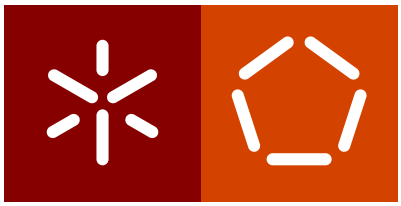
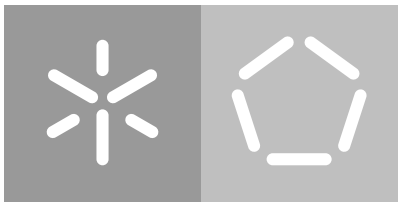


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tânia da Conceição Araújo Esteves

Sistemas de Armazenamento Configuráveis e Seguros

Novembro 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Tânia da Conceição Araújo Esteves

Sistemas de Armazenamento Configuráveis e Seguros

Dissertação de mestrado
Mestrado Integrado em Engenharia Informática

Dissertação orientada por
João Tiago Medeiros Paulo
José Orlando Pereira

Novembro 2018

AGRADECIMENTOS

Gostaria de expressar a minha grande gratidão para com as pessoas sem as quais não seria possível a realização desta dissertação.

Ao meu orientador, Doutor João Paulo, por me ter guiado e motivado durante todo o processo e pela disponibilidade prestada para ajudar sempre que necessário. Ao meu co-orientador, Professor José Orlando Pereira, pela ajuda prestada, principalmente nas decisões mais complexas da implementação.

Ao Ricardo Macedo, por toda a ajuda prestada quer na dissertação quer na escrita e realização do artigo científico. Ao Bernardo Portela e ao Danny Harnik (da IBM), pela colaboração na realização do artigo. Em especial ao Bernardo pela ajuda na elaboração da análise do modelo de segurança do protótipo implementado.

Ao Rogério Pontes e ao André Marinho, pela ajuda em certos tópicos, nomeadamente, em assuntos realizados com a tecnologia utilizada. Aos meus colegas do Grupo de Sistemas Distribuídos do HASLab, pela disponibilidade em ajudar e pelo bom ambiente de trabalho.

Aos meus amigos, pelo incentivo e apoio prestado, e por me aturarem sempre que falava sobre este trabalho. E finalmente, à minha família, por todo o apoio e incentivo constante durante todo o percurso académico.

ABSTRACT

Digital data has been increasing exponentially over the last few years. More and more users store their files in cloud services such as *Dropbox* or *Google Drive* and use devices connected to the Internet of Things (IoT), which suggests that by 2025 the amount of worlwide digital information being stored will reach a new mark of 163 zettabytes.

This data surge raises unprecedented challenges for storage systems which are related to both the efficiency of data management, as well as, the security and privacy concerns raised by third-party storage services. On one hand, cloud services resort to data reduction techniques (e.g., deduplication, compression) to optimize storage costs. On the other hand, in order to protect sensitive personal information, it is common practice for users to encrypt their data before sending it to their cloud services. However, using traditional encryption schemes makes it impossible to use content-oriented storage techniques, since encrypted data does not retain the properties of the original content.

This dissertation presents TrustFS, an open-source secure, programmable and modular stackable file system framework for implementing content-aware storage functionalities that will run in a secure trusted hardware environment (Intel SGX). The prototype leverages the security offered by SGX and the flexibility and modularity of the SafeFS system to allow different content-oriented storage functionalities to be offered safely and with a minimal re-implementation of existing systems. To demonstrate its feasibility, this dissertation presents a secure deduplication prototype that introduces a novel secure epoch-based SGX deduplication scheme which provides different trade-offs in terms of performance, security and space-savings when compared to state-of-the-art secure deduplication solutions. The evaluation experiments show that it is possible to develop a secure stackable file system with secure SGX-enabled storage functionalities while introducing negligible storage performance overhead, below 2% for most workloads.

RESUMO

A informação digital tem aumentado exponencialmente com o passar dos tempos. Cada vez mais utilizadores guardam os seus ficheiros pessoais em serviços de nuvem, como o *Dropbox* ou *Google Drive*, e recorrem a dispositivos ligados à Internet das Coisas (IoT), o que leva a crer que em 2025 serão armazenados, à escala mundial, cerca de 163 zettabytes de dados.

Este crescimento de dados e da informação pessoal contida nos mesmos levanta diversos desafios para o armazenamento na nuvem em termos de gestão e custos de armazenamento, confidencialidade e segurança oferecidas por estes serviços. De forma a otimizar os custos de armazenamento, os serviços de nuvem recorrem a técnicas de redução de espaço (deduplicação, compressão) do conteúdo armazenado. Por outro lado, de forma a proteger informação sensível pessoal, é comum os utilizadores cifrarem os seus dados antes de os enviarem para os respetivos serviços na nuvem. No entanto, o recurso à criptografia tradicional, impossibilita a utilização de técnicas de armazenamento orientadas ao conteúdo, uma vez que os dados cifrados não mantêm as propriedades do conteúdo original.

Esta dissertação apresenta a plataforma TrustFS, uma plataforma *open-source* segura, programável e modular para sistemas de armazenamento, que permite implementar funcionalidades de armazenamento orientadas ao conteúdo que executam em ambientes de execução confiáveis oferecidos pelo Intel SGX. Em mais detalhe, o protótipo final tira partido da segurança oferecida pelo Software Guard Extensions (SGX) e da flexibilidade e modularidade do sistema SafeFS de modo a permitir que diferentes funcionalidades de armazenamento orientadas ao conteúdo possam ser oferecidas de forma segura e com um esforço reduzido de reimplementação dos atuais sistemas. Para demonstrar a sua viabilidade, o protótipo implementa um novo esquema seguro de deduplicação por épocas dotado de SGX que possibilita novos compromissos em termos de segurança quando comparado com a solução do estado da arte de deduplicação. Os resultados obtidos com o protótipo mostram que é possível implementar um sistema de armazenamento com técnicas orientadas ao conteúdo com esta tecnologia com um impacto mínimo no desempenho (inferior a 2% na maioria dos cenários).

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EXCERTOS DE CÓDIGO

1

INTRODUÇÃO

Nos dias que correm tem-se verificado uma adesão crescente a serviços de nuvem, a qual é acompanhada por um crescimento exponencial dos dados armazenados em serviços remotos. Existem vários serviços de armazenamento na nuvem conhecidos, como *Amazon S3*, *Dropbox*, *Google Drive* [3, 24, 29], que oferecem soluções eficientes e competitivas, disponibilizando ao cliente uma combinação complexa de funcionalidades de armazenamento que tiram partido de mecanismos de *caching*, efetuam deduplicação, compressão, replicação, cifram os dados, entre outras.

Um estudo recentemente realizado pela *Intel* [35] a mais de 1400 profissionais revelou que cerca de 93% das empresas atualmente recorrem a serviços de nuvem, sendo que 74% destas armazenam uma quantidade significativa de dados confidenciais na nuvem pública. Segundo a *Clutch* [16], em 2017, as empresas estavam dispostas a investir ainda mais dinheiro em serviços na nuvem. Num estudo similar, a *International Data Corporation* prevê que, até 2025, a quantidade de dados globais crescerá para 163 zettabytes, dez vezes mais que os 16.1ZB de dados gerados em 2016 [59]. Estes serviços prestados pela nuvem tornam-se bastante apelativos, principalmente para as organizações, pois permitem reduzir os seus custos ao oferecerem a possibilidade de não serem obrigados a manter a sua própria infraestrutura de armazenamento.

No entanto, a migração de dados para a nuvem e o crescimento exponencial dos mesmos impõe desafios relativamente à segurança, eficiência e confidencialidade dos sistemas de armazenamento atuais. Ao colocar os seus dados na nuvem, o utilizador perde o controlo sobre os mesmos e vê-se obrigado a confiar neste serviço externo, ficando sujeito a ataques por utilizadores e administradores de sistema maliciosos. Exemplo disso, foi o caso de março de 2016, em que um funcionário da empresa *Ofcom* utilizou indevidamente dados sensíveis da empresa [32]. Com a intenção de controlar e proteger os dados pessoais de um utilizador, surgiu recentemente o Regulamento Geral sobre a Proteção de Dados (RGPD/GDPR) [26] que visa regulamentar o tratamento dos dados pessoais e a livre circulação desses dados. Uma empresa que opere na Europa é obrigada a declarar explicitamente qualquer recolha de dados que efetue, qual o enquadramento jurídico que permite essa recolha, a finalidade do processamento desses dados e o tempo que os manterão ar-

mazenados. O cliente tem direito de exigir uma cópia dos dados recolhidos ou até mesmo a eliminação destes em determinadas circunstâncias.

Neste sentido, e de forma a fornecer confidencialidade e integridade dos dados, são adotadas medidas de segurança baseadas na encriptação dos mesmos. Tipicamente são utilizadas cifras simétricas aleatórias que geram criptogramas diferentes para duas mensagens iguais. Neste caso, é necessária uma chave para cifrar os dados, a qual tem de ser gerida pelo cliente e não deve ser partilhada com o sistema de armazenamento onde os dados são armazenados, pois tal compromete a segurança do sistema.

1.1 PROBLEMA E OBJETIVOS

A utilização de esquemas criptográficos para aumentar a segurança dos dados introduz algumas implicações na gestão dos sistemas de armazenamento. De forma a otimizar os serviços oferecidos pela nuvem, são utilizadas técnicas orientadas ao conteúdo que permitem pesquisar mais rapidamente pelo conteúdo dos dados, como indexação, ou técnicas de redução de espaço de armazenamento, como compressão e deduplicação. No entanto, o uso destas técnicas torna-se incompatível com o emprego de esquemas criptográficos utilizados para manter a confidencialidade dos dados.

No caso, por exemplo, da indexação, os índices são geralmente construídos a partir de palavras-chave que mapeiam o conteúdo dos dados para os respetivos ficheiros permitindo otimizar, por exemplo, pesquisas sobre ficheiros que contêm um certo tópico. De forma semelhante, a deduplicação recorre à indexação e análise do conteúdo dos ficheiros para encontrar dados duplicados.

Consequentemente, ao cifrar os dados, estas técnicas tornam-se muito ineficientes ou até mesmo impossíveis. Recorrendo a cifras probabilísticas, duas cópias iguais geram criptogramas diferentes. No caso da deduplicação, por exemplo, isto torna-se um problema uma vez que impede a deteção de dados duplicados. Além disso, mesmo recorrendo a esquemas determinísticos de encriptação, o uso de chaves diferentes origina criptogramas diferentes, impossibilitando assim a deduplicação entre múltiplos clientes, ou obrigando à partilha de uma chave de encriptação comum entre os vários utilizadores. Existem algumas propostas que visam ultrapassar estes problemas e permitir o uso deste tipo de técnicas de forma segura e confidencial. Muitas destas soluções recorrem a esquemas de criptografia convergente que permitem que dados iguais gerem o mesmo criptograma. Contudo estas soluções apresentam ainda muitas falhas de segurança. A criptografia convergente permite que um atacante efetue o ataque "*confirmation of a file*" – "confirmação de um ficheiro". Se o atacante tiver acesso a um ficheiro na sua forma original, consegue verificar se um dado utilizador possui esse mesmo documento. Por exemplo, com este tipo de ataque é possível verificar se um determinado utilizador possui armazenados um filme ou uma música de

forma ilegal. Um outro ataque conhecido da criptografia convergente é o ataque "*learn the remaining information*" – "aprendizagem da restante informação". Neste ataque, o atacante conhece o modelo de um dado documento e tenta descobrir certas informações. Por exemplo, as cartas de um banco costumam seguir sempre o mesmo modelo, sendo que apenas o número da conta e a palavra-chave mudam de utilizador para utilizador. Se o atacante tiver acesso a esse mesmo modelo, poderá ser capaz de descobrir um número de conta e a respetiva palavra-chave que correspondam a algum utilizador [74].

Com o crescimento de plataformas de *hardware* confiável, surge a possibilidade de adotar esta tecnologia para aumentar a segurança dos sistemas de armazenamento e permitir o uso de técnicas de armazenamento orientadas ao conteúdo. Com o auxílio de tecnologias de *hardware* confiável seria possível isolar a aplicação de forma a que esta executasse num ambiente de execução confiável e protegido. Desta forma, os dados podiam ser enviados pela rede de forma protegida, isto é, cifrados probabilisticamente. Quando fosse necessário processar os mesmos na sua forma original, empregava-se este ambiente seguro e isolado para proceder à decifragem e ao processamento dos dados. Para além de permitir combinar técnicas de cifragem com funcionalidades de armazenamento orientadas ao conteúdo, esta plataforma permitiria também evitar a necessidade de recorrer a protocolos complexos de partilha de chaves entre os vários utilizadores. Contudo, estas plataformas estão ainda subexploradas em termos de desempenho e funcionalidades permitidas. Além disso, estas arquiteturas requerem que as aplicações tenham de ser reescritas para suportar esta tecnologia e implementar as funcionalidades de armazenamento de acordo com as bibliotecas permitidas pelo *hardware* confiável.

Por outro lado, o desenvolvimento, integração, reutilização e manutenção, das diversas funcionalidades de armazenamento apresentam outro grande desafio para os sistemas de armazenamento. Para colmatar este problema surgiram o armazenamento definido por *software* (SDS) e os sistemas de armazenamento por camadas, que permitem a criação de sistemas configuráveis e facilitam o desenvolvimento e integração de novas funcionalidades de armazenamento [34, 72, 56, 63]. O SafeFS [56] é um exemplo de um sistema baseado nos princípios de armazenamento definido por *software* que apresenta uma arquitetura por camadas. Esta arquitetura permite uma fácil integração de novas funcionalidades de armazenamento, bem como, uma maior flexibilidade na configuração do sistema, de forma a fornecer as funcionalidades (p.e., segurança, redução de espaço, replicação) que melhor se adaptam às necessidades de diferentes aplicações.

Como objetivo desta dissertação, pretende-se tirar partido de soluções de sistemas programáveis e empilháveis como o SafeFS, e integrar soluções de *hardware* configurável, como o SGX, assim como avaliar o impacto introduzido no desempenho e nos recursos utilizados. Adicionalmente, outro objetivo consiste em dotar sistemas de deduplicação atuais e

open-source com novos esquemas seguros que tirem partido da segurança oferecida pelo SGX.

1.2 CONTRIBUIÇÕES

Esta dissertação apresenta quatro conjuntos de contribuições: (i) uma revisão do estado da arte, (ii) a arquitetura e o desenvolvimento de uma plataforma de armazenamento por camadas dotada de SGX, (iii) a introdução de um novo esquema seguro de deduplicação, e (iv) a avaliação experimental do protótipo. Mais detalhadamente:

- Esta dissertação expõe uma revisão do estado da arte com foco em duas vertentes: *hardware* confiável, onde se investiga as diferentes tecnologias relacionadas com o *Intel* SGX e as suas semelhanças/desigualdades para com este. É ainda feito um levantamento do estado da arte relativamente ao tópico de armazenamento seguro, onde são descritas as diversas formas adotadas atualmente.

- Retrata uma arquitetura configurável e modular, intitulada por TrustFS, que permite executar funcionalidades de armazenamento do sistema SafeFS num ambiente confiável, recorrendo à tecnologia de *hardware* confiável da *Intel* (SGX)

- Exibe um protótipo com a funcionalidade de armazenamento de deduplicação como caso de estudo, que recorre ao SGX para efetuar as operações críticas num ambiente seguro. Adicionalmente, este protótipo apresenta um esquema de deduplicação segura baseada em épocas que, recorrendo ao SGX, oferece diferentes níveis de segurança e desempenho.

- Por último, apresenta uma avaliação experimental composta por micro e macro testes. Os micro testes foram executados de modo a avaliar o desempenho e os recursos utilizados pelo SGX na computação das operações críticas realizadas na camada da deduplicação. De modo a comparar o desempenho entre várias implementações, estes testes foram efetuados para duas implementações diferentes dos algoritmos para um ambiente confiável e comparados com uma implementação realizada num ambiente normal (não confiável). Os macro testes foram feitos num ambiente remoto (cliente-servidor) onde foi testado o protótipo e comparado com a solução do estado da arte mais recorrente, nomeadamente, a criptografia convergente.

Este trabalho deu origem ao artigo "TrustFS*: A Secure SGX-enabled Stackable File System Framework*", submetido para a conferencia USENIX FAST'19, onde é apresentado o sistema TrustFS e o novo esquema de deduplicação segura.

1.3 ESTRUTURA DA DISSERTAÇÃO

No capítulo 2 é feito o levantamento do estado da arte. Numa primeira parte é introduzia a noção de computação confiável e são apresentadas soluções de *hardware* confiável semelhantes ao SGX. De seguida, são enumeradas algumas aplicações atuais que recorrem ao uso do SGX para aumentar a segurança dos seus sistemas. Por último, são indicadas as soluções atuais que visam combinar esquemas de criptografia com deduplicação com ou sem *hardware* confiável.

No capítulo 3 é apresenta a arquitetura de TrustFS e no capítulo 4 é detalhado o modo de funcionamento do protótipo do sistema e apresentado um novo esquema de deduplicação segura por épocas. Os capítulos 5 e 6 introduzem a metodologia utilizada para testar o protótipo e os resultados obtidos na avaliação experimental, respetivamente.

Por último, no capítulo 7, são expostas as conclusões obtidas nesta dissertação e apresentado o trabalho futuro.

2

ESTADO DA ARTE

Os sistemas de armazenamento enfrentam frequentemente a necessidade de aceder aos dados sobre a sua forma original (não cifrados). Exemplo disso é a aplicação de técnicas de indexação ou redução de espaço para otimização das pesquisas orientadas ao conteúdo ou para otimização do espaço em disco, respetivamente. Contudo, o uso de esquemas criptográficos para assegurar a confidencialidade dos dados dos clientes limita o uso de técnicas orientadas ao conteúdo dos dados.

Neste capítulo é apresentada a noção de computação confiável (seção 2.1.1) e efetuada uma revisão sobre as técnicas de *hardware* confiável, similares ao SGX (seção 2.1.2), assim como enumeradas as diferentes aplicações que recorrem à tecnologia SGX para aumentar a segurança dos seus sistemas (seção 2.2). Por último, é feito o levantamento das soluções de deduplicação segura com e sem *hardware* confiável (seção 2.3). Este ponto é essencial uma vez que, como a deduplicação é o caso de estudo desta dissertação, é importante perceber quais as abordagens utilizadas na literatura.

2.1 COMPUTAÇÃO CONFIÁVEL

Existem várias soluções que visam fornecer um ambiente confiável para a execução de aplicações. Nesta seção será apresentada a noção de Computação Confiável e os seus requisitos. De seguida, será feito o levantamento de algumas técnicas de *hardware* confiável e realizada uma comparação entre elas. Por último, serão explicados alguns conceitos chave do SGX importantes para a melhor compreensão das abordagens seguidas nesta dissertação.

2.1.1 *Computação Confiável e os seus requisitos*

A computação confiável concentra-se na garantia de que uma entidade se comporta da maneira esperada, recorrendo a mecanismos de medição de integridade, armazenamento seguro e certificação do *software* [61]. A mesma consiste num conjunto distinto de propostas e iniciativas com o objetivo de criar uma arquitetura mais segura, que ofereça garantias sobre qual o *software* da aplicação que está a ser executado e que permita que aplicações

comuniquem com segurança com outras aplicações e/ou servidores. Computação confiável refere-se, portanto, à adição de funcionalidades de *hardware* e/ou *software* a um sistema de forma a adicionar algum nível de confiança nas entidades com as quais o computador interage e naquilo que o sistema está a executar [47].

Para concretizar uma plataforma confiável é necessária a implementação de alguns conceitos essenciais para computação confiável:

SISTEMA DE ARRANQUE SEGURO oferece certezas de que o sistema é inicializado num sistema operativo confiável que adere a alguma política de segurança dada.

ISOLAMENTO DE MEMÓRIA previne programas, sistemas operativos ou até mesmo *debuggers* de inapropriadamente ler ou escrever em memória que não lhes pertence.

***INPUT/OUTPUT* SEGUROS** a segurança das entradas e saídas consiste na validação dos dados recebidos através de *checksums* para verificar se o *software* utilizado para o efeito não foi alterado.

ARMAZENAMENTO SELADO permite o armazenamento de código e dados confidenciais de forma criptograficamente segura.

CERTIFICAÇÃO DE *SOFTWARE* permite que um programa se autentique. A certificação de *software* remota é um meio para que um sistema faça afirmações confiáveis sobre o software que está a executar em outro sistema. Ainda, a certificação permite detetar modificações não autorizadas no *software* gerando certificados cifrados para todas as aplicações de um computador. Por exemplo, considerando o caso em que um cliente pretende efetuar uma transação no *site* do seu banco. O *software* da aplicação do cliente apenas iria ser executado se o *software* do banco conseguisse provar que o seu código estaria a ser computado de uma forma segura.

MEDIÇÃO DE INTEGRIDADE é o cálculo de resumos criptográficos (*hash*) de código executável, dados de configuração e outras informações do estado do sistema para validar se a sua integridade foi ou não comprometida.

Como referido anteriormente, as tecnologias recorrem à adição de *software* ou *hardware*, que implementam estes conceitos, com o objetivo de alcançar uma plataforma confiável. A principal característica de uma arquitetura confiável é a Base de Computação Confiável (TCB). O TCB de um sistema informático consiste no conjunto de todos os componentes de *hardware*, *firmware* e/ou *software* que são críticos para a segurança dos sistemas. Qualquer mau funcionamento no TCB devido a falhas ou vulnerabilidades pode comprometer a segurança de todo o sistema.

