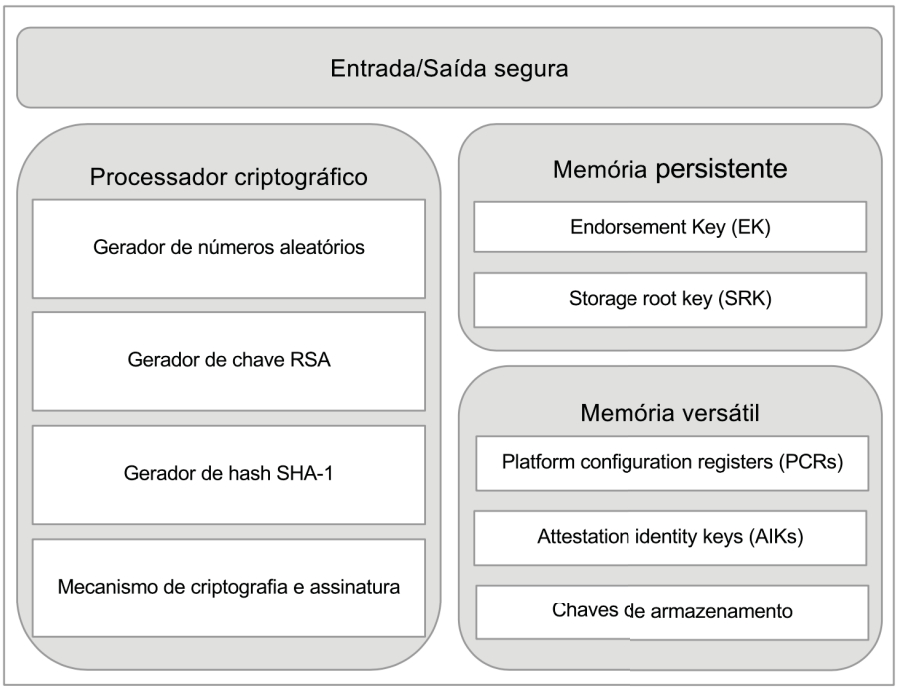


Figura 3.1: Arquitetura de um TPM. Adaptado de [Vacca, 2016].

Normalmente este elemento é a CPU controlada pelo *Core Root of Trust for Measurement* (CRTM), que é o primeiro conjunto de instruções executadas quando uma nova cadeia de confiança é estabelecida, por exemplo, quando o sistema é reiniciado. O CRTM envia valores que indicam sua identidade ao RTS.

- ***Root of Trust for Storage* (RTS):** Usada para armazenar elementos como as medidas da plataforma, a *Endorsement Key* (EK)[1], a *Storage Root Key* (SRK)[2]. A memória do TPM tem seu acesso bloqueado a qualquer entidade, exceto o próprio TPM. Se, no entanto, o elemento de memória contém informações não sensíveis, como no caso de algum *Platform Configuration Register* (PCR), o TPM permite a leitura.

- ***Root of Trust for Reporting* (RTR):** 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.

Estes três *Roots of Trust* usam certificação e atestação para fornecer evidências da confiabilidade da informação. Por fim, a plataforma mantem o *log* das suas mudanças de estado para posterior validação.

3.3.3 Intel *Trusted Execution Technology*

Intel *Trusted Execution Technology* (Intel TXT) é uma tecnologia da Intel que utiliza componentes de *hardware* para criar ambientes de execução seguros contra ameaças físicas e virtuais, de modo a complementar proteções em tempo de execução, como *softwares* de anti-virus.

[1] A EK é criada durante a fabricação e não pode ser alterada. Ela é usada no processo de autenticação.
[2] A SRK é usada para armazenamento criptografado. É gerada em tempo de execução.

O Intel TXT necessita de alguns componentes fundamentais: extensões integradas ao Intel Xeon e *chipset* Intel; módulos de código autenticados (*Authenticated Code Modules* - ACMs); ferramentas de LCP (*Launch Control Policy*); um módulo TPM; BIOS e hipervisor ou sistema operacional habilitado para o Intel TXT. Se qualquer destes componentes faltar ou falhar a plataforma irá ser carregada no seu estado não-confiável[Greene, 2012].

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

Existem dois métodos de estabelecimento de confiança através de medição (RTM): estático (S-RTM) e dinâmico (D-RTM). O S-RTM começa a medição em um evento de *reset* da plataforma com a medição da BIOS, passando pela medição do hipervisor (ou sistema operacional) e seus componentes. As medições realizadas são confrontadas com aquelas armazenadas no TPM. Se as medições forem equivalentes àquelas consideradas seguras, o sistema é carregado com o indicativo de confiável. Caso contrário, o sistema Intel fornecerá um indicativo de inicialização não-confiável. Apesar de ser mais simples, a S-RTM resulta em um TCB grande e difícil de gerenciar. Uma vez estabelecida a confiança, qualquer mudança ou atualização da plataforma ocasionará na necessidade de migração ou atualização dos segredos. Já no D-RTM, as propriedades de confiança dos componentes podem ser ignorados até a chamada de um evento seguro. Os componentes anteriores ao evento seguro serão excluídos do TCB e não poderão ser executados depois que a confiança do sistema for estabelecida. Isto resultará em uma TCB menor, o que é desejável.

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 DMAs não autorizadas e garantir o isolamento de dados no sistema.

3.3.4 ARM TrustZone

TrustZone é uma tecnologia de proteção baseado em hardware disponível nos controladores Cortex-A, Cortex-M23 e Cortex-M33. Este mecanismo provê isolamento em nível de hardware entre dois domínios de execução: o domínio seguro pode acessar recursos como memória, registradores da CPU e periféricos do domínio normal, mas o contrário não é permitido. Qualquer aplicação ou parte da aplicação pode ser desenhada para ser executada no domínio seguro.

O TrustZone provê a infraestrutura necessária para a criação de uma plataforma segura. Para isto cada processador físico tem um núcleo virtual seguro e um núcleo virtual inseguro.

Os domínios compartilham a CPU por divisão no tempo. O endereçamento de memória é particionado em regiões marcadas como segura e não-segura pelo *TrustZone Address Space Controller* [Sun et al., 2015]. Os sistemas que suportam TrustZone vem, ainda, com hardware complementar para suporte como chaves seguras não-voláteis, *Real Time Clock* seguro e aceleradores criptográficos. Os autores [Lesjak et al., 2015] ressaltam, ainda, que é possível configurar que determinadas interrupções devem ser sempre direcionadas ao domínio seguro e também dedicar determinados periféricos a este domínio.

Segundo a ARM, para consolidação de um ambiente seguro um sistema operacional e 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*). As propriedades de segurança do TEE suportam a confidencialidade e a integridade de múltiplas aplicações confiáveis [ARM, 2017].

Diversos trabalhos envolvendo esta tecnologia foram desenvolvidos. O trabalho de [Sun et al., 2015] usa o TrustZone para criar um mecanismo de geração de *One-Time Password* (OTP) em *smart phones* com a flexibilidade dos *tokens* OTP baseados em *software*, mas aproximando ao nível de segurança dos *tokens* baseados em *hardware*. O trabalho de [Zheng et al., 2016] propõe o *framework TrustPAY* para realização de pagamentos com privacidade e segurança. Os autores de [Jing et al., 2013] propuseram um sistema operacional seguro baseado no TrustZone.

3.3.5 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 aos núcleos do processador principal. Este subsistema proporciona um ambiente para execução de dados sensíveis isolado do sistema principal.

Para ser uma base confiável em *hardware*, podendo ser usada para 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 se trata de um núcleo virtual, mas de um núcelo fisicamente separado, integrado ao SoC, que dispõe de uma SRAM dedicada e acesso direto ao coprocessador criptográfico [Arthur e Challener, 2015]. Além disso, o PSP dispõe de acesso aos recursos de memória do sistema e uma DRAM encriptada, 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 esta chave para validar o *firmware* do PSP que, por sua vez, é carregado e inicia a execução da aplicação do sistema.

3.3.6 Comparativo entre os mecanismos apresentados

Esta seção apresentou de forma sucinta uma coletânea dos principais mecanismos de segurança baseadas em *hardware*. A Tabela 3.1 apresenta um breve comparativo entre as principais características dos mecanismos abordados neste texto. Nota-se que o processador Intel SGX abrange as principais características necessárias para o desenvolvimento de aplicações seguras. Por outro lado, o AMD *Secure Processor* consegue as mesmas características através da utilização do *TrustZone* e do TPM em um SoC.

Tabela 3.1: Comparativo entre os mecanismos de proteção baseados em *hardware*.

Tecnologia	Armazenamento seguro	Atestação	Isolamento de memória	Acelerador criptográfico
Cripto-processadores				✓
TPM	✓	✓		✓
Intel TXT	✓	✓	✓	
ARM TrustZone			✓	✓
AMD Secure Processor	✓	✓	✓	✓
Intel SGX	✓	✓	✓	✓

3.4 Mecanismos de proteção por virtualização

Além dos mecanismos de segurança baseados em *hardware*, o isolamento da camada de virtualização pode ser utilizado para melhorar a proteção de dados e execuções sensíveis. O *TrustVisor*, o *Overshadow* e o *Flicker* são exemplos de diferentes abordagens do uso da virtualização para melhorar a segurança de codigo e dados.

3.4.1 TrustVisor

Trustvisor é um hipervisor que, assim como o SGX, possui mecanismos de isolamento da memória para proteger partes sensíveis da aplicação contra ataques de sistemas operacionais maliciosos [McCune et al., 2010].

O ambiente é inicializado por um processo chamado *TrustVisor Root of Trust for Measurement* (TRTM) que interage com uma parte do hipervisor chamado *micro-TPM* (µTPM) que, por sua vez, provê somente as funcionalidades básicas de geração de números aleatórios, *measurement*, atestação e selagem de dados.

Aplicações legadas necessitam de modificações para se beneficiar do TrustVisor, mas os autores afirmam que tecnologia atinge um elevado nível de segurança pois protege código sensível em uma granularidade bastante fina e possui uma base de código bastante reduzida (6000 linhas de código, das quais metade são referentes a operações criptográficas), tornando possível a verificação e mantendo a TCB pequena e com um *overhead* menor que 7% no uso comum.

3.4.2 Overshadow

Os autores propuseram um sistema de proteção que estende a capacidade de isolamento da camada de virtualização para proteger entidades dentro da máquina virtual contra sistemas operacionais maliciosos sem a necessidade de hardware adicional ou modificações nas aplicações legadas. É proposto um mecanismo chamado *multi-shadowing*, que apresenta “pontos de vista”

da memória de execução diferentes dependendo do contexto que tenta realizar o acesso. Um monitor de máquinas virtuais (VMM) típico mantém um mapeamento de "um para um" do endereço físico do *guest* para os endereços da máquina virtural. O *multi-shadowing* substitui este modelo por um mapeamento de "um para muitos" proporcionando múltiplas visões da memória [Chen et al., 2008].

Este mecanismo é usado em uma técnica que os autores chamaram de *cloaking*: apresentar uma visão em claro de suas páginas para a aplicação enquanto apresenta uma visão encriptada destas páginas para o sistema operacional que pode gerenciar recursos sem comprometer a privacidade ou integridade da aplicação. Para manter a compatibilidade entre as aplicações legadas e sistema operacional ao carregar a aplicação, é introduzido um módulo para mediar a comunicação com o sistema operacional. Com o auxílio do VMM, este módulo intercepta eventos como chamadas de sistema, modificando sua semântica para habilitar o compartilhamento seguro de recursos entre a aplicação e o sistema operacional não-confiável.

3.4.3 Flicker

A tecnologia da AMD *Secure Virtual Machine - SVM* permite a realização de atestação do estado corrente da plataforma (*Trusted Platform Module* - TPM) a uma entidade externa e também a selagem de dados usando uma chave RSA de 2048 bits que nunca sai da TPM em claro. O SVM permite também o *late launch*, ou lançamento tardio do VMM. Para proteger o endereço de memória onde a VMM será carregada, chamado de (*Secure Loader Block - SLB*), o processador implementa uma série de medidas de segurança: DMA às páginas de memória contidas na SLB são desabilitados, interrupções e acesso a debug também não são permitidos. A Intel *Trusted eXecution Technology - TXT* apresenta funcionalidades análogas ao AMD SVM.

Os autores [McCune et al., 2008] propuseram o Flicker que se beneficia destas funcionalidades, mais especificamente o *late launch*, para promover execução isolada de código sensível. Ao invés de carregar o VMM, o Flicker pausa o ambiente de execução corrente para executar o trecho de código protegido antes de retornar ao ambiente de execução anterior. O desenvolvedor da aplicação deve fornecer a parte que deve ser protegida, chamada pelos autores de *Piece of Application Logic (PAL)*. Para criar uma SLB, o PAL deve ser ligado ao módulo *SLB Core* responsável por inicializar e finalizar a sessão Flicker. Para executar o SLB resultante, a aplicação passa o resultado para um módulo do Kernel do Linux desenvolvido pelos autores (*flicker-module*). Antes de retornar à execução regular, a memória utilizada é apagada para evitar vazamento de segredos. Desta forma nenhum software executando antes do Flicker ser iniciado poderá monitorar ou interferir na sua execução. O armazenamento de dados selados podem ser usados quando for necessário o compartilhamento de segredos entre dois ou mais PAL. Ainda, como o Flicker pausa a execução do restante do sistema, pode ser desejável dividir um segmento de trabalho em múltiplas sessões Flicker, dando tempo para que o restante do sistema possa operar.

Os autores exemplificam que esta tecnologia pode ser utilizada para que uma Autoridade Certificadora assine certificados tendo sua chave privada protegida mesmo quando o adversário possui o controle da BIOS, sistema operacional e dispositivos com DMA.

3.5 Casos de uso da arquitetura SGX

Apesar de ser uma tecnologia relativamente nova (lançada no final de 2015) diversos trabalhos sobre o SGX estão disponíveis. O OpenSGX [Jain et al., 2016] é uma plataforma aberta que emula o Intel SGX em nível de instruções através da extensão do emulador *software*

livre QEMU. Para mostrar o caso de uso do OpenSGX, os autores o aplicaram na proteção de informações sensíveis de nós Tor e avaliaram os potenciais impactos em desempenho. A plataforma OpenSGX ajuda o avanço da exploração das possibilidades trazidas pelas extensões SGX, uma vez que processadores com esta tecnologia ainda não estão completamente difundidos no mercado.

O trabalho Haven [Baumann et al., 2015] implementa um protótipo de sistema que permite a execução segura de aplicações *Windows* não-modificadas, como por exemplo servidores SQL e Apache fazendo uso do SGX. A idéia é colocar dentro da TCB, ou neste caso do Enclave, o binário não modificado juntamente a um subconjunto do sistema operacional e um módulo com funcionalidades do *kernel* e interface com o *host* não confiável.

