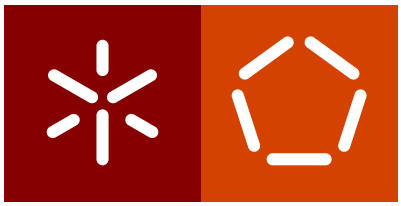
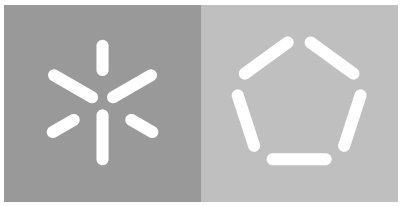


Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Alexandre Pinheiro Marinho

Sistemas Inovadores de Segurança em Bases de Dados

July 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Alexandre Pinheiro Marinho

Sistemas Inovadores de Segurança em Bases de Dados

Dissertação de mestrado
Mestrado Integrado em Engenharia Informática

Dissertação orientada por
Rogério António da Costa Pontes
Rui Carlos Mendes Oliveira

July 2019

AGRADECIMENTOS

Ao meu orientador, Professor Rui Carlos Oliveira, pela sua experiência, conhecimento e profissionalismo, que me motivou a desenvolver uma dissertação com qualidade, e pela oportunidade que me deu para a desenvolver num ambiente rodeado de pessoas que me ajudaram nos mais variados temas abordados na dissertação.

Ao Rogério Pontes, pela incansável ajuda no desenvolvimento da dissertação, paciência para com as minhas distrações e erros, disponibilidade sempre que necessária, por me ter guiado e motivado durante todo o processo, pelas horas que não pode trabalhar nos seus projetos e ajuda inesgotável na revisão dos documentos.

Ao Ricardo Macedo pelos esclarecimentos e toda a ajuda prestada em tópicos relacionados com o sistema base sobre o qual esta dissertação foi realizada.

À Tânia Esteves pela ajuda e discussões, saudáveis e não saudáveis, sobre assuntos relacionados com a tecnologia utilizada.

A todos os meus amigos, que sempre estiveram por perto não só para me tirar o sono e fazer companhia nas noites mais longas, mas também para inventar a alegria e manter o ânimo necessários para encarar o próximo dia com determinação.

Finalmente, um especial agradecimento aos meus pais e irmã, que apesar de não me verem todos os dias, nunca duvidaram das minhas capacidades, sempre estiveram do meu lado nos momentos mais complicados e mais importante, sempre souberam a palavra certa a dizer. Não seria capaz de realizar este trabalho sem a ajuda incondicional deles.

ABSTRACT

In today's world, there is a notorious increase of digital data that is handled, processed and analyzed at a global scale. To handle this data surge, organizations have started to outsource storage and computation to cloud services. These services can store vast amounts of data and handle thousands of concurrent users at a fraction of the cost of what organizations would have to spend to have the same computational power.

Amazon, Google and Microsoft are just some of the cloud market players that over the years gained their clients trust by having highly available and ubiquitous services. However, this trust has been affected by data leaks that compromised organization's confidential data and individuals' privacy.

To address the concerns of existing applications that deal with sensitive and confidential data, several privacy-aware databases technologies have been proposed to securely outsource storage and computation to the cloud. However, the core research effort has been to develop new cryptographic schemes that protect data in encrypted text, so that databases can process queries as if they were plaintext. Furthermore, most of the research has explored SQL databases due to its wide applicability. In this dissertation we propose a different approach to secure databases by proposing a novel privacy-aware NoSQL Database, *TrustNoSQL* that leverages secure hardware processing technologies. In detail, this work has three main contributions. The first, is a detailed state-of-the-art of the current privacy-aware databases, SQL and NoSQL, and the security guarantees ensured by these systems. The second contribution is the system *TrustNoSQL*, the first NoSQL privacy-aware database that securely processes queries with Intel SGX. The final contribution is a detailed system evaluation with an industry-standard benchmark.

RESUMO

Nos dias de hoje, tem-se notado um aumento do número e diversidade de dados digitais que circulam, são tratados, analisados e utilizados à escala global. Os números são significativos, e, por isso, as empresas começam a tomar partido de serviços de terceiros, para beneficiar das vantagens de computação que estes proporcionam.

Posto isto, serviços de nuvem disponibilizados pela *Amazon*, *Google* ou *Microsoft*, são utilizados por essas empresas, que procuram garantias não só de disponibilidade mas também de proteção dos seus dados. Como temos observado ao longo dos anos, os serviços de nuvem têm vindo a sofrer imensos ataques, onde falhas de segurança nos servidores de armazenamento acabam por ser responsáveis pela libertação de enormes quantidades de informação confidencial.

De modo a resolver as preocupações existentes de aplicações que lidam com dados sensíveis e confiáveis foram propostas várias bases de dados capazes de armazenar e processar dados de forma segura na *cloud*. Contudo, o maior esforço de investigação encontra-se em desenvolver novos esquemas criptográficos que protegem os dados em texto cifrado de tal modo que as bases de dados consigam processar interrogações como se fosse texto simples. Esta abordagem apesar de eficiente acaba por libertar informação sensível que pode ser utilizada para quebrar a segurança dos sistemas. Para além disso, a investigação existente tem dado prioridade às bases de dados SQL devido à sua grande aplicabilidade. Esta dissertação toma uma abordagem diferente e apresenta uma nova base de dados NoSQL com processamento seguro, *TrustNosQL*, assente nas propriedades de segurança de *hardware* confiável. Mais precisamente, este trabalho tem três contribuições principais. O primeiro é uma análise compreensiva do estado da arte atual em base de dados com processamento seguro. Este estudo permite posicionar o sistema apresentado em relação às capacidades e propriedades de segurança dos sistemas existentes. A segunda contribuição é a base de dados NoSQL com processamento seguro, *TrustNoSQL*, a primeira base de dados NoSQL que processa de forma segura as interrogações utilizando a tecnologia Intel SGX. A última contribuição é uma extensa avaliação do sistema apresentado com uma plataforma de avaliação de base de dados reconhecida pela indústria.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE INTERFACES

LISTA DE ABREVIATURAS

A

AES Advanced Encryption Standard. 11, 17, 20
API Application Programming Interface. 21
ARM Advanced RISC Machine. 6, 8
AWS Amazon Web Services. 1

C

CPU Central Processing Unit. 4, 5, 8

D

DBMS Database Management System. 10–14, 17
DET Deterministic Encryption. 12, 13, 15
DRAM Dynamic Random Access Memory. 4–9

H

HOM Homomorphic Encryption. 12, 13, 15

I

I/O Input/Output. 4, 14, 15
IBM International Business Machines. 16, 23

J

JNI Java Native Interface. 27
JOIN JOIN e OPE-JOIN. 12, 13

L

LLC Last Level Cache. 9

O

OPE Order-Preserving Encryption. 12–14

R

RND Randomized Encryption. 12, 13

S

SEARCH Word Search. 12, 13
SGX Software Guard Extensions. 3, 8, 22–24, 27
SoC System-On-Chip. 6
SQL Structured Query Language. 10–12, 20, 23
SRAM Static Random Access Memory. 6

T

TCB Trusted Computing Base. 4–9
TPM Trusted Platform Module. 4, 5
TXT Intel's Execution Engine. 5

U

UDF User-defined functions. 12

V

VMX Virtual Machine Extensions. 5

W

WISR Worldwide Infrastructure Security Report. 1

Y

YCSB Yahoo! Cloud Serving Benchmark. 28

1

INTRODUÇÃO

Ao longo dos anos, com o desenvolvimento e globalização da tecnologia, temos assistido a um aumento exponencial de quantidade e diversidade de dados digitalizados que circulam. Posto isto, tornou-se regular a utilização de serviços *online*, de modo a atribuir o esforço computacional e de armazenamento a serviços de terceiros. Os benefícios da utilização deste serviço são bem conhecidos: recursos virtualmente infinitos, ausência de custos de manutenção e investimento em infraestruturas e o acesso em qualquer lugar, através de qualquer dispositivo e a qualquer momento. A utilização de bases de dados de terceiros tornou-se um passo lógico para muitas empresas tecnológicas, de modo a reduzir custos e fornecer uma boa qualidade de serviço. As *clouds* como fornecedoras de serviços que armazenam dados pessoais e empresariais, são responsáveis não só por assegurar uma disponibilidade constante dos mesmos como também pela confidencialidade da informação que armazenam.

De modo a acompanhar a digitalização global, os dados não podem perder a sua disponibilidade, isto é, todos os dados têm de estar acessíveis em qualquer altura e por vários meios de acesso. Já são várias as empresas que nos disponibilizam estes serviços, como *Amazon Web Services (AWS)* [1], *Google Drive* [9], *Microsoft OneDrive* [14] e *Dropbox* [7].

A *Google Drive* anunciou em Maio de 2017 que já conta com mais de 800 milhões de utilizadores ativos [26] e mais de dois triliões de ficheiros guardados nos seus servidores [34]. Esta é apenas uma pequena percentagem da real utilização destes serviços, sendo que empresas tecnológicas executam centenas de tarefas nestes serviços para processar grandes quantidades de dados todos os dias. Por exemplo a *Yelp* usa o *Amazon S3* para armazenar diariamente *logs* e fotos, produzindo cerca de 1,2TB de *logs* por dia [15].

Com a adoção destes serviços de terceiros para armazenamento e computação, os clientes das *clouds* abdicam do controlo total dos dados. A segurança e privacidade dos dados fica dependente dos mecanismos de segurança e confidencialidade existentes nos fornecedores do serviço. Dado que estes serviços têm vindo a ser vítimas de imensos ataques, é necessário resolver vários desafios de segurança emergentes. Falhas de segurança nos servidores de armazenamento acabam por ser responsáveis pela libertação de enormes quantidades de informação crítica e privada dos clientes. No 11º Relatório Anual sobre Segurança da Infraestrutura Global Worldwide Infrastructure Security Report (WISR) da

Arbor Networks [6], foi divulgado que nos 11 anos (2005-2016) que envolveram esta pesquisa, o tamanho do maior ataque cresceu em mais de 60 vezes [3]. O estudo divulga também que em 2012 19% dos participantes observaram ataques direcionados aos seus serviços, sendo que, dois anos mais tarde, esse número aumentou para 24% e em 2016 para 33%.

1.1 PROBLEMA E OBJETIVOS

Nos dias de hoje, as bases de dados desempenham um papel fundamental no funcionamento das empresas, seja no armazenamento dos seus dados ou serviços que proporcionam. Posto isto, estas empresas estão reticentes quanto à utilização dos serviços de *cloud*, uma vez que são geridos e administrados por terceiros.

Uma das soluções para este problema passa pela utilização de técnicas criptográficas, de modo a manter os dados protegidos contra adversários maliciosos, que tenham o objetivo de espiar, corromper ou roubar a informação. No estado da arte atual já existem propostas para resolver este problema utilizando diferentes esquemas criptográficos, onde o processamento pode ocorrer sobre os dados cifrados, como acontece com o *CryptDB* [33], *Monomi* [38], *SafeNoSQL* [28] e *Opaque* [40]. O *SafeNoSQL* [28] é um exemplo de um sistema baseado no processamento de conjuntos restritos de cálculos a serem executados sobre dados cifrados, como igualdades e pesquisas sobre bases de dados NoSQL.

Os clientes que utilizam computação em nuvem, tomam partido da segurança apenas dos dados armazenados estaticamente (*Encryption at Rest*), sendo que estes não estão protegidos durante o processamento e transferências dos dados. Já existem bases de dados comerciais que integram estes mecanismos de segurança dos dados estáticos (e.g. *MySQL* [20], *Postgres* [30]), contudo, o processamento seguro ainda é um desafio presente. Ainda assim, podemos facilmente identificar alguns problemas nestas técnicas, desde a seleção de técnicas criptográficas não recomendadas, até ao desenho da arquitetura dessas soluções e à sobrecarga causada pelas primitivas de processamento sobre dados cifrados. Outra solução passa por manter os dados, ou apenas a computação sobre esses dados, na aplicação com a qual o cliente tem contacto direto. Assim, o cliente tem a certeza que os dados estão desprotegidos apenas na sua máquina, sendo que sobrecarregaria a aplicação do lado do cliente.