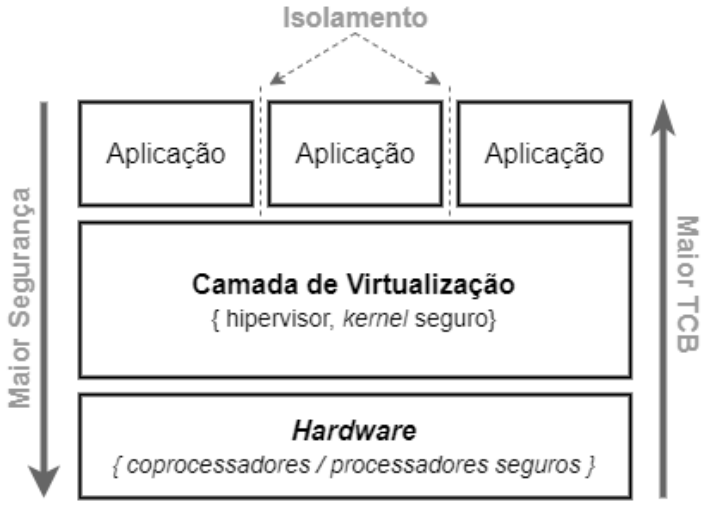


Figura 1: Camadas de abstração de um Sistema Operativo

A figura 1 detalha as diferentes camadas existentes num Sistema Operativo (OS) e demonstra onde as tecnologias podem incidir de modo a produzir uma plataforma confiável. A nível da camada da aplicação é recorrente o uso de compartimentos isolados, conhecidos como *containers*, para executar aplicações críticas. Docker [46], por exemplo, fornece uma camada adicional de abstração e automação de virtualização do nível do OS. As máquinas virtuais são outra técnica similar que fornecem o isolamento das aplicações. Por outro lado, existem tecnologias que se baseiam num *kernel* seguro ou num hipervisor, localizados entre a camada da aplicação e o *hardware*, que implementam algumas propriedades de segurança como canais confiáveis e isolamento entre processos. Outras tecnologias focam-se mais na implementação dessas funcionalidades via *hardware*. Como se pode induzir pela imagem anterior, as tecnologias focadas nas camadas mais inferiores oferecem uma maior segurança. Consequentemente, as soluções baseadas nas camadas superiores implicam um TCB mais complexo pois confiam em tudo o que lhes é inferior.

Na seção seguinte serão apresentadas algumas das tecnologias de *hardware* confiável, o seu modo de funcionamento e as funcionalidades de computação confiável que estas oferecem, e a constituição do seu TCB.

2.1.2 *Tecnologias de Hardware Confiáveis*

Existem várias tecnologias propostas com o objetivo de oferecer uma maior segurança recorrendo a *hardware*. Algumas soluções baseiam-se em coprocessadores seguros, como o IBM 4758 [25], que possuem os seus próprios periféricos seguros: Unidade Central de Processamento (CPU), aceleradores criptográficos, caches, Memória Dinâmica de Acesso Aleatório (DRAM) e controladores I/O, oferecendo ambientes resistentes a alterações. Exis-

tem duas alternativas neste sentido: um coprocessador externo, que consiste num módulo de *hardware* à parte do processador principal, não partilhando nenhum recurso com este; ou um coprocessador embutido no processador principal, possuindo a capacidade de partilhar alguns recursos com o sistema principal e manter na mesma o seu isolamento. Outra solução baseada em coprocessadores seguros é o TPM (*Trusted Platform Module*) [31]. Esta tecnologia confia num circuito integrado, na placa-mãe, com um dispositivo periférico para segurança. Este oferece geração aleatória de números em *hardware* e um conjunto de operações criptográficas (para gerar chaves, assinar, cifrar e calcular resumos criptográficos (*hash*)). Esta arquitetura introduziu o modelo de certificação de *software* para assegurar ao utilizador que o seu código foi produzido sem alterações e que este é executado num ambiente protegido pelo *hardware* confiável. O TCB desta solução engloba a placa-mãe (o circuito integrado TPM, CPU, DRAM, barramentos) e todo o *software* do computador. Esta arquitetura é vulnerável a ataques físicos visto que não fornece isolamento do *software* do computador. Além disso, é também suscetível a ataques no barramento de comunicação entre o processador e o circuito integrado TPM. TXT (*Trusted eXecution Technology*) [30] é uma arquitetura escalável baseada em *hardware* que valida a confiabilidade da plataforma durante a inicialização e execução, permitindo uma avaliação confiável da plataforma de computação e do seu nível de proteção. TXT utiliza o TPM, uma máquina virtual e técnicas criptográficas para fornecer medições do *software* e dos componentes da plataforma (BIOS (*Basic Input/Output System*), hipervisor, *etc.*), com a finalidade de serem usadas para decisões de confiança. O isolamento do *software* no TXT é alcançado através de um compartimento que garante controlo exclusivo sobre todo o computador enquanto ativo. Para isso, esta arquitetura recorre um módulo de código autenticado de inicialização segura (SINIT ACM) que provoca a reinicialização do sistema antes de iniciar a máquina virtual do compartimento. O seu TCB inclui o mesmo *hardware* que o do TPM, mas a nível de *software* inclui apenas o módulo SINIT ACM e a máquina virtual. Esta solução fornece proteção contra ataques a nível de *software*, destinados a roubar informações confidenciais, que corrompem o código do sistema ou da BIOS, ou modificam a configuração da plataforma. No entanto, é vulnerável a ataques físicos à DRAM uma vez que não recorre a técnicas criptográficas (cifragem/HMAC) para a proteger.

As soluções baseadas em coprocessadores sofrem, no entanto, de uma sobrecarga associada à transferência de dados entre o CPU e o coprocessador. Além disso, normalmente o coprocessador possui menos poder de computação que o processador principal. Como alternativa a estas arquiteturas, algumas tecnologias optam por adicionar segurança física e algumas funcionalidades adicionais ao processador principal.

XOM (*eXecute-Only Memory*) [42] é um exemplo de uma arquitetura baseada nesta estratégia. XOM assume que nenhum programa, nem mesmo o OS, pode ser confiável e, portanto, recorre a *hardware* para fornecer segurança para o sistema. Nesta arquitetura,

o *hardware* é responsável pela separação completa dos processos, sendo que nenhum processo deve conseguir ler ou alterar os dados privados ou instruções de outro processo. Adicionalmente, protege também contra ataques físicos à memória externa ao processador, recorrendo a mecanismos de criptografia para cifrar os programas e os dados de cada processo. AEGIS [68] baseia-se nos mesmos princípios de XOM e explora a "quantidade" de confiança que se pode por no *software*. Enquanto que XOM coloca toda a sua confiança no *hardware*, AEGIS recorre a um pequeno *kernel* de segurança para fornecer alguns recursos de segurança em *software*. Ambas as arquiteturas assumem que o processador e os seus componentes internos (como a memória) estão seguros contra leitura ou alterações. Contudo, tudo o que é externo ao processador é considerado vulnerável a ataques. Por isso, tanto XOM como AEGIS fornecem métodos para cifrar o código compilado para um determinado processador e executar esse código num ambiente protegido contra certos ataques, como engenharia reversa e adulteração. Sempre que os dados abandonam o processador são cifrados, e só são decifrados quando são carregados da memória principal para a cache do processador. Ambas as soluções admitem um TCB que inclui o processador seguro, a nível de *hardware*. Relativamente ao *software*, XOM inclui no TCB o código da aplicação e o hipervisor enquanto que no caso do AEGIS o hipervisor é substituído pelo *kernel* de segurança. Tanto XOM como AEGIS são vulneráveis a ataques de *side-channel*, isto é, ataques baseados na monitorização do consumo de energia e emissões eletromagnéticas enquanto um dispositivo está a executar operações criptográficas.

A ARM propôs outra tecnologia, TrustZone [2], que se baseia num conjunto de módulos de segurança baseadas em componentes *hardware* AMBA (*Advanced Microcontroller Bus Architecture*) para os processadores ARM. Esta tecnologia introduziu os conceitos de modos de execução "seguro" e "normal". Esta separação entre modos de execução seguros e não seguros é similar à separação habitual entre modos privilegiados e não privilegiados (*kernel*/utilizador), embora estes dois últimos coexistam com os dois primeiros. O modo "seguro" é isolado do modo "normal" e executa um *kernel* de segurança, *drivers* de dispositivos e aplicações de forma segura e confiável. De modo a interagir entre os modos "seguro" e "normal", TrustZone recorre a um monitor de segurança responsável por controlar as trocas de ambiente de execução de forma segura. A entrada no modo seguro pode ser acionada pelo *software* que executa uma instrução dedicada (SMC – *Secure Monitor Call*) ou por uma série de mecanismos de exceção. O código do monitor geralmente guarda o estado do modo "seguro" atual e restaura o estado do modo "seguro" para o qual está a ser alterado. Desta forma, TrustZone garante que o código no modo "normal" não consegue comprometer o *software* no modo "seguro". O conceito de modos seguros e normais estendem-se para além do processador, de modo a abranger memória, software, transações de barramento, interrupções e periféricos dentro de um SoC (Sistema-em-um-Chip). Os módulos de segurança em *hardware* adicionais permitem isolar, através de *hardware*, a memória física e

os periféricos da máquina. No entanto, esta arquitetura requer um *kernel* de segurança para garantir o isolamento seguro de processos, a sua gestão e os mecanismos de certificação de *software*. Para além disso, TrustZone não contempla contramedidas para ataques físicos, sendo que a ARM recomenda armazenar todo o código e dados no mundo seguro. Ainda, esta abordagem coloca limites significativos na funcionalidade do compartimento seguro uma vez que a Memória Estática de Acesso Aleatório (SRAM) no processador é muito mais cara que a DRAM com a mesma capacidade. O TCB de TrustZone contempla o processador seguro e os modos seguros (*firmware*, OS e aplicação).

A Intel apresentou recentemente a tecnologia SGX [17]. Esta arquitetura oferece um ambiente de execução seguro em infraestruturas não confiáveis através de um conjunto de instruções no CPU. Estas instruções permitem que o utilizador declare parte do seu espaço de endereçamento virtual como *enclaves* seguros para proteger código e dados sensíveis. O *enclave* tem acesso à memória da aplicação anfitriã, mas o contrário nunca acontece. Ou seja, qualquer tentativa externa de acesso à memória do *enclave* é barrada pelo processador. SGX não confia no OS e como tal o CPU supervisiona todas as suas operações de gestão do *enclave*. Além disso, toda a memória do *enclave* é cifrada quando escrita para a DRAM através de módulos *hardware* de criptografia. Esta tecnologia oferece, para além de isolamento de memória, mecanismos para verificar o *enclave*, para armazenar dados confidenciais de forma criptograficamente segura e para certificação de *software*, demonstrando a integridade do código e que este está a ser executado dentro do *enclave*. Quando um *enclave* é carregado para memória, o CPU calcula o resumo criptográfico do seu conteúdo e armazena-o num registo denominado *MRENCLAVE*. Antes de o executar, o processador compara o valor calculado com a versão fornecida pelo autor do código e aborta a sua execução em caso de incompatibilidade. De forma complementar, um processador com SGX contém uma chave privada assimétrica única que apenas pode ser acedida via um *enclave* especial denominado *Quoting Enclave* (QE). O QE assina resumos criptográficos de *enclaves* locais e gera umas estruturas denominadas por *QUOTE*. A certificação de *software* remota utiliza este *QUOTE* para verificar a integridade do código a ser executado num determinado *enclave* e que este pertence a um processador SGX genuíno. O SGX reduz o seu TCB incluindo apenas o processador e o código dos *enclaves*. Esta tecnologia é, no entanto, vulnerável a ataques *side-channel*. Iso-X [27] e Sanctum [18] são duas arquiteturas muito similares à do SGX. As principais vantagens do Iso-X são o facto de não limitar o local onde as páginas de memória dos *enclaves* residem, oferecendo uma maior flexibilidade no uso da memória, e permitir o crescimento dinâmico dos *enclaves*. Iso-x requer apenas a adição de seis instruções ISA necessárias para a gestão dos *enclaves* e outras duas instruções opcionais para suportar o intercâmbio de páginas, que não precisam de constar no TCB. Estas alterações de hardware adicionam, no entanto, sobrecarga ao tempo de ciclo do processador e não protegem contra ataques de temporização de *cache*. Sanctum

introduziu mecanismos para contornar os ataques de *software*, nomeadamente ataques de *cache-timing* e *address translation*. Para tal, Sanctum apenas adiciona *hardware* às interfaces entre blocos do CPU e recorre a um monitor seguro confiável que verificar as decisões de alocação de recursos do OS assegurando-se, por exemplo, que uma região DRAM não é acessível para dois compartimentos diferentes. Nesta arquitetura, a DRAM do computador é dividida em regiões contínuas de tamanhos iguais e cada região usa conjuntos distintos no último nível da *cache* partilhado. Cada região DRAM é alocada a exatamente um compartimento, oferecendo isolamento aos compartimentos tanto a nível de DRAM como do último nível da *cache* (LLC - *Last Level Cache*). A principal melhoria em relação ao SGX é a defesa contra ataques de *software* que analisam os padrões de acesso à memória de um compartimento isolado para inferir informações privadas. No entanto, Sanctum apenas considera este tipo de ataques (de *software*) e não oferece proteção contra qualquer ataque físico. O seu TCB inclui, para além do processador, o modulo da aplicação e o monitor seguro.

A AMD apresentou a mais recente tecnologia de *hardware* confiável que cifra e protege a memória do sistema: AMD *Memory Encryption Technology* [38]. Esta tecnologia é focada primeiramente em infraestruturas da nuvem publica e, mais em específico, em infraestruturas com serviço (IaaS – *Infrastructure as a Service*). Como principais componentes possui o Secure Memory Encryption (SME) e o Secure Encrypted Virtualization (SEV). Ambos são geridos pelo sistema operativo ou hipervisor e não requerem nenhuma modificação no código da aplicação. SME usa *hardware* dedicado nos controladores de memória para cifrar dados gravados na DRAM e decifrá-los quando lidos, lidando assim com os ataques de acesso físico. As chaves de cifragem, geridas pelo processador seguro da AMD, são geradas aleatoriamente em cada reinicialização do sistema e não visíveis para o *software* que está a ser executado nos núcleos principais do CPU. SEV é um componente de segurança que visa lidar com os ataques executados por parte de *software* privilegiado através do isolamento por máquinas virtuais cifradas. O espaço de memória das máquinas virtuais é cifrado, recorrendo a uma chave de encriptação específica da máquina, e protegido do hipervisor ou de outras máquinas virtuais da mesma plataforma. Embora os componentes SME e SEV forneçam proteção contra ataques físicos ou ataques por *software* privilegiado, a nova tecnologia da AMD não fornece proteção relativamente à integridade da memória. Isto torna-se um fator problemático uma vez que permite que um hipervisor malicioso com acesso privilegiado consiga alterar ou manipular as páginas de memória cifrada da máquina virtual. Comparativamente com o SGX, esta tecnologia da AMD apresenta um TCB maior, uma vez que inclui o sistema operativo e o hipervisor.

	Isolamento	Certificação de software	Armazenamento Selado	Confidencialidade do código	Resistência a ataques Side-Channel	Proteção da ataques físicos
TPM	✗	✓	✓	✓	✗	∼
TXT	✓	✓	✓	✓	✓	∼
XOM	✓	✗	✗	✓	✗	✓
AEGIS	✓	✓	✓	✓	✗	✓
TrustZone	✓	✗	✗	✗	✗	✗
SGX	✓	✓	✓	✓	✗	✓
Iso-X	✓	✓	✗	✗	✗	✓
Sanctum	✓	✓	✓	✓	✓	✗
AMD SEV/SME	✓	✓	✓	✓	✗	✓

Tabela 1: Comparação de algumas arquiteturas de computação confiável baseadas em *hardware*
✓e ✗ indicam o suporte ou não da funcionalidade e ∼ indica suporte parcial

De uma forma geral, existem duas grandes abordagens para tecnologias de *hardware* confiável: a adição de um coprocessador seguro ou a alteração do próprio processador da máquina de forma a garantir os principais conceitos da computação confiável. Algumas arquiteturas apresentam a noção de um modo seguro separado do modo normal de execução que oferece uma execução isolada do código da aplicação. As diversas tecnologias apresentam diferentes soluções para alcançar uma computação confiável, mas todas elas apresentam algumas vulnerabilidades. Um ponto muito forte na computação confiável e que tem sido dos principais objetivos no desenvolvimento de *hardware* confiável consiste na redução do TCB. Quanto menor for o número de componentes que são necessários confiar, menos vulnerável se torna a aplicação a ataques. A tabela 1 apresenta uma comparação entre algumas das tecnologias anteriormente abordadas. À exceção do TPM, todas elas oferecem isolamento da aplicação. Contrariamente, a resistência a ataques *side-channel* é a maior complicação destas soluções, sendo que apenas o TXT e o Sanctum suportam esta defesa. A tecnologia Sanctum, embora apresente mecanismos capazes de resistir a este tipo de ataques, é vulnerável a ataques físicos. O mesmo acontece com a tecnologia TXT, que apenas oferece uma proteção parcial para estes ataques. Além disso, ambas as tecnologias apresentam um TCB mais complexo do que o do SGX, o que se pode traduzir num maior número de componentes que podem comprometer a segurança do sistema. Ambas as tecnologias da Intel (SGX) e da AMD (*Memory Encryption Technology*) suportam praticamente todas estas características, sendo que apenas não conseguem oferecer resistência a ataques *side-channel*. Segundo o estudo realizado por Mofrad et al., a AMD *Memory Encryption Technology* efetua um processamento mais rápido que o SGX quando a aplicação protegida requer um conjunto alargado de recursos de memória. Por outro lado, o SGX fornece proteção da integridade da memória oferecendo uma maior confidencialidade que a tecnologia da AMD. Este estudo concluiu que a AMD *Memory Encryption Technology* é mais adequada para proteger aplicações complexas enquanto que o SGX é mais adequado para cenários mais pequenos, mas mais sensíveis em termos de segurança.

Devido ao seu reduzido TCB, às características de computação confiável que disponibiliza e à segurança e confidencialidade que oferece, optou-se por recorrer à tecnologia SGX como

tecnologia de *hardware* confiável a integrar num sistema de armazenamento modular e configurável. A seção seguinte descreve em mais detalhe esta tecnologia e alguns dos seus conceitos mais relevantes.

2.1.3 *Intel SGX: Visão geral*

Como referido anteriormente, a tecnologia SGX consiste num conjunto de instruções adicionadas ao CPU que permitem o isolamento das aplicações numa área reservada de memória. Estas áreas (*enclaves*) são restritas à aplicação em causa e impedem o acesso por parte de aplicações externas ou mesmo pelo próprio *kernel*. Para fazer a "ponte" entre o ambiente confiável e o não confiável, o SGX possui funções que permitem a entrada/saída do *enclave* de forma segura e para cifrar os dados quando estes o abandonam.

Controlo de acessos

O SGX reserva uma região da memória denominada por PRM (*Processor Reserved Memory*). Esta região é protegida pelo processador de qualquer tentativa de acesso externa ao *enclave*, incluindo por parte do *kernel*, do hipervisor ou acessos diretos à memória por periféricos. A figura 2 mostra a divisão destas áreas de memória.

O PRM inclui o Enclave Page Cache (EPC), que consiste em páginas de 4KB que armazenam o código e os dados do *enclave*. A cada *enclave* são atribuídas uma ou mais páginas EPC, por parte do *software* do sistema, e o CPU monitoriza o estado de cada página através do EPCM (*Enclave Page Cache Metadata*).

Sempre que é efetuado um pedido de acesso a determinado endereço é verificado primeiramente se o objetivo é aceder à memória de algum *enclave*. Se for esse o caso, verifica-se se o endereço consta no EPC e em caso afirmativo o acesso é permitido se a aplicação estiver autorizada a fazê-lo. Caso contrário o acesso é negado.

Se não pretender aceder à memória do *enclave* mas o endereço constar no EPC, o acesso é igualmente negado. Por último, se o endereço não constar no *EPC* o acesso é concedido.

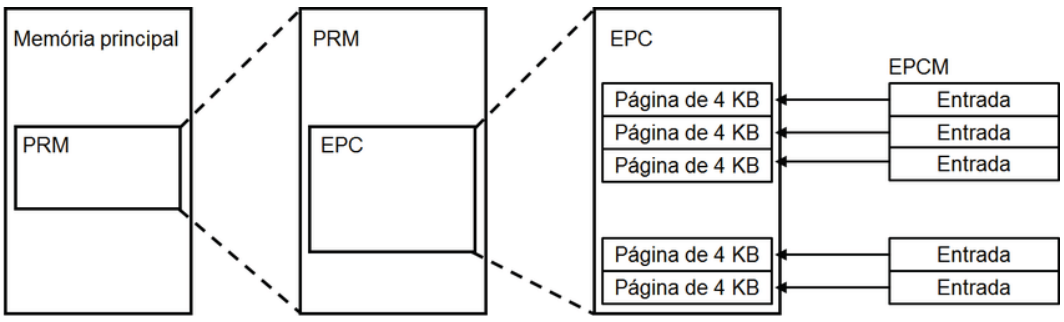


Figura 2: Memória do SGX (Adaptado de Costan and Devadas)

Ciclo de vida do enclave

A primeira fase de vida do *enclave* passa pela sua inicialização. Durante esta fase, o *software* do sistema pede ao processador para copiar os dados iniciais do *enclave* da memória desprotegida (fora do PRM) para páginas do EPC e atribuir essas páginas ao *enclave* que está a ser inicializado. Após o carregamento de todas as páginas do *enclave* para o EPC, o *enclave* é marcado como inicializado e o método de inicialização é desativado. Durante o processo de inicialização, o conteúdo do *enclave* vai sendo resumido criptograficamente. Quando concluído o processo, o resumo criptográfico resultante torna-se no *MRENCLAVE* do *enclave*. Após este ponto, existem instruções especiais do CPU que permitem entrar/sair do *enclave* de forma protegida. De igual forma, a criação e a destruição do *enclave* é realizada por primitivas similares.