No trabalho de [Hoekstra et al., 2013] são propostos modelos de soluções utilizando as instruções SGX. A primeira delas é um modelo de gerador de *One-Time Password* seguro. A segunda é um modelo para gerenciamento de direitos empresariais que visa garantir a segurança de elementos cruciais, como controle de acesso, distribuição, e confidencialidade de documentos sensíveis; políticas de uso e log das atividades do usuário. Por fim, um modelo de videoconferência segura, considerando que: as chaves criptográficas usadas para o *Secure Real Time Transport Protoco*l (SRTP) podem ser roubadas da memória da aplicação durante o processamento; o *stream* AV pode ser capturado durante o processamento da mídia ou durante a decriptação do SRTP; e as identidades do usuários podem ser roubadas, comprometendo a autenticação dos participantes.

Os *logs* de sistema são, frequentemente, alvo dos atacantes que visam apagar os vestígios dos seus ataques. O trabalho de [Karande et al., 2017] usa o SGX para redesenhar o sistema de *logs* de modo a garantir a integridade e confidencialidade dos *logs*. Neste esquema, os autores recebem os *logs* processados dentro do enclave e utilizam a selagem para o armazenamento. Apesar de evitar modificações não autorizadas, este esquema não consegue evitar que o atacante delete os *logs*. Este é um problema muito difícil de ser resolvido. Em casos extremos podem ser utilizados discos de escrita somente, mas esta é uma solução extremamente cara. Outra possibilidade é realizar o armazenamento em vários servidores. Neste caso vale ressaltar que a abertura dos *logs* só pode ser feita pelo mesmo processador que realizou a selagem.

A aplicação de mídia social *Signal* [Marlinspike, 2017] recentemente empregou o SGX para garantir a privacidade dos seus usuários. Essa aplicação usa a agenda dos usuários para construir o grafo social necessário para descobrir os contatos que também são usuários de *Signal*. Para garantir a privacidade dos usuários esses grafos não são armazenados de forma persistente. No entanto, para aumentar a transparência e para que o usuário não tenha que confiar no funcionamento correto da aplicação, os desenvolvedores empregaram o uso de enclaves SGX para proceder a busca de contatos mediante requisição dos clientes. Nesta solução, os clientes irão se atestar com o servidor e enviar seus dados cifrados, que serão processados dentro do enclave e retornados cifrados para o cliente. Como o sistema operacional e o serviço *Signal* não têm acesso às informações dentro do enclave não será possível registrar nenhuma informação sobre a requisição do usuário. Técnicas para evitar ataques de canal lateral também são aplicadas para prevenir vazamento de dados.

3.6 Limitações, questões em aberto e vulnerabilidades do SGX

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

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 novamente. Essa é uma questão também importante quando se trata de questões de balanceamento de carga em servidores, seja na utilização de múltiplos servidores ou de servidores com múltiplos processadores. Em virtude desta limitação, ainda não há disponível no mercado processadores *multi-socket* com arquitetura SGX disponível. 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 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 [Hoekstra et al., 2013].

O trabalho de [Davenport e Ford, 2014] também aponta 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. Os autores [Schwarz et al., 2017] implementaram um ataque de canal lateral ao enclave através de um *malware* que, por rodar dentro de outro enclave, fica protegido pelo isolamento provido pela arquitetura SGX e não pode ser identificado por anti-vírus ou analisado pelo sistema operacional. Procedimentos forenses também ficam limitados devido à proteção da memória.

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 estes 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. Os autores [Moghimi et al., 2017] utilizam este mesmo método para extrair chaves AES de um processamento do enclave e [Brasser et al., 2017] conseguiram, 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, os dados na *cache* rodam 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 esta técnica pelo menos duas primitivas devem ser atendidas: o *malware* deve ser capaz de sobreescrever determinados endereços da *cache* usados pela vítima; e o *malware* 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 código cujo acesso à memória é dependente da informação secreta (a chave por exemplo). Na sequencia o atacante faz

uma requisição dos seus dados e mede o tempo para recuperá-lo. 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.

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.

O trabalho de [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 granularidde 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 deste fato. No entanto, até o presente momento, não foram encontrados trabalhos demonstrando o uso destas vulnerabilidades para efetivamente descobrir segredos do enclave.

Por fim, além da Intel precisar 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.

3.7 Considerações finais

As Seções 3.1 e 3.2 apresentam alguns trabalhos recentes voltados ao aumento da segurança em sistemas de autenticação em aplicações diversas. Percebe-se que muitas destas soluções são baseadas na proteção das senhas através de algoritmos de resumo criptográficos. É interessante notar que a solução apresentada no Capítulo 4 pode ser adaptada para utilização de modo complementar às soluções apresentadas nestes trabalhos adicionando uma camada extra de segurança à solução. Isto pode ser feito utilizando o SGX para selar os arquivos de credenciais, evitando, assim, vazamentos de senhas (mesmo que na forma de *hash*) e que modificações sejam desapercebidas.

As Seções 3.3 e 3.4 apresentaram mecanismos de proteção de aplicações baseados, respectivamente, em *hardware* e em virtualização, que de algum modo se relacionam, ou apresentam semelhanças ao Intel SGX. Nota-se que dentre os mecanismos estudados, o SGX abrange as principais características necessárias para o desenvolvimento de aplicações seguras.

A Seção 3.5 apresenta trabalhos relacionados ou utilizando o SGX para aplicações diversas como, por exemplo, geração de OTP e videoconferência seguros. Por fim, a Seção 3.6 abordou as principais vulnerabilidades e limitações conhecidas do SGX.

Capítulo 4

Proposta

Este capítulo apresenta a proposta de um módulo de autenticação, para a plataforma PAM, que proporciona uma maior proteção para o arquivo de armazenamento de credenciais. Além do contexto e da justificativa da aplicação proposta, serão apresentados o modelo de ameaças, a descrição da solução, uma análise de segurança e as limitações da proposta.

4.1 Contexto de aplicação

Segredos industriais, comerciais e intelectuais, gerência de sistemas de produção e de sistemas de tráfego são apenas alguns exemplos de dados armazenados ou atividades exercidas por sistemas computacionais. Normalmente a última barreira entre uma pessoa não autorizada e esses bens, que muitas vezes constituem o cerne das principais empresas e governos do mundo, são sistemas de autenticação. Desta forma, sistemas de autenticação costumam ser altamente visados por atacantes com objetivos específicos. Conforme exposto na Seção 2.1.2, os métodos de ataques são muitos e têm ficado cada vez mais sofisticados, uma vez que o sucesso de um ataque, muitas vezes, pode representar para o atacante o ganho de montantes enormes de dinheiro ou até mesmo, quando este não for o objetivo, pode causar a falência de empresas ou prejuízos ao governo e à população. Atualmente existem sistemas de proteção contra ataques bastante eficientes, mas ainda não representam obstáculos intransponíveis contra atacantes motivados.

A história recente nos mostra que nem mesmo grandes empresas da área de TI, com acesso a recursos abundantes estão livres de serem violadas. O *LinkedIn* teve informações de mais de 117 milhões de usuários roubadas e colocadas a venda em um ataque em 2012 [Franceschi-Bicchierai, 2016]. Dentre elas, emails e senhas processadas com SHA1 sem *salt*. De uma amostra de 6,5 milhões de senhas cifradas que foi colocada *online*, 90% foram quebradas em menos de 72 horas. O Yahoo confirmou que pelo menos 500 milhões de contas de usuários foram roubadas em um dos maiores vazamentos de informação de todos os tempos [Fiegerman, 2016]. Estas informações incluíam nomes, emails, telefones, datas de nascimento, *hashes* de senhas e, em alguns casos, perguntas de segurança em claro. O ataque teria ocorrido no final de 2014. Credenciais de mais de 68 milhões de contas do serviço de armazenamento em nuvem da plataforma *DropBox* foram roubadas [Khandelwal, 2016]. A empresa foi obrigada a notificar os consumidores de que iria realizar uma reinicialização forçada das senhas. Por fim, nem mesmo a RSA ficou livre do potencial ofensivo dos atacantes. A empresa de segurança da informação teve que substituir 40 milhões de *tokens SecurID*[1] devido a um ataque ocorrido em março de 2011. A

[1] *SecurID* é um token gerador de OTP, números pseudo-aleatórios usados como segundo fator de autenticação. Estes números são gerados a partir de um segredo (*seed*) aplicando-se um algoritmo desenvolvido pela RSA. Cada Token tem um *seed* diferente. Se o *seed* de um *token* é descoberto por um atacante ele se torna inútil.

RSA, no entanto, não divulgou a quais informações o *hacker* teve acesso [Bright, 2011]. Nesse contexto, este trabalho propõe uma arquitetura de proteção de arquivos de senhas centrada em uma nova tecnologia de segurança lançada pela Intel em 2015, o *Intel Software Guard Extensions*.

4.2 Justificativa

A tecnologia SGX, apresentada na Seção 2.3, proporciona novas possibilidades para a proteção de parcelas sensíveis de aplicações contra diversos tipos de ataques, inclusive aqueles que fazem uso de *kernel* ou sistema operacional malicioso. Por ser uma tecnologia recente ainda é pequeno volume de trabalhos explorando e testando estas novas possibilidades. Três conceitos constituem os alicerces que sustentam estas possibilidades. O primeiro é o conceito de *enclave*, uma região criptografada da memória, protegida pelo hardware da CPU; o segundo é o conceito de *selagem* de dados que trata da aplicação de uma criptografia em dados sensíveis que só pode ser revertida dentro do mesmo enclave ou, opcionalmente, dentro de enclaves do mesmo autor e; por fim, o conceito de *atestação*, um mecanismo desenvolvido para comprovar à uma entidade externa de que determinado enclave pode ser confiado e para estabelecer um canal seguro de comunicação a ser utilizado para troca de segredos.

Desta forma, vislumbrou-se a possibilidade de aplicar esta tecnologia em benefício dos sistemas de autenticação e proteção dos arquivos contendo informações confidenciais dos usuários. Conforme exemplificado na Seção 4.1, estes elementos constituem parte extremamente sensível de aplicações e bancos de dados que requerem confidencialidade e são frequentemente comprometidos por ataques bem sucedidos.

4.3 Metodologia de desenvolvimento

O presente trabalho foi desenvolvido conforme a seguinte metodologia:

1. Determinar o modelo de ameaças ao arquivo de credenciais utilizado para autenticação no Linux, conforme Seção 4.4.

2. Esboçar o funcionamento de um módulo PAM utilizando SGX, conforme Seção 4.4.

3. Analisar a solução do ponto de vista da segurança (Seção 4.5) a fim de identificar e, se possível, sanar vulnerabilidades no esboço do módulo.

4. Estudo do funcionamento do módulo `pam_unix.so`, de modo a identificar e isolar os componentes relativos à autenticação.

5. Integrar a solução com SGX ao módulo `pam_unix.so` através da substituição da função relativa à checagem de credenciais por uma chamada ao enclave (ECALL).

6. Fazer uma aplicação para preparar o ambiente de testes. Isto é necessário tendo em vista que o módulo SGX consulta um arquivo de credenciais selado, que não existe originalmente no Linux. Esta aplicação tem a função de selar o arquivo de credenciais `etc/shadow`.

7. Especificar o ambiente, equipamentos e condições de testes.

8. Determinar os cenários a serem testados e as métricas a serem obtidas com a finalidade de proporcionar resultados quantitativos da influência dos processos de segurança do SGX no desempenho da solução.

9. Realizar os testes nas condições e cenários especificados, consolidar e analisar os resultados obtidos.

4.4 Arquitetura de proteção de arquivos de senhas de autenticação utilizando SGX

O objetivo desta solução é proteger o arquivo de credenciais do Linux, utilizado pela plataforma de autenticação PAM, contra vazamento de dados. A arquitetura de proteção proposta envolve a aplicação dos conceitos de **enclave** e de **selagem** da tecnologia SGX para que o arquivo contendo as credenciais seja armazenado encriptado, e o processo de validação das credenciais seja feito dentro do enclave. O armazenamento seguro de chaves criptográficas é um ponto altamente sensível quando se trabalha com dados encriptados. Utilizando-se a selagem para a criptografia dos dados armazenados fica dispensado o armazenamento da chave criptográfica utilizada, uma vez que a mesma é gerada pela CPU a partir dos dados do enclave e nunca atravessa os limites da CPU. Assim, a superfície de ataque ao arquivo de senhas é reduzida e consequentemente, as chances de vazamento das informações de usuário contidas no arquivo de senhas também são minimizadas. Esta forma de autenticação é aplicável à grande maioria dos sistemas de autenticação estudados pois, conforme exposto na Seção 2.1, o armazenamento e manipulação de credenciais são fatores críticos da segurança da autenticação.

A solução consiste na implementação de um módulo PAM que consulta um arquivo de credenciais selado e faz a comparação das credenciais de autenticação dentro de um enclave. Escolheu-se o módulo `pam_unix.so` para servir como referência de implementação por este ser o módulo de autenticação padrão do Linux. Os detalhes da implementação serão tratados na Seção 5.1. A Figura 4.1 apresenta um fluxograma do funcionamento da solução proposta.

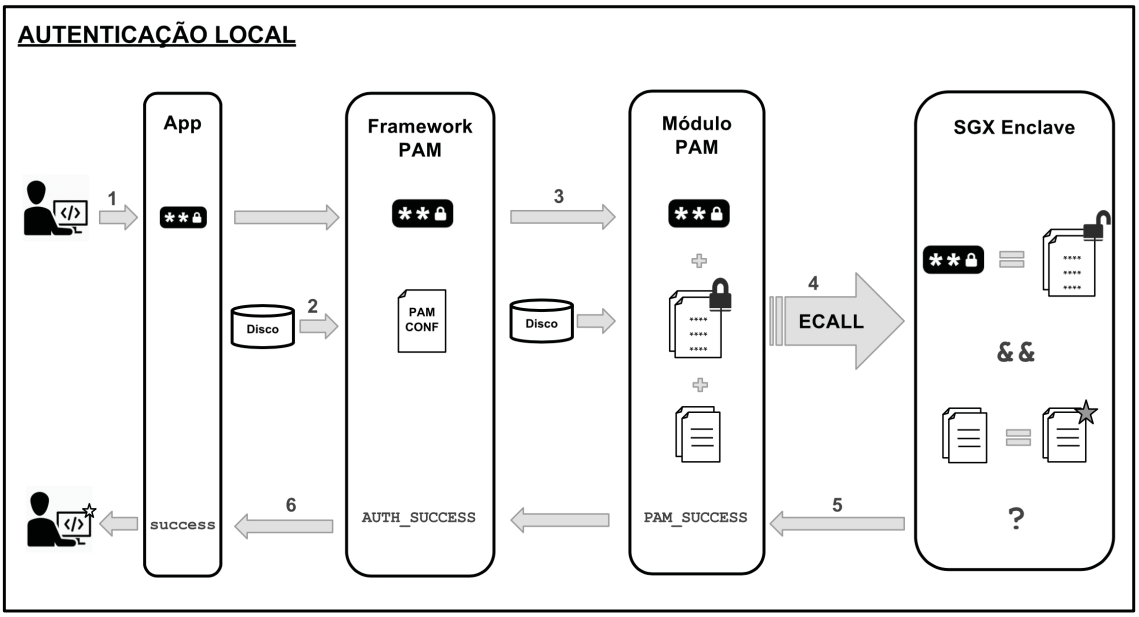


Figura 4.1: Funcionamento da solução de autenticação PAM com proteção do arquivo de senhas usando SGX.