Outra solução passa pela utilização de *hardware* confiável, que permite estudar novos compromissos entre segurança e eficiência nas bases de dados, propondo uma solução alternativa a bases de dados protegidas com esquemas criptográficos. Já foram desenvolvidos vários sistemas assentes nestas propriedades do *hardware* confiável, como o *Opaque* [40] e *TrustedDB* [19]. Contudo, nenhum destes sistemas analisou as garantias de segurança que são possíveis em bases de dados não relacionais e qual o seu impacto em termos de desempenho.

De modo a tomar partido da ampla distribuição de máquinas com processadores *Intel* e com o objetivo de garantir a segurança em zonas críticas de código, a *Intel* propôs recentemente nos seus novos processadores, unidades de processamento isoladas (*Intel Software Guard Extensions (SGX)*), que nos permitem abordar o problema da segurança em bases de dados de outra forma. Os processadores equipados com esta ferramenta permitem processar dados de forma segura através do uso do *Enclave*, que é uma área do processador protegida para execução em memória, tirando partido de novas instruções incorporadas nestes processadores.

Como objetivo desta dissertação, pretende-se tirar partido de soluções de sistemas que se apoiam nas tradicionais técnicas criptográficas para garantir a privacidade da informação sensível, como o *SafeNoSQL*, e integrar soluções de *hardware* confiável, como o SGX. É também necessário, proceder à avaliação do sistema, de forma a avaliar o impacto introduzido no desempenho depois da integração.

1.2 CONTRIBUIÇÕES

Esta dissertação apresenta três conjuntos de contribuições distintas: (i) uma revisão do estado da arte, (ii) o desenho de uma arquitetura e desenvolvimento de um sistema de computação segura para bases de dados NoSQL baseado nas propriedades do *hardware* confiável, e (iii) uma avaliação experimental do protótipo. Mais detalhadamente:

- Esta dissertação apresenta uma revisão do estado da arte centrada em duas vertentes: *hardware* confiável, onde se apresentam várias tecnologias com propriedades comparáveis com o *Intel SGX*; bases de dados seguras, onde são descritos vários sistemas desenvolvidos em bases de dados SQL e NoSQL, assim como algumas soluções assentes em *hardware* confiável.
- A segunda contribuição desta dissertação foi o desenho de uma arquitetura de uma base de dados *NoSQL* segura, assente num modelo de segurança baseado nas propriedades de hardware confiável, com o respetivo desenvolvimento de um protótipo que valide esta arquitetura. Este novo sistema deverá permitir processar os dados de forma segura no *Enclave*, área protegidas no espaço de endereçamento da aplicação que garante privacidade e integridade, sem as limitações das técnicas criptográficas.
- Como terceira contribuição foi realizada uma avaliação experimental detalhada do protótipo, através de uma avaliação da eficiência do sistema, comparando com o sistema *SafeNoSQL*, medindo o custo que o hardware confiável coloca no sistema.

Estas contribuições permitiram o desenvolvimento do sistema *TrustNoSQL*, um sistema baseado numa base de dados NoSQL com garantias de privacidade assentes em *hardware* confiável.

2

HARDWARE CONFIÁVEL

Os sistemas de *hardware* confiável permitem estabelecer uma base de confiança, Trusted Computing Base (TCB), para executar código remoto numa infraestrutura de terceiros. Num modelo clássico cliente-servidor onde o cliente migra a lógica do servidor para uma infraestrutura fora do seu controlo, por exemplo um serviço de *cloud*, estes expõem a aplicação a ataques maliciosos. Os *hardwares* confiáveis dispõem uma camada de proteção que pode ser vista como uma *Reverse Sandbox*, pois protege a aplicação da infraestrutura onde está alojada em vez de proteger a infraestrutura da aplicação.

Na realidade, existe um vasto conjunto de soluções que utilizam diferentes componentes criptográficos para estabelecer a TCB. Alguns sistemas dependem de coprocessadores externos que isolam completamente o processamento de *software* que lida com dados sensíveis do resto da infraestrutura, enquanto que outros sistemas estão integrados nos *chips* das unidades de processamento. Adicionalmente, a base de confiança estabelecida por cada sistema difere, podendo cobrir um sistema inteiro desde o processo de arranque de uma máquina até ao sistema operativo, ou apenas uma subsecção da lógica aplicacional. Este capítulo apresenta os sistemas mais relevantes, as garantias de segurança que permitem estabelecer e o conjunto de componentes protegidos pela TCB.

2.1 IBM 4765

IBM 4765 [18] é um coprocessador seguro que consiste num sistema computacional inteiro, protegido num invólucro resistente a ataques físicos. Estes coprocessadores encapsulam um sistema computacional completo, incluindo um Central Processing Unit (CPU), um acelerador criptográfico, *caches*, Dynamic Random Access Memory (DRAM) e um gestor de Input/Output (I/O) num ambiente inviolável. Dentro destes coprocessadores é possível executar *software* crítico que adquire fortes garantias de seguranças. Contudo estas soluções requerem conhecimento muito especializado e possuem um custo económico elevado.

As garantias de segurança dos coprocessadores advêm de uma zona de memória protegida, que só pode ser acedida pelo código do sistema, graças a verificações do controlo de acessos implementadas no *hardware* do barramento do sistema. O *hardware* dedicado é

usado para apagar toda a informação secreta e desligar o sistema aquando da deteção de um ataque físico.

2.2 TRUSTED PLATFORM MODULE

O Trusted Platform Module (TPM) é um coprocessador seguro que consiste num micro controlador dedicado, concebido para apenas autenticar o *software* executado numa máquina. Ao contrário do coprocessador anterior que protege um sistema isolado, as garantias de segurança do TPM assentam num *chip* integrado no CPU, sem necessitar de modificações adicionais. O *chip* TPM é usado para executar a certificação de *software*, processo que permite que um programa se autentique, ou seja, que o cliente consiga garantias da confidencialidade e integridade quando o seu código é executado, e armazenar a chave de certificação. Este sistema foi pioneiro ao introduzir o conceito de certificação de *software* remota, tornando o *design* do TPM invulnerável a ataques de *software*, uma vez que confia em todo o *software* no computador, apresentando um TCB muito amplo. O TCB desta tecnologia engloba todo o *software* do computador e a *motherboard*, juntamente com o respetivo CPU, coprocessador TPM, DRAM e os barramentos. No entanto, um sistema baseado em TPM é vulnerável a um adversário que tenha acesso físico à máquina, já que o *chip* TPM não fornece nenhum isolamento para o *software* no computador. Além disso, é também vulnerável a ataques no barramento de comunicação entre o processador e o circuito integrado TPM.

2.3 INTEL'S EXECUTION ENGINE

O Intel's Execution Engine (TXT) é um sistema baseado em TPM. O TXT usa o modelo de certificação de *software* do TPM e o *chip* auxiliar. Contudo, reduz o *software* dentro do invólucro seguro para uma máquina virtual hospedada pelos recursos de virtualização de *hardware* do CPU. O TXT isola o *software* dentro do invólucro de um *software* não confiável e assegura que o invólucro tem controlo total sobre todo o sistema não confiável sempre que está ativo.

Assim como os projetos baseados em TPM, o sistema TXT é vulnerável a ataques DRAM físicos. Além disso, as implementações iniciais de TXT eram vulneráveis a ataques onde um sistema operativo malicioso poderia programar um dispositivo, como uma placa de rede, para executar transferências com acesso direto a memória para a região DRAM, que estaria a ser usada por um invólucro TXT. Nos CPUs *Intel* recentes, o controlador de memória está integrado no CPU, e, portanto, o módulo de autenticação pode configurar o controlador de memória de forma segura para rejeitar este tipo de acessos.

O TCB desta tecnologia, semelhante ao que acontece quando é utilizado apenas o TPM, engloba a *motherboard*, juntamente com o respetivo CPU, coprocessador TPM, DRAM e os barramentos, sendo que a nível de *software*, a base de confiança se extende ao código autenticado de inicialização segura (SINIT ACM) e à máquina virtual criada, desde o sistema operativo ao código da aplicação.

2.4 ARM TRUSTZONE

O *TrustZone* [35] é uma tecnologia de segurança de hardware incorporada nos processadores Advanced RISC Machine (ARM) mais recentes. Esta tecnologia consiste na integração de extensões de segurança para um ARM System-On-Chip (SoC) que envolve o processador, a memória e os periféricos.

A base fundamental do ARM *TrustZone* é a introdução de um mundo seguro e um mundo inseguro como modos de operação nos núcleos dos processadores. Estes modos possuem espaços de endereço de memória independentes e privilégios diferentes. Enquanto o código em execução no mundo inseguro não pode aceder ao espaço de endereçamento do mundo seguro, o inverso pode acontecer. Os núcleos dos processadores que contêm esta tecnologia alternam entre os dois modos de operação à medida que executam código. Quando um coprocessador muda de mundo seguro para não seguro limpa os registos em *cache* para evitar libertar informação sensível.

Os componentes do *TrustZone* não apresentam proteção contra ataques físicos. No entanto, o *TrustZone* permite que um *design* proteja uma região dentro de uma memória estática no SoC, como uma Static Random Access Memory (SRAM). Posto isto, um sistema que segue as recomendações na documentação do *TrustZone* não será exposto a ataques físicos. A documentação *TrustZone* recomenda que todo o código e dados armazenados e executáveis no mundo seguro sejam armazenados na SRAM *on-chip*. [17] No entanto, esta abordagem coloca limites significativos na funcionalidade do invólucro seguro, uma vez que a SRAM no *chip* é muito mais dispendiosa que uma DRAM com a mesma capacidade a nível de custo de *hardware*. O TCB do *TrustZone* inclui o processador seguro e todo o *software* do mundo seguro, desde o *firmware*, ao sistema operativo e ao código da aplicação.

2.5 AEGIS SECURE PROCESSOR

O *Aegis Secure Processor* [37] assume que todos os componentes confiáveis estão contidos num processador seguro, constituído por um único *chip* que inclui todos os recursos de segurança necessários e as chaves secretas. A arquitetura do sistema assume que o processador e os seus componentes internos estão seguros contra leituras ou alterações, assumindo que o seu estado interno não pode ser adulterado ou observado diretamente por

meios físicos. Por outro lado, todos os componentes fora do *chip* do processador, incluindo a memória externa e os periféricos, são assumidos como inseguros.

Uma vez que todos os componentes confiáveis estão contidos num único processador, é possível construir uma plataforma de computação segura e económica. Ao contrário de uma abordagem com coprocessadores, na abordagem *Aegis*, os ataques físicos aos barramentos externos não comprometem a segurança do sistema. Por outro lado, manter todos os componentes confiáveis no *chip* do processador apresenta novos desafios, como a necessidade de manter as chaves secretas no processador principal com segurança, sem aumentar o custo do processador.

Este sistema não considera *side-channel attacks*, como ataques de padrões de acesso à memória, nem problemas de segurança relacionados com falhas ou erros no *software*, assume no entanto que o processador conta com um gerador de números aleatórios de modo a evitar possíveis ataques de repetição na comunicação. O TCB desta tecnologia engloba o processador seguro, o código da aplicação e o *kernel*.

2.6 BASTION

A arquitetura do sistema *Bastion* [21] introduziu a utilização de um *hypervisor* confiável, capaz de fornecer invólucros seguros para aplicações inseridas em sistemas operativos não confiáveis. O *hypervisor* do *Bastion* garante que o sistema operativo não interfere com os invólucros seguros.

O *hypervisor* reforça os mapeamentos de memória requeridos pelos invólucros nas tabelas do sistema operativo. Cada invólucro *Bastion* possui um segmento de memória responsável por manter todos os endereços virtuais e as permissões de todas as suas páginas, enquanto o *hypervisor* mantém uma tabela de estado do módulo que armazena um mapa de página invertido, associando cada página de memória física ao seu invólucro e endereço virtual.