De uma forma geral, o modo de funcionamento de uma aplicação dotada de SGX é o seguinte:

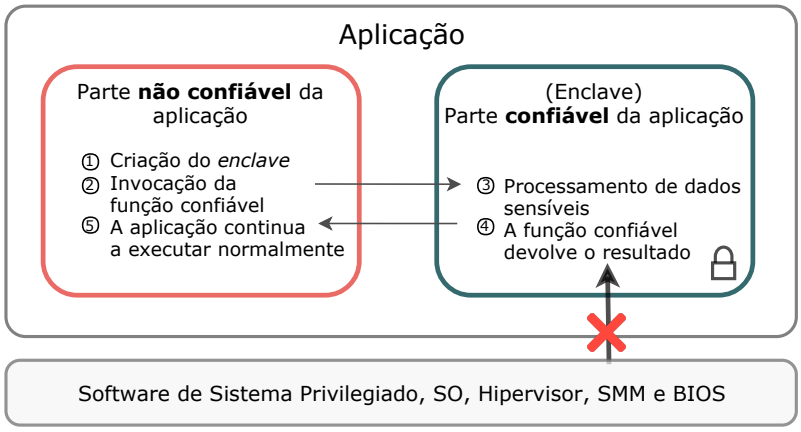


Figura 3: Funcionamento geral de uma aplicação dotada de SGX.

A aplicação é dividida em duas partes: a parte não confiável e a parte confiável (o *enclave*). O primeiro passo da aplicação é criar e executar o *enclave* (que se encontra localizado numa área de memória segura conhecida como EPC) (figura 3-①). De seguida é feita a invocação das funções confiáveis do *enclave* e é feita uma troca de contexto de execução do ambiente não confiável para o *enclave* (figura 3-②). Uma vez lá dentro, é feito o processamento necessário sobre os dados sensíveis de forma segura (figura 3-③). Após terminado o processamento, as funções seguras devolvem os resultados à aplicação e existe novamente uma troca de contexto, neste caso, do *enclave* para o ambiente não confiável (figura 3-④). A partir deste ponto a aplicação continua a executar normalmente (figura 3-⑤).

Ecalls/Ocalls

O conceito do SGX baseia-se na execução de código crítico num ambiente de execução confiável. Desta forma, o código da aplicação é dividido em duas partes: o código confiável, isto é, o código crítico que deve ser executado dentro do *enclave*, e o código não confiável, que corresponde ao código da aplicação que é executado no ambiente de execução normal.

Ao construir um *enclave* é necessário definir uma interface Enclave Definition Language (EDL), como exemplificado no excerto de código 2.1, onde se especificam as funções *ECALL* (*"enclave call"*) e as funções *OCALLs* (*"outside the enclave"*). As primeiras são o ponto de entrada para o *enclave* a partir da aplicação não confiável (*trusted*). As funções OCALL são aquelas que são invocadas de dentro do *enclave* para executar determinado código no ambiente não confiável e depois retornar ao *enclave* (*untrusted*). O Kit de desenvolvimento de software (SDK) do SGX inclui uma ferramenta denominada por *Edger8r* que através da análise do ficheiro *.edl* gera automaticamente quatro ficheiros: o *.c* e o *.h* com a implementação das funções confiáveis (aquelas que são definidas no *enclave*) e o *.c* e *.h* com a implementação das funções não confiáveis (aquelas executadas pela parte não confiável da aplicação que invocam as funções do *enclave*).

```
enclave {
    trusted {
        public void test_ecall_user_check([user_check] int * ptr);
        public void test_ecall_in([in] int * ptr);
        public void test_ecall_out([out] int * ptr);
        public void test_ecall_in_out([in, out] int * ptr);
    };
    untrusted {
        void test_ocall_user_check([user_check] int * ptr);
        void test_ocall_in([in] int * ptr);
        void test_ocall_out([out] int * ptr);
        void test_ocall_in_out([in, out] int * ptr);
    };
};
```

Excerto de Código 2.1: Exemplo da sintaxe do ficheiro *.edl*

Quando estas funções possuem apontadores como argumento é necessário indicar a direção destes: *in*, *out* ou a combinação dos dois. Este atributo de direção instrui o *proxy* responsável pela separação/ligação entre os ambientes confiáveis e não confiáveis para efetuar uma cópia do *buffer* assinalado pelo apontador. Posto isto, o atributo de direção deve ser acompanhado pelo tamanho dos dados a copiar (regra geral: total de *bytes*). Caso o tamanho a copiar não seja indicado, serão copiados o número de *bytes* equivalente ao resultado da aplicação da função *sizeof* ao apontador.

IN Indica que o conteúdo do apontador deve ser passado da aplicação para o *enclave* (onde serão processados), no caso das *ECALLs* (por exemplo, a função *test_ecall_in* do excerto de código 2.1), ou do *enclave* para a aplicação, no caso das *OCALLs* (função *test_ocall_in*), aquando a invocação da função. No caso de uma *ECALL*, é alocado um *buffer* na memória interna ao *enclave* para o qual será copiado o conteúdo do apontador. Qualquer alteração efetuada sobre estes dados dentro do *enclave* não será visível para a aplicação uma vez que estes foram copiados. No caso das *OCALLs* o processo é idêntico, com a exceção que os dados são copiados do *enclave* para um novo *buffer* na aplicação.

OUT Indica que os dados devem ser transferidos (tal como no caso do *in*) no fim da invocação da função. Quando a função é invocada, é alocado um novo *buffer* e inicializado a zeros. No final da função, o conteúdo desse *buffer* é então copiado para o processo original (do *enclave* para a aplicação no caso das *ECALLs* – função *test_ecall_out* – e da aplicação para o *enclave* no caso das *OCALLs* – função *test_ocall_out*).

É ainda possível combinar ambas as direções (*in* e *out*), como exemplificado pelas funções *test_ecall_in_out* e *test_ocall_in_out*). Desta forma os dados serão copiados aquando o início da função, processados e novamente copiados no fim para o processo original.

Existem situações em que as restrições impostas pelos parâmetros *in* e *out* são difíceis de cumprir. Por exemplo, se o *buffer* for demasiado grande para ser mantido na memória do *enclave* é necessário fragmentá-lo e processá-lo em blocos mais pequenos e invocar o *enclave* várias vezes. Em alternativa, é possível utilizar a opção *user_check*, que permite que o *enclave* aceda diretamente à memória externa. Embora esta opção possa trazer melhorias em termos de desempenho quando comparada com as restantes, se não for aplicada corretamente pode introduzir espaço para falhas de segurança. Por isso, utilização desta opção deve ser sempre acompanhada por verificações adicionais no código para evitar a fuga de informações confidenciais do *enclave*.

Seal/Unseal

Por vezes é necessário armazenar, de forma segura, certos dados no meio de armazenamento (como o disco) e que, posteriormente, serão novamente utilizados no *enclave*. Este processo, armazenamento selado, pode ser feito através da operação *SEAL*. Esta função cifra os dados com uma chave privada *seal*, que é única para cada plataforma. Existem dois tipos de chave *seal*. Uma delas é derivada a partir da chave do processador juntamente com o resumo criptográfico do *enclave* (*MRENCLAVE*), sendo que apenas o mesmo *software* a ser executado no mesmo processador poderá executar a operação inversa (*UNSEAL*). A outra chave *seal* é derivada a partir da chave do processador em conjunto com o resumo criptográfico da chave pública atribuída pelo ISV aquando a compilação do *software* (*MR-*

SIGNER). Quando utilizada esta chave é possível manter a compatibilidade entre diferentes versões do mesmo *enclave* ou partilhar segredos entre *enclaves* no mesmo processador.

Certificação de software local/remota

Uma aplicação pode ser constituída por diferentes *enclaves* que interagem uns com os outros. De forma a permitir que um *enclave* prove a sua identidade e autenticidade para uma outra parte na mesma plataforma recorre-se ao mecanismo de certificação de *software* local (*local attestation*). Neste mecanismo, o *hardware* gera uma credencial (conhecida por *REPORT*) que inclui uma prova criptográfica de como o *enclave* existe naquela plataforma.

O mecanismo de certificação de *software* remota permite avaliar a identidade de um *enclave*, a sua estrutura, a integridade do seu código e garante que este executa num processador SGX genuíno. Após um pedido de certificação de *software* remota é gerada a credencial local e convertida para uma credencial remota (*QUOTE*) que será enviado para a entidade que efetuou o pedido e que o irá verificar. No final do processo de certificação de *software* e consoante os atributos do *enclave*, a entidade que requisitou a certificação de *software* decide se deve ou não confiar no *enclave*. Além disso, este mecanismo proporciona um segredo partilhado preliminar entre a entidade que efetuou o pedido de certificação e a aplicação do *enclave* que ajuda a estabelecer um canal seguro de comunicação pela rede não confiável.

2.2 Computação Confiável em Armazenamento

A recente tecnologia da *Intel* tem sido alvo de pesquisas em diferentes áreas entre as quais base de dados, sistemas de armazenamento, *machine learning*, etc.

Existem propostas, como *Graphene-SGX* [15], *Haven* [7], *SCONE* [4] e *Panoply* [65], que incidem em portar, desenvolver e executar aplicações em ambientes de execução confiável com o mínimo de modificações necessárias. Estes sistemas garantem confidencialidade e integridade, através da proteção e do isolamento das aplicações e dos seus dados, de acesso não autorizado ou processos não confiáveis (OS, hipervisor, *etc.*).

Alguns sistemas, como *Opaque* [77] e *VC3* [62], permitem, com recurso ao SGX, a execução de operações analíticas sobre dados com fortes garantias de segurança, utilizando plataformas de análise de dados como *Apache Spark* [76] e *Hadoop MapReduce* [22]. Outro tópico que também tem sido alvo de investigação com o auxílio do SGX é *machine learning*. Ohrimenko et al. [50] e Küçük et al. [40] introduzem sistemas focados na preservação da privacidade em *machine learning* e na execução confiável sobre fontes de dados *multi-party*.

No que toca a gestão de base de dados, existe o *EnclaveDB* [57], um motor de base de dados SQL seguro que assegura confidencialidade e integridade dos dados, mantendo todas as tabelas, índices e interrogações na memória do *enclave*, protegendo assim contra entidades não autorizadas ou servidores/administradores de base de dados maliciosos. *HardIDX*

[28] é um sistema que propõe uma abordagem baseada em *hardware* para pesquisas sobre dados cifrados, permitindo a implementação de um índice de base de dados com grande desempenho. *REARGUARD* [69] é outro mecanismo de pesquisa sobre dados cifrados que utiliza SGX. Ainda no tema de pesquisas, *X-Search* [49] introduz um mecanismo de pesquisas na Web que recorre aos métodos de isolamento e segurança do SGX.

SecureKeeper é uma versão melhorada do serviço de coordenação *ZooKeeper* que utiliza SGX para preservar a confidencialidade e a integridade dos dados manuseados por este. Este sistema recorre a vários *enclaves* para garantir que os dados fornecidos pelo cliente no *ZooKeeper* estão sempre devidamente cifrados quando abandonam o *enclave* e que todo o processamento sobre a forma original dos dados é feito de forma segura.

Relativamente a sistemas de ficheiros, a *Intel* lançou a biblioteca *Protected File System Library for SGX* [70] que permite criar, trabalhar e eliminar ficheiros dentro do *enclave*. Uma vez que as funções por ela oferecida possuem algumas falhas de segurança, nomeadamente no que diz respeito à observação das interações entre o *enclave* e o *kernel*, surgiu recentemente o sistema *OBLIVIATE* [1] que, adaptando o protocolo ORAM (Oblivious RAM) para escritas e leituras de um ficheiro dentro do *enclave*, tenta esconder os padrões de acesso aos ficheiros.

Uma vez que é recorrente a utilização de técnicas de encriptação e, por vezes, é necessária a troca de chaves entre o cliente e o servidor, muitos estudos incidem na troca segura destas chaves. Neste sentido, *Poster* [44] propõe uma plataforma que visa fornecer as chaves dos utilizadores para os *enclaves* do SGX de forma segura.

Em suma, existem já algumas propostas que recorrem ao *hardware* para aumentar a segurança em sistemas de armazenamento, mas estas são ainda muito escassas. As soluções em cima apresentadas visam permitir o uso de certas técnicas, nomeadamente a indexação e deduplicação, em sistemas de armazenamento na nuvem de uma forma mais confiável. Contudo, estes sistemas divergem do objetivo desta dissertação uma vez que, nestes casos, o SGX é utilizado para a segurança de sistemas com funcionalidades específicas. Esta dissertação pretende dotar com SGX um sistema de armazenamento modular e configurável de forma a permitir implementar de forma segura qualquer funcionalidade de armazenamento.

Como caso de estudo desta dissertação foi escolhida a técnica da deduplicação de dados. A próxima seção apresenta o conceito desta funcionalidade e enumera as atuais soluções para deduplicação segura com e sem *hardware* confiável.

2.3 DEDUPLICAÇÃO DE DADOS SEGURA

Como referido anteriormente, existem muitas técnicas essenciais para a gestão de um sistema de armazenamento que são orientadas ao conteúdo e como tal não podem ser apli-

cadas sobre dados cifrados. Esta dissertação recorre à técnica de deduplicação de dados como caso de estudo, uma vez esta apresenta-se como um bom exemplo de uma funcionalidade de armazenamento orientada ao conteúdo, que contempla uma ampla investigação no tópico de deduplicação segura, e combina problemas de redução de espaço com indexação. Esta seção apresenta a descrição desta técnica de armazenamento e expõe as várias soluções da literatura que pretendem permitir a aplicação da deduplicação sobre dados cifrados.

O termo "deduplicação de dados" refere-se a técnicas que eliminam dados redundantes, por exemplo ficheiros ou blocos, armazenando apenas uma única cópia e criando uma referência lógica para esta cópia de forma a que diferentes pedidos lógicos de armazenamento para o mesmo conteúdo sejam redirecionados para a mesma cópia única [54]. A deduplicação reduz os requisitos de espaço e, quando aplicada remotamente, a largura de banda dos serviços de armazenamento de dados, sendo mais eficaz quando aplicada sobre os dados de diversos utilizadores.

Na deduplicação entre vários utilizadores (*cross-user deduplication*), cada bloco ou ficheiro é comparado com os dados de outros utilizadores e é deduplicado se já existir uma cópia idêntica disponível no servidor. Deduplicação *single-user* é utilizada para remover redundância de dados pertencentes a um único utilizador. A deduplicação entre vários utilizadores possui um maior potencial para poupar espaço de armazenamento, principalmente perante ficheiros grandes que são partilhados por muitos utilizadores.

Existem duas estratégias de deduplicação, relativamente à granularidade: deduplicação ao nível do ficheiro ou ao nível do bloco. Na primeira, a redundância dos dados é explorada ao nível dos ficheiros, sendo apenas armazenada uma cópia de cada documento e as restantes sendo associadas a esta cópia. Na segunda estratégia, cada ficheiro é dividido em múltiplos blocos e a redundância de dados é explorada ao nível do bloco.

Por outro lado, a deduplicação pode ser efetuada quer ao nível do servidor quer ao nível do cliente. Na deduplicação do lado do servidor (*server-side deduplication*), o servidor verifica se o ficheiro [1] enviado pelo cliente já foi previamente armazenado. Em caso afirmativo, o servidor não o armazena novamente, mas, em vez disso, regista o cliente como proprietário desse ficheiro e permite-lhe o acesso ao ficheiro partilhado. Esta abordagem atinge o objetivo de reduzir espaço, mas exige que o cliente envie sempre o ficheiro que pretende armazenar. Na deduplicação do lado do cliente (*client-side deduplication*), antes de carregar um ficheiro, o cliente verifica se este já se encontra armazenado na nuvem, enviando ao servidor o resumo criptográfico do conteúdo para que o mesmo verifique a sua existência no armazenamento remoto. Se o ficheiro já estiver armazenado na nuvem, o cliente não necessita de o enviar, e o servidor permite-lhe o acesso ao documento da mesma

1 No caso da deduplicação ao nível do bloco, o processo é semelhante.

forma descrita anteriormente. Neste caso, a deduplicação do lado do cliente consegue não só reduzir os requisitos de armazenamento, mas também os de largura de banda.

De forma a encontrar, de forma eficiente, os dados duplicados, a deduplicação baseia-se em índices do conteúdo dos ficheiros/blocos. Para manter estes índices reduzidos, são utilizados resumos criptográficos desses mesmos conteúdos. Devido a este pormenor, os dados cifrados tornam-se um impasse para a deduplicação, uma vez que com os dados cifrados probabilisticamente torna-se difícil detetar dados idênticos, pelo que a criptografia utilizada deve permitir adotar esta estratégia. Com este objetivo em mente, foram surgindo várias soluções para uma deduplicação segura. De seguida serão expostas as soluções do estado da arte que pretendem aplicar a deduplicação sobre dados cifrados, mantendo a confidencialidade e integridade destes.

2.3.1 *Soluções tradicionais de deduplicação segura*

Como explicado anteriormente, a criptografia probabilística impossibilita a deduplicação. Assim, Whiting and Dilatush introduziu, em 1995, o método de combinar chaves baseadas no conteúdo com a deduplicação denominado por Criptografia Convergente (CE) (do inglês "*Convergent Encryption*") [12, 23]. A CE utiliza o resumo criptográfico do conteúdo dos ficheiros como chave para cifrar os dados. Desta forma, dados idênticos originaram o mesmo criptograma, permitindo assim a aplicação de técnicas de deduplicação sobre dados cifrados. Para comparar dois criptogramas são utilizadas etiquetas (T) que, neste caso, equivalem ao resumo criptográfico do respetivo criptograma, $T = Hash(C)$. Apesar da criptografia convergente assegurar a privacidade dos dados perante deduplicação, é vulnerável a ataques de força bruta. Uma vez que a CE é um esquema de criptografia simétrica determinística, o seu espaço-chave é limitado e, através da geração da chave convergente de cada mensagem e do cálculo do respetivo criptograma, a mensagem alvo pode ser inferida.

Farsite [13] e Pastiche [19] são um sistema de armazenamento distribuído e um sistema de cópias de segurança, respetivamente, que utilizam criptografia convergente para aplicar deduplicação. Farsite desenvolveu a CE para aplicar deduplicação a nível do ficheiro. O cliente calcula o resumo criptográfico do conteúdo do ficheiro e utiliza-o como chave para cifrar o documento. De seguida, cifra também esse resumo criptográfico com as chaves públicas dos leitores autorizados do ficheiro e anexa os valores cifrados como metadados. Por sua vez, Pastiche propôs a vertente HCE (*Hash Convergent Encryption*) para deduplicação ao nível de blocos. Em HCE, os dados são cifrados com o seu resumo criptográfico e a etiqueta (T) é gerada calculando o resumo criptográfico da chave utilizada para cifrar, $T = Hash(K)$ com $K = Hash(F)$. HCE é vulnerável a ataques de força bruta e ataques de falsos duplicados, onde um utilizador honesto pode ficar incapaz de recuperar

o seu ficheiro original, pois este pode ser substituído por um falso sem que a substituição seja detetada.

Mais tarde, Bellare et al. [9] propôs a noção de MLE (*Message-Locked Encryption*) com o objetivo de resolver o problema da deduplicação a nível do ficheiro entre múltiplos utilizadores, e utiliza uma etiqueta para referenciar o documento. Este desenvolveu várias vertentes da CE (CE, HCE1, HCE2 e RCE) e verificou o seu nível de segurança. Entre essas vertentes destaca-se a RCE (*Randomized Convergent Encryption*). Nesta vertente, a mensagem é cifrada com uma chave aleatória e essa chave aleatória é cifrada com o resumo criptográfico da mensagem. Neste caso, a etiqueta corresponde ao resumo criptográfico da chave aleatória. O documento cifrado corresponde à concatenação dos dois criptogramas gerados com a etiqueta. RCE é também vulnerável ao ataque de falsos duplicados. Para colmatar este problema, Bellare and Keelveedhi propôs uma versão interativa de RCE, IRCE (*Interactive Randomized Convergent Encryption*). Nesta versão, um utilizador honesto pode verificar a coerência da etiqueta interagindo com o servidor e comprovar que o criptograma original está armazenado. Se um adversário carregar um criptograma modificado, este criptograma estará incoerente em relação à etiqueta do ficheiro correspondente. Esta incompatibilidade permite detetar que o texto cifrado está incorreto.

Com uma estratégia diferente surge DupLESS (*Duplicateless Encryption for Simple*) [10], apresentando uma arquitetura que utiliza criptografia convergente com ajuda de um servidor para resistir aos tradicionais ataques de força bruta da CE. Em DupLESS, os clientes cifram os seus dados com uma chave convergente gerada com a ajuda de um servidor de chaves adicional (separado do servidor de armazenamento). O servidor de chaves gera a chave convergente com base em duas entradas: o resumo criptográfico da mensagem e uma chave do sistema. Nesta arquitetura, o cliente obtém a chave convergente através da interação com o servidor de chaves executando uma função pseudoaleatória inconsistente (OPRF). Para evitar ataques de força bruta *online* por clientes comprometidos, DupLESS utiliza a estratégia de limitar o número total de consultas que um cliente pode fazer num determinado tempo. Contudo, o servidor de chaves constitui um ponto único de falha e, além disso, esta técnica não é adequada para deduplicação ao nível de bloco uma vez que provoca um aumento no tempo necessário para armazenar um documento.