1. O usuário faz uma requisição de autenticação a uma aplicação que utiliza o PAM para se autenticar (por exemplo a aplicação *login*).
2. O *framework* PAM consulta seu arquivo de configuração para constatar que a autenticação deve ser feita através de um módulo PAM customizado.
3. O módulo customizado é carregado pelo *framework* PAM. Se o usuário ainda não forneceu suas credenciais, funções de conversação da biblioteca PAM serão chamadas pelo módulo para sua obtenção.
4. O módulo customizado realiza uma chamada para o enclave SGX (ECALL) passando como parâmetro o *login* e senha (ou o *hash* da senha) colhidos do usuário e o arquivo selado contendo as credenciais armazenadas.
5. Por sua vez, o enclave SGX desela o arquivo de senhas e realiza a validação da senha daquele usuário e retorna sucesso ou erro de autenticação para o módulo PAM.
6. O *framework* PAM autentica, ou não, o usuário para a aplicação.

4.5 Análise de segurança

A análise de segurança será dividida em três partes para melhor compreensão. Na primeira será apresentado o modelo de ameaças enquanto que na segunda será realizada uma análise de segurança baseada nas características, mecanismos de segurança implementados e nas vulnerabilidades na arquitetura SGX. Por fim, a segurança da solução proposta, frente aos diversos tipos de ataque a sistemas de autenticação elencados na Seção 2.1.2, será analisada.

4.5.1 Modelo de ameaças

O modelo de ameaças deste trabalho considera que o adversário tem o objetivo de capturar credenciais contidas em arquivos usados para autenticação de usuários. Estas credenciais poderiam, por exemplo, ser utilizadas como trampolins para acesso à outros sistemas, haja visto que usuários frequentemente escolhem a mesma senha para múltiplas aplicações. Pode ser que o agente malicioso já possua o controle ou mesmo acesso físico à máquina atacada. Considera-se, portanto, que o atacante tem acesso privilegiado à BIOS, *kernel*, sistema operacional e, consequentemente, ao arquivo de credenciais. Assim, o adversário pode capturar este arquivo para realizar um ataque *offline* em ambiente de alta capacidade de processamento. O atacante pode, ainda, ter acesso físico à máquina e realizar *dumping* de memória para tentar extrair as credenciais de autenticação contidas no enclave. Além disso, ele pode subverter a parte não-confiável do módulo de autenticação para tentar extrair informações do enclave usado na comparação das credenciais de autenticação.

Supõe-se que a tecnologia empregada funciona adequadamente conforme suas especificações. Dessa forma, a arquitetura SGX impede que o adversário faça alterações diretamente no enclave ou construa outro enclave capaz de decifrar o arquivo selado. Para isso, no entanto, o ambiente de desenvolvimento do enclave que realiza a autenticação deve ser confiável.

4.5.2 Análise de segurança do SGX

A base de computação confiável (TCB) consiste no menor conjunto de elementos que precisam ser confiados para construção de um sistema seguro. No caso do SGX, a TCB consiste

na CPU e seus elementos internos, como a lógica do *hardware*, microcódigo, registradores e memória *cache*. Além da CPU, integram a TCB componentes de *software* utilizados na atestação remota, em especial o *quoting enclave*.

O SGX protege os enclaves de qualquer componente malicioso fora da TCB, em particular: do sistema operacional ou hipervisor; da BIOS, *firmware* ou *drivers* da máquina; de *software* SMM; do *software* de gerenciamento da Intel (ME); e de atacantes que possam comprometer quaisquer dos elementos citados.

A documentação da Intel aponta, no entanto, algumas limitações de segurança. O SGX não é capaz de evitar que atacantes explorem padrões de acesso à cache e execuções cujo segredo é dependente do tempo em programas vulneráveis a ataques por canal lateral. Ataques físicos contra a CPU, como aqueles onde o atacante executa ataques como injeção invasiva de falhas e, ainda, ataques ao microcódigo onde o atacante consegue reprogramar as funcionalidades do código de máquinas não são evitados.

Algumas vulnerabilidades além daquelas pontuadas na documentação da Intel são apontadas por [Aumasson e Merino, 2016]. Possíveis *bugs* nas bibliotecas confiáveis da Intel, na lógica do *harware* ou no microcódigo são vulnerabilidades levantadas. No entanto, esses *bugs* só podem ser detectados no estilo "caixa preta", uma vez que a documentação detalhada do *hardware* e microcódigo não foram disponibilizadas. Esta pesquisa não encontrou relatos de *bugs* identificados.

Além destas, se o ambiente de desenvolvimento estiver já comprometido durante o desenvolvimento do enclave, ou se o SDK for obtido através de um canal inseguro, o enclave poderá não ter o comportamento desejado. Por fim, logicamente, o desenvolvedor tem a responsabilidade de programar de modo seguro. Problemas clássicos de corrupção de memória ou de concorrência, manipulação de ponteiros e vazamento de segredos através de chamadas fora do enclave devem ser evitados pelo desenvolvedor do enclave.

A segurança do SGX depende de uma série de algoritmos criptográficos. Sete algoritmos principais são usados nas diversas funcionalidades suportadas pelo SGX [Aumasson e Merino, 2016], dos quais os importantes para este trabalho são:

- RSA-3072 PKCS#1 v1.5 com SHA-256, para computar a assinatura do enclave;
- assinaturas ECDSA sobre a curva p256 NIST, com SHA-256, para a política de checagem do lançamento do enclave;
- AES-GCM para selagem de dados;
- AES-CMAC para derivação de chaves e;
- AES-CTR para criptografia da memória.

Os algoritmos AES e esquemas de curva elíptica sobre p256 têm segurança de 128 bits. O RSA-3072 e RSA-2048 têm segurança de 112 e 96 bits respectivamente. Todos estes algoritmos são confiáveis, padronizados, testados no tempo e, portanto, improváveis de serem quebrados. Algumas implementações em *software* do AES são notoriamente vulneráveis a ataques do tipo *cache-timing*. Para resolver esta questão, as implementações do AES em componentes críticos do SGX são combinadas com instruções de *hardware* para funcionar em tempo constante, de forma a minimizar a possibilidade de vazamento de dados por este tipo de ataque [Intel, 2016b].

A capacidade de gerar números aleatórios também é fundamental para algoritmos criptográficos. A biblioteca confiável SGX não suporta a função `rand()` da biblioteca `libc`. Esta função é um gerador de números aleatórios determinístico que não deve ser utilizado para

operações criptográficas, uma vez que poderá gerar segredos previsíveis. Ao invés disso o SGX fornece a função `sgx_read_rand()`, que usa um gerador de números aleatórios baseado no *hardware* da CPU, que retorna resultados mais confiáveis.

Outro ponto importante da segurança do SGX é que cada CPU SGX possui uma chave única de 128 bits cujo valor é desconhecido até mesmo para a Intel. Todas as chaves criptográficas utilizadas pela arquitetura SGX são derivadas desta chave. Isso elimina a necessidade do armazenamento de chaves em memória e permite que a TCB seja mantida dentro da CPU.

A Seção 3.6 expõe casos onde pesquisadores conseguiram recuperar chaves de processamento criptográfico ocorrendo dentro do enclave. No entanto, esta pesquisa não encontrou nenhum caso em que as chaves referentes às implementações da Intel dos algoritmos usados na arquitetura SGX fossem comprometidas.

4.5.3 Análise de segurança da solução

Na Seção 2.1.2 foi enumerada uma série de ataques a sistemas de autenticação. Nesta Seção será discutido como a solução proposta protege as credenciais dos usuários e evita alguns tipos de ataques. Os ataques foram agrupados por afinidade para tornar o texto mais conciso.

- **Ataques de Força bruta, por dicionário, senhas padrão e *rainbow tables***: Em qualquer sistema de autenticação baseado em senhas o atacante sempre poderá tentar estes tipos de ataques. As credenciais são testadas individualmente, não ameaçando o arquivo de credenciais como um todo. O tempo necessário para que um atacante descubra uma senha depende da complexidade, ou força, desta senha e do tempo que o sistema demora para retornar o sucesso ou a falha de autenticação. Na solução proposta, quando uma tentativa de autenticação não obtém sucesso é adicionado um atraso no seu retorno. Isso não interfere na percepção de qualidade do usuário, uma vez que durante uso comum a senha correta é fornecida na maioria das vezes. Por outro lado, durante um ataque de força bruta a senha incorreta é fornecida na grande maioria das vezes. O pequeno atraso incluído neste caso gera um enorme atraso para o sucesso do ataque. Para se ter uma ideia quantitativa da eficiência desta simples medida, suponha que o atacante está tentando descobrir, através de força bruta, uma senha fraca de seis dígitos compostos de 26 opções de caracteres (somente letras minúsculas) e que ele consiga testar uma senha a cada 2 milissegundos. Neste cenário, em 7 dias, no máximo, a senha seria descoberta. Se um atraso de apenas 5 segundos for adicionado ao retorno da falha de autenticação o prazo máximo para a descoberta desta senha seria de aproximadamente 49 anos. Desta análise decidiu-se usar a função `sleep()` para gerar o atraso no caso de erro de autenticação. Para viabilizar o uso desta função é necessário realizar uma chamada para fora do enclave (OCALL). Um atacante que tenha o domínio da máquina poderá contornar a execução desta função, uma vez que ela executa fora do enclave. Implementar um método que gere este atraso dentro do enclave (através de um laço `for()`, por exemplo) é uma abordagem mais segura. A inclusão deste atraso, no entanto, é ineficaz quando o usuário usa senhas extremamente fracas, como senhas padrão, ou vulneráveis à ataques por dicionário. Para se precaver contra esses casos uma medida possível seria bloquear a senha do usuário após um certo número de tentativas incorretas.

- **Ataques *offline***: Um atacante que tenha escalado privilégio na máquina ou com acesso físico poderia roubar o arquivo de credenciais selado pelo SGX para tentar montar um ataque *offline*, em um ambiente com alta capacidade de processamento, para descobrir

a lista de senhas armazenadas. Segundo a *Wikipedia* o supercomputador mais rápido do mundo, o chinês *Sunway Taihu Light*, tem velocidade de pico de 93,01 PFLOPS (*Peta Floating Points Operations per Second*) o que equivale à 93,01 x 10^{15} FLOPS. Assumindo que um ataque de força bruta à chave de selagem seja realizado neste supercomputador, e que cada tentativa de chave gaste somente 100 FLOPS (o que é uma suposição extremamente otimista) o atacante será capaz de testar 93,01x10^{13} combinações por segundo. A chave de selagem é uma chave AES de 128 bits, o que resulta em um total de 2^{128} = 3,40x10^{38} combinações possíveis. Assim, considerando que um ano tem 365x24x60x60 = 31.536.000 segundos, o atacante pode levar até 3,40x10^{38}/(93,01x10^{13}*31536000) = 1,16x10^{16} anos, ou seja, 11,6 quatrilhões de anos. Desta análise percebe-se que as credenciais dentro do arquivo selado poderiam ser armazenadas em claro sem grande prejuízo à sua segurança.

- **Ataques à memória** (DMA e *dumping* de memória): Conforme explanado na Seção 2.3.1, o controlador de memória da CPU SGX rejeita transferências via DMA para o PRM de forma a proteger os enclaves dos acessos de dispositivos periféricos e de ataques por DMA. Além disso, como a memória do enclave é encriptada, a análise da quebra de chave AES de 128 bits feita no item anterior também se aplica ao caso do atacante conseguir realizar um *dumping* de memória para posterior análise.

A análise acima mostra que o arquivo de armazenamento das credenciais está, de fato, protegido. Proteger este arquivo é de fundamental importância, já que sua violação além de comprometer o sistema alvo do ataque pode comprometer também outros sistemas de determinados usuários, pois, como visto na Seção 2.1.1, é comum usuários utilizarem a mesma senha para múltiplos sistemas. O modelo de ameaças deste trabalho, no entanto, considera que o adversário já tem o controle da máquina atacada, podendo o sistema operacional ser malicioso ou ainda ter acesso físico à máquina. Desta forma, a proteção do arquivo de credenciais não é suficiente para garantir a segurança de todo o sistema de autenticação. O módulo de autenticação proposto é vulnerável à alguns tipos de ataques. Abaixo são apresentados os principais.

- **Ataques de canal lateral**: O conjunto de instruções *Advanced Encryption Standard New Instructions* (AES-NI) utilizado pelo SGX foi desenvolvido para funcionar em tempo constante de forma a evitar ataques de canal lateral baseados no tempo [Intel, 2016a]. Sendo assim, as chaves utilizadas no procedimento de selagem de arquivos ficam protegidas contra este tipo de ataque [Intel, 2016b]. No entanto, processamentos dentro dos enclaves devem ser realizados através de implementações resistentes à ataques de canal lateral. Ataques baseados no tempo, monitoramento de energia ou monitoramento das páginas sendo acessadas podem ser aplicados para tentar descobrir credenciais sendo utilizadas durante a autenticação procedida pelo enclave.

- **Contornar o sistema de autenticação**: Se a aplicação a ser utilizada não for segura o suficiente e o atacante tiver privilégios de administrador no sistema ele poderá usar o acesso a todos arquivos e processos para depurar a aplicação e burlar seu esquema de autenticação, por exemplo, trocando o comando que chama a autenticação PAM por um comando que retorna sucesso na autenticação. Uma alternativa para solucionar este problema seria reescrever as aplicações de modo a utilizar enclaves para, antes da sua execução, verificar a integridade da aplicação e se o processo de autenticação ocorreu corretamente. Outro ataque possível para um adversário com privilégios de administrador seria simplesmente modificar o arquivo de configuração

do PAM para não utilizar o módulo de autenticação do SGX. Este problema também pode ser resolvido com o SGX através de modificações na plataforma do PAM. Uma possibilidade é selar o arquivo de configuração e usar os enclaves como interface para o seu acesso e modificação. Desta forma, o acesso direto do administrador ao arquivo de configuração e modificações não autorizadas seriam impedidas. Essas abordagens não foram implementadas uma vez que este trabalho se concentra na proteção do arquivo de senhas e são possibilidades de trabalhos futuros.