O *Bastion*, à semelhança do *Aegis Secure Processor*, oferece a mesma proteção contra ataques DRAM físicos, não necessitando que todos os dados de um invólucro sejam armazenados apenas numa faixa DRAM contínua.

O *hypervisor Bastion* permite que um sistema operativo não confiável, não consiga aceder a páginas de invólucros seguros. Quando alguma informação é requerida no exterior do invólucro, as páginas são cifradas e cobertas por uma árvore de *Merkle* mantida pelo *hypervisor*. Assim, o *hypervisor* garante a confidencialidade, autenticidade e atualização das páginas. No entanto, esta capacidade de enviar páginas dos invólucros para o exterior, permite que um sistema operativo malicioso observe os acessos de memória de um invólucro.

O sistema *Bastion* não confia no *firmware* da plataforma e calcula a *hash* criptográfica do *hypervisor* após o *firmware* terminar a sua parte no processo de inicialização. A *hash* representa a identidade do *hypervisor* a ser utilizado, na ligação com a sua área segura de

memória. Em resumo, o TCB do *Bastion* inclui o processador seguro, o código da aplicação e o *hypervisor*.

2.7 INTEL SGX

A *Intel* propôs recentemente nos seus novos processadores a tecnologia *Intel SGX* [24]. Esta arquitetura oferece um ambiente de execução seguro em infraestruturas não confiáveis, tirando partido de novas instruções incorporadas nestes processadores. O conceito central do *Intel* SGX é o *Enclave*, uma área de memória protegida. Os *Enclaves* são protegidos por um novo conjunto de instruções incorporados nos processadores criando assim um ambiente seguro que contém código e dados relativos a uma computação sensível.

Este sistema permite que as aplicações criem este novo *container* protegido, o *Enclave*. De uma forma mais prática, os *Enclave*s consistem em espaços de memória protegidos, que só podem ser acedidos por código armazenado no *Enclave*, ou seja, nem mesmo um processo com o privilégio mais alto tem acesso a estas regiões de memória. O SGX segue o modelo de mundo seguro e inseguro apresentado no ARM. Os acessos à memória protegida, são controlados pelo processador, de modo a evitar acessos inválidos por *software* do mundo inseguro. Posto isto, para fornecer garantias de privacidade e integridade contra adversários que tentam aceder a este espaço de memória do *Enclave*, quando são copiadas da memória protegida para a memória principal, as páginas são protegidas com técnicas criptográficas clássicas.

Os processadores habilitados com SGX fornecem computação confiável, isolando o ambiente de cada *Enclave*, do *software* não confiável fora do *Enclave* e implementando um esquema de certificação (*attestation*) de *software*, que garante ao cliente que a sua execução está a ser processada de forma íntegra e confidencial num servidor de terceiros.

Os mecanismos de isolamento do SGX destinam-se a proteger a confidencialidade e a integridade da computação realizada dentro de um *Enclave*, de ataques provenientes de *software* malicioso, que executam no mesmo computador, bem como de um conjunto limitado de ataques físicos.

O TCB deste sistema a nível de *hardware* consiste em todo o CPU, sendo que a nível de *software* apenas confia no código da aplicação mantido no *Enclave*. O *Intel SGX* reduz assim o seu TCB incluindo apenas o processador seguro e o código do *Enclave*.

2.8 SANCTUM

O *Sanctum* [25] apresentou um sistema com as mesmas garantias de segurança que o SGX, introduzindo proteção contra ataques de padrões de acesso, como ataques de monitorização de falhas de páginas e *cache-timing attacks*.

O *Sanctum* utiliza um esquema de particionamento de cache simples, onde a DRAM é dividida em regiões contínuas de tamanho igual, em que cada região usa áreas de memória distintas no nível de cache partilhada (Last Level Cache (LLC)). Cada região é alocada para exatamente um invólucro, de modo que os invólucros são isolados tanto na DRAM quanto na LLC.

Em semelhança ao que acontece com o *Aegis* e o *Bastion*, o *Sanctum* também considera que o *hypervisor*, o sistema operativo e a camada aplicacional pertencem a um invólucro separado. Os invólucros são assim protegidos do *software* externo não confiável da mesma forma que se protegem uns dos outros. O *Sanctum* conta com um monitor de segurança confiável, que garante que nenhuma região DRAM seja acessível para dois invólucros diferentes.

Cada invólucro do *Sanctum* gere não só as suas próprias tabelas de paginação, mapeando a sua área de memória DRAM, como também lida com as suas próprias falhas de página. Posto isto, um sistema operativo malicioso não tem como aceder aos endereços virtuais que possam causar alguma falha de página no invólucro. As modificações de *hardware* do *Sanctum* funcionam em conjunto com o monitor de segurança, de modo a certificar-se que as tabelas de paginação de um invólucro apenas referenciam a memória dentro das regiões DRAM desse invólucro.

O design do *Sanctum* concentra-se completamente nos ataques de *software* e não oferece proteção contra qualquer ataque físico, de modo que para que este sistema ofereça garantias de segurança nesta vertente, as modificações de *hardware* possam ser combinadas com as proteções a ataques físicos de outros sistemas, como por exemplo do *Aegis*.

O TCB desta tecnologia inclui o processador seguro, o código da aplicação e o monitor de segurança.

3

ESTADO DA ARTE

Nos dias de hoje, já é possível assegurar uma computação segura em bases de dados, tanto em sistemas relacionais como não relacionais, através de diferentes técnicas criptográficas. Para além disso existe já trabalho desenvolvido em *hardware* confiável, sendo que esta vertente não está totalmente explorada em processamento seguro nas bases de dados.

Este capítulo contém uma descrição dos sistemas mais relevantes nestas áreas, desde bases de dados seguras através de esquemas criptográficos em *software*, até mecanismos de hardware confiável. Posto isto, irá ser descrito o estado atual da arte neste setor da segurança em bases de dados, e comparadas várias propostas já existentes em ambos os tipos de bases de dados.

3.1 BASES DE DADOS SEGURAS APENAS COM CRIPTOGRAFIA

3.1.1 *Segurança em bases de dados SQL*

As bases de dados relacionais, (ou bases de dados Structured Query Language (SQL)), são uma coleção de objetos com relações pré-definidas entre si. Estes objetos, definidos como tabelas, estão organizados por um conjunto de colunas com as mais variadas características. Cada coluna numa tabela guarda um tipo específico de dados, cada linha da tabela armazena os valores das várias colunas, podendo ser identificada por uma chave única que poderá ser relacionada com outras tabelas através de chaves estrangeiras, criando assim o conceito de relação da informação.

3.1.1.1 *CryptDB*

Com o intuito de corrigir as falhas de segurança das bases de dados relacionais como a exploração de *bugs* do sistema que permitem a adversários ganhar acesso a dados confidenciais ou até mesmo administradores maliciosos, surgiu o *CryptDB* [33], uma base de dados SQL que executa interrogações sobre dados cifrados e explora uma forma de fornecer confidencialidade para aplicações que usam um Database Management System (DBMS)

hospedado em servidores não confiáveis. Para tal, a abordagem do *CryptDB* é executar interrogações sobre dados cifrados, usando um conjunto bem definido de operadores capazes de executar eficientemente operações sobre dados cifrados.

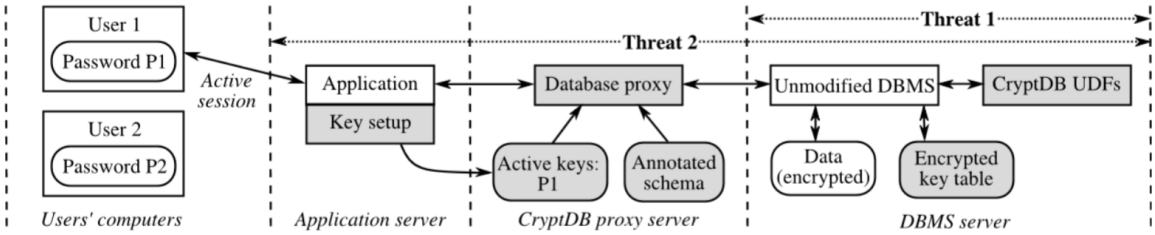


Figura 1: Arquitetura do sistema *CryptDB*.

Como podemos observar na figura 1 [33], este sistema considera dois tipos de ameaças, a primeira é um administrador malicioso que tenta aceder a dados privados fazendo uma análise no servidor do DBMS. Neste caso, o *CryptDB*, uma vez que executa interrogações sobre dados cifrados sobre a camada DBMS, protege os dados confidenciais, reduzindo a área de ataque às fugas de informação das próprias técnicas criptográficas utilizadas. A segunda ameaça a este sistema acontece quando um adversário ganha controlo completo sobre aplicações ou servidores DBMS. Nesse caso, o *CryptDB* não fornece nenhuma garantia para a confidencialidade dos dados dos utilizadores que estão ativos na aplicação durante o ataque, uma vez que o adversário tem acesso às chaves de cifragem e decifragem e, consequentemente aos dados armazenados no DBMS. O adversário pode nesta posição também aceder a todos os dados submetidos para a camada aplicacional em texto simples. O *CryptDB* pode assim apenas garantir a confidencialidade dos dados dos utilizadores sem sessão iniciada no sistema.

De forma a tentar proteger a informação destas duas classes de atacantes, o *CryptDB* depara-se com dois desafios. O primeiro reside na necessidade de minimizar a quantidade de informação confidencial revelada a um adversário, mantendo a capacidade de executar de forma eficiente as interrogações necessárias. Por exemplo, ao proteger os dados com primitivas fortes e eficientes como o Advanced Encryption Standard (AES), estaria a ser diminuido o desempenho do servidor DBMS ao executar uma quantidade avultada de interrogações SQL. O segundo desafio passa por minimizar a quantidade de dados revelados quando um adversário ganha controlo sobre a camada aplicacional para além do servidor DBMS.

O *CryptDB* aborda estes desafios usando três ideias chave, estratégia de cifragem *SQL-aware*, cifragem ajustável à interrogação e chaves de cifragem em cadeia.

A primeira passa por executar interrogações SQL sobre dados cifrados. O *CryptDB* toma partido do facto de que todas as interrogações SQL serem compostas por um conjunto bem definido de operadores primitivos, como verificações de igualdade, comparações, somas

e associações. De modo a tornar possível esta técnica, o *CryptDB* cifra cada elemento de forma a permitir que o DBMS execute sobre os dados transformados. Os valores das tabelas são protegidos com esquemas criptográficos explicados mais à frente, de modo a permitir executar certas interrogações sem ter que converter os valores para texto simples.

O *CryptDB* permite que o servidor DBMS execute interrogações SQL em dados cifrados, como em questões de dados de texto desprotegido, ou seja, o plano de execução das questões é quase o mesmo que o original. Contudo os operadores que compõem a consulta em si, como seleções, projeções, junções e agregações são executados sobre o texto protegido e usam operadores modificados.

O *CryptDB* usa um *proxy* que armazena uma *Master Key* secreta MK, o *schema* da base de dados e as camadas de cifragem atuais de todas as colunas, as *onions layers*. Estas camadas podem ser vistas como um grupo de esquemas criptográficos que estão agrupados de forma hierárquica, segundo as garantias de segurança assentes. O servidor DBMS tem por sua vez um *schema* anónimo, dados de utilizadores cifrados, algumas tabelas auxiliares e ainda User-defined functions (UDF)s específicas do *CryptDB* que permitem que o servidor compute sobre texto cifrado para determinadas operações.

A execução de uma interrogação no *CryptDB* passa por quatros passos. Inicialmente, a aplicação envia uma interrogação, que é intercetada e alterada pelo *proxy*. Este anonimiza os identificadores das tabelas e colunas e, usando a *Master Key* MK, cifra os resultados da interrogação com um esquema de cifragem mais adequado. Em segundo, o *proxy* verifica se o servidor DBMS tem necessidade de ajustar as camadas de cifragem que utiliza antes de executar a interrogação e, em caso afirmativo, envia uma interrogação *UPDATE* para o servidor DBMS que desencadeia uma UDF para ajustar a camada de cifragem das colunas em questão. No terceiro passo, o *proxy* envia a interrogação cifrada para o servidor DBMS, que o executa usando o padrão de SQL. Por último, o servidor DBMS retorna o resultado cifrado, que irá ser decifrado pelo *proxy* e o resultado posteriormente enviado à camada aplicacional.