Sendo as chaves da CE um elemento de extrema importância e que exige uma gestão complexa destas (por parte do cliente) e algum espaço de armazenamento para as guardar, Dekey e SecDep apresentam soluções para a sua gestão e segurança. Dekey [41] é um esquema eficiente e confiável de gestão de chaves da CE para deduplicação ao nível de bloco. Este divide a chave convergente de cada bloco em segredos e distribui-os por diferentes servidores de chave, com base no esquema RSSS (*Ramp Secret Sharing Scheme*) [11, 21]. Os fornecedores de serviços de gestão de chaves (KM-CSP: *key-management cloud service provider)* mantêm as chaves convergentes dos utilizadores e fornecem serviços de

armazenamento e computação para facilitar a gestão de chaves. Em vez de cifrar as chaves convergentes por cliente, Dekey cria segredos partilhados a partir dessas chaves e distribui-os pelos KM-CSPs. Desta forma, os vários utilizadores que possuam um bloco idêntico conseguem aceder à mesma chave convergente, reduzindo significativamente a sobrecarga de armazenamento de chaves convergentes. Ao contrário das propostas anteriores, o objetivo de Dekey não é a eficiência do espaço, mas sim garantir a confiabilidade das chaves convergentes, pelo que ainda sofre enormes gastos gerais de espaço com chaves. SecDep [78] explora a distribuição redundante de dados entre utilizadores internos e externos, e usa variantes de CE para chegar a um compromisso entre a segurança de dados e a deduplicação. Esta tecnologia propôs duas abordagens: UACE (*User-Aware Convergent Encryption*) para resistir a ataques de força bruta e reduzir os custos de computação (tempo), e MLK (*Multi-Level Key*) para assegurar a segurança das chaves e reduzir as despesas gerais do espaço das chaves. UACE combina deduplicação ao nível de ficheiro entre múltiplos utilizadores com deduplicação ao nível de bloco interna ao utilizador. O servidor, recorrendo a CE, gera as chaves convergentes ao nível ficheiro (para garantir a segurança da deduplicação entre vários utilizadores) enquanto que as chaves a nível de bloco são geradas pelo cliente. MLK pretende reduzir a sobrecarga do espaço das chaves utilizando as chaves do nível de ficheiro para cifrar as chaves do nível de bloco. MLK divide as chaves do nível de ficheiro em chaves de partilha via SSSS (*Shamir's Secret Sharing Scheme*) [64] e distribui por vários servidores de chaves para garantir a segurança e confidencialidade das chaves do nível de ficheiro. Segundo Zhou et al., SecDep mostrou-se mais eficiente em termos de tempo e de espaço que as abordagens de deduplicação segura que a antecedem.

Outras propostas foram feitas neste âmbito, como acordar a chave de cifragem entre os vários clientes [43] recorrendo ao protocolo PAKE (*Password Authenticated Key Exchange*); cifrar com dois esquemas criptográficos diferentes os ficheiros menos populares e cifrar os mais populares apenas com criptografia convergente [67]; ou esquemas que levam em consideração a popularidade dos blocos de dados e que utilizam técnicas de *Perfect Hashing* [58].

As abordagens existentes para deduplicação de dados segura concentram-se em abordar os grandes desafios de segurança para a deduplicação entre múltiplos utilizadores, especialmente na nuvem. Desativar a deduplicação entre múltiplos utilizadores claramente iria fornecer a maior segurança possível, uma vez que iria impedir alguns dos ataques, nomeadamente fuga de informação entre os utilizadores. Contudo, esta não é uma solução tendo em conta a grande necessidade de otimizar o espaço em disco.

Desafios semelhantes aos apontados anteriormente podem ser encontrados também em outras funcionalidades de armazenamento, como por exemplo na compressão. A aplicação da técnica de compressão sobre dados cifrados reduz fortemente o desempenho desta

técnica, uma vez que se perde a noção de redundância dos dados. Neste caso, cifrar os dados com um esquema determinístico continua a impossibilitar o sucesso desta técnica. O *hardware* confiável pode ser utilizado como solução para resolver estes problemas permitindo, por exemplo, decifrar os dados e aplicar sobre estes as técnicas orientadas ao conteúdo num ambiente de execução protegido e confiável, sem comprometer a confidencialidade e integridade dos mesmos. De seguida são apresentadas as soluções existentes que recorrem a tecnologias de *hardware* confiável para oferecer uma deduplicação de dados segura.

2.3.2 *Deduplicação segura com hardware confiável*

Atualmente existem já algumas abordagens que recorrem à segurança oferecida pelo *hardware* confiável para tornar o armazenamento mais seguro. Por exemplo, Baracaldo et al. [6] apresentou uma plataforma que reconcilia criptografia de ponta-a-ponta com compressão e deduplicação, garantindo confidencialidade dos dados. Esta plataforma introduz um módulo confiável, *Trusted Decrypter* (TD), que é colocado no lado do fornecedor de serviços de armazenamento, e que é responsável por procurar as chaves dos clientes, decifrar os dados, processá-los e reconfigurar os dados compactados e deduplicados no armazenamento do servidor. Embora esta arquitetura esteja planeada tendo em conta as características das tecnologias de *hardware* confiável, esta plataforma não possui ainda nenhuma implementação real com *hardware* confiável.

No que diz respeito a deduplicação segura recorrendo a SGX existem ainda poucos sistemas. Dang and Chang [20] apresentaram recentemente uma arquitetura de três camadas, com deduplicação no lado do servidor, que permite poupar a sobrecarga de largura de banda provocada pela deduplicação no lado do servidor e ao mesmo tempo evitar perdas de informação da deduplicação do lado do cliente. Com recurso ao SGX, implementaram um protocolo de deduplicação que garante privacidade e confidencialidade dos dados e das informações de propriedade. A arquitetura proposta (figura 4) é composta por três camadas: o servidor de armazenamento (S), um *proxy* local (P) e os clientes (C). Nesta estrutura, os clientes (por exemplo, os funcionários de uma empresa) conectam-se ao *proxy* da sua empresa que, por sua vez, está ligado ao servidor de armazenamento. Os ficheiros duplicados, enviados pelos clientes, são detetados e removidos no *proxy*. A deduplicação ao nível dos *proxies* é realizada no lado do servidor. Esta arquitetura evita que o cliente consiga retirar informações sobre a existência ou não de uma cópia já armazenada no servidor. O SGX foi utilizado para fornecer a privacidade dos dados do protocolo de deduplicação via *hardware*. Com o recurso a esta tecnologia, as aplicações são executadas num ambiente protegido e confiável, e a integridade dos seus códigos pode ser atestada remotamente. Nesta arquitetura, tanto o servidor como os *proxies* estão equipados com processadores SGX.

Para garantir a confidencialidade dos dados são utilizadas duas camadas de criptografia. A camada interna de criptografia é determinística e calculada pelo dono da informação. A chave utilizada é derivada do conteúdo e obtida através do protocolo *blind signature* [14] com o *enclave* do *proxy* da respetiva empresa. A camada externa de criptografia é aleatória e adicionada sobre uma chave secreta do processador SGX, impedindo partes não confiáveis de aprender informações de igualdade de dados terceiros. Esta solução, no entanto, é também vulnerável a ataques *online* de força bruta.

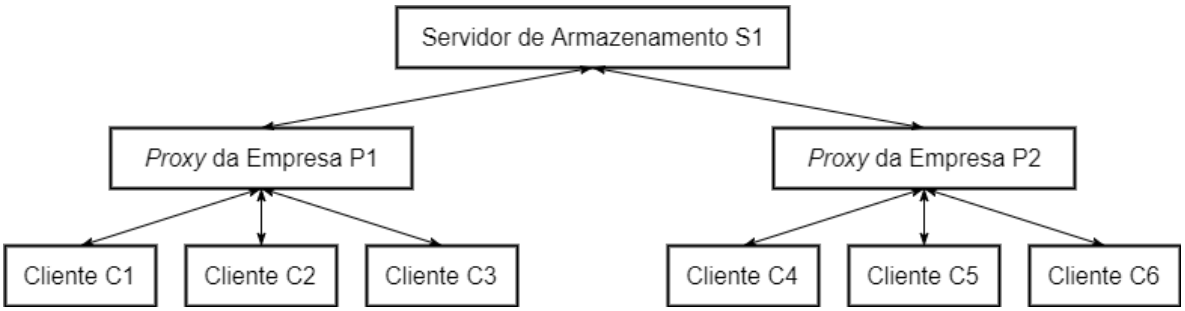


Figura 4: Arquitetura proposta por Dang and Chang

Como já referido, existem muitas poucas soluções para deduplicação que recorrem a *hardware* confiável. A última solução apresentada, uma vez que recorre ao uso do SGX num sistema de armazenamento, torna-se útil para esta dissertação no sentido em que demonstra a possibilidade de integrar esta tecnologia de *hardware* seguro com sistemas de armazenamento. No entanto, a mesma diverge desta dissertação uma vez que apresenta solução para apenas um problema em específico (deduplicação) enquanto que o objetivo desta dissertação foca-se no desenvolvimento de um sistema de armazenamento modular e configurável que permita a execução confiável de qualquer técnica utilizada nestes sistemas.

2.4 Discussão

Com o aumento de informação digital surge a necessidade de desenvolver soluções que consigam combinar as técnicas de redução e de pesquisa de dados com os métodos que fornecem segurança e confidencialidade dos mesmos. A revisão da literatura apresentada previamente permite observar as várias soluções existentes neste âmbito e as suas vantagens e defeitos.

No âmbito de computação confiável existem várias plataformas, focadas em *hardware* confiável, que fornecem ambientes de execução protegida e isolada e que permitem o armazenamento e processamento seguro de dados sensíveis, tentando minimizar os efeitos dos ataques quer físicos quer de *software*. Com o desenvolvimento que tem surgido na área de *hardware* confiável, começam a surgir algumas soluções que recorrem a este tipo de tecno-

logia para aumentar a segurança de diferentes tipos de sistemas. Nomeadamente, existem soluções que incorporam o SGX a diferentes áreas, como base de dados, sistemas de armazenamento, *machine learning*, *etc*. Embora estas soluções demonstrem a viabilidade da incorporação de tecnologias de *hardware* confiável, como o SGX, em sistemas já existentes, todas elas estão limitadas a um problema em específico.

As soluções tradicionais com o objetivo de permitir a aplicação de técnicas orientadas ao conteúdo, como a deduplicação, sobre dados cifrados optam por utilizar, na maioria dos casos, esquemas de criptografia determinísticos, como a criptografia convergente. Este tipo de esquemas permitem gerar o mesmo criptograma para dados idênticos mesmo por diferentes clientes. Deste modo, torna-se possível utilizar certas técnicas que não eram possíveis de aplicar sobre dados cifrados. Contudo, as soluções apontadas apresentam falhas de segurança e não permitem o uso de todas as técnicas de armazenamento orientadas ao conteúdo, como o caso da compressão.

Existem inclusive algumas propostas para deduplicação segura com SGX mas são de novo orientadas a aplicações específicas e esquemas de deduplicação específicos. Por outro lado, não existe ainda nenhuma solução com *hardware* confiável para sistemas de armazenamento modulares e configuráveis. Com esta dissertação pretende-se avançar o estado da arte atual e apresentar uma plataforma que aplique a tecnologia SGX a diferentes funcionalidades de armazenamento sem ser necessário reimplementar completamente os sistemas atuais.

3

ARQUITECTURA

Neste capítulo será apresentada a arquitetura definida para o TrustFS. Esta consiste na extensão da arquitetura do SafeFS de maneira a proporcionar a integração de soluções de *hardware* confiável, particularmente a recente tecnologia da *Intel*: o SGX. Deste modo, inicialmente serão explicados os princípios arquiteturais do sistema SafeFS, nomeadamente a sua programabilidade, modularidade e flexibilidade aquando a implementação, configuração e manutenção do sistema de armazenamento, e posteriormente serão expostas as alterações efetuadas para permitir a integração do SGX. Por último, será explicado o modo de funcionamento do TrustFS, nomeadamente o fluxo usual de um pedido neste sistema.

3.1 SAFEFS: VISÃO GERAL

SafeFS é uma plataforma para sistema de ficheiros empilháveis que permite implementar sistemas de ficheiros baseados na plataforma FUSE, de modo flexível, modular e extensível. Esta plataforma segue uma abordagem de camadas permitindo combinar camadas independentes que implementam funcionalidades de armazenamento específicas, em diferentes ordens. Os objetivos principais da plataforma SafeFS são os seguintes:

EFICÁCIA reduzir o custo da reimplementação de novos sistemas de ficheiros recorrendo a camadas independentes, empilháveis e reutilizáveis.

COMPATIBILIDADE fornecer uma integração de forma simples de sistemas de ficheiros baseados em FUSE como camadas individuais.

FLEXIBILIDADE configurar a pilha de camadas conforme os requisitos de aplicações heterogéneas.

USABILIDADE ser transparente e utilizável como os restantes sistemas de ficheiros FUSE.

Assim sendo, o SafeFS considera dois tipos de camadas: processamento e armazenamento. As camadas de processamento possuem uma função específica que manipula ou transforma os dados/metadados do ficheiro (p.ex., deduplica, comprime, cifra) e reencaminham o resultado para as camadas seguintes. As camadas de armazenamento são res-

ponsáveis pela persistência/pesquisa dos dados no meio de armazenamento designado (disco, armazenamento em rede ou serviços de armazenamento baseados na nuvem). Todas as camadas expõem uma interface idêntica à API fornecida pela biblioteca FUSE, permitindo assim que sejam empilhadas por qualquer ordem. Contudo, a ordem das camadas deve ser escolhida de modo a providenciar o melhor desempenho. Isto é, uma camada de compressão colocada depois de uma camada de encriptação resultará numa compressão ineficiente (dados cifrados possuem um grau de compressão menor que os dados na sua forma original).

Tal como nos outros sistemas baseados em FUSE, cada operação relacionada com sistemas de armazenamento (como *open*, *read*, *write*, *seek*, *flush*, *close*, *mkdir*, *etc*) é intercetada pelo módulo FUSE do *kernel* (figura 5-❶) e redirecionada para o SafeFS pela biblioteca FUSE no espaço do utilizador (figura 5-❷). De seguida, o pedido é ordenadamente transmitido pelas várias camadas, de forma a que cada camada apenas receba pedidos da camada imediatamente acima e envie pedidos apenas para a camada imediatamente abaixo (figura 5-❸). A camada final deve ser sempre uma camada de armazenamento de modo a providenciar persistência ao sistema. No final, o resultado combinado das funções de cada camada equivale à implementação de um sistema de armazenamento.

Algumas funções de armazenamento podem conter algoritmos e implementações alternativas. Por exemplo, a camada de encriptação pode implementar diferentes tipos de cifras (p.e., AES, DES, *etc*), sendo probabilísticas (`PRB`) ou determinísticas (`DET`), através de diferentes modos de operação (ECB, CTR, GCM, *etc*). Outro exemplo é a camada de deduplicação, onde o cálculo do resumo criptográfico pode ser feito com recurso a diferentes funções de cálculo do resumo criptográfico (MD5, SHA256, *etc*). Contudo, estas implementações alternativas partilham o mesmo fluxo de execução das mensagens. Por isso, de modo a maximizar a reutilização do código de uma camada e assim tornar a plataforma mais modular, o SafeFS fornece a noção de *driver*. Respeitando uma dada API, cada camada pode implementar diferentes quantidades de *drivers* (algoritmos/implementações alternativas). Essa API pode ser alterada de acordo com a especialização e as características da camada.

3.2 Arquitetura do TrustFS

O TrustFS estende a arquitetura da plataforma SafeFS de modo a dotar o sistema com soluções de *hardware* confiável, como o SGX, e permitir a execução segura de funcionalidades de armazenamento orientadas ao conteúdo em serviços/servidores não confiáveis. De um modo geral, o servidor não confiável onde o TrustFS será executado, irá receber dados cifrados de múltiplos clientes remotos pela rede, enquanto que o TrustFS recorrerá ao ambiente de execução isolado do SGX (*enclave*) para executar as computações necessárias sobre os dados na sua forma original.

De modo a ser compatível com o SGX, a arquitetura do TrustFS (figura 5) introduz um componente *middleware* (*proxy* SGX) que atua como intermediário entre o ambiente de execução não confiável e o ambiente seguro do *enclave*. Este proxy é responsável pela gestão do ciclo de vida do *enclave* (criação, inicialização e destruição) e por reencaminhar pedidos e receber as respostas respetivas de um determinado *enclave*.

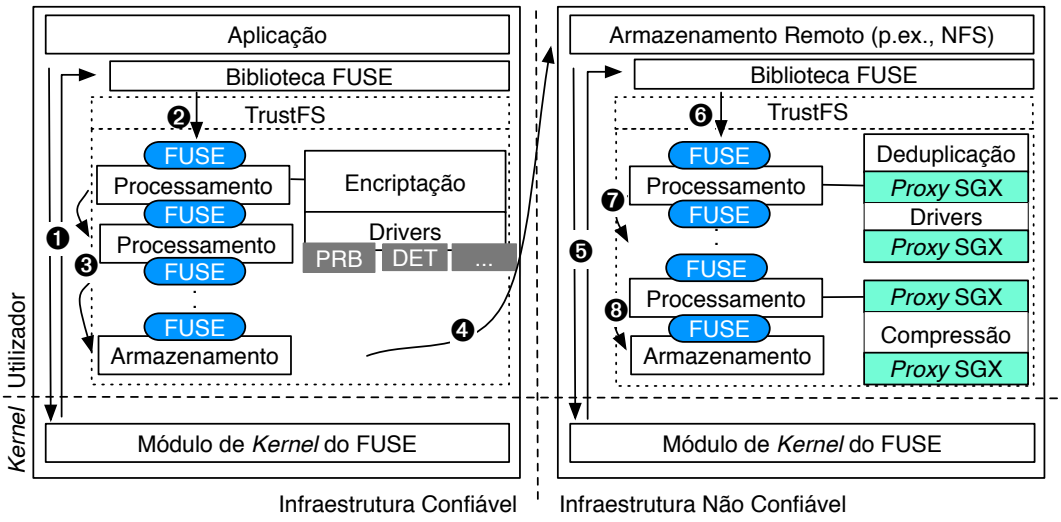


Figura 5: Arquitetura do TrustFS

Mantendo a arquitetura e os princípios do SafeFS, o TrustFS foi desenhado de modo a maximizar a reutilização do código existente das camadas e das *drivers*, permitindo que esse mesmo código executasse dentro do *enclave*. Contudo, devido às limitações do SGX, nem todas as funções/bibliotecas podem ser executadas dentro do *enclave*, sendo necessário em alguns casos alterar/replicar algum do código existente. Como tal, o TrustFS disponibiliza duas opções: aplicar o *proxy* do SGX ao nível da camada (toda a camada é executada no ambiente confiável) ou ao nível da *driver* (apenas as operações críticas são executadas no *enclave*).

Por um lado, existem casos em que toda a camada é considerada crítica, isto é, possui muitas operações que necessitam ser executadas num ambiente confiável. Por exemplo, numa camada de compressão, para atingir o melhor desempenho, é necessário decifrar os dados, aplicar o algoritmo de compressão/descompressão e voltar a cifrá-los. Todo este processo é considerado crítico, sendo por isso um bom exemplo para aplicar o *proxy* do SGX ao nível da camada inteira (como a camada de compressão representada na figura 5). Nestes casos, toda a camada é executada no ambiente seguro do SGX permitindo aceder aos dados na forma original sem que qualquer outro processo ou até mesmo o próprio *kernel* tenha acesso. Isto implica que o *proxy* esteja preparado para receber os pedidos da

camada superior, redirecioná-los para o *enclave* e receber o resultado que irá reencaminhar para a camada inferior. Além disso, é necessário atender à memória necessária, uma vez que ainda existem certas restrições de memória no SGX.

Por outro lado, colocar o código da camada inteira dentro do *enclave* pode nem sempre ser a melhor opção, quer por questões de memória quer por questões de desempenho. Por exemplo, no caso da deduplicação de dados, o tamanho do índice e dos metadados pode superar os limites de memória impostos pela atual arquitetura do SGX. Além disso, por questões de limitações de memória ou por questões de persistência e tolerancia a dados, o índice e os metadados têm de ser armazenados em disco. Desta forma, era necessário efetuar vários pedidos de I/O entre o *enclave* e o meio de armazenamento onde estes metadados estão persistidos, o que é substancialmente mais lento do que o fazer diretamente no ambiente não confiável. Isto introduz um impacto no desempenho do sistema que pode vir a ser significativo. Assim sendo, o TrustFS permite aplicar um *proxy* do SGX ao nível da *driver* (como na *driver* da camada de deduplicação representada na figura 5). Ou seja, apenas o código crítico da *driver* é processado no ambiente confiável. No caso da deduplicação, a camada recebe o pedido e invoca o *enclave*, que por sua vez decifra, calcula o resumo criptográfico dos dados originais e devolve apenas esse resumo à aplicação, tornando assim possível processar os dados cifrados sem que o servidor tenha acesso à informação na sua forma original.

O proxy SGX é então utilizado para ligar os dois componentes: o código não confiável do servidor com o código do *enclave*. Este possui as funções necessárias para criar e inicializar o *enclave* com os dados necessários, assim como invocar as funções confiáveis com os dados no formato exigido.

3.3 Fluxo dos pedidos do TrustFS

A arquitetura do TrustFS foi motivada pelo modelo cliente-servidor remoto, onde o cliente possui dados sensíveis que pretende armazenar, num serviço ou servidor, de forma segura. Em mais detalhe, o cliente recorre a cifras probabilísticas para preservar a confidencialidade e integridade dos dados que envia para o servidor. Simultaneamente, o servidor visa armazenar de forma segura os dados dos múltiplos clientes e aplicar sobre os mesmos técnicas de redução de espaço sem nunca comprometer a confidencialidade dos mesmos.

Desta forma, o fluxo de pedidos do TrustFS segue o percurso similar ao do SafeFS. Considerando o caso de um pedido de escrita enviado pelo cliente para o servidor. Assume-se o estabelecimento de um canal seguro entre a aplicação cliente e o *enclave* do servidor, durante a inicialização da aplicação. Este canal seguro é utilizado para efetuar a troca dos dados necessários do cliente com o *enclave*, como por exemplo a chave que o cliente utilizou para cifrar os seus dados. O estabelecimento de um canal seguro e a troca de chaves não

são o foco desta dissertação, mas seria possível recorrer a protocolos já existentes, como os apresentados por Bahmani et al. [5] e Pass et al. [51], e integrá-los com o TrustFS.