- ***Keyloggers***: A segurança dentro do enclave é garantida pela tecnologia SGX, mas a proteção dos dados antes de serem passados para o enclave ainda é uma questão em aberto. Por exemplo, esta solução não consegue evitar um ataque por *keylogger* onde o atacante pode capturar a senha do usuário no instante em que é digitada. Para evitar este tipo de ataque deve-se utilizar um caminho confiável para o tráfego das informações de entrada de dados. Uma das formas para se estabelecer o caminho confiável para a entrada de dados é o uso de um dispositivo de entrada dedicado. Este dispositivo deve suportar criptografia nativa, o que permite que seja criado um canal seguro para a entrada de dados. Já existem trabalhos como o SGXIO [Weiser e Werner, 2017] que utilizam enclaves SGX para auxiliar o estabelecimento de um caminho confiável protegendo os dados de entrada/saída de ataques lógicos. O SGXIO é composto de uma pilha confiável com um pequeno hipervisor e um ou mais *drivers* de entrada/saída seguros, além de um enclave para inicialização confiável. Aplicações que necessitam se comunicar de modo seguro com o usuário abrem um canal de comunicação criptografado com um *driver* de entrada/saída formando um túnel pelo sistema operacional não-confiável. Para isto, o hirpervisor é usado exclusivamente para ligar os dispositivos do usuário aos *drivers* correspondentes. Os *drivers* usam o enclave de inicialização confiável para atestar que o caminho confiável dos dados está sendo utilizado corretamente.

- **Ataques de rede** (*sniffers*, *man-in-the-middle*, repetição da autenticação, reduzir a força da autenticação e uso de servidores falsos): Estes tipos de ataques se aplicam em soluções de autenticação remota, o que não é o caso do módulo proposto. A Seção 6.2 apresenta uma breve discussão de como a atestação SGX pode ser utilizada para evitar estes tipos de ataques.

- **Ataques não-computacionais** (engenharia social, *shoulder surfing*, *dumpster diving* e roubo de identidade): Ataques não computacionais só podem ser prevenidos através da capacitação e conscientização adequada dos usuários do sistema. Estes tipos de ataque não estão no modelo de ameaças desta solução.

4.6 Limitações da solução

A implementação proposta trata de um módulo de autenticação, envolvendo apenas o serviço *auth* do PAM, na forma de uma prova de conceito da aplicação do SGX para proteção de arquivos de credenciais. Soluções de autenticação relativa aos demais serviços PAM (*session*, *account* e *passwd*) poderão ser objetos de trabalhos posteriores.

A memória dedicada aos enclaves possui tamanho limitado. O processador da máquina de testes suporta enclaves de até 128MB. Isto implica em uma limitação do tamanho do arquivo de credenciais, que na solução proposta é carregado integralmente para dentro do enclave. Este problema pode ser resolvido dividindo o arquivo de credenciais em pedaços menores e fazendo a busca nos arquivos para a comparação de usuário e senha. É possível, ainda, instanciar diversos

enclaves para soluções que demandam o processamento de volume de dados maior que os limites do enclave. No entanto, tratando-se de uma prova de conceito, esta situação não foi tratada na implementação proposta. Esta solução também supõe que a Intel é confiável e que o SGX funciona corretamente conforme suas especificações.

Por fim, uma vez autenticado, diversas aplicações ainda precisam acessar informações do usuário que estão nos arquivos `/etc/passwd`, `/etc/shadow`, e `/etc/group`. Por este motivo, em um cenário de uso real, não é possível manter o `/etc/shadow` selado sem antes realizar determinadas adaptações. Para isso ser possível, as aplicações que criam, gerenciam e manipulam os usuários Linux e a plataforma PAM devem ser modificadas de forma que todas as consultas a estes arquivos sejam mediadas por enclaves SGX. Como alternativa de compatibilidade com o legado, pode-se simplesmente deixar de armazenar credenciais no `/etc/shadow` não selado, disponibilizando neste arquivo somente os demais dados dos usuários.

4.7 Relação com outros trabalhos

A coletânea de trabalhos sintetizados na Seção 3.1 fornece exemplos de que o arquivo de credenciais é fator crítico de segurança na maioria dos sistemas de autenticação. A revisão bibliográfica aponta ainda, na Seção 3.2, que as principais abordagens dos esquemas de proteção de credenciais são baseados no uso de funções (*hash*) e operações criptográficas aplicadas individualmente em cada uma das credenciais. Sabe-se que senhas fortes cifradas são computacionalmente difíceis de quebrar mesmo em um ataque offline montado em ambiente de alta capacidade de processamento. No entanto, estudos como os de [Slain, 2016] e [Doel, 2013] mostram que parcela considerável dos usuários usam senhas fracas, e ataques reais, como o ataque ao *LinkedIn* [Franceschi-Bicchierai, 2016], mostram que neste caso o uso de funções *hash* perdem eficiência e as senhas podem ser quebradas dentro de um curto intervalo de tempo.

A solução apresentada neste trabalho se distingue das demais por propor a cifragem do arquivo de credenciais inteiro ao invés de cifrar as credenciais separadamente. Mesmo senhas fortes possuem baixa entropia quando comparadas ao arquivo que armazena todas as credenciais de autenticação. Assim, montar um ataque para decifrar um arquivo selado contendo várias credenciais é muito mais custoso do que um ataque sobre uma única credencial cifrada. Esta proposta, portanto, representa um mecanismo de proteção de credenciais mais eficiente do que os apresentados na Seção 3.2. Quando se pensa no conceito de segurança em camadas, ou segurança em profundidade, no entanto, nota-se que esta solução não é concorrente das demais, mas pode ser utilizada de forma complementar. A camada extra de segurança do esquema proposto oferece proteção de arquivos sensíveis (como o arquivo de credenciais), evitando a recuperação de informações pelo atacante mesmo que ele tenha acesso ao arquivo armazenado e capacidade para montar ataques diversos contra este arquivo.

Além de cifrar o arquivo de credenciais inteiro, este trabalho aplica a tecnologia SGX para propor um método seguro para manipulação desse arquivo, de modo a garantir que seus dados nunca fiquem em claro, nem mesmo para atacantes com privilégio em nível de sistema ou com acesso à periféricos capazes de realizar DMA. Conforme exposto na Seção 3.3 existem outras tecnologias que fornecem suporte em *hardware* para proteção de dados e execução. A Tabela 3.1 mostra, inclusive, que algumas delas, como o AMD Secure Processor, apresentam mecanismos que também poderiam dar suporte à uma solução similar à apresentada neste trabalho. O SGX, no entanto, é a única que concentra todas as funcionalidades necessárias em uma única unidade de *hardware*, o processador, o que resulta na menor superfície de ataque dentre as opções apresentadas.

4.8 Considerações finais

Este capítulo apresentou a proposta de pesquisa, o contexto de aplicação a justificativa para sua realização, a arquitetura de proteção de arquivos de senhas de autenticação utilizando SGX, uma análise de segurança e das limitações da solução e a relação com outros trabalhos. O Capítulo seguinte irá apresentar os detalhes de implementação bem como os resultados obtidos.

Capítulo 5

Validação da proposta

Uma prova de conceito da solução proposta no Capítulo 4 foi implementada para a validação. Neste capítulo serão discutidos os detalhes da implementação e a avaliação do protótipo. Além da avaliação do desempenho do módulo de autenticação, foram realizadas pequenas modificações no protótipo para possibilitar a medição dos impactos de desempenho devido ao uso da arquitetura SGX. Em especial, foram avaliados o custo da inicialização do enclave; o custo de realizar a autenticação em memória encriptada; e o custo de trabalhar com arquivo de credenciais selado.

5.1 Implementação

O módulo de autenticação padrão do Linux, `pam_unix.so`, foi escolhido para servir como base para a implementação do protótipo da solução proposta. Desta forma, para um melhor entendimento da implementação, primeiramente será discutido em detalhes o funcionamento deste módulo e, em seguida, serão apresentadas as modificações e adaptações necessárias para a integração desta solução ao uso de enclaves SGX.

5.1.1 Funcionamento do `pam_unix.so`

O código fonte do `pam_unix.so` é constituído por 4 arquivos principais responsáveis pelos 4 serviços PAM implementados pelo módulo. Como o foco deste trabalho é o serviço de autenticação, somente o arquivo `pam_unix_auth.c` será examinado em maiores detalhes. A Figura 5.1 apresenta, de modo resumido, os principais pontos do fluxo de execução de uma autenticação realizada por este módulo.

Conforme visto na Seção 2.2.4, o método `pam_sm_authenticate()` é o responsável pela autenticação de um módulo PAM. No caso do `pam_unix.so`, este método começa configurando algumas variáveis de controle através da função `_set_ctrl()` que configura manualmente uma série de *flags* baseado nos arquivos de configuração, ou nos argumentos enviados ao módulo. Dentre estas *flags* cita-se, por exemplo, a `PAM_SILENT`, que diz ao módulo para não emitir mensagens. Aqui o arquivo `/etc/login.defs` é consultado para verificação do tipo de criptografia pré-definido e é lido o número de vezes que o algoritmo criptográfico deve rodar.

O próximo passo é obter o nome do usuário. Esta tarefa é desempenhada pela função `pam_get_user`, da biblioteca `security/pam_modules.h`. Pode ser que no instante que esta função for chamada o PAM já tenha obtido o nome do usuário de um módulo anterior. Se o

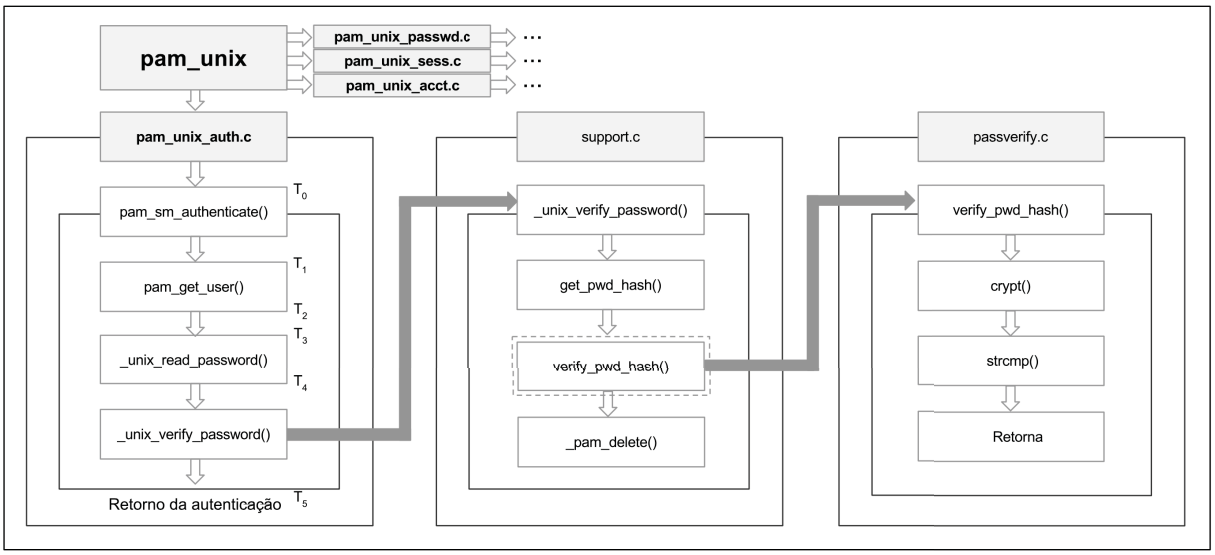


Figura 5.1: Resumo do fluxo de execução das principais funções do módulo de autenticação Linux `pam_unix.so` original.

nome do usuário não for conhecido pelo PAM, automaticamente uma função de conversação com a aplicação será iniciada para obtê-lo.

Em seguida o módulo trata os casos de erro relacionados ao nome de usuário, usando a função `_pam_syslog()` para registro dos *logs*. A função auxiliar `_unix_blankpasswd()` é utilizada para verificar se o usuário em questão tem senha nula. Só então a função `_unix_read_password()` é chamada para obter o *token* de autenticação do usuário e os erros do retorno desta função são tratados.

Por fim, o nome de usuário e a senha obtidas são passados como parâmetro para a função `_unix_verify_password()` definida no arquivo `support.c` para a realização da validação das credenciais de autenticação. Inicialmente esta função localiza o usuário e obtém as suas credenciais de autenticação do arquivo de credenciais através da função `get_pwd_hash()`. Após o tratamento dos erros a função `verify_pwd_hash()` é chamada para realizar a validação das credenciais. Os erros de autenticação são contabilizados e a função `_pam_delete()` para limpar os dados sensíveis do usuário.

A função `verify_pwd_hash()` recebe como parâmetros a senha fornecida pelo usuário, o *hash* com *salt* da senha obtida do arquivo de credenciais pelo método `get_pwd_hash()` e a *flag* `nullok`, obtida do arquivo de configuração do PAM. Os primeiros caracteres da *string* do *hash* são verificados para obter o tipo de criptografia utilizada no *hash*. Então a senha fornecida pelo usuário é convertida para a forma de *hash* para ser confrontado, através da função `strcmp()`, com aquele obtido do arquivo de credenciais.

5.1.2 Funcionamento do módulo implementado

Da análise da parte de autenticação do módulo `pam_unix.so` pode-se concluir que é possível realizar uma adaptação da solução original do Linux para a arquitetura de proteção do arquivo de credenciais proposta. Para isto as alterações necessárias foram realizas na função `_unix_verify_password()` definida no arquivo `support.c`.

Como o arquivo de credenciais agora é armazenado selado, a função `get_pwd_hash()` perde sua funcionalidade e é substituída pelo processo de inicialização do enclave e tratamento dos erros de inicialização. A função `verify_pwd_hash()` é substituída por uma ECALL

que absorve também as responsabilidades da função `get_pwd_hash()`. Esta ECALL foi chamada de `ecall_verify_pwd_hash()`.

Inicializado o enclave, o arquivo de credenciais selado é carregado integralmente para uma variável para ser enviado como parâmetro para a função `ecall_verify_pwd_hash()`, que também deve receber o tamanho do arquivo selado, uma variável para retorno do tipo inteiro além dos parâmetros da `verify_pwd_hash()` original. A Figura 5.2 apresenta os principais pontos de alteração em contraposição ao módulo original.