No que se refere a esquemas criptográficos, o *CryptDB* utiliza Standard Encryption (STD) (também conhecido como Randomized Encryption (RND)), Deterministic Encryption (DET), Order-Preserving Encryption (OPE), Homomorphic Encryption (HOM), JOIN e OPE-JOIN (JOIN) e Word Search (SEARCH).

O esquema RND fornece a máxima segurança, o esquema é probabilístico, ou seja, dois valores iguais são muito provavelmente mapeados para diferentes textos cifrados. Por outro lado, o RND não permite que qualquer computação seja executada de forma eficiente sobre o texto cifrado. Por estas razões, o RND é utilizado nas camadas mais externas das *onions* de igualdade e ordenação, apresentando garantias de segurança muito fortes.

O esquema DET apresenta uma garantia de segurança ligeiramente mais relaxada, uma vez que liberta mais informação que o esquema RND. Este esquema filtra os valores cifra-

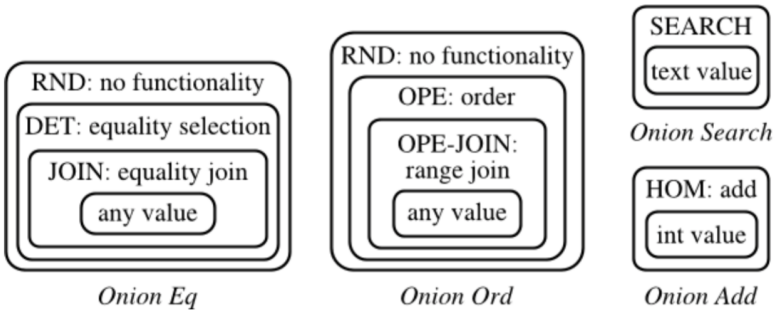


Figura 2: *Onions* criptográficas com as respetivas camadas de cifragem.

dos que correspondem ao mesmo valor de dados, gerando deterministicamente o mesmo texto cifrado para o mesmo texto simples. Esta camada permite que o servidor execute verificações de igualdade, o que significa que é bastante útil para comparações de igualdade, junções de igualdade, GROUP BY, COUNT e DISTINCT.

O esquema OPE permite que as relações de ordem entre os dados sejam estabelecidas com base nos seus valores cifrados, sem revelar os dados em si, ou seja, a ordem dos textos simples e dos respetivos textos cifrados é mantida. O OPE é um esquema criptográfico mais fraco do que DET pois revela mais informação, uma vez que revela a ordem dos textos simples pela análise dos textos cifrados, que mantêm a mesma ordem. Assim, o *proxy CryptDB* só irá revelar colunas cifradas com OPE para o servidor, se os utilizadores solicitarem pedidos sobre essas colunas.

O HOM é um esquema criptográfico probabilístico seguro, permitindo que o servidor execute cálculos em dados cifrados com o resultado final cifrado no *proxy*.

O esquema JOIN suporta todas as operações permitidas pelo DET, permitindo ainda que o servidor determine a repetição de valores entre duas colunas.

Por fim, para realizar operações de pesquisa, o esquema SEARCH é utilizado na medida em que representa a camada única da *onion* de pesquisa. Para cada coluna que precisa de SEARCH, o texto é dividido em palavras-chave, usando delimitadores padrão. Em seguida, as repetições são removidas, as posições das palavras são permutadas aleatoriamente e cada uma das palavras é cifrada usando o esquema *Song et al* [36]. Contudo, a operação de pesquisa apenas funciona corretamente para palavras completas e não para expressões regulares, prefixos ou sufixos.

A segunda ideia-chave, cifragem ajustável à interrogação, visa ajustar dinamicamente a camada de cifragem no servidor DBMS. Utilizando um sistema de cifragem que utiliza *onions* de igualdade, ordem, pesquisa e adição, o objetivo é cifrar cada elemento dos dados recebidos com uma ou mais *onions*. Cada camada de cada *onion* permite tipos de funcionalidades diferentes, por exemplo, as camadas mais externas, como RND e HOM, oferecem segurança máxima, possibilitando poucas ou até nenhumas funcionalidades, ao passo que

camadas internas, como a OPE, oferecem mais funcionalidades, diminuindo as garantias de segurança.

Para cada camada de cada *onion*, o *proxy* usa a mesma chave para cifrar valores na mesma coluna e chaves *onions* para cifrar em tabelas, colunas, *onions* e camadas. Usar a mesma chave para todos os valores da mesma coluna, permite que o *proxy* execute operações sobre toda a coluna. Usar diferentes chaves em diferentes colunas evita que o servidor tome conhecimento sobre qualquer relação adicional.

Por último, a terceira ideia, chaves de cifragem em cadeia sobre *passwords* dos utilizadores, define que a cada item da base de dados apenas pode ser decifrado utilizando uma cadeia de chaves derivada da *password* de um utilizador. Posto isto, se o servidor DBMS for comprometido, se o utilizador não estiver ligado nem o adversário tiver conhecimento da sua chave, todos os seus dados não podem ser decifrados.

Relativamente à segurança, o *CryptDB* está suscetível a ataques de criptoanálise p.e., ataques por análise de frequência, e ainda a ataques por compilação de interrogações, que visam identificar a identidade dos utilizadores [16].

3.1.1.2 *Monomi*

À semelhança do que acontece no *CryptDB*, o *Monomi* [38] é um sistema capaz de executar interrogações sobre dados cifrados. Este sistema é baseado no *CryptDB*, superando este no que refere ao processamento de grandes quantidades de dados, enfrentando assim três desafios principais que tornam difícil este processamento.

Em primeiro lugar, consultas sobre grandes quantidades de dados são geralmente limitadas pelo I/O, causando uma diminuição da eficiência do processamento da interrogação uma vez que são utilizados esquemas criptográficos que aumentam o tamanho dos dados.

Em segundo lugar, as consultas analíticas exigem cálculos complexos, que podem ser ineficientes, ou até impossíveis de executar sobre dados cifrados. Posto isto, a estratégia deste sistema passa por utilizar técnicas criptográficas eficazes, que realizam um número mais reduzido de computações e, por isso, temos a necessidade de dividir a interrogação em partes que podem ser executadas usando as técnicas criptográficas disponíveis em servidores não confiáveis, e partes sensíveis que devem ser executadas no cliente confiável.

Em terceiro lugar, algumas das técnicas para processar consultas sobre dados cifrados podem otimizar certas consultas e tornar outras mais lentas. Como resultado, pode ser mais vantajoso decifrar e executar sobre dados individuais no cliente ao invés de executar sobre dados agregados e cifrados no servidor.

O *Monomi* aborda esses desafios de três maneiras.

Como podemos ver na figura 3 [38], em primeiro lugar, o *Monomi* apresenta um modelo de execução dividida cliente-servidor (*split client-server execution model*), permitindo a divisão de consultas complexas. Esta divisão permite ao sistema executar um plano com

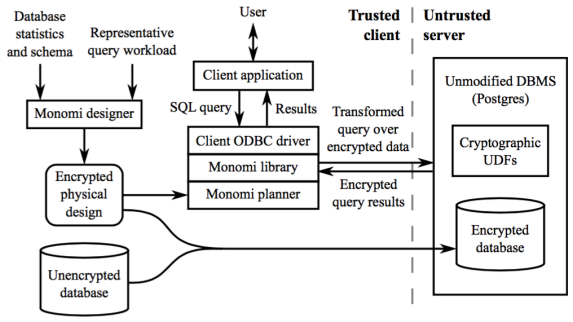


Figura 3: Arquitetura do sistema *Monomi*.

duas fases distintas. Numa primeira fase, é executado no servidor, sobre os dados cifrados. Numa segunda fase, as partes da interrogação que o servidor não consegue executar ou cujo processamento é mais eficiente no cliente, são enviadas para um cliente confiável, que decifra e executa sobre esses dados decifrados normalmente.

Em segundo lugar, o *Monomi* apresenta um conjunto de técnicas que melhoram o desempenho para certos tipos de consultas, como pré-processamento por linha, criptografia eficiente relativamente à localização dos dados, adição homomórfica agrupada e pré-filtragem. Para otimizar a execução de algumas interrogações, o *Monomi* adiciona colunas auxiliares cujo valor depende de operações entre colunas. Para minimizar o espaço adicional gerado pelos textos cifrados quando comparado com o texto simples e mitigar problemas de desempenho derivado do I/O, o *Monomi* usa esquemas criptográficos que conservam o formato dos textos cifrados através de esquemas Format-Preserving Encryption (FPE) [27].

A última solução proposta pelo *Monomi* para colmatar os desafios assentes no sistema é a criação das entidades *designer* e *planner*. O *designer* é utilizado para otimizar a camada de dados do servidor. O *planner* serve para decidir como será particionada a execução de uma interrogação entre o cliente e o servidor.

A nível de segurança do sistema, comparativamente ao *CryptDB*, o *Monomi* assume as mesmas propriedades, uma vez que a sua construção é baseada no *CryptDB*. No que diz respeito ao desempenho, devido à otimizações propostas e ao modelo de execução *split client-server*, o *Monomi* é mais eficiente. Contudo, uma vez que o processamento das interrogações é dividido, obriga a que o cliente possua poder computacional.

3.1.2 *Segurança em bases de dados NoSQL*

Com a direção que o desenvolvimento das arquiteturas dos computadores tomou, como a computação em nuvem e a necessidade crescente de proporcionar serviços escaláveis para satisfazer o consumo de armazenamento e computação, o desenvolvimento de bases de dados está a tomar um rumo onde o ponto fulcral é a escalabilidade horizontal. As bases

de dados *NoSQL* aparecem de modo a proporcionar o armazenamento dos dados com técnicas que visam satisfazer esse requisito. Já foram desenvolvidos alguns sistemas, sendo alguns deles o *BigTable* da Google [22], *International Business Machines (IBM) Cloudant* [11] e o *DynamoDB* da *Amazon* [8]. Existem também alguns exemplos de *software open-source*, como o *Apache Cassandra* [4] e o *Apache HBase* [10].

As bases de dados *NoSQL* surgiram então para abordar as necessidades de alta escalabilidade e disponibilidade de algumas aplicações.

3.1.2.1 *MiniCrypt*

De modo a colmatar os problemas de segurança em bases de dados não relacionais, surgiram sistemas como o *MiniCrypt* [41]. O *MiniCrypt* é um sistema implementado sobre o *Cassandra*, o primeiro a integrar computação segura e compressão sobre os dados de um modo ótimo, reduzindo os problemas inerentes de juntar as duas técnicas.

Cifrar grandes quantidades de dados é problemático, uma vez que a aleatoriedade das cifras geradas pelos esquemas criptográficos limitam a capacidade dos algoritmos de compressão. Uma vez que os dados não podem ser decifrados no servidor não seguro, o processamento sobre os dados tem de ser feito do lado do cliente, exigindo assim que o cliente em algum instante deste processo tenha uma grande quantidade de dados, mesmo que só precise de utilizar uma linha.

A ideia principal deste sistema é agrupar os pares chave-valor em conjuntos pequenos, chamados *packs* (como podemos observar na figura 4) [41], com a finalidade de encontrar a melhor proporção entre a compressão dos dados e a facilidade com que podemos processar interrogações sobre estes. Este *pack* é posteriormente cifrado e armazenado na base de dados.

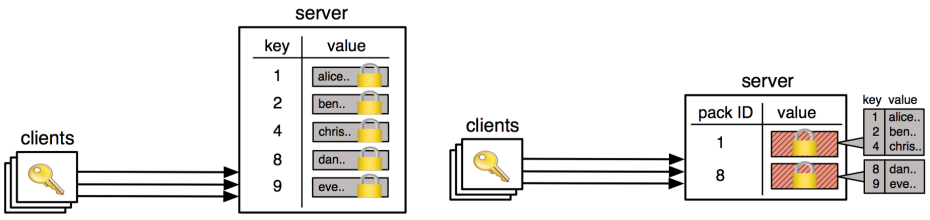


Figura 4: Modelo de cifragem do *MiniCrypt*.