Após a inicialização, o cliente cifra os dados e envia-os para o servidor através do protocolo de armazenamento remoto (por exemplo, NFS). De forma a tornar estes passos transparentes para o cliente é possível executar uma instância do TrustFS na máquina do cliente, configurada com a camada de encriptação juntamente com uma camada de armazenamento remoto. Novamente, os pedidos intercetados pelo FUSE *kernel* são redirecionados para o FUSE no espaço do utilizador (figura 5-❶) e encaminhados para o TrustFS do cliente (figura 5-❷), passando pelas várias camadas (figura 5-❸) até serem enviados para o servidor (figura 5-❹). No servidor não confiável, é executado um *daemon* para o protocolo de armazenamento remoto (NFS), enquanto que os dados são requisitados/armazenados pela pilha de camadas do TrustFS. Tal como referido anteriormente, o modulo FUSE do *kernel* interceta os pedidos de armazenamento e reencaminha-os para a biblioteca no espaço do utilizador (figura 5-❺), atingindo a camada do topo da instância do TrustFS, por exemplo a camada de deduplicação (figura 5-❻).

Uma vez que os dados estão cifrados (de forma probabilística), a camada de deduplicação necessita de decifrar estes dados de forma a conseguir encontrar dados redundantes. Assim sendo, os pedidos de escrita que chegam à camada de deduplicação são redirecionados pelo *proxy* SGX para um *enclave* que decifra e calcula o resumo criptográfico dos dados de forma segura. De seguida, o *proxy* devolve à camada de deduplicação o resumo criptográfico dos dados e esta continua a efetuar as restantes operações de deduplicação. No final da computação, a camada de deduplicação redireciona o pedido para a camada seguinte, por exemplo, a camada de compressão (figura 5-❼). Aplicar a camada de compressão após a camada de deduplicação permite poupar ainda mais espaço de armazenamento através da compressão dos blocos únicos que a camada de deduplicação persistir em disco. Neste caso, o *proxy* SGX redireciona os dados para o *enclave* que decifra, comprime e cifra os blocos comprimidos antes de os devolver ao servidor. O pedido é novamente redirecionado para as camadas seguintes até os dados serem persistidos pela camada de armazenamento (figura 5-❽).

A resposta do pedido consiste no sucesso ou insucesso da operação em causa e segue o caminho reverso ao do pedido, sendo propagada pelas diversas camadas de volta até ao *daemon* de armazenamento remoto. Nessa altura, a resposta é transmitida para a máquina do cliente e divulgada à aplicação do utilizador. O fluxo das restantes operações (leituras ou operações sobre os metadados de ficheiros/diretorias) é muito similar ao das escritas, aqui apresentado. No entanto, no caso das leituras, ocorre o processo inverso. Por exemplo, no caso de um pedido de leitura de um dado ficheiro, os respetivos blocos são enviados ao *enclave* da camada de compressão onde são decifrados com a chave segura do servidor, descomprimidos e cifrados com a chave do cliente.

4

PROTÓTIPO

As técnicas orientadas ao conteúdo utilizadas pelos serviços de armazenamento podem ter uma ou mais operações críticas, onde é necessário aceder aos dados decifrados sem comprometer a confidencialidade dos mesmos. Isto torna-se possível recorrendo a soluções de *hardware* confiável.

O protótipo do TrustFS, conforme descrito anteriormente permite combinar diferentes camadas e recorrer ao SGX de forma a alavancar operações sobre conteúdo de forma segura. Este capítulo utiliza a deduplicação como caso de estudo para validar as funcionalidades desta plataforma. Mostramos ainda que ao utilizar esta plataforma é possível integrar sistemas existentes de deduplicação numa nova camada e dotar estes sistemas de um novo esquema de deduplicação segura que recorre ao SGX.

Neste capítulo será explicado em detalhe o funcionamento geral de um sistema de deduplicação e apresentado o novo esquema de deduplicação segura. Por último, será apresentada a pilha de camadas do protótipo e o seu modo de funcionamento.

4.1 DEDUPLICAÇÃO: VISÃO GERAL

A deduplicação é uma técnica de redução de espaço bastante utilizada por serviços de armazenamento, que se baseia na redução das diversas cópias de um dado ficheiro ou bloco. Como descrito na secção 2.3, esta técnica pode ser aplicada em sistemas centralizados ou distribuídos e a diferentes níveis de granularidades (ao nível do ficheiro ou ao nível do bloco). Além disso, dependendo dos requisitos do serviço de armazenamento e/ou da aplicação, a deduplicação pode seguir uma abordagem *in-line*, onde os duplicados são identificados e removidos diretamente no caminho critico dos pedidos a disco, ou uma abordagem *off-line* onde os duplicados são identificados e removidos como uma tarefa em *background*.

Em ambas as abordagens, e quando a deduplicação é feita ao nível do bloco, tipicamente existe um índice e um conjunto de receitas de metadados onde se encontram os resumos criptográficos dos blocos armazenados, bem como quais os blocos que compõem cada ficheiro e os respetivos endereços de armazenamento. Na figura 6 estão representados, por

uma caixa com limites azuis, os componentes típicos de um sistema de deduplicação *in-line*, como o DDUMBFS [66] e o LESSFS [60]. As setas pretas representam o fluxo típico de uma operação.

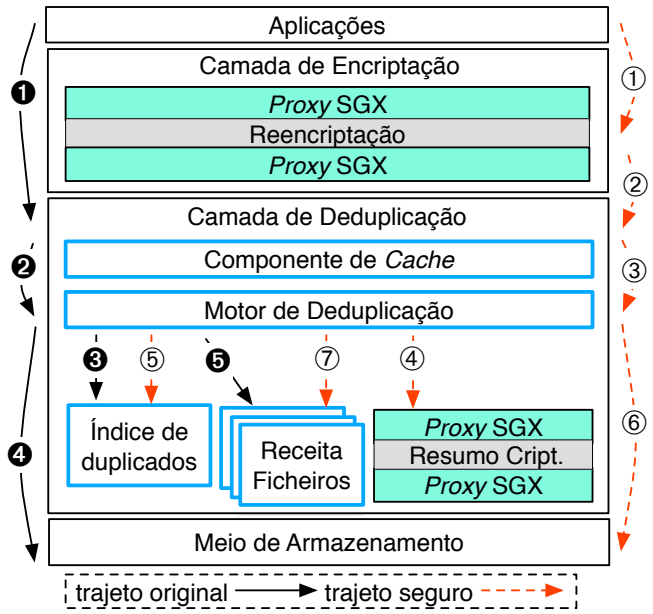


Figura 6: Componentes de reencriptação e deduplicação e fluxo de operações do TRUSTFS

Primeiramente, os pedidos de escrita de um ficheiro são intercetados e o ficheiro é dividido em blocos de tamanho fixo. Normalmente, o tamanho do bloco é escolhido como um parâmetro de configuração aquando a inicialização do sistema. De seguida, esses blocos são guardados temporariamente num componente de *cache* (figura 6-❶) e são posteriormente armazenados no meio de armazenamento, dependendo da política de *flush* da *cache* definida pelo sistema.

Antes de serem armazenados, é calculado o resumo criptográfico de cada bloco (figura 6-❷) e recorre-se ao índice dos resumos criptográficos para verificar, de forma eficiente, se já foi armazenada anteriormente alguma cópia dos blocos (figura 6-❸). Caso não seja encontrada nenhuma cópia do bloco significa que se trata de um bloco único e procede-se ao armazenamento do mesmo (figura 6-❹), guardando uma referencia para o endereço desse bloco na receita de metadados do respetivo ficheiro que esta a ser escrito (figura 6-❺). Caso exista já algum bloco idêntico no meio de armazenamento, o novo bloco não precisa de ser armazenado. Neste caso, basta guardar na receita de metadados do ficheiro uma referência para o endereço de armazenamento do bloco idêntico já armazenado.

Quando o mesmo ficheiro está a ser lido, se os blocos que o constituem se encontrarem em *cache*, estes são imediatamente devolvidos. Caso contrário, recorre-se à receita dos metadados do ficheiro para descobrir quais os endereços onde os blocos daquele ficheiro se encontram armazenados de forma a retornar o conteúdo do ficheiro ao utilizador.

Estes sistemas de deduplicação possuem uma arquitetura complexa que requer o processamento dos dados na sua forma original (como é o caso do cálculo do resumo criptográfico) e a partilha de informação entre múltiplos utilizadores. Aplicar esta técnica sobre dados cifrados pelo cliente torna-se muito complicado/ineficiente principalmente tendo em conta que cada cliente possui a sua própria chave. Para poder aplicar este tipo de técnicas orientadas ao conteúdo sobre dados cifrados recorre-se a esquemas de criptografia convergente [10], que permitem gerar criptogramas iguais para ficheiros idênticos, ou a soluções de *hardware* confiável, como o SGX [20], onde as operações críticas são executadas num ambiente isolado e confiável.

4.2 Camada de Deduplicação

O protótipo do TrustFS contempla uma nova camada de deduplicação que integra as implementações dos sistemas de deduplicação DdumbFS e LessFS. Desta forma, é possível tirar partido da técnica de deduplicação como uma camada empilhável e permitir ao utilizador escolher a solução de deduplicação que melhor se adapta aos requisitos das aplicações.

Com o intuito de integrar a tecnologia SGX nestes sistemas e assim oferecer uma camada de deduplicação segura, substituíram-se as funções críticas da camada, nomeadamente o cálculo do resumo criptográfico, de ambos os sistemas para atuarem como *drivers* da camada, permitindo assim não só recorrer ao *enclave* mas também adicionar suporte a novas implementações (SHA-256, MD5, *etc*). Assim, no ficheiro de configuração do TrustFS é possível definir a solução de deduplicação pretendida e escolher entre as diferentes implementações de cálculo do resumo criptográfico disponibilizadas. É ainda possível escolher entre executar no modo não confiável ou recorrer ao SGX, sendo que na segunda opção as funções das *drivers* são executadas dentro do *enclave*.

A integração do SGX na camada de deduplicação tinha como objetivo permitir detetar duplicados entre múltiplos utilizadores de forma segura e sem ser necessário recorrer à partilha de uma chave de encriptação entre os vários clientes. Isto apresenta dois desafios. Em primeiro lugar, tendo em conta que cada utilizador tem uma chave de cifragem única, dois utilizadores com um bloco idêntico vão gerar criptogramas diferentes, dificultando assim a deteção dos blocos duplicados. Além disso, o cliente pode ainda recorrer a esquemas de criptografia probabilística, sendo que o mesmo bloco cifrado com a mesma chave dará sempre origem a criptogramas diferentes. Em segundo lugar, atendendo ao facto que o servidor armazena apenas uma cópia de cada bloco, essa cópia tem de poder ser acedida por todos os clientes que a possuem, sem ser necessário recorrer a um protocolo de partilha de chaves. Por exemplo, o cliente A cifra um bloco b e envia ao servidor para proceder ao seu armazenamento. Posteriormente, o cliente C envia também uma cópia do mesmo bloco ao servidor, mas cifrada com a sua própria chave. O servidor deteta que se trata do mesmo

bloco e apenas dá um apontador para o bloco original ao cliente *C*. Uma vez que o bloco armazenado corresponde ao bloco cifrado pelo cliente *A*, quando o cliente *C* tentar aceder ao bloco não vai conseguir decifrá-lo. Uma solução para este problema seria a partilha de uma mesma chave de cifragem entre os vários clientes. No entanto, este tipo de protocolos costuma apresentam uma grande complexidade e novos desafios de segurança.

De modo a resolver estes desafios, o protótipo do TrustFS possui dois *enclaves*, um *enclave* que é utilizado para reencriptar de forma segura os dados do cliente (`EnclaveRE` – na camada de encriptação) e outro para efetuar o cálculo do resumo criptográfico (`EnclaveCRC` – na camada de deduplicação). Aquando o processo de inicialização do sistema, ocorre uma troca de chaves entre o cliente e o `EnclaveRE` do servidor via um canal de comunicação seguro, sendo que este *enclave* possui a chave de encriptação de cada cliente. Esta informação encontra-se protegida pelos mecanismos do SGX, não sendo acedida pelo servidor. Com a introdução do `EnclaveRE`, os dados são recifrados com a chave segura do servidor aquando a operação de escrita e novamente recifrados com a chave do cliente aquando a operação de leitura. Desta forma é possível armazenar no lado do servidor os dados dos múltiplos clientes no mesmo formato e evitar a necessidade de partilhar uma chave de cifragem entre os múltiplos utilizadores do sistema. O processo de reencriptar os dados é feito numa camada à parte (camada de encriptação) e implementado como uma *driver* de forma a permitir ao programador escolher entre as diferentes implementações de cifras assim como fornecer novos algoritmos. Além disso, esta separação permite reutilizar este processo para vários casos de estudo, para além da deduplicação, que precisem desta funcionalidade. Por outro lado, como os dados estão cifrados com o mesmo algoritmo e com a mesma chave (chave segura do servidor), o `EnclaveCRC` na camada de deduplicação consegue facilmente decifrar e calcular o resumo criptográfico do conteúdo original dos dados.

A figura 6 apresenta o fluxo das operações com este esquema seguro (setas vermelhas). O primeiro passo corresponde ao estabelecimento de um canal seguro entre o utilizador e o `EnclaveRE`. Após efetuada a troca das chaves do cliente com o *enclave*, o cliente pode começar a efetuar pedidos para o servidor.

Para cada pedido de escrita, os blocos do ficheiro são cifrados pelo utilizador e enviados via NFS para o servidor remoto de deduplicação. No servidor, o pedido é inicialmente tratado pela camada de encriptação que recorre ao `EnclaveRE` para reencriptar os blocos (figura 6-①). Ou seja, decifra cada bloco com a chave do cliente e volta a cifrá-lo com uma chave segura do servidor. Esta chave é gerada pelo *enclave* e armazenada em disco através do mecanismo de *sealing* (descrito na secção 2.1.3) sendo apenas acessível pelo *enclave*. Quando o cliente requisita os blocos, ocorre o processo inverso, isto é, os blocos são decifrados com a chave segura do servidor e cifrados novamente com a chave do cliente que fez o pedido.

De seguida, os blocos reencriptados são encaminhados para a camada de deduplicação, onde serão temporariamente guardados em cache (figura 6-②) até serem armazenados no meio de armazenamento do sistema. Nessa altura (figura 6-③), será efetuado o cálculo do resumo criptográfico de cada bloco recorrendo à execução segura da função no `EnclaveCRC` (figura 6-④) . Uma vez que os blocos chegam ao *enclave* cifrados com um esquema probabilístico, o código da *driver* contempla a decifragem do bloco com a chave segura do servidor e a aplicação da função HMAC-SHA2 ao bloco na sua forma original, de forma a gerar o resumo criptográfico do mesmo. No fim, apenas o resumo criptográfico do bloco é devolvido ao servidor, permitindo assim pesquisar pelos duplicados do bloco (figura 6-⑤), armazenar o bloco cifrado com um esquema probabilístico e com a chave segura do servidor (figura 6-⑥) e atualizar a receita de metadados do ficheiro (figura 6-⑦).

Este conjunto de passos permite que o protótipo consiga oferecer um cálculo de resumos criptográficos de forma segura e, ao mesmo tempo, fornecer garantias de segurança equivalentes aos tradicionais esquemas de criptografia convergente.

4.3 Deduplicação baseada em épocas

A utilização da técnica de deduplicação sobre dados cifrados apresenta grandes desafios, em específico, conseguir atingir um elevado desempenho sem comprometer a confidencialidade e privacidade dos dados do cliente.

Através da análise do estado da arte (como descrito na secção 2.3) é possível concluir que as soluções apresentadas para este desafio baseiam-se em criptografia convergente. Com esta técnica, diferentes utilizadores com o mesmo documento conseguem derivar a mesma chave de cifragem e, recorrendo a esquemas determinísticos, geram o mesmo criptograma. Desta forma, é possível proteger os utilizadores de serviços da nuvem maliciosos ou de atacantes que estejam à escuta da rede a tentar ler dados sensíveis que estão a ser armazenados, e ao mesmo tempo alcançar um elevado desempenho em termos de espaço poupado (equivalente a realizar deduplicação sobre dados não cifrados). Contudo, esta solução apresenta ainda algumas falhas de segurança, nomeadamente permitir ao atacante deduzir se determinado ficheiro já foi ou não previamente persistido.

A solução de deduplicação do TrustFS, tal como os outros esquemas de criptografia convergente, mantém o índice e as receitas de metadados do ficheiro na memória ou no meio de armazenamento do servidor de forma desprotegida, tornando-se assim suscetível aos mesmos ataques de deteção de blocos duplicados da criptografia convergente. No entanto, como os dados são cifrados com esquemas probabilísticos e transmitidos por um canal seguro para o servidor, um atacante que esteja à escuta da rede já não consegue tirar as mesmas ilações, reduzindo assim a vulnerabilidade do sistema apenas para o lado do servidor.

Para reduzir ainda mais a fuga de informação desta solução segura de deduplicação, TrustFS introduz uma abordagem de deduplicação baseada em épocas. Esta abordagem contempla a noção de épocas, ou seja, períodos temporais, entre os quais existe diferentes chaves de época. O `EnclaveCRC` gera no início de cada época uma chave que é utilizada como segredo na derivação dos resumos criptográficos. Desta forma, blocos idênticos que pertencem à mesma época originaram o mesmo resumo criptográfico e serão deduplicados, atingindo propriedades de segurança equivalentes à criptografia convergente. Blocos iguais pertencentes a diferentes épocas não serão deduplicados, uma vez que terão resumos criptográficos diferentes, mas atingirão o mesmo nível de segurança oferecida pela criptografia probabilística. O uso da função HMAC para o cálculo do resumo criptográfico permite recorrer ao SGX para periodicamente gerar uma chave de época diferente e assim introduzir uma noção de épocas. Desta forma, apenas serão deduplicados blocos que pertençam à mesma época, possibilitando assim oferecer uma segurança superior à dos esquemas de criptografia convergente entre as diferentes épocas.

As épocas podem ser determinadas quer por intervalos de tempo ou por número de mensagens (pedidos I/O) e correspondem a um parâmetro central que permite balancear entre a segurança e a eficácia do sistema, permitindo escolher entre maior segurança ou mais espaço poupado.

De forma a permitir aceder aos dados na sua forma original, o protocolo de deduplicação baseada em épocas é executado juntamente com um componente externo de encriptação (`EnclaveRE`). Este componente é utilizado para decifrar os dados com a chave do cliente e cifrá-los novamente com a chave segura do servidor. Desta forma, o *enclave* que calcula o resumo criptográfico dos blocos necessita estar em acordo com esse componente externo de forma a utilizar o mesmo mecanismo de decifragem e a mesma chave segura do servidor.

Assim, o `EnclaveRE` é constituído por dois métodos principais: *trusted_enclave_encode* e *trusted_enclave_decode* (como representado no excerto de código 4.1). O primeiro método, *trusted_enclave_encode*, equivale a decifrar os dados com a chave do cliente e cifrar de novo com a chave segura do servidor. O método *trusted_enclave_decode* executa o processo contrário: decifrar com a chave segura do servidor e voltar a cifrar com a chave do cliente.

```
int trusted_enclave_encode(uint8_t *dest, size_t dest_size, uint8_t* src,
    size_t src_size);
int trusted_enclave_decode(uint8_t *dest, size_t dest_size, uint8_t* src,
    size_t src_size);
```

Excerto de Código 4.1: Métodos do *enclave* da camada de encriptação

O `EnclaveCRC`, na camada de deduplicação, possui apenas uma função principal denominada por: *trusted_compute_hash* (como representado pelo excerto de código 4.2). Esta função contempla o processo de atualizar a chave de época (quando necessário), decifrar os dados e calcular o resumo criptográfico.

```
int trusted_compute_hash(uint8_t *digest, size_t digest_size, uint8_t* src,
    size_t src_size) {
    newEpoch(epoch_key);
    decrypt(src, src_size, plaintext, plaintext_size);
    tag(digest, epoch_key, plaintext);
    return digest;
}
```

Excerto de Código 4.2: Métodos do *enclave* da camada de deduplicação

Este protocolo torna-se viável graças à tecnologia de *hardware* confiável utilizada, uma vez que num sistema com uma arquitetura típica de criptografia convergente seria necessário partilhar e sincronizar a chave de época pelos múltiplos clientes.

4.4 Configuração de camadas do protótipo

O protótipo TrustFS, como extensão da arquitetura do SafeFS, inclui as camadas de granularidade, encriptação e de armazenamento local, assim como as respetivas *drivers*.

A camada de granularidade atua como um *middleware* entre camadas que operam os dados em diferentes níveis de granularidade. Por exemplo, a biblioteca do espaço do utilizador de FUSE reporta operações de escrita e leitura com diferentes tamanhos. No entanto, certos algoritmos de cifragem ou deduplicação funcionam com um tamanho de bloco específico. Esta camada divide os dados em blocos com o tamanho configurado de forma a permitir o funcionamento da camada seguinte. Possuir esta divisão de blocos como camada permite reutilizar código e evita a implementação desta transformação de granularidade em cada camada. A camada de encriptação é utilizada para a proteção de informação sensível através da cifragem dos dados. Esta camada possui várias *drivers* que implementam diferentes esquemas criptográfico. Os dados escritos ou lidos são intercetados por esta camada e são respetivamente cifrados ou decifrados. A camada de armazenamento local permite redirecionar as operações de armazenamento para diferentes diretorias locais na máquina onde a informação está a ser persistida.

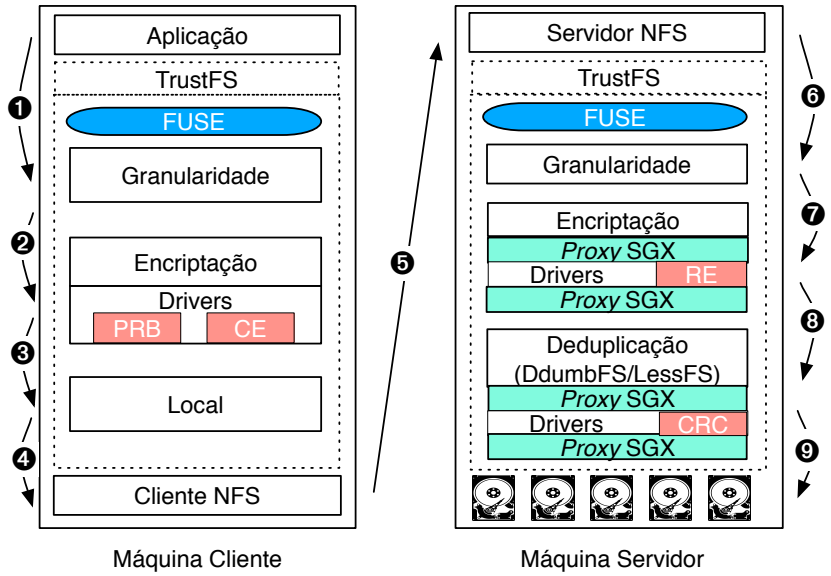


Figura 7: Protótipo do TrustFS e respetiva pilha de camadas.