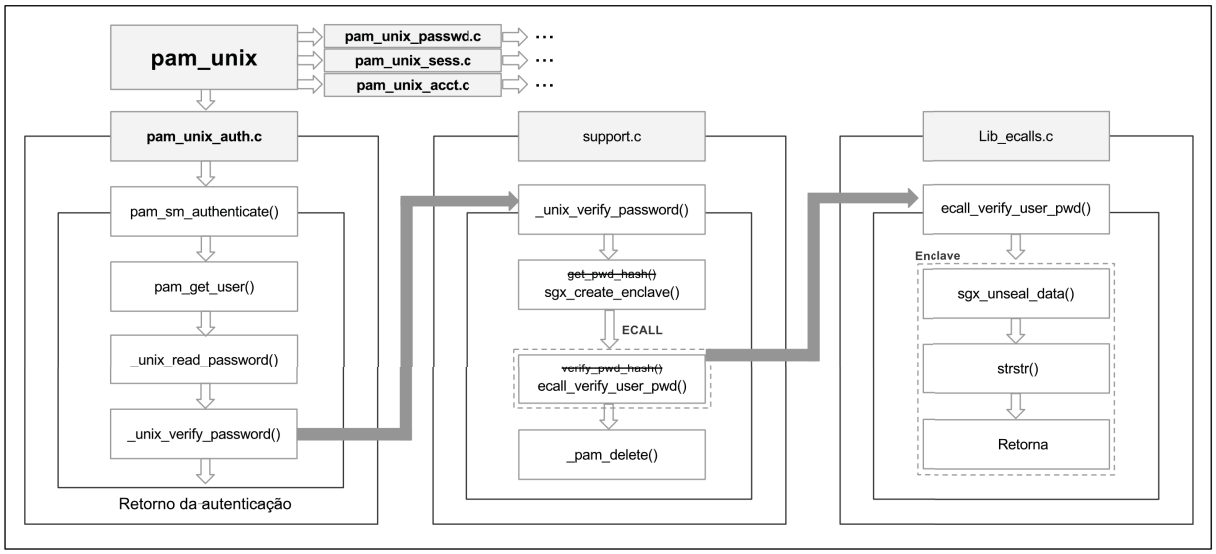


Figura 5.2: Resumo do fluxo de execução das principais funções do módulo de autenticação baseado no `pam_unix.so` utilizando o SGX para a proteção do arquivo de credenciais.

O enclave, por sua vez, ao receber o arquivo de credenciais através da `ecall_verify_pwd_hash()` aloca memória protegida para proceder a sua abertura através da função confiável `sgx_unseal_data()`. Em seguida é realizada a a busca das credenciais de autenticação do usuário e são tratados os erros, como por exemplo, o caso do usuário não existente. Esta busca é feita através da função `strstr()` da biblioteca `string.h` do SDK SGX, que retorna um ponteiro para o início da ocorrência do conteúdo buscado. A partir de então esta ECALL desempenha as tarefas da função `verify_pwd_hash()` de modo análogo, mas com uma diferença: sempre que ocorrer um erro de autenticação, independente do motivo, é realizada chamada a função `ocall_sleep` que força um atraso na resposta da autenticação, conforme discutido na Seção 4.5.

Para gerar *logs* do processamento ocorrendo dentro do enclave seria necessária a implementação de chamadas para o ambiente inseguro (OCALLs). Dentro da premissa de manter a interface do enclave a menor possível e evitar chamadas para o ambiente não-confiável, nenhum *log* é gerado durante a execução dentro do enclave. As informações necessárias para o desempenho da função `_unix_verify_password()` (como por exemplo, se o usuário existe no arquivo de credenciais ou se a senha de usuário confere) são passadas para fora do enclave através da variável de retorno passada como parâmetro na ECALL. Este retorno é passado na forma de um inteiro e é o único valor que atravessa a interface do enclave para o ambiente externo. À exceção da execução do enclave, os demais *logs* são gerados tais como no módulo original.

O módulo original implementa em seu código fonte apenas o algoritmo de criptografia de senhas para o *hash* MD5. Assim o algoritmo do MD5 foi modificado para a execução dentro do enclave. Essas modificações consistem na eliminação das dependências de bibliotecas dinâmicas

e na substituição de funções inseguras não suportadas pelo SGX conforme [Intel, 2016b] (como por exemplo as funções de manipulação de *strings* como `strcpy()` e `strcat()`). Para os demais algoritmos é utilizada a função `crypt()` do Linux que é ligada dinamicamente e não foi modificada para funcionamento dentro do enclave.

A Figura 5.3 representa o esquema de desenvolvimento, compilação e assinatura que geram o enclave e o módulo de autenticação. Conforme apresentado na Seção 2.3.4 um cuidado especial deve ser tomado com a interface do enclave. Dessa forma é necessário declarar em um arquivo EDL todas as funções que realizam transições da aplicação para o enclave e vice-versa. Para isto foi criado o arquivo `Enclave.edl` com a declaração das funções `ecall_verify_pwd_hash()` e `ocall_sleep`. A partir deste arquivo a ferramenta da Intel *edger8er* é utilizada para gerar as rotinas de borda que realizam a manipulação segura dos parâmetros pela interface do enclave. Para que isso seja possível a declaração da função no arquivo EDL deve conter atributos indicando o sentido do tráfego dos dados (se o dado vai entrar ou sair do enclave) e o tamanho dos dados que serão manipulados. O fonte destas rotinas devem ser compiladas juntamente ao enclave e encontram-se nos arquivos `Enclave_t.h` e `Enclave_t.c` referente à interface do ambiente confiável e `Enclave_u.h` e `Enclave_u.c` referente à interface do ambiente não-confiável.

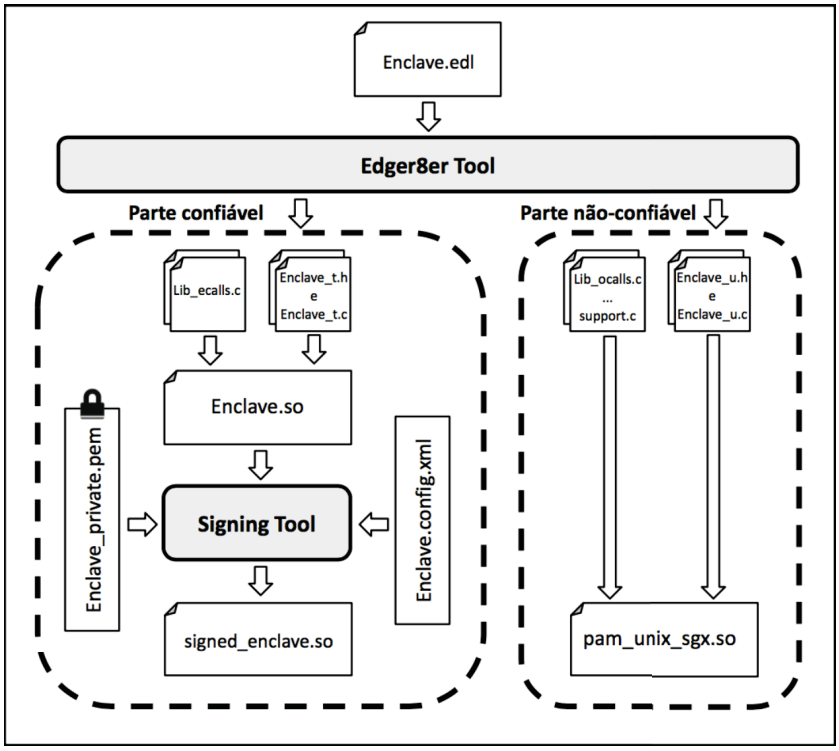


Figura 5.3: Processo de desenvolvimento do enclave e do módulo PAM.

Conforme Apêndice A, a memória necessária para a execução do enclave deve ser definida antes da sua inicialização. A definição dos tamanhos máximos da *heap* e da *stack* são realizadas no arquivo de configuração do enclave. Neste arquivo também são definidos o identificador de produto PRODUCTID e o número de versão de segurança ISVSVN, dentre outros parâmetros de configuração. Estas configurações constam no arquivo `Enclave.config.xml`. Conforme [Intel, 2016b], se este arquivo não tivesse sido fornecido valores padrão seriam utilizados.

Por fim, para tornar possível a utilização do enclave, após a compilação, o enclave foi assinado com a ferramenta de assinatura da Intel *Enclave Signing Tool* utilizando-se a chave constante no arquivo `Enclave_private.pem`.

5.2 Avaliação do protótipo

Para a avaliação do protótipo foram medidos os tempos de resposta do processo de autenticação do módulo implementado para ser comparado ao módulo original, `pam_unix.so`. Além disto, cenários intermediários entre o módulo original e o proposto foram implementados no intuito de medir a influência da inicialização do enclave, da cifragem da memória da EPC e da selagem dos dados.

5.2.1 Métricas utilizadas

Conforme explicado nas Seções 2.2.4 e 5.1.1, o método `pam_sm_authenticate()` é responsável pelo serviço de autenticação do PAM. Para mensurar o impacto gerado pelo SGX no desempenho da autenticação, o tempo gasto para executar este método deve ser considerado. Desta forma, as marcações de tempo de T_0 a T_5 foram inseridas no código do módulo em todos os cenários, inclusive no original. A Figura 5.1 ilustra o posicionamento dos marcadores de tempo no módulo de autenticação. T_0 representa o instante do início da execução do método, enquanto T_5 representa o instante imediatamente antes do seu retorno.

O tempo de interação com o usuário, no entanto, não deve ser considerado na aferição do desempenho. Por este motivo foram inseridos os marcadores T_1 e T_2 imediatamente antes e depois, respectivamente, da função responsável por coletar o nome de usuário (`pam_get_user`) e os marcadores T_3 e T_4 imediatamente antes e depois, respectivamente, da função responsável por coletar a senha do usuário (`_unix_read_password`). Desta forma o tempo considerado para que o módulo proceda a autenticação é dado pela soma dos intervalos conforme fórmula abaixo:

$$T = [T_0, T_1] + [T_2, T_3] + [T_4, T_5]$$

O método `clock_gettime()` da biblioteca `time.h`, foi utilizado para gerar os marcadores de tempo e em seguida calcular o tempo em milissegundos. Ressalta-se que esta foi a única alteração no arquivo `pam_unix_auth.c` original do módulo `pam_unix.so`. Este arquivo contém a parte relativa ao serviço de autenticação do `pam_unix.so` e foi aproveitado em todos os cenários de testes, inclusive na versão final utilizando o SGX, sem nenhuma outra alteração.

5.2.2 Cenários considerados

Para medir o impacto no desempenho da solução como um todo e os impactos parciais devido às peculiaridades do SGX foram estabelecidos 4 cenários de testes que foram contrapostos com o serviço de autenticação do módulo *pam_unix.so* original.

- *Cenário 1 - `pam_unix` original*: Neste cenário foram inseridas as marcações de tempo no serviço de autenticação do módulo `pam_unix.so` que foi compilado separadamente dos demais serviços. Este cenário foi montado para aferir o tempo gasto para realizar autenticações pelo módulo `pam_unix.so` original.

- *Cenário 2 - Custo da inicialização do enclave*: Neste cenário o conteúdo da função `_unix_verify_password()` foi excluído e substituído pelo processo de inicialização de um enclave. Nenhuma chamada ao enclave é realizada e esta função retorna sucesso sempre que o enclave é inicializado corretamente. Este cenário foi montado para aferir o tempo gasto para realizar a mera inicialização de um enclave.

- *Cenário 3 - sem selar*: Neste cenário a verificação das senhas é realizada dentro de memória protegida do enclave. No entanto, o arquivo de credenciais é enviado em claro para o enclave, da mesma forma em que é armazenado originalmente no Linux. Aqui o enclave é responsável por buscar o usuário e o *hash* da senha no arquivo de credenciais, aplicar o algoritmo de *hash* na senha fornecida pelo usuário e fazer a comparação dos *hash*, retornando sucesso ou falha de autenticação.

- *Cenário 4 - unix_sgx*: O arquivo de credenciais é armazenado selado pelo SGX e é enviado para dentro do enclave que, por sua vez, irá proceder a sua abertura. Em seguida, a autenticação é realizada de maneira análoga ao cenário 3. Desta forma, a diferença de tempo obtida entre os cenários 4 e 3 corresponde à sobrecarga adicionada para se trabalhar com o arquivo de credenciais selado. Este cenário foi implementado exatamente conforme a descrição da Seção 5.1.

- *Cenário 5 - sem hash*: A análise da Seção 4.5 mostrou que armazenar o arquivo de credenciais selado proporciona segurança suficiente para tornar desnecessário o armazenamento das senhas processadas com funções de *hash*. Assim, neste cenário é utilizado um arquivo de credenciais hipotético que é armazenado selado, mas dentro do qual os usuários e senhas são armazenados em claro. À exceção do exposto, o funcionamento do módulo se dá exatamente como no Cenário 4. Deve-se ter em mente que, apesar do emprego da selagem SGX na proteção de credenciais ser mais eficaz frente ao uso de funções de *hash*, a opção pela implementação deste cenário implica na eliminação de uma camada de segurança comparado ao cenário anterior. Neste caso, deve-se tomar cuidado especial com as vulnerabilidades SGX como, por exemplo, ataques de canal lateral.

5.2.3 *Setup* utilizado (equipamentos e softwares)

Nos testes foi utilizada uma máquina Dell OptiPlex 5040, com processador Intel Core i7-6700, 3.40Hz, de 4 núcleos e 8 *threads*, com SGX habilitado na BIOS, 8GB de memória RAM, 512GB de HD, sendo que 479GB estavam disponíveis. O tamanho máximo de 128MB do enclave foi configurado na BIOS. Além disso, o *TurboBoost*, Intel *SpeedStep* e o *HyperThread* foram desabilitados na BIOS para padronizar as condições de avaliação e evitar oscilações na medição dos resultados. A máquina estava executando o sistema operacional Ubuntu 14.04 LTS com kernel versão 4.2.0-42-generic. Os testes de desempenho foram realizados imediatamente após a inicialização da máquina que estava desconectada da internet.

Uma aplicação customizada foi criada no intuito de realizar testes com o módulo desenvolvido. Ela consiste numa implementação simples que utiliza os métodos convencionados expostos na Seção 2.2.4 para se autenticar utilizando o PAM. Sua execução consiste simplesmente na iniciação de uma sessão PAM através da criação de uma estrutura tipo `pam_handle_t` e da chamada da função `pam_start()` além de uma chamada da função `pam_authenticate()` para realizar a autenticação. Ao terminar o processo, a aplicação indica a falha ou sucesso da autenticação através de uma impressão na tela.