O *MiniCrypt* apresenta um esquema de mapeamento simples, que permite ao servidor identificar o pacote que contém a chave, evitando a sobrecarga de pesquisar o ID do pacote numa tabela ou índice num servidor remoto.

O sistema é composto por duas partes, um cliente seguro onde é feito o processo de cifragem/decifragem e compressão/descompressão, e por um servidor não-confiável onde está o DBMS inalterado.

Desta forma, o processamento sobre a informação passa a ser feito sobre *packs* e não sobre linhas. Assim, se quisermos consultar a informação de uma determinada linha, a interrogação é feita ao *package* correspondente. Este *pack* é então enviado para o cliente, que o decifra, descomprime e verifica qual o valor da linha que pretende. Para efetuar alterações sobre os dados, o pacote é de igual forma enviado para o cliente, onde este irá criar um novo *package* com a informação atualizada.

No que diz respeito à segurança, o *MiniCrypt* considera adversários *honest-but-curious*, participantes legítimos do protocolo de comunicação, mas que tentam aprender todas as informações possíveis a partir de mensagens legitimamente recebidas. Este sistema protege os *packs* com um esquema criptográfico de *Standard Encryption*. Relativamente aos identificadores de linha, se não demonstrarem conteúdo sensível, não será aplicada nenhuma técnica criptográfica, caso contrário, são protegidos com um esquema de *Deterministic Encryption*.

3.1.2.2 *Arx*

O *Arx* [32] foi o primeiro sistema de base de dados que utiliza apenas técnicas criptográficas com fortes garantias de segurança para cifrar todos os dados, quase exclusivamente *Standard Encryption*. Isto não só proporciona garantias de segurança fortes, mas também ajuda no desempenho, devido às implementações em *hardware* da cifra AES. Para alcançar isso, o *Arx* apresenta um conjunto novo de técnicas, que em vez de incorporar a computação sobre esquemas criptográficos especiais como no *CryptDB*, o *Arx* incorpora a computação por estruturas de dados.

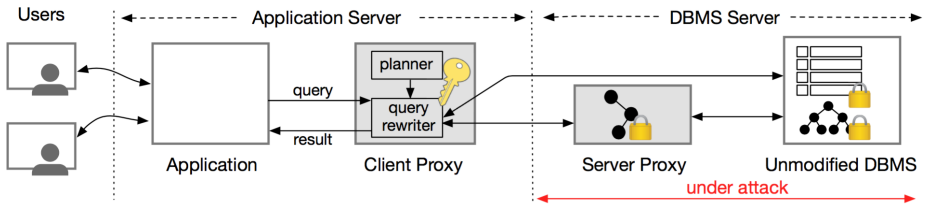


Figura 5: Arquitetura do sistema *Arx*.

Como pode ser visto na figura 5 [32], este sistema é composto por um *client proxy* (cliente seguro), alojado no servidor aplicacional, e por um *server proxy* (servidor não confiável), alojado com a base de dados. O papel do *client proxy*, com a ajuda do *planner* e do *querie rewritter*, é intercetar as interrogações e convertê-las em operações seguras. Estas interrogações são então submetidas diretamente para a base de dados caso não compreendam informação sensível ou então reencaminhadas para o *server proxy* no caso de se tratarem de operações seguras.

Para evitar a alteração da base de dados original, o *server proxy* assume o papel de cliente de base de dados, intercetando as mensagens enviadas pelo cliente e traduzindo-as

em operações sobre a base de dados. Esta entidade armazena numa estrutura os dados relativos às propriedades do texto simples, enquanto que a base de dados, armazena a informação com um esquema criptográfico forte.

De forma a possibilitar o processamento da informação com estas fortes garantias de segurança, o *Arx* apresenta quatro novos índices de base de dados: *Arx-Range* que permite realizar pesquisas e comparações de ordem entre valores, *Arx-Eq* para consultas de igualdade entre valores, *Arx-Agg*, que permite fazer operações aritméticas sobre os dados, e ainda o *Arx-Join*, que permite efetuar *joins* sobre tabelas.

Considere-se uma estrutura de dados de índices, como uma árvore binária, em que cada um dos nodos está protegido com uma cifra segura. Assim, o *Arx-Range* armazena em cada nodo um programa ofuscado (mais propriamente um *Garbled Circuit* [39]) que efetua a comparação de forma segura. Posto isto, dado um texto cifrado, o servidor pode encontrar o texto simples correspondente ao avaliar a função ofuscação em cada um dos nodos. Esta avaliação retorna o caminho a seguir na árvore. Ao fim da travessia da árvore, o índice deverá ser reconstruido de forma a fornecer um novo programa *Garbled Circuit*.

O *Arx-EQ* funciona incorporando um contador em cada valor de repetição. Isto garante que ao serem cifrados dois valores iguais, o resultado seja diferente. Esta é uma solução alternativa que apenas possibilita comparações de igualdade mas necessita também de uma árvore binária como índice (neste caso, armazenada no *client proxy*). Neste cenário, são considerados dois tipos de valores. O primeiro é quando os valores são únicos (p.e., identificadores de linha), estes são protegidos com *Deterministic Encryption* e armazenados diretamente na base de dados. O segundo tipo de valores é quando os valores são repetidos, o *client proxy* conta os valores repetidos e guarda o par valor/quantidade.

O *Arx-Agg* permite que o *Arx* calcule somas e contagens com uma estrutura de dados semelhante à do *Arx-Range*, o *server proxy* agrega os valores pedidos num conjunto, envia-os para o *client proxy* e a agregação final é feita no lado do cliente.

Em relação ao *Arx-Join* existem duas operações a serem consideradas, comparações de igualdade entre valores e comparações da ordem dos valores. Assumindo que queremos unir duas tabelas, em que a segunda possui uma chave estrangeira que aponta para a chave primária da primeira. No primeiro caso, o *Arx* segue a estratégia seguida no índice *Arx-Eq*, sendo que para cada chave estrangeira é aplicado o algoritmo de cifragem de *Arx-Eq* e o *server proxy* obtém o mapeamento para a chave primária respetiva. Para o segundo caso, o *Arx* segue a estratégia seguida no índice *Arx-Range*, sendo armazenados na árvore binária não só os valores das chaves primárias mas também das chaves estrangeiras. Para gerar os resultados da interrogação, o *server proxy* percorre a árvore, retornando ao *client proxy* os nós correspondentes às chaves estrangeiras. Por sua vez, decifrados os valores das chaves estrangeiras, o *client proxy* cifra estas chaves com *Arx-Eq* e envia-as para o *server proxy* que mapeia os conteúdos cifrados com as chaves primárias.

3.1.2.3 *SafeNoSQL*

A proteção dos dados é geralmente garantida usando técnicas criptográficas com garantias de segurança fortes. Contudo, essas abordagens impedem que a computação sobre os dados seja feita com estes cifrados. Isto levou a que o desenvolvimento destes sistemas tomasse um rumo direcionado ao desenvolvimento de conjuntos restritos de cálculos a serem executados sobre dados cifrados, como igualdades e pesquisas. Posto isto, se for adotado um sistema com garantias de segurança fortes, irá resultar num sistema que restringe a escalabilidade e a disponibilidade dos dados. Por outro lado, dar prioridade ao desempenho em detrimento da privaciadde, pode levar a falhas de segurança e privacidade da informação.

Este sistema aborda o problema admitindo que a segurança e privacidade da informação deve ser suportada por uma arquitetura modular e escalável, de modo a permitir uma especificação mais ampla de requisitos funcionais e de segurança. Isto maximiza o desempenho do sistema, garantindo um nível de privacidade adequado para que o sistema seja implementado de forma segura em máquinas de terceiros.

O *SafeNoSQL* [28] apresenta três grandes contribuições. A primeira é uma arquitetura genérica que suporta motores de bases de dados *NoSQL* já existentes, capazes de suportar os requisitos de privacidade e desempenho de diferentes aplicações. Esta estrutura possui uma arquitetura modular e escalável, que permite o processamento de dados com várias técnicas criptográficas, aplicadas no mesmo esquema de base de dados. O *SafeNoSQL* foi dos primeiros sistemas a trazer esquemas criptográficos já existentes em bases de dados SQL (como o *CryptDB* e o *Monomi*) para *NoSQL*. A segunda contribuição é um protótipo *SafeNoSQL* baseado no *Apache HBase*, juntamente com um conjunto de bibliotecas que implementam diferentes técnicas criptográficas de processamento de dados. A terceira contribuição consiste numa extensa avaliação do protótipo com micro e macro *workloads* em diferentes cenários de aplicação. Os resultados das macro-*workloads* demonstram que a sobrecarga média no desempenho das operações *NoSQL* é inferior a 15% em relação a uma implementação do *HBase* sem garantias de privacidade.

No que se refere a esquemas criptográficos, o *SafeNoSQL* utiliza várias primitivas, *Standard Encryption*, *Deterministic Encryption* e *Order-preserving Encryption*.

O esquema *Standard Encryption* considera um algoritmo de cifragem probabilístico. Isto implica um nível de segurança bastante robusto, o que significa que é inexequível para um adversário computacionalmente limitado, retirar informações significativas dos *plaintexts* associados.

Os algoritmos criptográficos clássicos que garantem a segurança semântica, não são desenhados para produzir textos cifrados sobre os quais se pode realizar cálculos significativos. A aplicabilidade dessas técnicas é, portanto, limitada pelo nível de proteção dos dados. Por exemplo, recuperar um valor cifrado utilizando esta primitiva, exigiria uma

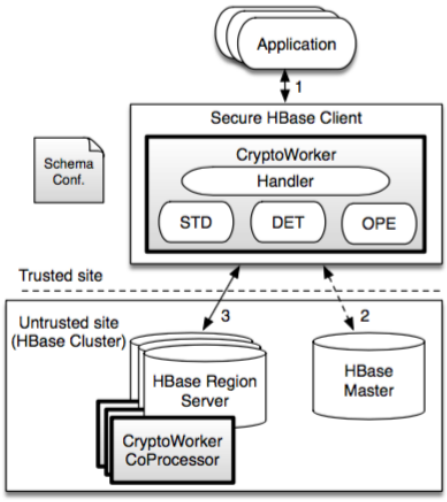


Figura 6: Implementação do *SafeNoSQL* recorrendo ao *Apache HBase*.

consulta completa da base de dados e consequente decifragem, o que é inviável para qualquer implementação de base de dados realista.

O segundo esquema utilizado, *Deterministic Encryption*, considera um algoritmo de cifragem determinístico, que garante que se cifrarmos duas mensagens iguais com a mesma chave, o resultado também será igual. A implementação deste esquema é conseguida adaptando o AES para que este se comporte de forma determinística. Um exemplo de uma aplicação que reflete este requisito é a cifragem dos números de segurança social, que provavelmente partilham prefixos, mas não estão de alguma forma relacionados. A partir das propriedades destes algoritmos determinísticos, os textos cifrados podem ser comparados sem a necessidade de utilizar a chave associada. Nesta configuração, a consulta de um valor cifrado é trivial, contudo, a consulta sobre um intervalo de dados exigiria uma consulta e decifragem completa da base de dados, uma vez que a comparação de igualdade não é suficiente.

O terceiro esquema utilizado, *Order-preserving Encryption*, considera um algoritmo de cifragem determinista, que garante que se cifrarmos duas mensagens com a mesma chave, os respetivos textos cifrados mantêm a ordem original das mensagens.

Uma tabela pode ser dividida horizontalmente em várias regiões, cada uma contendo um subconjunto das linhas para essa tabela. Este esquema de particionamento é uma característica fundamental que torna o *HBase* altamente escalável.

A figura 6 [28] mostra como o *SafeNoSQL* é instanciado sobre uma base de dados NoSQL. Esta implementação apresenta um cliente *HBase* modificado que exporta a Application Programming Interface (API) original do *HBase* (1).