O protótipo TrustFS apresenta duas configurações diferentes: uma que equivale ao esquema seguro de deduplicação proposto nesta dissertação e outra que exemplifica a solução do estado da arte (criptografia convergente proposta por Bellare et al.). Em ambas as configurações, o protótipo segue um modelo cliente-servidor para um sistema remoto de deduplicação.

ESQUEMA SEGURO DE DEDUPLICAÇÃO COM SGX No lado do cliente, a aplicação recorre a uma instância do TrustFS, configurada com as três camadas anteriormente referidas, para proceder à escrita/leitura de ficheiros. Uma vez que o cliente pretende manter a confidencialidade dos seus dados é necessária a utilização da camada de encriptação para cifrar os dados. O protótipo acrescenta ainda uma nova *driver* com implementação de um esquema de criptografia probabilística, nomeadamente com o modo GCM (*Galois/Counter Mode*). Como referido anteriormente, este tipo de algoritmos trabalha sobre blocos de dados com um tamanho específico, por isso a camada superior do sistema corresponde à camada de granularidade (figura 7-❶), responsável por criar uma abstração de blocos para a camada seguinte: camada de encriptação (figura 7-❷). Por último, a instância do TrustFS do cliente possui uma camada de armazenamento local (figura 7-❸) que redireciona os pedidos do sistema de armazenamento para a diretoria onde se encontra montado o cliente NFS-v3 (figura 7-❹). O servidor possui um *daemon* NFS-v3 a executar, que recebe os pedidos vindos do cliente (figura 7-❺) e os reencaminha para uma outra instância do TrustFS responsável por persistir/pesquisar os dados (figura 7-❻). Esta instância é constituída por uma camada de granularidade, uma camada de encriptação e uma camada de deduplicação. Tal como no caso do cliente, a camada de granularidade é necessária para abstrair a lógica de blocos

para a camada seguinte (figura 7-❼). A camada de encriptação é utilizada para, recorrendo à *driver* de reencriptação segura com SGX, reencriptar os dados com a chave segura do servidor (`RE`). Uma vez reencriptados, os blocos são reencaminhados para a última camada: camada de deduplicação (figura 7-❽). Esta camada é configurada para executar as soluções de deduplicação do DdumbFS ou LessFS e recorrer à *driver* de cálculo seguro do resumo criptográfico dotada de SGX (`CRC`). Como camada terminal, atua também como camada de armazenamento, responsável pela persistência dos dados no meio de armazenamento local (figura 7-❾).

SOLUÇÃO DO ESTADO DA ARTE: CRIPTOGRAFIA CONVERGENTE De forma a contemplar uma solução do estado da arte foi adicionada uma nova *driver* à camada de encriptação que implementa o algoritmo da criptografia convergente. Em primeiro lugar aplicam-se as funções de SHA-2 para calcular o resumo criptográfico de cada bloco. De seguida, cifra-se o bloco com um esquema determinístico recorrendo ao modo de operação GCM e um IV fixo, e utiliza-se resumo criptográfico do bloco como chave de encriptação. Para esta solução, a configuração das camadas do lado do cliente é a mesma da solução dotada de SGX, com a exceção da *driver* utilizada na camada de encriptação. Neste caso, o sistema é configurado para utilizar a *driver* da camada de encriptação correspondente à implementação do algoritmo de criptografia convergente. No lado do servidor, tendo em conta que diferentes clientes geram o mesmo criptograma para blocos idênticos, não é necessário recorrer ao `EnclaveRE` da camada de encriptação para reencriptar os dados, nem ao `EnclaveCRC` para decifrar os mesmos antes de calcular o resumo criptográfico. Além disso, como ambas as implementações do DdumbFS e do LessFS estão preparadas para tratar de blocos com tamanhos variáveis, não é necessária a utilização da camada de granularidade. Desta forma, a instância do TrustFS do servidor é configurada apenas com a camada de deduplicação (e a respetiva *driver* do cálculo do resumo criptográfico num ambiente não confiável).

5

METODOLOGIA

Com o intuito de avaliar o impacto da deduplicação segura em ambientes de execução confiável foram realizados testes de desempenho para dois cenários diferentes. Numa primeira fase, executaram-se micro testes de forma a avaliar o impacto no desempenho introduzida pelo SGX nas operações seguras do protótipo. Posteriormente, executaram-se testes remotos que avaliaram a sobrecarga de desempenho do protótipo TrustFS num ambiente de armazenamento remoto e realista.

Neste capítulo será detalhado a metodologia utilizada, desde a descrição dos diferentes testes e cenários, à configuração experimental utilizada para avaliar o TrustFS.

5.1 MICRO TESTES

O protótipo do TrustFS contempla duas operações críticas que são executadas no ambiente seguro oferecido pelo SGX: a reencriptação de um bloco (`RE`) e o cálculo do resumo criptográfico de um bloco cifrado de forma probabilística (`CRC`). De forma a analisar o desempenho isolado destas operações seguras foram realizados micro testes para avaliar a execução isolada destas operações no *enclave*.

Cada teste contempla as trocas de ambiente de execução (do ambiente não confiável para o confiável e vice-versa), a computação da operação em si e a troca dos dados entre o *enclave* e a aplicação. A operação `RE` considera a reencriptação de um bloco, ou seja, o processo de decifrar os dados com a chave do cliente e cifrá-los novamente com a chave segura do servidor. A função `CRC` corresponde a decifrar um bloco com a chave segura do servidor e calcular o resumo criptográfico dos dados originais. O algoritmo destas funções equivale ao utilizado no protótipo TrustFS, descrito na secção 4.3.

Os testes foram executados para três cenários diferentes. O primeiro cenário corresponde à execução no ambiente não confiável recorrendo às funções criptográficas e de calculo de resumos criptográficos da biblioteca `OpenSSL` (v1.1.0f [71]). A avaliação deste cenário é importante uma vez que permite, posteriormente, comparar os resultados e verificar o impacto no desempenho introduzido pelo SGX. Os outros dois cenários recorrem ao ambiente de execução confiável. O cenário `SGX_SDK` utiliza as funções da biblioteca oferecida pelo SDK

do SGX ([36]) enquanto que o cenário `SGX_SSL` recorre às funções oferecidas pela biblioteca SGXSSL ([37]) - biblioteca com as primitivas do OpenSSL compatíveis com o SGX.

De forma a avaliar o impacto da transferência de dados entre o *enclave* e a aplicação executaram-se os testes para cinco tamanhos de dados diferentes: 4KB, 16KB, 32KB, 64KB e 128KB. Cada teste foi executado durante 10 minutos e repetido 5 vezes para cada operação e por cada cenário, com os diferentes tamanhos de dados.

5.2 TESTES REMOTOS

O segundo conjunto de testes avalia o protótipo de deduplicação do TrustFS num ambiente de armazenamento remoto realístico, com duas máquinas isoladas: cliente e servidor. Para cada teste foram medidos o desempenho, os recursos utilizados e o impacto no espaço poupado. Estes testes foram divididos em duas fases:

Impacto da integração Comparação entre as versões originais dos sistemas DumbFS e LessFS (`Vanilla`) com o protótipo do TrustFS que integra ambas as implementações na camada de deduplicação (`Layered`). Neste caso, o protótipo do TrustFS contempla apenas a camada de deduplicação e com a *driver* do cálculo do resumo criptográfico padrão, isto é, sem SGX. Estes sistemas não consideram dados protegidos pelo cliente, pelo que os dados são enviados para o servidor na sua forma original.

Desempenho da solução segura de deduplicação com SGX. Comparação entre o esquema seguro de deduplicação dotado de SGX do TrustFS (`SGX`) e a solução tradicional do estado da arte que recorre a criptografia convergente (`CE`). Neste caso, ambas as soluções consideram a proteção dos dados pelo que os dados são cifrados no lado do cliente antes de serem enviados para o servidor.

Para avaliar o protótipo utilizou-se o *benchmark* `DEDISbench` (commit #8633fc7 [52, 53]) como gerador de dados no lado do cliente. A escolha deste *benchmark* deve-se à sua capacidade de simular conteúdo realístico e diferentes padrões de acesso de armazenamento. Além disso, este *benchmark* já foi utilizado várias vezes para avaliar o desempenho de diferentes sistemas de deduplicação ([55, 75, 79]).

Os testes realizados contemplaram diferentes tipos de operações I/O e vários padrões de acesso ao disco: escritas e leituras com padrões de acesso sequenciais ou aleatórios. À exceção das escritas sequenciais, todos os restantes testes consideraram a escrita prévia de um ficheiro de 32GB.

Os testes de escrita e leitura para avaliar a sobrecarga da integração, `Vanilla` e `Layered`, foram configurados com um tamanho do bloco igual a 4KB. No caso dos testes de desempenho da solução segura de deduplicação com SGX e da solução de criptografia convergente o tamanho do bloco foi configurado para 4068KB. A razão deste tamanho deve-se ao facto

do esquema criptográfico utilizado adicionar um *padding* de 28 bytes (12 bytes do vetor de inicialização – `IV` e 16 bytes de autenticador da mensagem – `MAC`) e tanto o DdumbFS como LessFS apenas suportarem tamanhos de blocos que sejam potências de 2. Assim, optou-se por gerar blocos com o tamanho de 4068KB de forma a que o servidor recebesse blocos cifrados de 4KB, e evitando assim ser intrusivo no código destes dois sistemas. Uma vez que os testes de ***Sobrecarga da integração*** são apenas para avaliar o desempenho dos sistemas originais, os dados não são cifrados, podendo neste caso ser utilizado um tamanho de bloco de 4KB. Esta configuração do tamanho do bloco corresponde apenas à configuração do *benchmark* (lado do cliente). A configuração do tamanho do bloco dos sistemas de armazenamento (lado do servidor), quer os `Vanilla` quer o próprio protótipo, foi em todos os casos igual a 4KB. No caso das camadas de granularidade do lado do cliente, a configuração do bloco foi então de 4068KB para que, após a camada de encriptação, os blocos ficassem com um tamanho de 4KB.

Cada teste foi repetido 3 vezes com um tamanho de escrita/leitura de 32GB, no caso dos testes sequenciais, ou por 20 minutos, no caso dos testes aleatórios. O `DEDISbench` foi configurado para utilizar a distribuição de conteúdo *dist_highperformance* para os testes de escrita de forma a imitar um cenário de armazenamento realístico com cerca de 17.5% de blocos duplicados nos 32GB escritos/lidos.

Como protocolo de armazenamento remoto entre o cliente e o servidor utilizou-se o `NFSv3`. Inicialmente optou-se pela versão mais recente deste protocolo (`NFSv4`), contudo verificou-se que o desempenho da versão 3 era superior. O `NFS` do lado do cliente foi configurado com a opção *async*, permitindo que o cliente não fique bloqueado à espera da resposta do servidor e aumentar assim a taxa de transferência de dados.

Como referido anteriormente, os testes `Vanilla` foram efetuados com as versões originais do DdumbFS (v1.1 commit #1aa2b08 [66]) e do LessFS (v1.7 commit #5a753c4 [60]) montados na máquina do servidor como sistema de armazenamento local, para o qual eram encaminhados os pedidos de armazenamento intercetados pelo *daemon* do `NFS` do servidor. Nos testes `Layered`, o sistema de armazenamento TrustFS foi configurado no lado do servidor com uma única camada de deduplicação que integra o código do LessFS e do DdumbFS, podendo escolher uma abordagem ou outra via o ficheiro de configuração do TrustFS.

Os testes de `CE` e do `SGX` seguiram uma configuração idêntica à discutida na secção 4.4. Para ambos os casos, uma instância do TrustFS foi montada na máquina do cliente com a camada de granularidade, a camada de encriptação, e a camada de armazenamento local. Nos testes de `CE` utilizou-se um esquema de criptografia convergente para cifrar os dados, enquanto que nos testes de `SGX` foi utilizado um esquema de criptografia probabilística (AES-GCM). Os pedidos foram reencaminhados pelo `NFS` do cliente para o armazenamento remoto do servidor que por sua vez possuía também uma instância do TrustFS montada.

No caso dos testes de CE, esta instância contemplava apenas a camada de deduplicação. No entanto, no caso dos testes de SGX, uma vez que era necessário reencriptar e aceder ao *enclave* para processar os dados, o TrustFS foi configurado não só com a camada de deduplicação mas também com as camadas de granularidade e encriptação, assim como a *driver* da camada de deduplicação com suporte SGX.

5.3 Setup experimental

Os testes foram realizados em máquinas idênticas, com um CPU dual core 3.90GHz (Intel Core i3-7100 CPU), com 8GB de Memória de Acesso Aleatório (RAM) e um SSD SK SC311 SATA de 128GB.

As máquinas comunicavam entre si através de um switch de 1 gigabit e executavam uma distribuição do Ubuntu 16.04.4 LTS de 64-bit com a versão 4.4.0 do Linux kernel e o sistema de ficheiros Ext4.

A versão de *hardware* do SGX corresponde ao SGX 1.0 e foram instaladas as versões v2.1.102 do SDK e da Intel SGX Platform Software (PSW) do SGX.

6

AVALIAÇÃO EXPERIMENTAL

Neste capítulo serão apresentados os resultados dos testes executados para avaliar o protótipo do TrustFS. Numa primeira fase serão descritos os testes de desempenho das operações seguras executadas num ambiente de execução confiável (secção 6.1). De seguida serão apresentados os resultados dos testes remotos do protótipo (secção 6.2). No final serão retratados os recursos utilizados nas diferentes vertentes testadas.

6.1 MICRO TESTES

Nos micro testes foram medidos o débito (MB/s) e a latência (μs) alcançados pelas operações seguras de reencriptar (`RE`) e calcular o resumo criptográfico (`CRC`) de um bloco. Os gráficos das figuras 8 e 9 representam os resultados obtidos para os diferentes cenários (`OpenSSL`, `SGX_SDK` e `SGX_SSL`) e para os vários tamanhos de dados (4KB, 16KB, 32KB, 64KB e 128KB).

Nos cenários que recorrem ao SGX (`SGX_SDK` e `SGX_SSL`), cada operação segura considera a troca de contexto, isto é, a troca do ambiente não confiável para o ambiente confiável e vice-versa, assim como a transferência de dados entre a aplicação e o *enclave*. Contrariamente, ao executar as mesmas operações num ambiente de execução não seguro não é necessária esta troca de contextos e de dados. Por essa mesma razão, o cenário `OpenSSL` (que não recorre ao SGX), alcançou, em ambos os testes e para todos os tamanhos de blocos, melhor desempenho que os restantes cenários.

Para a operação `RE`, os cenários com SGX atingiram os melhores resultados de desempenho para o tamanho de bloco de 32KB, nomeadamente 931.4 MB/s para o `SGX_SDK` e 1327.4 MB/s para o `SGX_SSL` ($\approx$54.84% e $\approx$78.11% do desempenho do `OpenSSL`, respetivamente). Os resultados das latências mostram uma diferença similar com uma média de 22.17μs para o `SGX_SDK` e 16.63μs para o `SGX_SSL`.

Estes resultados mostram que com blocos de tamanhos mais pequenos existe uma sobrecarga maior, devido ao aumento da quantidade de invocações às funções do *enclave* (para a mesma quantidade de informação, se os dados forem partidos em blocos mais pe-

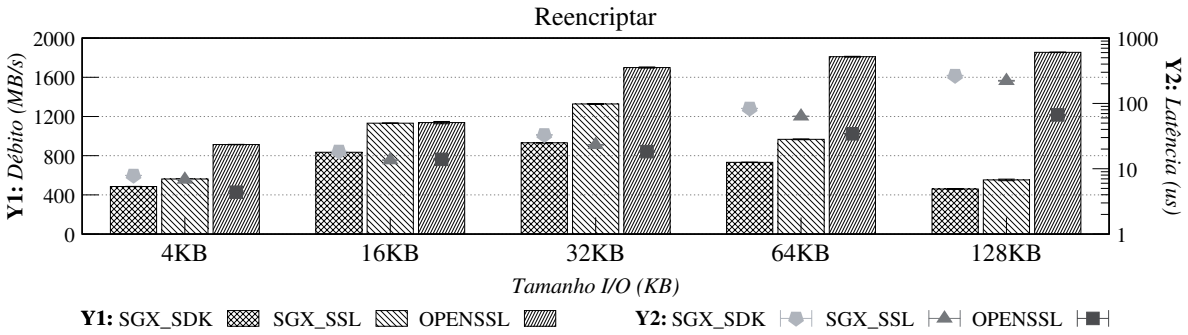


Figura 8: Resultados da operação de re-cifrar(RE)

quenos é necessário invocar mais vezes o *enclave*). Por outro lado, blocos com tamanhos maiores introduzem também uma sobrecarga significante uma vez que, embora o número de invocações ao SGX diminua, o tamanho dos dados necessários de transferir entre a aplicação e o *enclave* aumenta.

Verifica-se também que o cenário SGX_SSL apresenta melhores resultados que o cenário SGX_SDK. Esta diferença de desempenho é justificada pelas diferentes implementações das funções utilizadas para o cálculo da operação, nomeadamente $sgx\_rijndael128GCM\_encrypt$ no caso do SGX_SDK e $EVP\_Encrypt\{Init|Update|Final\}$ no caso do SGX_SSL.

Com o tamanho do bloco igual a 16KB verifica-se uma diferença menor entre o cenário não confiável (OpenSSL) e os dotados de SGX, sendo que o SGX_SSL atingiu ≈99.48% do desempenho do OpenSSL e o SGX_SDK alcançou ≈73.39%. Com isto é possível concluir que este tamanho de bloco oferece o melhor balanceamento entre o número de invocações ao *enclave*, o tamanho de dados a serem transferidos para dentro/fora do *enclave* e do processamento das operações efetuadas no *enclave*.

O gráfico da figura 9 mostra resultados similares para a operação CRC. Uma vez que as funções de cálculo do resumo criptográfico exigem mais do CPU que as funções de cifrar/decifrar, nota-se uma queda de desempenho entre os resultados dos testes de RE (≈1854 MB/s – OpenSSL – melhor desempenho alcançado) e os resultados da operação CRC (≈423 MB/s – OpenSSL – melhor desempenho alcançado). Além disso, o impacto no desempenho introduzido pelas invocações ao *enclave* e pela transferência de dados entre os ambientes confiáveis e não confiáveis encontra-se camuflado pelo impacto das operações do cálculo do resumo criptográfico, o que pode ser observado pela diminuição da diferença de desempenho entre o OpenSSL e os cenários dotados de SGX.

Tal como verificado nos testes da operação RE, devido à diferença de implementação entre as funções utilizadas ($ippsHMAC\_Message\ /HMAC$), o SGX_SSL atingiu sempre melhores resultados que o SGX_SDK. Para ambos os cenários dotados de SGX, o melhor desempenho foi igualmente atingido para blocos com 32KB de tamanho, com 391.9 MB/s no caso do SGX_SSL e 311.3 MB/s no caso do SGX_SDK (≈94.69% e ≈75.21% do desempenho do OpenSSL,

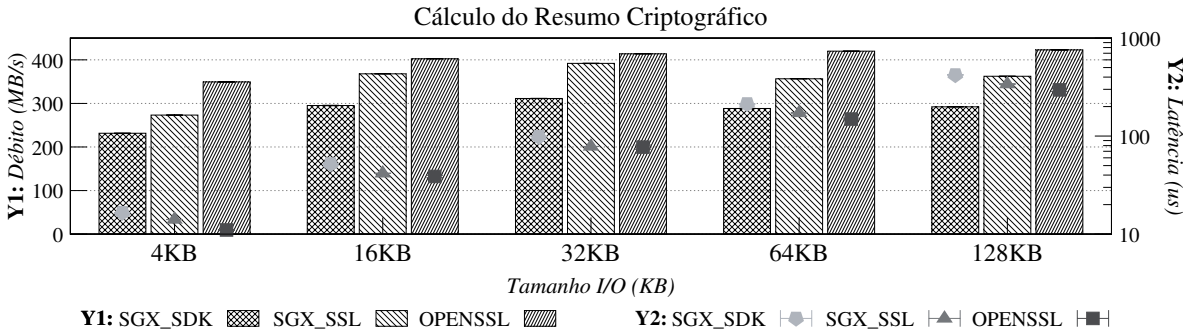


Figura 9: Resultados da operação do cálculo do resumo criptográfico (CRC)

respetivamente). Relativamente à latência, resultados similares podem ser observados, com 52.83 μs para o SGX_SSL e 66.50 μs para o SGX_SDK.

Como descrito na secção 4.3, a operação CRC segue uma abordagem configurável baseada em épocas. Os testes foram repetidos para modificarem a chave segura de época a cada 100 000 operações. A variação do desempenho foi quase nula quando comparada com os resultados do CRC sem épocas, exibindo uma média de variação de desempenho/latência inferior a 1% para os dois cenários dotados de SGX .

Estes testes são importantes para perceber qual o impacto introduzido pelo uso do SGX nestas operações, e concluir quais as melhores configurações para alcançar um melhor desempenho. A implementação do SGX_SSL parece ser a melhor opção para utilizar com SGX. Em todos os testes, o SGX_SSL atingiu sempre melhores resultados que o SGX_SDK, e com o tamanho do bloco igual a 16KB quase que atinge o desempenho do ambiente de execução não confiável. Além disso, o tamanho do bloco que oferece um melhor desempenho aparenta ser de 32KB.