Por fim, a execução da autenticação através do módulo implementado parte do pressuposto da existência de um arquivo de credenciais selado. Para ser possível a execução dos testes foi desenvolvida uma aplicação para selar o arquivo de credenciais. Para cumprir esta tarefa a aplicação carrega o arquivo de credenciais, no caso `/etc/shadow`, e o envia como parâmetro na função `ecall_seal_data` que realiza a selagem dos dados e chama a `ocall_save_data` para gravar os arquivo selado no disco. Foi utilizada a função `sgx_seal_data()` da biblioteca `sgx_tseal.h` do SDK SGX, que vincula a chave de selagem ao registrador `MRSIGNER` de modo que qualquer enclave do mesmo autor na mesma plataforma que realizou a selagem é capaz de abrir o arquivo de credenciais. As funções `ecall_seal_data` e `ocall_save_data` também foram declaradas no arquivo `Enclave.edl`, de modo que o enclave que selou os dados será o mesmo que irá deselar para realizar a consulta. Alternativamente poderia ter sido usada a função `sgx_seal_data_ex()` para escolher o `MRENCLAVE` para vinculação da chave de selagem desejada, conforme explicado na Seção 2.3.5. Modificações nas funções que criam usuários e credenciais no Linux para armazenar automaticamente as credenciais em um arquivo selado, e não mais no arquivo `/etc/shadow`, é uma possibilidade para trabalhos futuros.

5.2.4 Descrição dos experimentos realizados

Cada cenário foi testado procedendo tentativas de autenticação através de três aplicações distintas: Linux *sudo*, Linux *login* e uma aplicação customizada, desenvolvida especificamente para autenticar um usuário utilizando a biblioteca do PAM. Os testes foram repetidos variando-se o tamanho arquivo de credenciais entre 5KB, 100KB e 500KB. Os arquivos de credenciais tinham respectivamente 100, 1.600 e 8.300 entradas de usuários quando as senhas estavam armazenadas na forma de *hash* ou respectivamente 200, 4.000 e 20.000 entradas de usuário quando as senhas estavam armazenadas em claro. As senhas na forma de *hash* estavam armazenadas no arquivo `/etc/shadow` processadas com o algoritmo MD5.

Para o cenário 2 o uso de tamanhos distintos de arquivo de credenciais não faz sentido, já que o mesmo é destinado apenas à aferição do atraso devido ao tempo de inicialização de um enclave que tem tamanho constante. Conforme Anexo A, o tamanho da *heap* e da *stack* do enclave deve, ser definidos durante o seu desenvolvimento. Para trabalhar com arquivos de credenciais de até 500KB uma *stack* de 4KB e uma *heap* de 1024KB foram configuradas no arquivo `Enclave.config.xml`, para todos os testes descritos no parágrafo acima.

A memória do enclave é alocada durante seu processo de inicialização. Assim é esperado que o tempo de inicialização do enclave seja dependente do tamanho da *stack* e da *heap* configuradas. Dessa forma, para verificar a influência do tamanho da memória do enclave no seu tempo de inicialização foram medidos também os tempos de inicialização nas seguintes condições:

- Mantendo-se o tamanho da *heap* constante em 16KB e variando-se o tamanho da *stack* entre 64 e 2048KB.
- Mantendo-se o tamanho da *stack* constante em 16KB e variando-se o tamanho da *heap* entre 64 e 2048KB.

5.2.5 Resultados obtidos

A Figura 5.4 mostra a representação gráfica dos resultados da medição do tempo de execução do módulo em cada um dos cenários considerados. O eixo das ordenadas representa o tempo em milissegundos enquanto o eixo das abscissas representa cada um dos cenários testados.

Em cada ponto do eixo das abscissas estão apontados os valores médios, em milissegundos, auferidos em cada cenário. Os valores mínimo, médio, máximo e desvio padrão obtidos constam na Tabela 5.1.

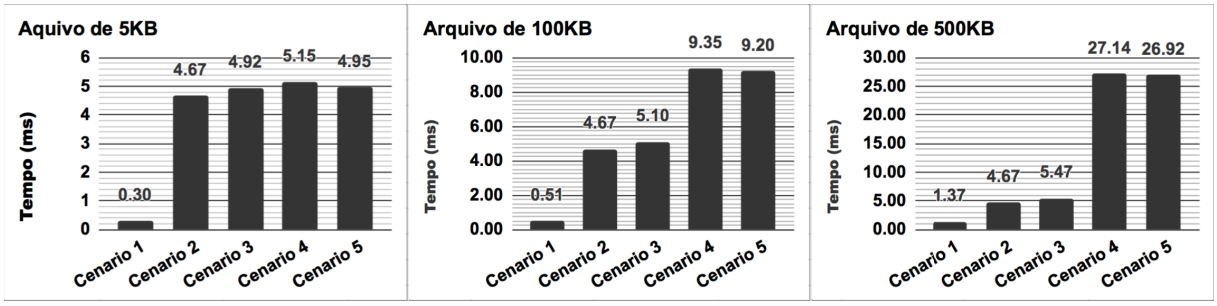


Figura 5.4: Resultados obtidos para os cenários 1, 2, 3, 4 e 5 relacionados na Tabela 5.1.

Primeiramente foram realizados os testes segundo o cenário 1, no módulo original modificado para realizar a aferição do tempo de resposta. Quando foi utilizado um arquivo de credenciais de 5KB o tempo médio de resposta foi de 0,3 milissegundos (ms). Aumentando-se o tamanho do arquivo para 100KB o tempo de resposta aumentou para 0,5ms e, por fim, para 500KB o resultado obtido foi de 1,4ms.

Os testes do cenário 2 mostraram que o tempo de inicialização do enclave com 4KB de *stack* e 1MB de *heap* e gera um atraso médio 4,7ms. Este processo ocorre uma vez em cada instância de autenticação e representa uma parcela de 17,3% do tempo de resposta do módulo de autenticação com SGX considerando o cenário 5 e o arquivo de credenciais de 500KB. O gráfico da Figura 5.5 mostra a influência do tamanho da *stack* e da *heap* no tempo de inicialização do enclave. É notável que este tempo sofre mais influência do aumento do tamanho da *stack* do que do aumento do tamanho da *heap*. Esta informação deve ser considerada pelo programador quando desempenho da aplicação for um aspecto relevante.

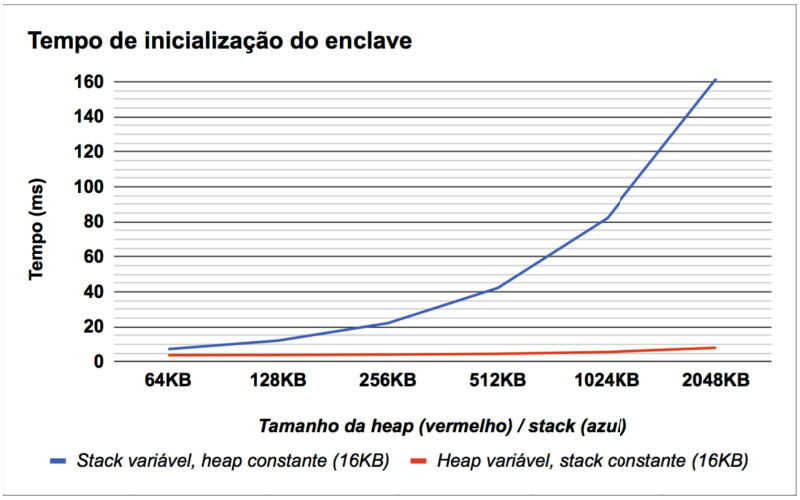


Figura 5.5: Dependência do tempo de inicialização do enclave em relação ao tamanho da *heap* e *stack* configurados.

O cenário 3, "sem selar", realiza a autenticação dentro do enclave, no entanto com o arquivo de credenciais em claro. Desta forma, pode-se estimar o custo do enclave trabalhar com memória de execução cifrada através dos resultados obtidos neste cenário, descontados aqueles obtidos no cenário anterior. O tempo de resposta obtido foi 4,9ms para o arquivo de 5KB,

5,1ms para o arquivo de 100KB e de 5,5ms para 500KB. Descontando-se os valores médios obtidos nos cenários anteriores é obtido um custo médio de 0,25ms, 0,44ms e 0,80ms quando trabalhando com arquivos de 5KB, 100KB e 500KB respectivamente. Esta sobrecarga está associada principalmente ao custo da passagem dos dados para dentro do enclave, da busca do usuário e do cômputo do *hash* MD5 em memória protegida.

Tabela 5.1: Resultados obtidos para os cenários testados.

Tempo (ms)					
		Mín	$\bar{X}$	Máx	σ
5KB	Cenário 1	0.29	0.30	0.30	0.01
	Cenário 2	4.61	4.67	4.72	0.04
	Cenário 3	4.88	4.92	4.96	0.03
	Cenário 4	5.03	5.15	5.22	0.08
	Cenário 5	4.90	4.95	5.01	0.06
100KB	Cenário 1	0.50	0.51	0.51	0.01
	Cenário 2	4.61	4.67	4.72	0.04
	Cenário 3	5.04	5.10	5.14	0.04
	Cenário 4	9.28	9.35	9.42	0.06
	Cenário 5	9.09	9.20	9.26	0.06
500KB	Cenário 1	1.36	1.37	1.38	0.01
	Cenário 2	4.61	4.67	4.72	0.04
	Cenário 3	5.40	5.47	5.56	0.07
	Cenário 4	27.05	27.14	27.22	0.07
	Cenário 5	26.81	26.92	27.13	0.13

No quarto cenário, "unix_sgx", tem-se o módulo em plenas condições de funcionamento, com duas camadas de segurança para as senhas: a selagem do arquivo e a *hash* MD5 individual das senhas. Nessas condições o tempo medido foi de 5,2ms, 9,3ms e 27,1ms quando trabalhando com arquivos de 5KB, 100KB e 500 KB respectivamente. A única mudança realizada do cenário 3 para o 4 é a selagem do arquivo de credenciais. Assim, a diferença do tempo obtido entre esses cenários representa a sobrecarga adicionada ao sistema por esse processo. Isto corresponde à um montante de 0,2ms, 4,2ms e 21,7ms para os arquivos de 5KB, 100KB e 500KB respectivamente.

Finalmente, o quinto cenário, "sem hash", é equivalente ao cenário anterior, porém as entradas individuais de usuário encontram-se em claro dentro do arquivo selado. Com esta medida houve uma redução média de 0.19ms no tempo levado para a autenticação em relação ao cenário anterior. Ressalta-se que ao trocar o arquivo de credenciais com as senhas em *hash* pelo arquivo com as senhas em claro, a quantidade de entradas de usuário para arquivos com o mesmo tamanho quase triplica.

A Figura 5.6 mostra, separadamente, os resultados dos testes obtidos em cada uma das aplicações testadas. Analisando-se os três gráficos para cada cenário percebe-se resultados muito similares para as três aplicações testadas. Este resultado era esperado, tendo em vista que aplicações que utilizam o PAM delegam a ele a responsabilidade de autenticar os usuários, desta forma não exercendo influência no resultado ou no desempenho da autenticação.

Verificou-se que as maiores sobrecargas de desempenho foram ocasionadas pelos processos de inicialização do enclave e de deselagem do arquivo protegido. Conforme verificado na Figura 5.5 o tempo de inicialização do enclave está relacionado ao seu tamanho pré-configurado enquanto a Figura 5.4 mostra que o tempo de deselagem depende do tamanho do arquivo protegido. Por outro lado, para esta aplicação, a sobrecarga de se trabalhar com a memória de execução

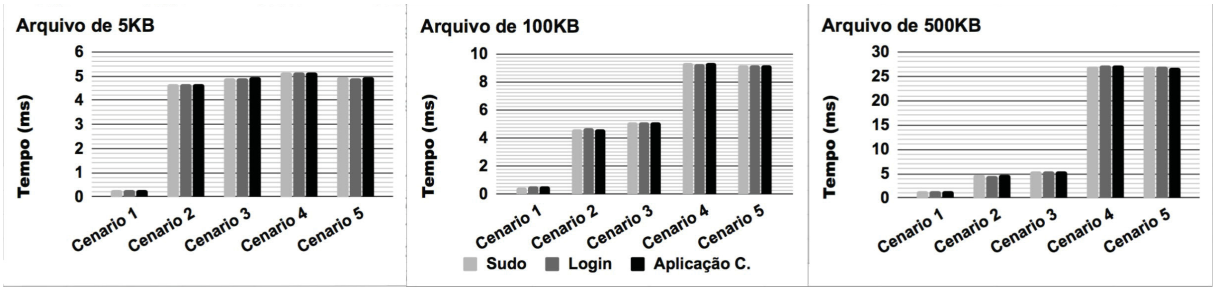


Figura 5.6: Resultados parciais dos Cenários 1, 2 3, 4 e 5 divididos em cada uma das três aplicações testadas: *sudo*, *login* e aplicação customizada.

cifrada exerceu pouca influência no desempenho, mostrando boa eficiência do *Memory Encryption Engine*.

5.2.6 Análise da escalabilidade da solução

Aplicações com quantidade de usuários muito elevada, como servidores de *e-mail* ou serviços de autenticação na nuvem, poderão demandar esforços extras. No modelo implementado para cada requisição de autenticação uma nova instância do módulo é carregado e, portanto, uma nova instância do enclave é carregado. Apesar de não existir impedimento de carregar diversos enclaves na simultaneamente EPC, sua memória é limitada à 128MB. Esta limitação implica em uma limitação ao tamanho do arquivo de credenciais a ser carregado dentro do enclave.

Para comportar a manipulação de arquivos de credenciais de 500KB foi necessário configurar o enclave com memória aproximada de 1,1MB. Assim, o tamanho do arquivo de credenciais tem influência direta na quantidade de enclaves que é possível carregar simultaneamente na EPC. A falta de memória na EPC para carregar uma nova instância do enclave causará um erro de inicialização de enclave, o que resultará em uma falha de autenticação. Consequentemente, ambientes que processam centenas de requisições de autenticação simultâneas devem tratar o caso de congestionamento para que novas requisições de autenticação possam ser enfileiradas e aguardar até que haja memória suficiente para carregar uma nova instância do enclave na EPC.

Os tempos de resposta para as requisições de autenticação dependerão da capacidade da CPU de processá-las simultaneamente e do tamanho do arquivo de credenciais, conforme mostrado na Seção 5.2.5. Se o fluxo de requisições for muito grande algumas medidas para melhoria do desempenho poderão ser necessárias. O uso da CPU para determinar a chave utilizada para a selagem/de-selagem dos dados, conforme Seção 3.6, causa restrições ao 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.