Os pedidos são enviados para o *cluster HBase* composto por um *HBase Master*, *RegionServers* e o *Secure HBase Client* recebe os pedidos desprotegidos da aplicação e transforma-os

em pedidos protegidos com técnicas criptográficas. O cliente entra em contacto com o *Master* quando precisa de localizar o *Region Server* que contem a região com as linhas que necessita (2). Com o *Region Server* localizado, os pedidos do cliente são feitos diretamente para este (3), que trata os dados desejados.

Na figura 6, estão representados os componentes do protótipo *SafeNoSQL*. As caixas a cinza apresentam os componentes que foram adicionados ao *HBase* pelo *SafeNoSQL*. A implementação do *CryptoWorker* utiliza a implementação do cliente *HBase* original e fornece garantias de confidencialidade nas operações NoSQL.

3.2 Bases de dados seguras com Hardware confiável

As soluções apresentadas nas secções anteriores são baseadas em *software*, contudo, surgiram recentemente estratégias que recorrem a *hardware* confiável para garantir uma computação segura sobre bases de dados relacionais.

A *Intel* e a *AMD* propuseram nos seus processadores mais recentes, unidades de processamento isoladas, *Intel SGX* [24] e *AMD Secure Processor* [2] respetivamente, o que nos permite abordar o problema da segurança em bases de dados de outra forma.

Os processadores equipados com esta ferramenta permitem processar dados de forma segura através do uso de áreas do processador protegidas para execução em memória, tirando partido de novas instruções incorporadas nestes processadores. O uso deste *hardware* confiável permite estudar novos compromissos entre segurança e eficiência nas bases de dados, propondo deste modo uma solução alternativa a bases de dados protegidas com esquemas criptográficos.

3.2.1 *Bases de dados SQL*

3.2.1.1 *Opaque*

O *Opaque* [40] é uma plataforma de análise de dados distribuída que oferece suporte a uma gama bastante ampla de consultas, oferecendo também fortes garantias de segurança. A utilização deste tipo de sistemas tem ainda, contudo, alguns problemas. A utilização de *Enclave*s ainda permite ataques que reconheçam padrões de acesso. Portanto, para garantir a confidencialidade dos dados, a computação deve ser *Oblivious* [31], isto é, não deve permitir a fuga de nenhum padrão de acesso.

Utilizando os *Enclave*s de hardware Intel SGX [24], o *Opaque* fornece garantias de segurança fortes, incluindo a integridade dos dados. Para tal, o principal desafio encontrado na conceção do *Opaque* é então, o de oferecer uma proteção contra os padrões de acesso. Para resolver esse desafio, o *Opaque* propõe uma solução dividida em duas partes.

Em primeiro lugar, é apresentado um conjunto de novos operadores relacionais distribuídos que protegem contra a libertação de dados que permitam reconhecer padrões de acesso à memória e à rede ao mesmo tempo. Estes incluem operadores para associações e agregações. Estes operadores apresentam garantias de integridade computacional, impedindo o adversário de comprometer o resultado da computação.

Em segundo lugar, são fornecidas novas técnicas de planeamento das interrogações, *rule-based* e *cost-based*, de modo a melhorar a eficiência da *Oblivious computation*.

As otimizações *rule-based* consistem na divisão de cada operador lógico em operadores *Opaque* menores, ou seja, ao aplicar regras específicas, alguns operadores podem ser combinados ou até removidos.

As otimizações *cost-based* consistem no desenvolvimento de um modelo de custo para operadores *oblivious* que permitem avaliar o custo de um plano físico. Utilizando a reordenação de *joins* para minimizar a quantidade de operadores *oblivious*. Uma vez que nem todas as tabelas de uma base de dados são sensíveis e o *Opaque* permite que o administrador escolha quais são essas colunas, podemos consultar as tabelas que não são sensíveis usando operadores *non-oblivious* de forma a melhorar o desempenho. No entanto, algumas tabelas sensíveis podem estar relacionadas com tabelas aparentemente não sensíveis. Para proteger as tabelas sensíveis neste caso, o *Opaque* toma partido de uma técnica chamada deteção de inferência, de modo a propagar a sensibilidade das tabelas com base nas informações do esquema.

O *Opaque* foi implementado usando o *Intel SGX* sobre o *Spark SQL* com modificações mínimas neste segundo. O *Opaque* pode ser executado em três modos. O primeiro é o *encryption mode*, que fornece cifragem e a autenticação dos dados, além de garantir a execução correta da computação. O segundo é o *oblivious mode*, em que o *Opaque* fornece uma execução que protege contra *access pattern leakage*. O último modo é o *oblivious pad mode*, que consiste num melhoramento do *oblivious mode* de modo a não permitir *size leakage*.

3.2.1.2 *TrustedDB*

O sistema *TrustedDB* [19] recorre a um coprocessador criptográfico em *hardware* da *IBM*. Uma vez que o coprocessador apresenta limitações relativas à capacidade de armazenamento, os dados são guardados no servidor e a computação será dividida entre o servidor e o coprocessador.

Os atributos são classificados como sendo públicos ou privados, consoante a informação seja ou não sensível. Os dados públicos podem ser processados pelo servidor, já os dados privados são cifrados e decifrados pelo cliente e pelo coprocessador, enquanto o processamento ocorre sobre os dados cifrados no coprocessador.

Ao contrário do que acontece em sistemas baseados em software, tanto o cliente como o coprocessador têm acesso às mesmas chaves de cifragem. Estas chaves são transmitidas en-

tre as duas entidades através de mensagens cifradas com cifragem pública. O processador permite que estas chaves sejam guardadas de forma segura pois é seguro contra adulteração de informação.

4

BASE DE DADOS TRUSTNOSQL

4.1 TRUSTNOSQL: VISÃO GERAL

De acordo com o capítulo anterior, existem soluções propostas para a computação segura sobre bases de dados SQL assentes em *hardware* confiável. Contudo, apesar do desenvolvimento das bases de dados estar a tomar um rumo onde o ponto fundamental é a escalabilidade horizontal com processamento seguro, nenhuma solução atual explorou *hardware* confiável. Posto isto, propomos o *TrustNoSQL*, uma solução que utiliza esta tecnologia para bases de dados NoSQL, de forma a permitir a integração de soluções em *hardware* confiável. Desta forma, é apresentada uma nova arquitetura em que o objetivo é utilizar *hardware* confiável para proteger código capaz de processar dados decifrados de uma forma confidencial, conforme os pedidos necessários a uma base de dados NoSQL, ao invés do armazenamento da totalidade dos dados dentro do *hardware* confiável.

Posto isto, numa fase inicial deste capítulo, irão ser explicados os princípios arquiteturais do sistema e, posteriormente, serão explicadas as camadas desenvolvidas para permitir a integração do *hardware* confiável. Por último será explicado o modo de funcionamento do sistema, particularmente o fluxo de vários pedidos.

4.2 ARQUITETURA DA SOLUÇÃO PROPOSTA

O *TrustNoSQL* é uma base de dados NoSQL com armazenamento e processamento seguro. As garantias de segurança deste sistema assentam nas propriedades garantidas pelo *hardware* confiável. A arquitetura do sistema, apresentada na figura 7, está desenhada para abstrair a tecnologia de processamento seguro e assume um componente capaz de garantir várias propriedades desde a execução de código arbitrário, a confidencialidade e integridade dos dados durante a execução, acesso a módulos criptográficos para cifragem e decifragem dos dados e garantias de certificação de *software*.

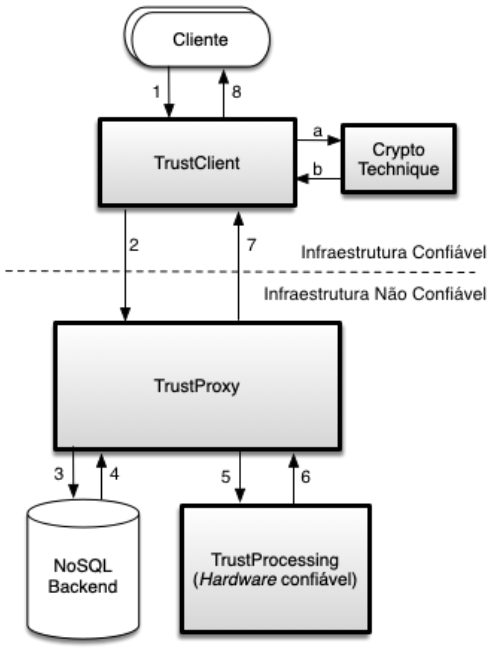


Figura 7: Arquitetura do TrustNoSQL.

A arquitetura do sistema está dividida em duas zonas, confiável e não confiável. A zona confiável representa uma infraestrutura segura do lado do cliente, onde os dados podem ser processados como texto claro. Assume-se que esta zona não é suscetível a ataques e pode corresponder a um computador pessoal ou a uma infraestrutura privada de onde provêm os pedidos ao sistema. A segunda zona não é confiável e está sujeita a ataques, de forma que dados presentes nesta zona podem ser intercetados. Posto isto, os dados que circulam ou são mantidos nesta zona ou têm de ser cifrados do lado do cliente e processados de forma segura no *TrustProcessing* (*hardware* confiável).

O cliente está inserido na zona confiável do sistema e tem uma completa abstração do que acontece na zona não confiável. Este apoia-se nas garantias de segurança provenientes da integridade e confidencialidade asseguradas pelo *hardware* confiável, e pela garantia do sistema de que nos restantes módulos da zona não confiável (*TrustProxy* e *NoSQL Backend*), os dados nunca são decifrados, uma vez que estes não têm acesso à chave necessária nem às primitivas de decifragem. Nesta zona, os pedidos/questões provenientes do cliente para a base de dados são intercetados pelo *TrustClient*, um módulo responsável por transformar pedidos inseguros (ou sobre texto desprotegido) em pedidos seguros. Esta transformação cifra esses pedidos, de acordo com o esquema criptográfico, e garante a confidencialidade, assegurando que estes não estejam decifrados em parte alguma da zona não confiável com a exceção do *TrustProcessing*. Para isso, o *TrustClient* tem acesso a um módulo chamado

CryptoTechnique, capaz de abstrair as cifras e os respetivos modos (AES-CBC, AES-GSM, 3DES, RSA) necessários ao funcionamento do sistema.

Na zona não confiável temos presente um *TrustProxy*, encarregue de identificar e tratar todos os pedidos do cliente, comunicando com o *NoSQL Backend* onde estão presentes os dados, reencaminhando os pedidos para o *TrustProcessing* e os resultados de volta para o lado do cliente. Nesta zona foi adicionada a camada de *hardware* confiável (*TrustProcessing*), que interage com o *TrustProxy*, permitindo-lhe realizar computações requeridas pelos clientes, de forma segura, garantindo confidencialidade e integridade. O *TrustProxy* tem acesso às primitivas desenvolvidas que utilizam o *TrustProcessing*, para que possa ser efetuado processamento sensível na zona não confiável. Este sistema na sua globalidade apenas oferece confidencialidade, sendo que as garantias de integridade oferecidas pelo *hardware* confiável são válidas apenas dentro do *TrustProcessing*.

O *TrustClient* envia pedidos seguros para a zona não confiável, que são intercetados pelo *TrustProxy*, que, por si, é capaz de carregar dados cifrados da base de dados, enviá-los para o *TrustProcessing* para o processamento desejado e, por fim, retornar o resultado, sempre de um modo transparente para o cliente e mantendo a confidencialidade de todos os dados.

O cliente quando faz pedidos ao sistema, tem de ter a garantia de que pode confiar no código executado no *TrustProcessing*. Para isso, o código precisa de ser verificado pelo cliente quando é carregado. Para que o *TrustProcessing* prove a sua identidade e autenticidade para com o cliente recorre-se à certificação de *software*. Este procedimento é o responsável por verificar o código a ser executado pelo *hardware* quando é carregado e é capaz de detetar modificações não autorizadas no *software*.