Nos testes remotos do protótipo do TrustFS foram utilizadas as funções do SGX_SSL tanto por este ter mostrado ser uma melhor opção, como por uma questão de compatibilidade para com o código já existente do SafeFS (que recorriam a funções da biblioteca OpenSSL). O tamanho de bloco utilizado nesses testes foi igual a 4KB. A razão desta escolha deve-se ao facto de ser o tamanho típico usado em sistemas de ficheiros e também utilizado em diversos sistemas de deduplicação. Além disso, como verificado pelos resultados dos micro testes, este é o caso em que serão feitas mais invocações ao SGX, sendo mais observável o impacto no desempenho de usar esta abordagem.

6.2 Testes Remotos

De modo a testar o desempenho da solução segura dotada de SGX foram executados testes remotos. Estes testes contemplam um modelo remoto cliente-servidor, onde o cliente cifra os seus dados e envia ao servidor que irá aplicar as técnicas de deduplicação e persistir

os dados. Para perceber o impacto no desempenho introduzido pelo SGX, repetiram-se os mesmos testes para a solução do estado da arte: criptografia convergente. Os resultados permitem tirar conclusões relativamente à integração desta tecnologia de *hardware* confiável num sistema real e remoto. Esta secção apresenta os resultados dos testes remotos, sendo dividida entre dois tipos de testes: *Testes do impacto da integração* e *Testes de desempenho de deduplicação segura com SGX*.

6.2.1 *Testes do impacto da integração*

De modo a entender o impacto no desempenho do protótipo do TrustFS e dos sistemas LessFS e DdumbFS *vanilla*, realizaram-se inicialmente testes (`Native`) para medir a latência (μs) e o débito (MB/s) alcançados num cenário com as máquinas do servidor e do cliente isoladas e a comunicar entre si via o protocolo NFS. O *benchmark* `DEDISbench` efetuou pedidos de escrita/leitura a partir da diretoria do `NFS` do cliente, enquanto que os pedidos foram reencaminhados para o *daemon* de `NFS` do servidor remoto e armazenados através de um sistema de armazenamento `Ext4`. De seguida, foram efetuados os mesmos testes, mas desta vez com as versões originais do DdumbFS e do LessFS montadas como sistema de armazenamento na máquina do servidor (testes `Vanilla`), de forma a medir o impacto no desempenho introduzido por estes dois sistemas. Como terceiro cenário, foram testadas as integrações das implementações destes dois sistemas na camada de deduplicação do TrustFS. Nestes casos, uma instância do protótipo do TrustFS foi montada no lado do servidor apenas com a camada de deduplicação (testes `Layered`).

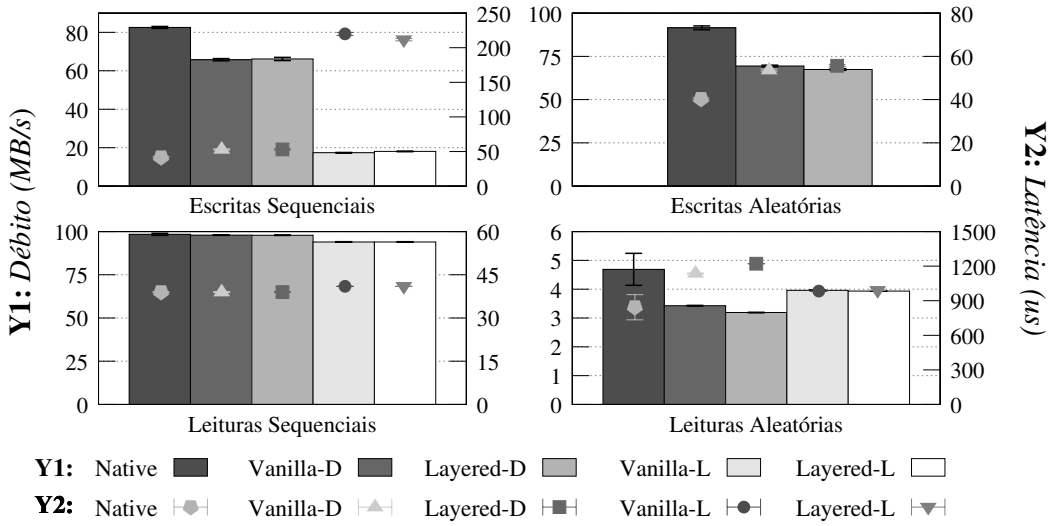


Figura 10: Resultados dos testes remotos `Native`, `Vanilla` e `Layered`

Os gráficos da figura 10 mostram os resultados obtidos para estes três cenários. Como esperado, o cenário `Native` alcançou o melhor desempenho entre os três cenários, atingindo 82.6 MB/s nas escritas sequências e 91.5 MB/s nas escritas aleatórias. As latências foram também as melhores (mais pequenas), sendo registados valores de $\approx 40\mu$s para ambos os testes de escrita. Como explicado anteriormente, o cliente `NFS` foi configurado para executar os pedidos de escrita de forma assíncrona, por isso o cliente não ficou à espera de uma resposta do servidor a indicar se pedido foi corretamente persistido. Isto explica a proximidade de valores obtidos entre os cenários de escrita sequencial e aleatória.

Nos testes de leitura verifica-se igualmente um melhor desempenho do cenário `Native`, com os valores de 98.4 MB/s e 4.7 MB/s para as leituras sequências e aleatórias, respetivamente. Mais uma vez, este cenário apresenta as menores latências também no caso das leituras, com 39μs para as leituras sequenciais e 844μs para as leituras aleatórias. Uma vez que a conexão da rede entre as máquinas era de 1 Gigabit *Ethernet*, nota-se uma limitação no desempenho das leituras sequenciais. Relativamente às leituras aleatórias, o seu desempenho foi constrangido pela forma como o protocolo `NFS` lida com os padrões de acesso ao meio de armazenamento.

Quando comparado com o cenário `Native`, os testes `Vanilla` mostram um impacto no desempenho tanto no caso das escritas sequenciais como nas escritas e leituras aleatórias. No caso das escritas sequenciais, o DdumbFS atingiu um desempenho de 65.78 MB/s com uma latência média de $\approx 53\mu$s ($\approx$80% do desempenho e $\approx$79% da latência do cenário `Native`), e o LessFS alcançou 17.4 MB/s com uma latência de $\approx 220\mu$s ($\approx$21% do desempenho e $\approx$19% da latência do cenário `Native`) . Nas escritas aleatórias, o DdumbFS registou valores de desempenho iguais a 69.43 MB/s também com uma latência de $\approx 53.7\mu$s ($\approx$76% e $\approx$75% do desempenho e da latência do cenário `Native`, respetivamente).

As leituras aleatórias mostram uma queda de desempenho de cerca de 27% em relação ao cenário `Native` no caso do DdumbFS (3.43 MB/s) e à volta de 16% no caso do LessFS (3.96 MB/s). As latências apresentam um impacto similar à do desempenho, 74% (1137μs) e 86% (983μs) do cenário `Native`, para o DdumbFS e LessFS respetivamente. Este impacto é expectável uma vez que é necessário consultar os metadados do índice e as receitas dos ficheiros muitas vezes diretamente ao disco, introduzindo uma demora no tempo de resposta dos pedidos de armazenamento [54].

As leituras sequenciais apresentam-se como uma exceção, com uma sobrecarga pouco significante quando comparado com a versão nativa, mais precisamente 0.5% no caso do DdumbFS e cerca de 6% no caso do LessFS. Novamente, o desempenho deste teste encontra-se limitado pela capacidade da conexão entre as duas máquinas (*switch* de 1 Gigabit), mascarando assim a sobrecarga introduzida pela deduplicação e fragmentação dos blocos.

Em regra geral (à exceção das leituras aleatórias), o sistema DdumbFS apresenta melhores resultados que o sistema LessFS, que pode ser explicado pelas diferentes abordagens

de implementação que cada sistema escolheu. O gráfico das escritas aleatórias apenas apresenta os resultados do sistema DdumbFS uma vez que, devido a um erro de *memory leak* encontrado no sistema LessFS não foi possível concluir estes testes.

Relativamente aos testes `Layered`, estes apresentaram um impacto pequeno quando comparados os resultados destes com os dos sistemas originais `Vanilla`. A maior queda de desempenho notou-se nas leituras aleatórias para o sistema DdumbFS, que apenas atingiu 93% (3.2 MB/s) do desempenho do correspondente teste `Vanilla`.

Estes resultados mostram o impacto introduzido no desempenho esperado ao integrar as implementações dos sistemas DdumbFS e LessFS na camada de deduplicação do protótipo e a queda de desempenho quando comparado com o cenário `Native`.

6.2.2 *Testes de desempenho de deduplicação segura com SGX*

A segunda fase de testes efetuada incidiu na medição do débito (MB/s) e da latência (μs) obtidos ao executar o protótipo num ambiente remoto e com os dois esquemas seguros de deduplicação: o esquema de deduplicação seguro dotado de SGX proposto nesta dissertação (`SGX`) e a solução do estado da arte conhecida por criptografia convergente (`CE`). A instalação de ambos os cenários seguiu as configurações descritas nas secções 4.3 e 5.2.

Os resultados da figura 11 mostram que a solução `SGX` alcançou valores similares de débito e latência aos da solução `CE`. A queda de desempenho verificada pela solução dotada de SGX em relação à solução `CE` foi, no caso das leituras, inferior a 4%. A solução do estado da arte alcançou, nos testes de leitura sequencial, um débito de 96.86 MB/s com a implementação do sistema DdumbFS e 92.43 MB/s com a implementação do sistema

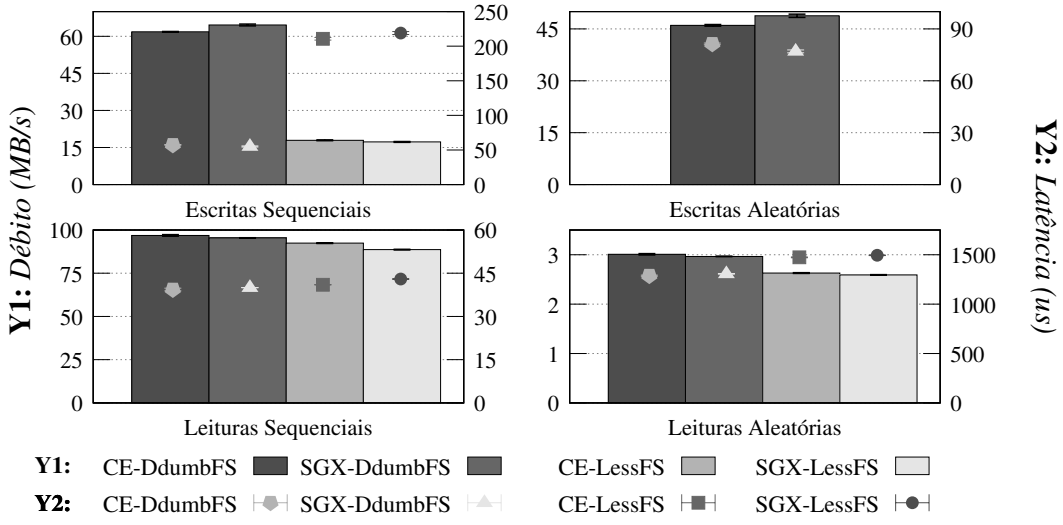


Figura 11: Resultados das soluções seguras de deduplicação `SGX` e `CE`

LessFS, com uma latência média de 39μs e 41μs, respetivamente. A solução SGX atingiu cerca de 98.6% (95.47 MB/s) do desempenho da solução CE com o DdumbFS e quase os 96% (88.67 MB/s) com o LessFS. As latências aumentaram ligeiramente tendo atingido os 40μs e 43μs respetivamente. Nas leituras aleatórias, SGX alcançou, tanto no caso do DdumbFS como no caso do LessFS, 98.6% do desempenho e da latência de CE.

No caso das escritas, os testes SGX com o sistema DdumbFS mostram uma melhoria no desempenho em relação aos testes CE, até 6%. Nas escritas sequenciais a CE atingiu $\approx$ 61.83 MB/s enquanto que o SGX alcançou $\approx$ 64.59 MB/s. Nas escritas aleatórias observou-se a mesma melhoria, com $\approx$ 48.79 MB/s para o SGX e $\approx$ 46.03 MB/s para a CE. Este comportamento pode ser explicado pela opção *async* utilizada nas configurações do NFS do cliente. O processamento feito na camada de encriptação no lado do cliente é maior no caso da solução CE. A solução baseada em SGX efetua o cálculo do resumo criptográfico com base do texto original dentro do *enclave* do lado do servidor. Assim, o cliente apenas necessita cifrar os seus dados. No caso da solução CE, os dados são cifrados com o seu resumo criptográfico como chave de forma a gerarem um criptograma determinístico para permitir ao servidor detetar os duplicados. Desta forma, a camada de encriptação da solução CE não só executa os mesmos passos que a solução de SGX, isto é, cifrar os dados, como também efetua o cálculo do resumo criptográfico para calcular a chave de cifragem. Uma vez que o cliente não fica bloqueado à espera da resposta do servidor, é compreensível a obtenção de melhores resultados para a solução SGX.

De modo a validar esta análise, foram executados mais dois tipos de testes para as escritas sequenciais e aleatória das duas soluções: um teste de *profiling* do tempo despendido na camada de encriptação do cliente, e um teste com a opção *sync* do NFS do cliente ativada, obrigando a que o cliente esperasse pelo fim do processamento do pedido do lado do servidor.

Os resultados dos testes de *profiling* mostraram que, nas escritas sequenciais, a solução CE gasta, em média, $\approx$66μs por operação na camada de encriptação do cliente, enquanto que a solução SGX despende apenas $\approx$63μs. Nas escritas aleatórias verificou-se um comportamento similar, com $\approx$99μs e $\approx$81μs despendidos pela CE e SGX, respetivamente. Como era de esperar, nos testes de escrita sequencial síncrona, a solução SGX já obteve um desempenho menor (0.411 MB/s) e uma latência maior (9.4 ms) em comparação com o desempenho (0.416 MB/s) e a latência (9.3 ms) da solução CE (uma queda de desempenho inferior a 2%). No caso das escritas aleatórias síncronas, a queda de desempenho da solução SGX em relação ao desempenho da solução CE foi inferior a 8%.

6.2.3 *Redução de espaço de armazenamento*

	1 **GB**	5 **GB**	10 **GB**	**Sem épocas**
DDUMBFS	5.38 GB	5.42 GB	5.46 GB	5.6 GB
LESSFS	5.41 GB	5.46 GB	5.49 GB	5.6 GB

Tabela 2: Redução de espaço com a distribuição *dist_highperformance*

Observando os resultados das escritas sequenciais, ambas as soluções alcançaram cerca de 17.5% de redução de espaço, equivalente a 5.6 GB (de 32 GB escritos) de espaço de armazenamento poupado. Estes valores correspondem ao melhor resultado, em termos de poupança de espaço, de um sistema de deduplicação. No caso da solução `CE`, uma vez que os dados são cifrados de forma determinística, foram detetados todos os blocos duplicados, sendo por isso esses resultados os valores base de comparação. Os resultados da solução `SGX` são equivalentes uma vez o sistema foi configurado para executar sem noção de épocas, ou seja, todos os blocos foram processados na mesma época, permitindo assim poupar o mesmo espaço de armazenamento que a solução `CE` (5.6GB).

De modo a verificar o impacto no espaço poupado com o novo esquema baseado em épocas repetiram-se os testes de escrita sequencial (três vezes cada um) para trocar de época a cada 10 GB, 5GB e 1GB de dados escritos. Os resultados representados na tabela 2 vão de encontro aos valores espectáveis: uma diminuição do espaço poupado com o aumento do número de épocas. O impacto no espaço poupado foi mínimo, com uma diminuição inferior a 4% no teste com mais épocas (1 GB).

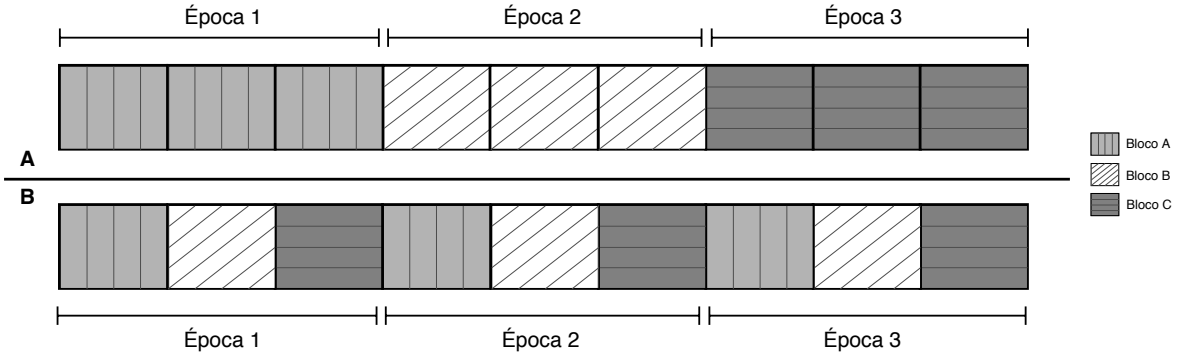


Figura 12: Exemplos de distribuições de blocos duplicados por épocas

No entanto, esta análise é dependente das propriedades de localidade temporal dos dados a armazenar. Por exemplo, em cenários de armazenamento onde se verifica a escrita de uma grande percentagem de blocos duplicados um dado espaço temporal, as épocas devem ser configuradas de forma a estarem de acordo com esse espaço temporal (figura 12–Exemplo `A`). Por outro lado, no caso de um sistema de armazenamento que periodicamente

efetua cópias de segurança, torna-se mais difícil escolher o parâmetro de épocas mais eficiente (figura 12–Exemplo B). No exemplo B existem três blocos diferentes (A,B,C), cujo armazenamento se repete para todos os blocos em todas as épocas. Neste caso, a redução do espaço de armazenamento vai ser bastante afetada. Como as cópias de cada bloco estão distribuídas pelas diferentes épocas, não serão detetadas como duplicadas, sendo que cada bloco será armazenado três vezes. No caso do exemplo A, as épocas coincidem com o espaço temporal dos diferentes blocos. Os blocos idênticos estão todos na mesma época, levando a que cada bloco seja armazenado apenas uma vez.

O DEDISbench imita a distribuição de blocos duplicados especificados na distribuição *dist_highperformance* e segue uma abordagem onde as cópias de um dado bloco são armazenadas uniformemente ao longo do tempo. Os resultados mostram que, para este cenário em específico, é possível ter um impacto reduzido no espaço de armazenamento poupado.

	1 **GB**	5 **GB**	10 **GB**	**Sem épocas**
DDUMBFS	0.85 GB	1.51 GB	2.08 GB	3.72 GB
LESSFS	0.85 GB	1.52 GB	2.09 GB	3.72 GB

Tabela 3: Redução de espaço com a distribuição *dist_personalfiles*

Realizaram-se os mesmos testes com a distribuição *dist_personalfiles* e observou-se um impacto maior consoante o aumento do número de épocas. A percentagem de duplicados desta distribuição é inferior à da distribuição anterior (cerca de 12% de blocos duplicados em 32GB), sendo que no teste sem épocas foram encontradas cerca de 3.72 GB de cópias. Tal como indicado na tabela 3, com a mudança de época a cada 10GB foram deduplicados ≈2.08 GB, o que equivale a 56% do espaço poupado na versão sem épocas. Consoante se aumentou o número de épocas observou-se uma diminuição ainda maior do espaço poupado, sendo que com cerca de 7 épocas alcançaram-se valores perto de 1.5 GB e com 32 épocas, o espaço poupado foi inferior a 1GB (0.85 GB), atingindo os 23% do espaço originalmente poupado.

	# blocos	**# cópias**	**% blocos duplicados com**	
			<32 cópias	**>= 32 cópias**
dist_highperformance	400 599 152	100 199 335	33.23 %	66.77 %
dist_personalfiles	29 530 586	5 081 176	85.1 %	14.9 %

Tabela 4: Análise das distribuições *dist_highperformance* e *dist_personalfiles*

Após a análise das distribuições do DEDISbench chegou-se à conclusão que a maior percentagem de redundância da distribuição *dist_highperformance* provem de blocos com uma grande quantidade de duplicados, enquanto que no caso da distribuição *dist_personalfiles* provem de uma grande quantidade de blocos com poucos duplicados. Como se pode ob-

servar na tabela 4, 33% das cópias encontradas na distribuição *dist_highperformance* resultam de blocos com menos de 32 cópias, enquanto que os restantes 67% resultam de blocos com mais de 32 cópias. Isto significa que, com a mudança de época a cada 1 GB num total de 32 GB de dados escritos, a maior parte dos blocos possuem mais cópias do que as épocas executadas, originando um impacto mínimo no espaço de armazenamento poupado. No caso da distribuição *dist_personalfiles*, 85% dos blocos duplicados provêm de blocos com menos de 32 cópias, sendo que neste caso 32 épocas já introduzem um impacto mais significativo na deteção dos blocos duplicados.

6.3 RECURSOS UTILIZADOS

Com a finalidade de perceber o impacto do SGX nos recursos utilizados foram medidas, para cada um dos testes, a utilização do CPU, da RAM e da rede. A tabela 5 apresenta a média dos resultados obtidos nos testes `Vanilla`, enquanto que a tabela 6 mostra os recursos utilizados (quer na máquina cliente, quer na máquina servidor) nos os vários testes das duas soluções de deduplicação segura (`CE` e `SGX`).