Dependendo do tamanho do arquivo de credenciais pode ser necessário dividi-lo em arquivos menores. Desde que se mantenha o controle do arquivo em que se encontra cada usuário, esta medida também poderá contribuir para a melhoria do desempenho, ou ainda, possibilitar o uso de múltiplos processadores, onde cada processador ficaria responsável pela autenticação de uma fração dos usuários. Outra medida poderia ser implementar o uso de banco de dados para o armazenamento das credenciais. Para manter o padrão de segurança o banco teria que ser armazenado selado e, consequentemente, o mecanismo de consulta teria que ser implementado dentro do enclave.

Por fim, foi verificado que o tempo de inicialização do enclave constitui parcela considerável no tempo de execução do módulo. Além disso, conforme visto na Figura 5.5, esse

tempo aumenta com o tamanho do enclave. Assim, outra maneira para reduzir o tempo de autenticação seria evitar a criação de um novo enclave a cada requisição. Isto pode ser feito mantendo um ou mais enclaves instanciados no servidor que recebe as requisições de autenticação. Desta forma, a cada requisição bastaria realizar uma ECALL para proceder a autenticação.

5.3 Considerações finais

Este Capítulo apresentou os detalhes do módulo implementado, baseado no módulo de autenticação padrão do Linux, bem como do ambiente de testes. Além disso foram mensurados os impactos de desempenho causados pelos processos típicos do SGX tais como a inicialização do enclave, o processamento de dados na região encriptada da memória de execução (EPC) e a selagem/de-selagem de dados.

É conhecimento comum que segurança e desempenho caminham em direções opostas. No entanto, quando componentes sensíveis de aplicações demandam um maior nível de segurança, é desejável que as técnicas utilizadas ocasionem o menor impacto possível na qualidade da experiência do usuário.

Nielsen afirma que, quando se trata da experiência do usuário durante a interação humano-computador, tempos de resposta inferiores a 100ms são percebidos pelo usuário como instantâneos [Nielsen, 1993]. Um estudo mais recente, [Kohrs et al., 2016], avalia os efeitos na atividade cerebral do usuário devido aos atrasos na interação humano-computador. Os autores chegaram ao resultado de que o limiar da capacidade de percepção de um atraso é, na média, 327,2ms ± 89,7, e que assim, pode-se considerar, com boa margem de segurança, que atrasos inferiores a 200ms não são percebidos pelo usuário.

Os resultados mostrados neste Capítulo apontam que no pior caso avaliado, utilizando um arquivo de credenciais de 500KB e 20.000 entradas de usuários, a autenticação utilizando o arquivo de credenciais selado demanda, em média, 26,9ms. Apesar de corresponder à, aproximadamente, dezenove vezes o tempo demandado pela autenticação convencional do Linux este atraso não é suficiente para ocasionar nenhum impacto na experiência do usuário ou, ainda, para ser percebido. Esta afirmação continua verdadeira mesmo se for considerada a amostra com resposta mais lenta obtida no cenário mais lento (27,2ms). Desta forma pode-se concluir que a solução implementada além dos benefícios de segurança discutidos no Capítulo 4, também apresenta desempenho adequado para utilização em cenário real.

Capítulo 6

Conclusão

Em um contexto onde a segurança de sistemas computacionais tem se tornado fator cada vez mais crítico, sistemas de autenticação e a proteção das credenciais de autenticação têm um papel fundamental. A arquitetura Intel SGX serve de infraestrutura para o desenvolvimento de aplicações e armazenamento de dados com elevado nível de segurança. O presente trabalho utilizou esta tecnologia de segurança baseada em *hardware* para melhorar a segurança do arquivo de armazenamento de credenciais no sistema operacional Linux.

6.1 Contribuições e resultados obtidos

Um módulo baseado na tecnologia SGX foi proposto e um protótipo deste módulo foi implementado e avaliado. A seguir estão relacionadas as principais contribuições deste trabalho.

- Um compilado resultante de pesquisa sobre os principais mecanismos de segurança computacional baseados em *hardware* da atualidade.
- Um estudo sobre o funcionamento da tecnologia SGX, bem como um resumo do estado-da-arte e das principais vulnerabilidades conhecidas do SGX.
- Proposta e implementação de uma prova de conceito de um módulo de autenticação utilizando o SGX para a proteção do arquivo de credenciais para a plataforma PAM Linux.
- Análise de segurança dos aspectos críticos da arquitetura SGX aplicada à sistemas de autenticação, em especial no tocante a criptografia da memória de execução e à selagem de dados.
- Testes de desempenho da solução implementada. Também foram realizados testes complementares para proporcionar um maior conhecimento acerca do impacto das técnicas ou ações do SGX necessárias para sua utilização, como por exemplo, o custo de se inicializar um enclave, de realizar processamento de dados dentro do enclave e, ainda, de abrir dados selados.

Os resultados obtidos mostram que, apesar da adição de uma tecnologia de segurança causar impacto no desempenho, o que já era esperado, o impacto causado pelo SGX é pequeno o suficiente para ser imperceptível para os usuários. Além disso, nenhum aspecto da usabilidade do sistema de autenticação foi alterado. Desta forma, mostrou-se que a tecnologia SGX é perfeitamente adequada para ser usada com a finalidade de robustecer, no ponto de vista da segurança, dados e sistemas de autenticação.

6.2 Trabalhos futuros

A arquitetura SGX traz novas oportunidades para a melhoria da segurança de softwares e dados para diversas aplicações. No entanto, ela exige que o desenvolvedor seja capaz de identificar pontualmente todos os elementos sensíveis que necessitam de proteção, o que demanda uma nova forma de pensar e programar a aplicação. Neste trabalho, o arquivo de credenciais do Linux foi identificado como um elemento digno de receber uma proteção especial. Por outro lado, outros aspectos e/ou situações de autenticação também podem se beneficiar do emprego das possibilidades trazidas pelo SGX. As seções a seguir apresentam duas possibilidades de objetos para trabalhos futuros.

6.2.1 Extensão da solução para o restante da Plataforma PAM

Para que o módulo implementado possa ser utilizado em ambiente de produção sem ocasionar uma sobrecarga para o administrador do sistema, algumas modificações no sistema operacional GNU/Linux e na plataforma PAM devem ser realizadas. A começar pela alteração da rotina que gera o arquivo `/etc/shadow` para que as senhas não sejam mais armazenadas ali, e sim em um arquivo selado. É importante frisar que o enclave utilizado para gerar este arquivo deve ser o mesmo dos demais módulos que o manipularão para realizar consultas e alterações. Assim, os módulos para alteração de senhas (serviço *passwd*) também deverão ser alterados. Alternativamente, pode-se vincular a chave de selagem ao *hash* da chave pública do autor para possibilitar a criação de enclaves específicos para cada função (autenticação, alteração de senhas, criação de usuários, etc) mantendo a capacidade de trabalhar com os mesmos arquivos selados.

Conforme apresentado na Seção 2.2.2, a plataforma PAM permite a utilização de diversas formas de autenticação, sendo adaptável às necessidades particulares de cada aplicação. Esta adaptabilidade é possível através da configuração e/ou adição dos módulos a serem utilizados. No entanto, um atacante que consiga, através de algum método qualquer, privilégios de administrador do sistema poderá alterar o arquivo de configuração do PAM, substituir os módulos de autenticação ou ainda alterar as bibliotecas do PAM. Mesmo que, ainda assim, o atacante não conseguiria acesso ao arquivo de credenciais em claro (nem mesmo ao *hash* das senhas), ele poderia obter acesso não autorizado à aplicação.

Uma alternativa para dificultar este tipo de ataque seria modificações no linux e na plataforma PAM para utilizar o SGX para garantia da execução segura dos seus elementos críticos. Um resumo criptográfico das partes críticas do PAM, como seu arquivo de configuração poderiam ser armazenados selados. Assim, antes de realizar a validação das credenciais o enclave do PAM deselaria este arquivo e faria a verificação de integridade do sistema de autenticação.

6.2.2 Extensão da solução de autenticação local para autenticação remota

A aplicação do SGX para sistemas de autenticação conforme proposto na seção 4.4 pode ser adaptada para autenticação em servidores remotos. Um modelo que pode ser aplicado neste caso é apresentado na Figura 6.1. O usuário solicita autenticação na parte não confiável da aplicação e fornece sua credencial a ser validada (1). Uma ECALL é realizada passando como parâmetros um resumo criptográfico de elementos críticos para verificação da integridade da aplicação e a credencial fornecida pelo usuário (2). O enclave não poderá selar esses parâmetros para enviar ao servidor de autenticação pois conforme Seção 2.3.5 dados selados só podem ser abertos na plataforma em que foram selados. Então o conceito de *atestação remota*, explicado brevemente na Seção 2.10, pode ser utilizado para que o servidor reconheça o enclave da aplicação

como legítimo e estabelecer um canal seguro para o envio das credenciais de autenticação (3 e 4). Desta forma protege-se o sistema contra ataques de rede tais como o uso de *sniffers* e ataques de homem-no-meio.

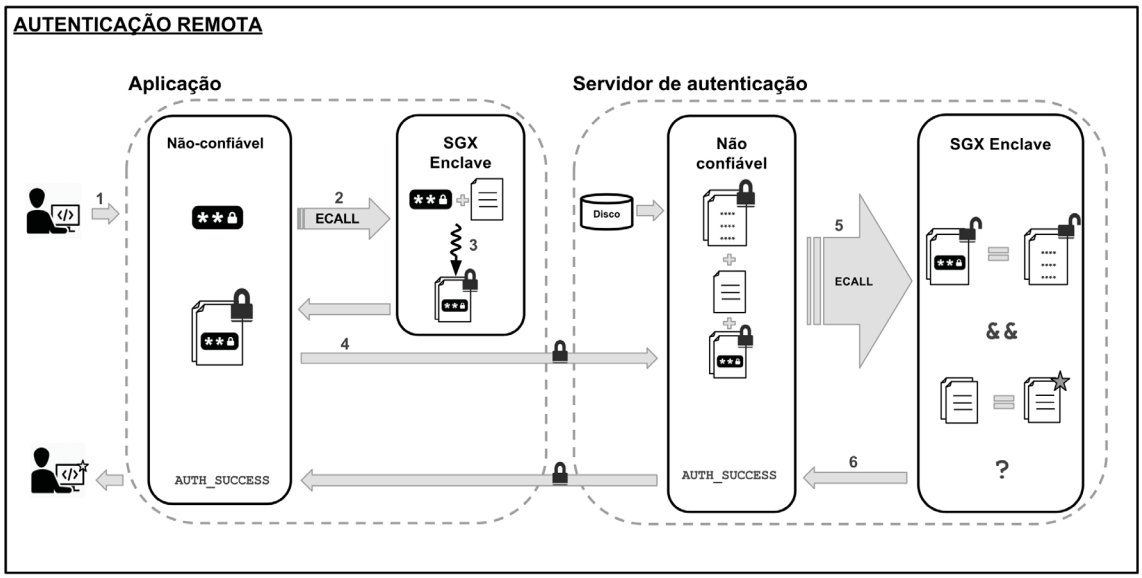


Figura 6.1: Ideia inicial de solução de autenticação remota e proteção do arquivo de senhas com SGX.

Ao receber as credenciais de autenticação, o servidor realizará uma ECALL passando como parâmetros o arquivo recebido, o arquivo de credenciais selado armazenado em disco e um resumo criptográfico para verificação da integridade dos elementos críticos para o funcionamento correto do servidor de autenticação (5). Por sua vez, o enclave realizará a verificação da integridade do sistema e a validação da credencial. Se tudo estiver correto retornará o sucesso da autenticação (6).

Conforme visto na Seção 5.2.5, a inicialização do enclave corresponde à uma parcela significativa do custo da autenticação. Caso o servidor receba grande demanda de autenticações por segundo, o desempenho da solução pode ser melhorado evitando a inicialização de uma nova instância do enclave à cada autenticação realizada. Isto pode ser feito através da manutenção dos enclaves inicializados para a realização das diversas autenticações.

Por fim, se o servidor utilizar um banco de dados relacional para o armazenamento das credenciais a plataforma de consulta ao banco de dados deverá ser adaptada para utilização dentro do enclave. Isto pode ser um problema se o tamanho do banco for grande. Neste caso, a plataforma do banco deverá ser adaptada para realizar a consulta por partes, ou para suportar a consulta distribuída em vários enclaves rodando paralelamente.

Referências

[Amin et al., 2008] Amin, M., Khan, S., Ali, T. e Gul, S. (2008). Trends and directions in trusted computing: Models, architectures and technologies. Em *International Multiconference Of Engineers and Computer Scientist*, volume 1, páginas 19–21.

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

[ARM, 2017] ARM (2017). ARM TrustZone. `https://www.arm.com/products/security-on-arm/trustzone`. Acessado em 15/04/2017.

[Arthur e Challener, 2015] 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.

[Aumasson e Merino, 2016] Aumasson, J. e Merino, L. (2016). SGX secure enclaves in practice–security and crypto review. *Black Hat*.

[Baumann et al., 2015] Baumann, A., Peinado, M. e Hunt, G. (2015). Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)*, 33(3):8.

[Bellovin e Merritt, 1993] Bellovin, S. M. e Merritt, M. (1993). Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. Em *Proceedings of the 1st ACM Conference on Computer and Communications Security*, páginas 244–250. ACM.

[Biryukov et al., 2016] Biryukov, A., Dinu, D. e Khovratovich, D. (2016). Argon2: new generation of memory-hard functions for password hashing and other applications. Em *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, páginas 292–302. IEEE.

[Blocki e Datta, 2016] Blocki, J. e Datta, A. (2016). Cash: A cost asymmetric secure hash algorithm for optimal password protection. Em *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*, páginas 371–386. IEEE.

[Bonneau, 2012] Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. Em *Security and Privacy (SP), 2012 IEEE Symposium on*, páginas 538–552. IEEE.

[Bossuet et al., 2013] 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 Computing Surveys (CSUR)*, 45(4):41.

[Brasser et al., 2017] Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S. e Sadeghi, A.-R. (2017). Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521*.

[Bright, 2011] Bright, P. (2011). Rsa finally comes clean: Securid is compromised. `https://arstechnica.com/security/2011/06/rsa-finally-comes-clean-securid-is-compromised/`. Acessado em 02/04/2017.

[Checkoway e Shacham, 2013] Checkoway, S. e Shacham, H. (2013). *Iago attacks: Why the system call api is a bad untrusted rpc interface*, volume 41. ACM.

[Chen et al., 2008] Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J. e Ports, D. R. (2008). Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. Em *ACM SIGARCH Computer Architecture News*, volume 36, páginas 2–13. ACM.

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

[Cox, 2017] Cox, J. (2017). Hackers: We will remotely wipe iphones unless apple pays ransom. `https://motherboard.vice.com/en_us/article/hackers-we-will-remotely-wipe-iphones-unless-apple-pays-ransom`. Acessado em 01/04/2017.

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

[Doel, 2013] Doel, K. (2013). Scary logins: Worst passwords of 2012 and how to fix them. *SplashData, 2012*.