O *TrustClient* é responsável por garantir a compatibilidade dos esquemas criptográficos com a introdução do *hardware* confiável e abstrai os detalhes de implementação em quatro operações de alto nível:

- *Inicialização()* permite inicializar o *CryptoTechnique* e os respetivos parâmetros do seu esquema criptográfico.
- *Destruição()* permite destruir o contexto da cifra assim como libertar as variáveis de estado da cifra.
- *Geração()* permite criar uma chave criptográfica aleatória do tipo do esquema criptográfico.
- *Cifragem(texto, chave)* permite cifrar os dados sensíveis dada uma chave criptográfica, retornando um criptograma. Esta operação utiliza uma primitiva de cifragem probabilística.
- *Decifragem(criptograma, chave)* permite decifrar um criptograma dada uma chave criptográfica, retornando os dados originais, previamente cifrados.

- *Validar(token)* permite verificar a validade do *token* proveniente do *TrustProcessing*, que permite a certificação do *software* presente no *hardware* confiável. Esta operação retorna novamente um *token* direcionado ao *TrustProcessing* para estabelecer um canal seguro com a lógica aplicacional executada no *hardware* confiável.

O módulo *TrustProcessing*, responsável pelo processamento no *hardware* confiável, está abstraído em cinco operações distintas:

- *Inicialização()* é a operação que permite a criação de uma unidade de processamento com uma região de memória privada. Esta operação é responsável por, no primeiro pedido que o cliente necessitar desta unidade, criar um novo módulo e guardar o seu *ID*, para que, em pedidos futuros seja utilizada essa mesma unidade e não forçar a criação de uma nova por cada pedido.
- *Destruição()* é a operação responsável por destruir e libertar a memória utilizada pelo *TrustProcessing* no final da execução do programa.
- *Processamento(dados_cifrados)* é a operação responsável pelo processamento dentro do *TrustProcessing*.
- *Inicialização da Certificação()* é a operação que gera um *token* para o cliente que vai permitir a certificação de *software*.
- *Finalização da Certificação(token)* é a operação que conclui a certificação de *software*. Se esta operação for bem sucedida, o cliente e o *hardware* confiável podem trocar mensagens de forma segura sem algum atacante conseguir decifrar.

Numa base de dados não relacional, os dados não têm tipos e são armazenados como *bytes*, contudo, à semelhança das bases de dados relacionais, são armazenados de forma estruturada através de tabelas com colunas. Estas tabelas são constituídas por pares chave-valor, ou seja, cada entrada da base de dados é constituída pelo identificador de linha (chave) e pelo valor que contém associado a essa chave. As bases de dados constituídas por pares chave-valor possuem a seguinte interface: *Put*, *Get* e *Scan* e *Delete*. O *Put* insere uma nova linha (par chave-valor) na base de dados, o *Get* retorna uma linha dada uma chave específica, o *Scan* devolve os vários resultados que satisfazem a condição de um eventual filtro presente e/ou as linhas em que a sua chave está contida num conjunto de valores e o *Delete* remove uma chave e os valores associados (linha) da base de dados.

4.3 Esquema fluxo de operações

De modo a ilustrar a interação entre os componentes da arquitetura apresentada e verificar como cada um dos módulos interage com os outros, vamos assumir um esquema *NoSQL*

em que as chaves e todas as colunas estão cifradas com uma técnica criptográfica oferecida pela interface presente no *CryptoTechnique*. Nestas descrições os números a ser indicados fazem referência às interações descritas na figura 7.

Considerando um pedido *Put* para um determinado par chave-valor proveniente do cliente, o módulo *TrustClient* interceta esse pedido (1). Com a ajuda do *CryptoTechnique*, o *TrustClient* inicia então o processo de construção do pedido cifrado para posteriormente enviar para a zona não confiável, começando por enviar o par chave-valor para o *CryptoTechnique* (a). Este módulo irá cifrar o par chave-valor com a técnica criptográfica definida pelo requisitos de segurança do cliente e retornar os criptogramas para o *TrustClient* (b). Este modulo vai então construir o pedido *Put* com os parâmetros já cifrados, enviar esse pedido para o *TrustProxy* (2), que posteriormente vai ser reencaminhado para o *NoSQL Backend*, onde irão ser armazenados os dados cifrados no servidor não confiável (3).

Dado um pedido *Get* sobre uma determinada chave x cifrada pelo processo explicado anteriormente, o módulo *TrustProxy* interceta o pedido proveniente do lado do cliente (2). Uma vez que as chaves na base de dados estão protegidas com um esquema probabilístico, este componente inicia um *Scan* interno sobre o *NoSQL Backend* (3) para percorrer todas as linhas da base de dados e recolher as chaves da tabela (4). No momento em que o *TrustProxy* tem já um conjunto de chaves, está preparado para enviar este conjunto de dados cifrados e a chave x para o *Trusted Hardware* (5). Este, por si, vai processar os dados de forma a manter a confidencialidade, e retornar para o *TrustProxy* o resultado (6). De forma a completar o pedido *Get*, o processo de carregamento de dados pelo *Scan* interno (3 e 4) e o processamento no *TrustProcessing* (5 e 6) são iterativos e só terminam quando todos os conjuntos de dados estiverem processados e o resultado enviado para o *TrustProxy*. Este, por sua vez, vai reencaminhar o resultado para o cliente (7 e 8).

Dado um pedido *Scan* com uma *start key* e um filtro sobre uma coluna, o módulo *TrustClient*, à semelhança dos exemplos anteriores, é responsável por intercetar o pedido (1) e criar um novo, cifrado com a ajuda do módulo *CryptoTechnique* (a e b). Após este processo, envia o pedido para o *TrustProxy* (2). Ao receber este pedido e perante a presença da *start key* e do filtro, o *TrustProxy* é capaz de fazer o processamento devido: iniciar o *Scan* interno sobre o *NoSQL Backend* para recolher não só as chaves da tabela, mas também os valores guardados na coluna pretendida (3 e 4) e executar, para todos os conjuntos de dados obtidos iterativamente, duas chamadas ao *TrustProcessing*. Uma primeira com as chaves para o processamento sobre a *start key*, em que o *TrustProcessing* retorna todas as linhas em que a sua chave tem o valor superior à *start key* (5 e 6), e uma segunda chamada, em que o pedido feito ao *hardware* confiável (5) tem presente a condição e tipo do filtro proveniente do cliente, e em que os dados considerados são apenas aqueles correspondentes ao resultado da primeira operação. Esta segunda resposta por parte do *Trusted Hardware* (6) corresponde

ao resultado não só processado pela *start key*, mas também pelo filtro por coluna. Este resultado é finalmente transmitido ao cliente (7 e 8).

5

IMPLEMENTAÇÃO

Conforme apresentado no capítulo 4, o sistema TrustNoSQL permite dotar um sistema baseado numa base de dados NoSQL de processamento seguro, através da introdução de *hardware* confiável. A integração deste módulo permite acrescentar a este sistema não só propriedades de segurança relacionadas com as cifras utilizadas, mas também características de segurança como a confidencialidade dos dados, assentes no *hardware* confiável. Neste capítulo será apresentada uma descrição detalhada da implementação dessa arquitetura, descrevendo as várias tecnologias utilizadas, a base de dados abordada, as técnicas criptográficas utilizadas e as funcionalidades implementadas.

5.1 VISÃO GERAL DA IMPLEMENTAÇÃO DO TRUSTNOSQL

O sistema *TrustNoSQL* é implementado como uma extensão do sistema *SafeNoSQL*, com a adição de componentes que possibilitam a execução de operações seguras utilizando *hardware* confiável. Os componentes alterados estendem a funcionalidade do *SafeNoSQL* no lado seguro e não seguro. Do lado seguro foi desenvolvida uma nova *CryptoBox* encarregue de fornecer as primitivas necessárias no que compete à cifragem e decifragem de dados. No lado não seguro, são adicionados dois componentes. Um primeiro correspondente a uma unidade de processamento de *hardware* confiável *TrustProcessing*, e um segundo *TrustProxy* responsável por intercetar os pedidos provenientes do lado seguro, recolher todos os dados necessários do *NoSQL Backend*, fazer os pedidos ao *TrustProcessing* e retornar os resultados ao lado do cliente.

Antes de descrever as alterações feitas no sistema, são aqui descritos alguns detalhes relevantes do *HBase* e da arquitetura do *SafeNoSQL* que não foram descritos na secção do Estado da Arte.

5.2 APACHE HBASE

Como base, o sistema *SafeNoSQL* utiliza a base de dados NoSQL *Apache HBase*, uma base de dados não relacional, distribuída, altamente escalável e *open-source*, que foi desenvolvida a partir da base de dados *BigTable* [22] da Google.

No *Apache HBase* os dados são armazenados de forma estruturada através de tabelas com colunas. Estas tabelas são compostas pelos identificadores de linhas e pelas respetivas colunas. O identificador de linha é a *row key* e é sempre único. As colunas são compostas por dois níveis, as *column families* e os *column qualifiers* de forma a organizar todos os dados de forma estruturada. Para o *HBase* todas as linhas são *arrays* de *bytes*, o que faz com que esta seja uma base de dados sem tipos de dados. Podemos ver na tabela 1 um exemplo de uma estrutura de uma tabela no *HBase*.

Identificador (Chave)	Estado (CF)			Governo (CF)	
	Nome (CQ)	Capital (CQ)	População (CQ)	Governador (CQ)	Vice Governador (CQ)
16341	California	Sacramento	39536653	Gavin Newsom	Eleni Kounalakis
25385	Texas	Austin	28304596	Greg Abbott	Dan Patrick
34126	Florida	Tallahassee	20984400	Ron DeSantis	Jeanette Núñez

CF - *Column Family*, CQ - *Column Qualifier*

Tabela 1: Exemplo da estrutura de uma tabela no *HBase*.

A arquitetura do *HBase* é composta por várias camadas e está dividida em zona confiável e não confiável. O *HBase Client* é a camada onde as ações dos clientes são intercetadas, convertidas em interrogações ao *HBase* e por fim enviadas para o *HBase Backend*. Na figura 8 podemos ver uma ilustração da arquitetura do *HBase*.

No *HBase*, as tabelas são distribuídas dinamicamente pelo sistema, sempre que se tornam muito grandes (*Auto Sharding*). A unidade mais simples e fundamental de escalabilidade horizontal neste sistema é a região, um conjunto contínuo e ordenado de linhas que são armazenadas na base de dados. Posto isto, a arquitetura do *HBase* possui um só nodo de gestão *HBase Master* e vários *Region Servers*, em que cada *Region Servers* pode conter um conjunto de regiões que contém os dados armazenados. Sempre que um cliente envia uma interrogação, o *HBase Master* recebe o pedido e encaminha-o para o *Region Server* correspondente. O *HBase Master* é o processo que atribui regiões a *Region Servers* no *cluster* do *Hadoop* para balanceamento de carga. Os *Region Servers* são os nodos que executam as interrogações de leitura, escrita, atualização e exclusão provenientes dos clientes.

No que diz respeito às operações típicas de uma base de dados, o *HBase*, através de uma camada aplicacional fornece as operações *Put*, *Get*, *Scan* e *Delete*.

A operação *Put* é utilizada para inserções na base de dados. Esta operação necessita de uma *row key*, uma *column family* e um *column qualifier*. De acordo com a existência prévia

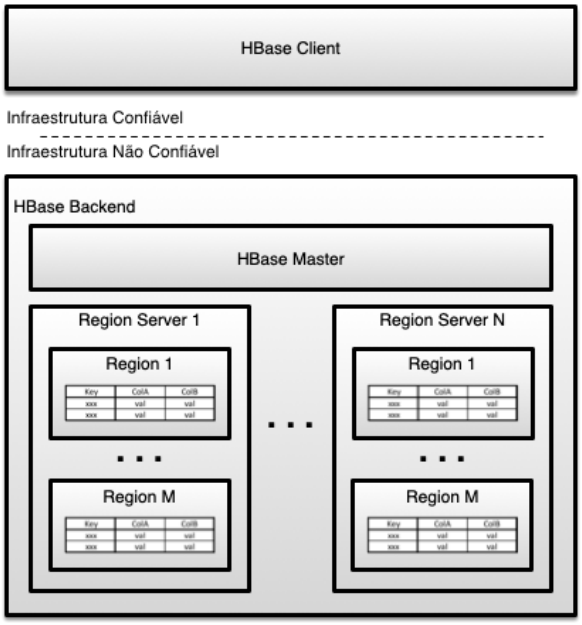


Figura 8: Arquitetura do *HBase*.

dessa chave na tabela, a operação é executada de duas formas diferentes. Se a chave não existir na tabela, ou essa chave não tiver dados na respetiva coluna, são adicionadas novas linhas, com os novos dados. No caso de já existir na tabela uma entrada com a chave, a *column family* e o *column qualifier*, os valores da tabela são atualizados com os novos dados.