	DDUMBFS		LESSFS	
	CPU	**RAM**	**CPU**	**RAM**
Escritas sequenciais	36.16%	7.15%	**83.26**%	2.9%
Escritas aleatórias	37.25%	7.15%		
Leituras sequenciais	**39.67**%	7.15%	41.61%	0.39%
Leituras aleatórias	10.22%,	7.15%	16.62%	0.3%

Tabela 5: Recursos utilizados (em média) pelos sistemas DDUMBFS e LESSFS

Através dos resultados `Vanilla` é possível obter duas observações interessantes: o sistema LESSFS requer menos utilização de memória RAM que o sistema DDUMBFS (≈3% e ≈7% do total da memória RAM do servidor, respetivamente) mas exige uma maior utilização do CPU, chegando a atingir os 83% no caso das escritas sequenciais, em oposição ao pior caso registado do DDUMBFS de quase 40% (nas leituras sequenciais). As leituras aleatórias apresentam os melhores resultados para ambos os sistemas, com ≈10% para o DDUMBFS e ≈17% para o LESSFS. A diferença observada entre os valores das leituras aleatórias e das leituras sequenciais deve-se ao facto de nas primeiras, uma vez que estão a aceder de forma aleatória, terem de aceder sempre ao disco, contrariamente às leituras sequenciais que podem ser alimentadas pelos valores previamente carregados para cache (*prefetching*). Isto explica igualmente a razão da diminuição do desempenho nas leituras aleatórias. Uma vez que é necessário aceder quase sempre ao disco, requerem menos consumo de CPU mas exige um tempo de resposta maior por cada pedido, diminuindo assim capacidade de resposta de pedidos por segundo.

		DdumbFS				LessFS			
		CPU		**RAM**		**CPU**		**RAM**	
		cliente	servidor	cliente	servidor	cliente	servidor	cliente	servidor
Escritas sequenciais	CE	42.0%	38.52%	13.7%	7.16%	20.79%	83.0%	10.82%	2.9%
	SGX	31.89%	53.55%	0.02%	7.17%	13.44%	87.25%	0.02%	2.93%
Escritas aleatórias	CE	26.12%	**44.65**%	29.97%	7.17%				
	SGX	22.9%	**65.92**%	0.02%	7.18%				
Leituras sequenciais	CE	41.12%	39.64%	21.92%	7.16%	42.07%	**41.1**%	26.41%	0.36%
	SGX	37.28%	48.12%	0.03%	7.17%	36.56%	**49.58**%	0.04%	0.33%
Leituras aleatórias	CE	6.62%	16.28%	18.22%	7.17%	6.58%	21.69%	18.15%	0.28%
	SGX	5.89%	22.45%	0.03%	7.18%	5.77%	26.46%	0.03%	0.3%

Tabela 6: Recursos utilizados (em média) pelas soluções CE e SGX

Os resultados observados nos testes das soluções de deduplicação segura mostram novamente que para a implementação do sistema LessFS, as escritas sequenciais exigem mais do CPU e que as leituras apresentam os melhores resultados, principalmente as leituras aleatórias. Relativamente ao DdumbFS, os piores resultados observados foram para os testes de escrita aleatória, que superaram os valores atingidos nos testes de leituras sequenciais.

Analisando os resultados das duas máquinas, é possível verificar que, na máquina do cliente, a solução CE utiliza uma maior percentagem de CPU que a solução SGX (chegando a atingir uma diferença de ≈10%). Esta diferença no caso das escritas é, mais uma vez, explicada pelo cálculo do resumo criptográfico que a solução CE necessita efetuar no lado do cliente. No caso das leituras, a razão incide na procura da chave para decifrar o bloco (chave essa que correspondente ao resumo criptográfico calculado na operação de escrita).

Relativamente ao consumo de memória ambas as soluções de CE e SGX apresentam valores idênticos no servidor: ≈7% com o DdumbFS e até 3% com o LessFS. No lado do cliente, a solução SGX apresenta um consumo de memória quase nulo, enquanto que a solução CE chegou a consumir uma média de quase 30% que, novamente, se deve à gestão das chaves para cifrar/decifrar os blocos.

Os valores obtidos relativamente ao consumo de rede foram os espectáveis, com o total de MB/s enviados pelo cliente a corresponderem ao total de MB/s recebidos pelo servidor e vice-versa, e a corresponderem aos valores de desempenho obtidos. Os resultados de rede das leituras sequenciais manifestam o limite imposto pela rede, com valores à volta dos 100 MB/s (DdumbFS) e 90 MB/s (LessFS).

Estes resultados mostram que a solução segura de deduplicação do TrustFS dotada de SGX introduz um pequeno impacto no desempenho das operações I/O, com uma diminuição até 2% na maioria dos testes e inferior a 8% no pior cenário (escritas síncronas), quando comparado com a solução do estado da arte. Relativamente à quantidade espaço poupado, a solução proposta apresenta um compromisso entre segurança e economia de espaço, sendo que aumentando o número de épocas, o sistema fica mais resistente a ataques sobre os quais a criptografia convergente é vulnerável, mas diminui o número de blocos duplicados encontrados. Nos testes apresentados, a redução de espaço sofreu uma diminuição inferior a 4% com a distribuição *dist_highperformance* e até 77% com a distribuição *dist_personalfiles*, ambos os casos com a mudança de épocas a cada 1GB. O impacto no desempenho para blocos de 4KB verificada nos micro testes foi aliviada pela computação adicional efetuada pelas diferentes camadas de armazenamento e pelos componentes da deduplicação.

7

CONCLUSÃO

Esta dissertação apresenta o sistema TrustFS, uma plataforma de sistemas de armazenamento empilháveis que recorre à tecnologia SGX para construir sistemas de armazenamento orientados ao conteúdo de forma segura e modular.

Pela análise do estado da arte foi possível concluir que existem várias propostas para encontrar uma solução segura e aceitável que permite conjugar técnicas que garantem a confidencialidade e integridade dos dados, nomeadamente técnicas de criptografia probabilísticas, com técnicas de armazenamento orientadas ao conteúdo, como o caso da deduplicação ou da compressão. A solução mais comum, no caso da deduplicação, mas que ainda apresenta algumas falhas de segurança é a criptografia convergente, que consiste em cifrar os dados com um esquema determinístico e usando o resumo criptográfico dos dados como chave. Por outro lado, existem também várias propostas de arquiteturas que visam oferecer o isolamento de aplicações contra entidades maliciosas, incluindo o próprio sistema operativo ou o hipervisor de um servidor malicioso. No entanto, existem muitas poucas propostas que combinem as tecnologias de *hardware* confiável com sistemas que usem técnicas orientadas ao conteúdo. O aparecimento das novas tecnologias de *hardware* confiável, como o SGX, tem provocado um aumento das propostas de soluções que confiam neste tipo de tecnologia. Contudo, ainda não surgiu nenhuma proposta para uma plataforma modular e programável para sistemas de armazenamento que tire partido deste tipo de tecnologia.

O objetivo da plataforma exposta nesta dissertação consiste em fornecer uma arquitetura modular e programável de forma a simplificar o desenvolvimento de funcionalidades de armazenamento dotadas de SGX, exigindo um esforço de reimplementação reduzido. Desta forma, o código das camadas e *drivers* existentes podem tirar partido desta tecnologia de *hardware* confiável recorrendo a um *proxy middleware*, que faz a comunicação entre a parte não confiável do servidor e o *enclave*. Além disso, o TrustFS possibilita a escolha, aquando o desenvolvimento ou configuração do sistema, do conjunto de camadas/*drivers* que devem executar num ambiente seguro consoante os requisitos da infraestrutura onde a solução do sistema de armazenamento será instalada. Esta dissertação mostrou as capacidades desta plataforma através de um caso de estudo focado na deduplicação, essencialmente. A mo-

dularidade do TrustFS e a sua aposta na segurança oferecida pelo SGX permitiu desenhar um novo esquema de deduplicação baseado em épocas que proporciona diferentes níveis de segurança e espaço poupado.

O protótipo do TrustFS integra dois sistemas de deduplicação *open-source*, nomeadamente os sistemas DdumbFS e LessFS, e recorre ao *enclave* para efetuar o cálculo do resumo criptográfico de cada bloco num ambiente de execução confiável. De modo a permitir que os blocos armazenados sejam acessíveis por todos os clientes que os possuem, os blocos são reencriptados quando chegam ao servidor com a chave segura deste. Desta forma, torna-se possível enviar os dados cifrados do cliente para o servidor (anulando os possíveis ataques realizados por algum individuo à escuta da rede) e permitir ao mesmo tempo a aplicação de técnicas orientadas ao conteúdo. Este processo é apenas viável (seguro) graças ao isolamento oferecido pelo SGX. O cálculo do resumo criptográfico é realizado sobre os dados decifrados e com o auxílio de uma chave de época (apenas acessível pelo *enclave*). Este processo permite introduzir um novo esquema de segurança, impedindo que um atacante tente perceber se dado bloco já se encontra armazenado entre as diferentes épocas. No entanto, ao recorrer a este esquema de deduplicação seguro por épocas, dados idênticos em épocas diferentes não serão deduplicados, reduzindo assim a taxa de sucesso desta técnica de redução de espaço. Desta forma, o esquema proposto deve ser configurado consoante as características temporais dos blocos e consoante os objetivos pretendidos (mais segurança ou mais espaço reduzido).

De modo a avaliar o protótipo TrustFS, implementou-se e testou-se de igual forma a solução do estado da arte: criptografia convergente. Esta comparação revelou que é possível providenciar premissas fortes de segurança com um impacto reduzido no desempenho, inferior a 2% na maioria dos testes realizados e até 8% com as configurações mais exigentes. Além disso, pelos resultados de espaço poupado obtidos é possível concluir que o novo esquema de épocas é bastante dependente dos dados sobre o qual está a ser aplicado e, em especial, das suas características de redundância e distribuição espacial. Desta forma, o número de épocas deve ser ajustado consoante os objetivos do sistema (segurança/redução de espaço) e tendo em conta as características dos dados.

Em conclusão, esta dissertação apresenta uma plataforma para sistemas de armazenamento dotada de SGX que possibilita, tanto aos programadores como aos utilizadores do sistema, construir e executar soluções seguras e flexíveis de armazenamento com as seguranças oferecidas pelo *hardware* confiável.

7.1 TRABALHO FUTURO

O protótipo apresentado nesta dissertação permite desenvolver e executar um sistema de armazenamento que aplica a técnica de deduplicação, para economizar o espaço de armazenamento, sobre dados cifrados de forma segura. Além disso, a arquitetura do protótipo foi desenhada de forma a facilmente recorrer ao espaço reservado do SGX (*enclave*) para outros fins além da deduplicação.

Embora o TrustFS ofereça uma camada de deduplicação (implementações dos sistemas DdumbFS e LessFS) e um novo esquema seguro de deduplicação baseado em épocas, seria relevante, como trabalho futuro, desenhar uma camada de deduplicação que tirasse maior partido de novas versões do SGX de forma a aumentar tanto a segurança como o desempenho e sem comprometer o espaço poupado pela técnica. As novas instruções do CPU introduzidas pelo SGX 2.0 possibilitam alocar dinamicamente a memória do *enclave* [45]. Assim, torna-se possível, por exemplo, manter o índice dos resumos criptográfico na memória protegida do *enclave*, evitando assim que o atacante consiga perceber quais os blocos armazenados.

Outro tópico interessante para trabalho futuro seria a integração de protocolos de estabelecimento de canais seguros ([39, 5, 51]) para assegurar uma comunicação segura entre os clientes e os *enclaves* do servidor. Além disso, num sistema distribuído com vários nodos de armazenamento, os utilizadores têm de ser capazes de trocar as chaves com um ou mais nodos. Da mesma forma, os *enclaves* dos diferentes nodos precisam igualmente de trocar a chave segura do servidor entre si, de forma a serem capazes de responder aos pedidos de diferentes clientes. Como tal, seria vantajoso avaliar a utilização de novas técnicas, como os métodos utilizados em [33], para assegurar esta funcionalidade.

Adicionalmente, seria interessante integrar no TrustFS outras funcionalidades de armazenamento orientadas ao conteúdo (como compressão, cache orientada ao conteúdo, *etc*) para tirarem partido da segurança oferecida pelo SGX.

BIBLIOGRAFIA

[1] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee. Obliviate: A data oblivious file system for intel sgx. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[2] T. Alves. Trustzone: Integrated hardware and software security. *White Paper*, 2004.

[3] Amazon. Amazon s3 web page. URL https://aws.amazon.com/s3/. Visitado em 2018-10-09.

[4] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703. USENIX Association, 2016.

[5] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi. Secure multiparty computation from sgx. In *International Conference on Financial Cryptography and Data Security*, pages 477–497. Springer, 2017.

[6] N. Baracaldo, E. Androulaki, J. Glider, and A. Sorniotti. Reconciling end-to-end confidentiality and data reduction in cloud storage. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 21–32. ACM, 2014.

[7] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, pages 267–283, 2014.

[8] M. Bellare and S. Keelveedhi. Interactive message-locked encryption and secure deduplication. In *IACR International Workshop on Public Key Cryptography*, pages 516–538. Springer, 2015.

[9] M. Bellare, S. Keelveedhi, and T. Ristenpart. Message-locked encryption and secure deduplication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 296–312. Springer, 2013.

[10] M. Bellare, S. Keelveedhi, and T. Ristenpart. Dupless: Server-aided encryption for deduplicated storage. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 179–194, Berkeley, CA, USA, 2013. USENIX Association. ISBN 978-1-931971-03-4.

[11] G. R. Blakley and C. A. Meadows. Security of ramp schemes. In *Crypto*, volume 84, pages 242–268. Springer, 1984.

[12] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *ACM SIGMETRICS Performance Evaluation Review*, volume 28, pages 34–43. ACM, 2000.

[13] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *ACM SIGMETRICS Performance Evaluation Review*, volume 28, pages 34–43. ACM, 2000.

[14] D. Chaum. Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, 1983.

[15] C. che Tsai, D. E. Porter, and M. Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658. USENIX Association, 2017.

[16] Clutch. Annual Cloud Computing Survey. https://clutch.co/cloud/resources/annual-cloud-computing-survey-2017, 2017. Visitado em 2018-10-09.

[17] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[18] V. Costan, I. A. Lebedev, and S. Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*, pages 857–874, 2016.

[19] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36(SI):285–298, Dec. 2002. ISSN 0163-5980.

[20] H. Dang and E.-C. Chang. Privacy-Preserving Data Deduplication on Trusted Processors. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pages 66–73. IEEE, 2017.

[21] A. De Santis and B. Masucci. Multiple ramp schemes. *IEEE Transactions on Information Theory*, 45(5):1720–1728, 1999.

[22] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[23] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 617–624. IEEE, 2002.

[24] I. Dropbox. Dropbox web page. https://www.dropbox.com. Visitado em 2018-10-09.

[25] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. Van Doorn, and S. W. Smith. Building the ibm 4758 secure coprocessor. *Computer*, 34(10):57–66, 2001.

[26] P. E. e Conselho da União Europeia. General data protection regulation 2016/679/eu. https://eugdpr.org/, 2016. Visitado em 2018-10-09.

[27] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley. Iso-x: A flexible architecture for hardware-managed isolated execution. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 190–202, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-6998-2.

[28] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A.-R. Sadeghi. Hardidx: Practical and secure index with sgx. *arXiv preprint arXiv:1703.04583*, 2017.

[29] Google. GoogleDrive web page. https://www.google.com/drive/. Visitado em 2018-10-09.

[30] D. Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach*. Intel Press, 1st edition, 2009. ISBN 1934053171, 9781934053171.

[31] T. C. Group. Tpm main specification. https://trustedcomputinggroup.org/tpm-main-specification, 2003. Visitado em 2018-10-09.

[32] T. Guardian. Ofcom tackles mass data breach of tv company information, 2016. URL https://www.theguardian.com/media/2016/mar/10/ofcom-tackles-mass-data-breach-of-tv-company-information. Visitado em 2018-10-09.

[33] D. Harnik, P. Ta-Shma, and E. Tsfadia. It takes two to# metoo-using enclaves to build autonomous trusted systems. *arXiv preprint arXiv:1808.02708*, 2018.

[34] J. S. Heidemann and G. J. Popek. File-System Development with Stackable Layers. *ACM Transactions on Computer Systems (TOCS)*, 12(1):58–89, 1994.

[35] Intel. Building Trust in a Cloudy Sky. https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/02/infographic-building-cloudy-sky.pdf, 2017. Visitado em 2018-10-09.

[36] Intel. Intel® software guard extensions for linux* os, 2018. URL https://github.com/intel/linux-sgx. Visitado em 2018-10-09.

[37] Intel. Intel® software guard extensions ssl, 2018. URL https://github.com/intel/intel-sgx-ssl. Visitado em 2018-10-09.

[38] D. Kaplan, J. Powell, and T. Woller. Amd memory encryption. *White paper, Apr*, 2016.

[39] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863*, 2018.

[40] K. A. Küçük, A. Paverd, A. Martin, N. Asokan, A. Simpson, and R. Ankele. Exploring the use of intel sgx for secure many-party applications. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*, page 5. ACM, 2016.

[41] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou. Secure deduplication with efficient and reliable convergent key management. *IEEE transactions on parallel and distributed systems*, 25(6):1615–1625, 2014.

[42] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. *SIGPLAN Not.*, 35(11): 168–177, Nov. 2000. ISSN 0362-1340.

[43] J. Liu, N. Asokan, and B. Pinkas. Secure Deduplication of Encrypted Data without Additional Independent Servers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 874–885. ACM, 2015.

[44] T. Machida, D. Yamamoto, I. Morikawa, H. Kokubo, and H. Kojima. Poster: A novel framework for user-key provisioning to secure enclaves on intel sgx. 2018.

[45] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, page 10. ACM, 2016.

[46] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014. ISSN 1075-3583.

[47] C. Mitchell. *Trusted computing*, volume 6. Iet, 2005.

[48] S. Mofrad, F. Zhang, S. Lu, and W. Shi. A comparison study of intel sgx and amd memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, page 9. ACM, 2018.

[49] S. B. Mokhtar, A. Boutet, P. Felber, M. Pasin, R. Pires, and V. Schiavoni. X-search: revisiting private web search using intel sgx. In *Middleware*, pages 12–12, 2017.

[50] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pages 619–636, 2016.

[51] R. Pass, E. Shi, and F. Tramer. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 260–289. Springer, 2017.

[52] J. Paulo, P. Reis, J. Pereira, and A. Sousa. DEDISbench. https://github.com/jtpaulo/dedisbench, 2010. Visitado em 2018-10-09.

[53] J. Paulo, P. Reis, J. Pereira, and A. Sousa. Dedisbench: A benchmark for deduplicated storage systems. In *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems"*, pages 584–601. Springer, 2012.

[54] J. a. Paulo and J. Pereira. A Survey and Classification of Storage Deduplication Systems. *ACM Comput. Surv.*, 47(1):11:1–11:30, 2014.

[55] J. a. Paulo and J. Pereira. Efficient deduplication in a distributed primary storage infrastructure. *Trans. Storage*, 12(4):20:1–20:35, 2016.

[56] R. Pontes, D. Burihabwa, F. Maia, J. a. Paulo, V. Schiavoni, P. Felber, H. Mercier, and R. Oliveira. SafeFS: A Modular Architecture for Secure User-space File Systems: One FUSE to Rule Them All. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR '17)*, pages 9:1–9:12. ACM, 2017.

[57] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database using SGX. In *EnclaveDB: A Secure Database using SGX*. IEEE, 2018.

[58] P. Puzio, R. Molva, M. Önen, and S. Loureiro. Perfectdedup: Secure data deduplication. In *International Workshop on Data Privacy Management*, pages 150–166. Springer, 2015.

[59] D. Reinsel, J. Gantz, and J. Rydning. Data age 2025: The evolution of data to life-critical. *Don't Focus on Big Data*, 2017.

[60] M. Ruijter. LessFS: deduplication filesystem. https://github.com/rootfs/lessfs, 2009. Visitado em 2018-10-09.

[61] R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 147–158. ACM, 2005.

[62] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud using SGX. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 38–54. IEEE, 2015.

[63] M. A. Sevilla, N. Watkins, I. Jimenez, P. Alvaro, S. Finkelstein, J. LeFevre, and C. Maltzahn. Malacology: A Programmable Storage System. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 175–190. ACM, 2017.

[64] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[65] S. Shinde, D. Tien, S. Tople, and P. Saxena. Panoply: Low-tcb linux applications with sgx enclaves. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, page 12, 2017.

[66] A. Spineux. DDumbFS: inline deduplication filesystem. https://github.com/eskyuu/ddumbfs, 2011. Visitado em 2018-10-09.

[67] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl. A secure data deduplication scheme for cloud storage. In *International Conference on Financial Cryptography and Data Security*, pages 99–118. Springer, 2014.

[68] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, pages 160–171, New York, NY, USA, 2003. ACM. ISBN 1-58113-733-8.

[69] W. Sun, R. Zhang, W. Lou, and Y. T. Hou. Rearguard: Secure keyword search using trusted hardware. *IEEE INFORM*, 2018.

[70] Surenthx. Overview of intel protected file system library using software guard extensions, 2016. URL https://software.intel.com/en-us/articles/overview-of-intel-protected-file-system-library-using-software-guard-extensions. Visitado em 2018-10-09.

[71] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. https://www.openssl.org, 2003. Visitado em 2018-10-09.

[72] E. Thereska, H. Ballani, G. O'Shea, T. Karagiannis, A. Rowstron, T. Talpey, R. Black, and T. Zhu. IOFlow: A Software-Defined Storage Architecture. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*, pages 182–196. ACM, 2013.

[73] D. L. Whiting and T. Dilatush. System for backing up files from disk volumes on multiple nodes of a computer network, July 7 1998. US Patent 5,778,395.

[74] Z. Wilcox-O'Hearn. Drew perttula and attacks on convergent encryption, 2008.

[75] H. Yu, X. Zhang, W. Huang, and W. Zheng. Pdfs: Partially dedupped file system for primary workloads. *IEEE Trans. Parallel Distrib. Syst.*, 28(3):863–876, Mar. 2017.

[76] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. *HotCloud*, 10(10-10):95, 2010.

[77] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *NSDI*, pages 283–298, 2017.

[78] Y. Zhou, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhang, and C. Li. Secdep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management. In *Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on*, pages 1–14. IEEE, 2015.

[79] Y. Zhou, Y. Deng, L. T. Yang, R. Yang, and L. Si. Ldfs: A low latency in-line data deduplication file system. *IEEE Access*, 6:15743–15753, 2018. ISSN 2169-3536.