[FBI, 2005] FBI (2005). Computer crime survay. `http://www.fbi.gov/publications/ccs2005.pdf`. Acessado em 01/02/2017.

[Fiegerman, 2016] Fiegerman, S. (2016). Yahoo says 500 million accounts stolen. `http://money.cnn.com/2016/09/22/technology/yahoo-data-breach/`. Acessado em 02/04/2017.

[Franceschi-Bicchierai, 2016] Franceschi-Bicchierai, L. (2016). Another day, another hack: 117 million linkedin emails and passwords. `https://motherboard.vice.com/en_us/article/another-day-another-hack-117-million-linkedin-emails-and-password`. Acessado em 02/04/2017.

[Geisshirt, 2007] Geisshirt, K. (2007). *Pluggable Authentication Modules*. Packt Publishing Ltd.

[Greene, 2012] Greene, J. (2012). Intel trusted execution technology, white paper. *Online: http://www. intel. com/txt*.

[Gueron, 2016] Gueron, S. (2016). A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016:204.

[Herley et al., 2009] Herley, C., Van Oorschot, P. e Patrick, A. (2009). Passwords: If we're so smart, why are we still using them? *Financial Cryptography and Data Security*, páginas 230–237.

[Hoekstra et al., 2013] Hoekstra, M., Lal, R., Pappachan, P., Phegade, V. e Del Cuvillo, J. (2013). Using innovative instructions to create trustworthy software solutions. Em *HASP@ ISCA*, página 11.

[Intel, 2014a] Intel (2014a). *Intel Software Guard Extensions Programming Reference*.

[Intel, 2014b] Intel (2014b). *Intel Software Guard Extensions Programming Reference*.

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

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

[Intel®, 2017] Intel® (2017). Intel® software guard extensions (intel® SGX) details. `https://software.intel.com/en-us/sgx/details`. Acessado em 01/2/2017.

[Jain et al., 2016] 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*.

[Jeong et al., 2015] Jeong, Y.-S., Park, J. S. e Park, J. H. (2015). An efficient authentication system of smart device using multi factors in mobile cloud service architecture. *International Journal of Communication Systems*, 28(4):659–674.

[Jing et al., 2013] Jing, L., Chunhua, J. e Xia, Y. (2013). Design and implementation of security os based on trustzone. Em *Electronic Measurement & Instruments (ICEMI), 2013 IEEE 11th International Conference on*, volume 2, páginas 1027–1032. IEEE.

[Ju et al., 2014] Ju, S.-h., Seo, H.-s., Lee, S.-j. e Kim, M.-s. (2014). A methodology on password based user authentication for multi-touch environment. *International Information Institute (Tokyo). Information*, 17(2):835.

[Karande et al., 2017] Karande, V., Bauman, E., Lin, Z. e Khan, L. (2017). Sgx-log: Securing system logs with sgx. Em *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, páginas 19–30. ACM.

[Khandelwal, 2016] Khandelwal, S. (2016). Dropbox hacked — more than 68 million account details leaked online. `http://thehackernews.com/2016/08/dropbox-data-breach.html`. Acessado em 02/04/2017.

[Kohrs et al., 2016] Kohrs, C., Angenstein, N. e Brechmann, A. (2016). Delays in human-computer interaction and their effects on brain activity. *PloS one*, 11(1):e0146250.

[Lesjak et al., 2015] Lesjak, C., Hein, D. e Winter, J. (2015). Hardware-security technologies for industrial iot: Trustzone and security controller. Em *Industrial Electronics Society, IECON 2015-41st Annual Conference of the IEEE*, páginas 002589–002595. IEEE.

[Marlinspike, 2017] Marlinspike, M. (2017). Technology preview: Private contact discovery for signal. `https://signal.org/blog/private-contact-discovery/`. Acessado em 16/12/2017.

[McCune et al., 2010] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V. e Perrig, A. (2010). Trustvisor: Efficient tcb reduction and attestation. Em *Security and Privacy (SP), 2010 IEEE Symposium on*, páginas 143–158. IEEE.

[McCune et al., 2008] McCune, J. M., Parno, B. J., Perrig, A., Reiter, M. K. e Isozaki, H. (2008). Flicker: An execution infrastructure for tcb minimization. Em *ACM SIGOPS Operating Systems Review*, volume 42, páginas 315–328. ACM.

[McKeen et al., 2016] 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 Hardware and Architectural Support for Security and Privacy 2016*, página 10. ACM.

[McKeen et al., 2013] 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*, HASP 2013, páginas 10:1–10:1, New York, NY, USA. ACM.

[Moghimi et al., 2017] Moghimi, A., Irazoqui, G. e Eisenbarth, T. (2017). Cachezoom: How SGX amplifies the power of cache attacks. *arXiv preprint arXiv:1703.06986*.

[Murillo-Escobar et al., 2015] Murillo-Escobar, M., Cruz-Hernández, C., Abundiz-Pérez, F. e López-Gutiérrez, R. (2015). A robust embedded biometric authentication system based on fingerprint and chaotic encryption. *Expert Systems with Applications*, 42(21):8198–8211.

[Nielsen, 1993] Nielsen, J. (1993). Response times: the three important limits. *Usability Engineering*.

[Schwarz et al., 2017] Schwarz, M., Weiser, S., Gruss, D., Maurice, C. e Mangard, S. (2017). Malware guard extension: Using SGX to conceal cache attacks. *arXiv preprint arXiv:1702.08719*.

[Shin et al., 2015] Shin, S., Yeh, H. e Kim, K. (2015). An efficient secure authentication scheme with user anonymity for roaming user in ubiquitous networks. *Peer-to-peer Networking and Applications*, 8(4):674–683.

[Slain, 2016] Slain, M. (2016). Announcing our worst passwords of 2016. `https://www.teamsid.com/worst-passwords-2016`. Acessado em 03/04/2017.

[Sun et al., 2015] Sun, H., Sun, K., Wang, Y. e Jing, J. (2015). Trustotp: Transforming smartphones into secure one-time password tokens. Em *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, páginas 976–988. ACM.

[TCG, 2008] TCG (2008). Trusted Platform Module (TPM) summary. `https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf`. Acessado em 02/04/2017.

[TCG, 2016] 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.

[Todorov, 2007] Todorov, D. (2007). *Mechanics of user identification and authentication: Fundamentals of identity management*. CRC Press.

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

[Weafer, 2016] Weafer, V. (2016). Report: 2017 threats prediction. Relatório técnico, McAfee Labs. Acessado em 29/03/2017.

[Weiser e Werner, 2017] Weiser, S. e Werner, M. (2017). Sgxio: Generic trusted i/o path for intel sgx. Em *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, páginas 261–268. ACM.

[Zheng et al., 2016] Zheng, X., Yang, L., Ma, J., Shi, G. e Meng, D. (2016). Trustpay: Trusted mobile payment on security enhanced arm trustzone platforms. Em *Computers and Communication (ISCC), 2016 IEEE Symposium on*, páginas 456–462. IEEE.

Apêndice A

Instruções e estruturas de dados SGX

O objetivo deste apêndice é apresentar de forma sumária as instruções e estruturas de dados SGX para proporcionar um maior entendimento do funcionamento da sua arquitetura. Maiores detalhes acerca do conteúdo ora apresentado podem ser encontrados no manual de referência [Intel, 2014b]. As instruções SGX não foram utilizadas diretamente no desenvolvimento deste trabalho, mas sim indiretamente através das bibliotecas confiáveis e não-confiáveis fornecidas junto ao SDK da Intel.

A.1 Instruções do processador SGX.

As instruções SGX para o enclave são organizadas sob dois mnemônicos de instrução: ENCLS (*ring 0*) e ENCLU (*ring 3*). O processador SGX, em seu primeiro lançamento (SGX1), implementa 18 novas instruções das quais 13 são instruções do supervisor SGX (Tabela A.1) e 5 de usuário (Tabela A.2). A expressão longa de instruções SGX assumem a forma ENCLx[LEAF_MNEMONIC] onde x pode ser ‘S’ or ‘U’. Cada função usa o registrador EAX para especificar o seu índice. Por sua vez, cada índice da função é associado de forma única com um "LEAF_MNEMONIC"específico. Registradores adicionais podem ser implicitamente necessários como entrada de parâmetros.

Tabela A.1: Instruções do supervisor SGX1

Instrução	Descrição
ENCLS[EADD]	Adiciona uma página à EPC
ENCLS[EBLOCK]	Bloqueia uma página da EPC
ENCLS[ECREATE]	Cria um enclave
ENCLS[EDBGRD]	Lê dados do *debugger*
ENCLS[EDBGWR]	Escreve dados no *debugger*
ENCLS[EEXTEND]	Faz a extensão da medição criptográfica de um *chunk* de 256 *bytes* da página da EPC
ENCLS[EINIT]	Inicializa o enclave
ENCLS[ELDB]	Carrega uma página da EPC como bloqueada
ENCLS[ELDU]	Carrega uma página da EPC como desbloqueada
ENCLS[EPA]	Adiciona um *version array*
ENCLS[EREMOVE]	Remove uma página da EPC
ENCLS[ETRACK]	Ativa checagens EBLOCK
ENCLS[EWB]	Desfaz/invalida uma página da EPC

Tabela A.2: Instruções de usuário SGX1

Instrução	Descrição
ENCLU[EENTER]	Entra em um enclave
ENCLU[EEXIT]	Sai de um enclave
ENCLU[EGETKEY]	Cria uma chave criptográfica
ENCLU[EREPORT]	Cria um *report*
ENCLU[ERESUME]	Reentra em um enclave

As extensões SGX1 permitem uma aplicação instanciar um enclave, no entanto, não permite que páginas sejam adicionadas ao enclave depois de sua inicialização e preparação para execução. Logo toda memória que o enclave irá utilizar deve ser alocada antes da sua inicialização, assim o desenvolvedor deve definir um tamanho estático para a *heap* e *stack* do enclave [McKeen et al., 2016]. Novas plataformas lançadas pela Intel contarão com extensões adicionais SGX2 para realização do gerenciamento dinâmico de memória, o que permitirá uma flexibilidade adicional no gerenciamento dos recursos e tarefas do enclave em tempo de execução. Para isso serão introduzidas 6 novas instruções. Estas instruções estão relacionadas nas Tabelas A.3 e A.4.

Tabela A.3: Instruções do supervisor SGX2

Instrução	Descrição
ENCLS[EAUG]	Aloca uma página à um enclave já existente. A página é adicionada com estado "pendente" e não será usada até o enclave executar um EACCEPT para esta página.
ENCLS[EMODPR]	Restringe as permissões de uma página existente na EPC
ENCLS[EMODT]	Cria um *Thread Control Structure* (TCS) para a página. Modifica o tipo de uma página existente na EPC

Tabela A.4: Instruções de usuário SGX2

Instrução	Descrição
ENCLU[EACCEPT]	Aceita uma página ou modificações no tipo de uma página em um enclave em execução
ENCLU[EMODPE]	Extende as permissões de acesso de uma página existente na EPC
ENCLU[EACCEPTCOPY]	Copia uma página existente na EPC para uma página pendente e aceita a página para o enclave em execução

A.2 Estruturas de dados SGX

A operação do enclave é gerenciada através de uma coleção de estruturas de dados das quais algumas contém, ainda, sub-estruturas. As estruturas de alto nível relacionadas à parâmetros usados na configuração e manutenção dos enclaves pelas instruções SGX ou por eventos AEX (*Asynchronous Enclave Exit*) são as seguintes:

1. **SGX Enclave Control Structure (SECS)**

   Representa um enclave. Contém informações como o tamanho do enclave, a base do seu endereço linear, identificador do enclave, informações sobre a assinatura etc.

2. **Thread Control Structure (TCS)**

   Cada *thread* executando em um enclave é associada à um TCS. Contém atributos como *flags* de execução, o ponto de entrada e um ponteiro para o SSA.

3. **State Save Area (SSA)**

   Na ocorrência de uma saída assíncrona do enclave (AEX) seu estado é salvo no SSA apontado pelo TCS da *thread*.

4. **Page Information (PAGEINFO)**

   É uma estrutura de dados arquitetural utilizada como parâmetro pelas instruções de gerenciamento da memória do enclave (EPC). Contém dados como o endereço linear, endereço efetivo, SECS e SECINFO.

5. **Security Information (SECINFO)**

   Contém os metadados sobre as páginas do enclave. *Flags* de permissão de leitura, escrita, tipo da página, etc.

6. **Paging Crypto MetaData (PCMD)**

   A estrutura PCMD é utilizada para armazenar cripto metadados relativos às páginas. Em conjunto com a PAGEINFO, a PCMD fornece as informações necessárias ao processador para verificar, decriptar e recarregar páginas da EPC.

7. **Enclave Signature Structure (SIGSTRUCT)**

   Esta estrutura contém informações sobre a assinatura do enclave, o que inclui um hash SHA256 do enclave além de informações como o ISVSVN, ISVPRODID e a assinatura sobre o cabeçalho e o corpo.

8. **EINT Token Structure (EINITTOKEN)**

   É utilizado pela instrução ENCLS[EINIT] para verificar se o enclave está pronto para o uso. Contém o MRENCLAVE, MRSIGNER além de outros atributos do enclave.

9. **Report (REPORT)**

   Esta estrutura é retornada da instrução ENCLU[EREPORT] contendo informações como o hash do enclave, informações do assinante, atributos do enclave e um conjunto de dados a serem utilizados na comunicação entre o enclave e o enclave de destino.

10. **Report Target Info (TARGETINFO)**

    Esta estrutura é um parâmetro de entrada para a instrução ENCLU[EREPORT]. O endereço do TARGETINFO é especificado no registrador RBX. É usada para identificar o enclave que vai ser capaz de verificar criptograficamente a estrutura REPORT retornada pela instrução ENCLU[EREPORT].

11. **Key Request (KEYREQUEST)**

    Esta estrutura é um parâmetro de entrada para a instrução ENCLU[EGETKEY]. É passada através de um endereço efetivo no registrador RBX e é usada para selecionar a chave apropriada bem como os parâmetros adicionais necessários para a geração desta chave.

12. **Version Array (VA)**

    O *Version Array* é uma página especial da EPC usada para armazenar seguramente as versões das páginas retiradas da EPC. Ele contém 512 slots de um número de versão de 8 bytes. Quando uma página é retirada da EPC um *slot* é escolhido para receber um número de versão único desta página. Caso essa página seja recarrega com sucesso esse *slot* é limpo.

13. **Enclave Page Cache Map (EPCM)**

    O EPCM é uma estrutura usada pelo processador para manter o controle dos conteúdos da EPC. Ele mantém uma entrada para cada página que está carregada na EPC e não é acessível por *software*.