Para remover dados de uma tabela é usada a operação *Delete*. Esta operação pode ser utilizada para remover todas as linhas associadas a uma chave, ou apenas dados mais específicos relativos a uma família ou qualificador. No primeiro caso apenas é necessário fornecer a *row key* que se pretende eliminar, sendo que no segundo já é necessário fornecer também a *column family* e/ou o *column qualifier*.

A operação *Get*, dada uma determinada chave, vai à base de dados buscar todos os valores de colunas associados a essa chave. Através de funções auxiliares do *Get*, é possível fazer pedidos à base de dados de *column family* ou *column qualifier* específicos.

Por último, a operação *Scan* permite não só ir à base de dados buscar os pares chave-valor associados a um conjunto contíguo de chaves, mas também a utilização de filtros, que permitem retornar um subconjunto de resultados mais restrito ao cliente. No âmbito desta tese foram abordados os filtros *RowFilter* e *SingleColumnValueFilter*. Quando o cliente faz uma interrogação *Scan*, é criado um objeto *Scan*, sobre o qual é criado um iterador na forma de um *ResultScanner*. Este *Scanner* pode ser percorrido invocando o método *next()*, em que cada invocação retorna para o cliente um novo resultado, na forma de um *Result*. Cada objeto *Result* pode ser percorrido e dividido de forma a que o cliente possa aceder não só

à chave mas também aos valores das colunas de acordo com a *column family* e o *column qualifier*.

O filtro *RowFilter* é utilizado para filtrar sobre os identificadores de linha (*row key*), ao passo que o *SingleColumnValueFilter* permite a filtragem sobre os vários valores presentes nas colunas. Estes filtros suportam os operadores de comparação (==, !=, <=, <, >=, >), o que permite ao filtro e consequentemente à operação *Scan*, uma versatilidade muito ampla de pedidos.

5.2.1 *Coprocessador*

Os coprocessadores do *HBase* são uma funcionalidade do *HBase* que permite estender o funcionamento base do sistema. Estes permitem executar código ao nível do servidor, sobre os dados armazenados no *HBase*. Existem dois tipos diferentes de coprocessadores, os *observers* e os *endpoints*. Os *observers* são acionados antes ou depois de um evento específico ocorrer. Os *endpoints* permitem exercer computação no local dos dados, ou seja, estender um conjunto de funcionalidades na base de dados em si.

Neste projeto vamos desenvolver um *observer*, mais concretamente um *RegionObserver*. Um *RegionObserver* permite intercetar os pedidos do cliente tais como o *Get*, *Put*, *Delete* e *Scan*. Desta forma e utilizando esta tecnologia é possível alterar a execução dos pedidos, sendo que é possível construir novos pedidos com a mesma informação sem decifrar os dados, de forma a tomar partido e respeitando as interfaces das novas *CryptoBoxes* adicionadas. O *TrustProxy* é implementado como um *RegionObserver*.

5.3 SAFENOSQL

Como visto na subsecção 3.1.2.3, a *framework SafeNoSQL* permite dotar as bases de dados NoSQL de armazenamento e processamento seguros, garantindo a privacidade da informação. Este é um sistema modular e extensível que permite fornecer de forma transparente, privacidade e segurança a bases de dados NoSQL. A arquitetura está apresentada na figura 9.

A *framework* é genérica e compatível com a maioria das bases de dados NoSQL do tipo chave-valor. Para isso, o sistema é composto por quatro módulos principais: *CryptoWorker*, *CryptoBox*, *Handler* e *SafeMapper*.

Os *CryptoWorkers* são entidades que abstraem a integração de múltiplas técnicas criptográficas no sistema e estão presentes em ambas as zonas, confiável e não confiável. Estas entidades fornecem uma API NoSQL transparente, de forma a possibilitar a integração com as bases de dados NoSQL atuais. As interrogações efetuadas à base de dados são intercetadas pelo *CryptoWorker* e transformadas em operações com a mesma semântica mas com

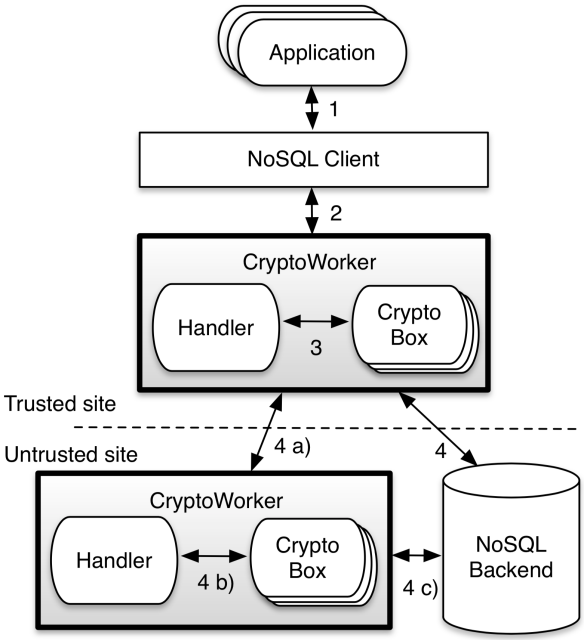


Figura 9: Arquitetura do sistema *SafeNoSQL*.

garantias de segurança. Dependendo da garantia de segurança estabelecida no esquema criptográfico, a operação é traduzida de forma diferente.

As *CryptoBoxes* são entidades que contém técnicas criptográficas utilizadas para proteger os dados sensíveis. Como *CryptoBoxes* foram adotadas as técnicas criptográficas *Standard Encryption* (STD) que não revela informação mas também não permite fazer nenhum tipo de processamento, *Deterministic Encryption* (DET) que revela igualdade entre valores cifrados e ainda *Order-Preserving Encryption* (OPE) que revela a ordem dos criptogramas. O módulo *SafeMapper* foi instanciado de forma a fornecer um mapeamento direto com as tabelas da base de dados *HBase* e os seus dados. O *SafeNoSQL* oferece um módulo *SafeMapper* que permite fazer o mapeamento entre o esquema de uma base de dados NoSQL e as respetivas propriedades de segurança assentes sobre cada coluna, descritas num ficheiro de configuração (*Schema Conf.*).

5.4 Integração com o SafeNoSQL

Ao sistema *SafeNoSQL* foi adicionada uma *CryptoBox* do lado do cliente (zona confiável), que adota técnicas criptográficas compatíveis com as técnicas utilizadas dentro do *Trust-Processing*. Esta *CryptoBox* é instanciada num *CryptoWorker* (*TrustClient*) responsável por intercetar os pedido provenientes de clientes e, cifrar os pedidos de acordo com o esquema criptográfico definido no esquema de configuração.

Foi também adicionada uma *CryptoBox* na zona não confiável, que é instanciada pelo *CryptoWorker* (*TrustProxy*) e responsável pela criação e destruição do *TrustProcessing*.

Na figura 10 está representada a implementação do *TrustNoSQL* sobre o sistema *SafeNoSQL*. As caixas a cinzento representam os componentes adicionados à proposta original do *SafeNoSQL*.

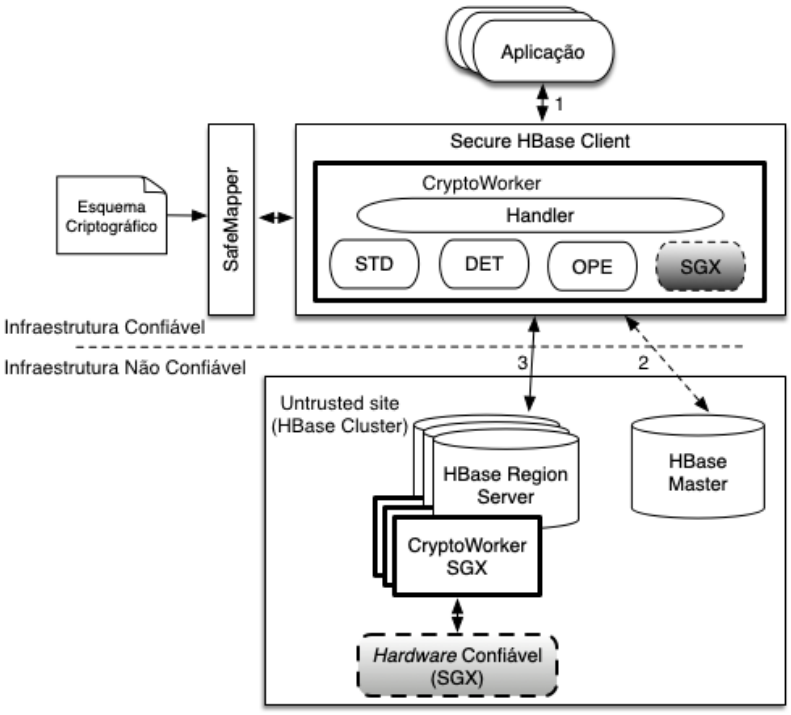


Figura 10: Implementação do sistema *TrustNoSQL*.

5.4.1 *TrustClient e SafeMapper*

O componente *TrustClient* é o responsável por intercetar os pedidos provenientes dos clientes. Estes pedidos contêm as chaves e/ou colunas sobre as quais o cliente pretende operar, decifradas, sendo que não podem ser transmitidas para a zona não confiável dessa forma. Para isso, o *SafeMapper* foi estendido com um novo *CryptoType*, representativo da nova técnica criptográfica adicionada, de modo a permitir ao *TrustClient* reagir e lidar corretamente com os pedidos em que esta técnica é usada.

Como descrito na arquitetura do *TrustNoSQL* (4), o papel do *CryptoWorker* instanciado pelo *TrustClient* é intercetar de forma transparente os pedidos à base de dados NoSQL e traduzi-los em operações seguras e privadas, isto é, é o módulo que se encarrega de intercetar os pedidos, recolher os dados da interrogação, cifrá-los e reconstruir os pedidos

de forma a poder enviá-los para a zona não confiável. Para isso foi necessário fazer as alterações necessárias para a construção dos pedidos em que a técnica criptográfica estabelecida no esquema corresponde à cifra utilizada dentro do *TrustProcessing*, o que resulta na necessidade de implementar o processo de construção da interrogação cifrada nas várias operações suportadas (*Put*, *Get* e *Scan*).

Em particular, para todos os pedidos *Put*, dada uma chave x (no momento decifrado) sobre a qual é necessário cifrar conforme a técnica criptográfica estabelecida, o *TrustClient* é responsável por cifrar este valor e construir um novo pedido *Put* com o valor cifrado e, posteriormente enviá-lo para a zona não confiável. Para isso, o *TrustClient*, com a ajuda do *SafeMapper*, consulta o esquema criptográfico (*tableSchema*) para saber se tem que cifrar os dados. Caso o esquema criptográfico indique que é necessário cifrar algum dos componentes (chave e/ou colunas), o *CryptoWorker* do *TrustNoSQL* é invocado para cifrar os dados respetivamente, e criar um novo pedido *Put*, que é reencaminhado para a zona não confiável.

Por sua vez, os pedidos *Get* sobre um valor y decifrado, provenientes de um cliente são intercetados pelo *TrustClient*. À semelhança da transformação dos pedidos *Put*, este valor y não pode ser enviado decifrado para a zona não confiável, portanto, o *CryptoWorker* encarrega-se de separar este valor do pedido, cifrá-lo utilizando a *CryptoBox* respetiva, e construir um novo pedido *Get* preparado para ser enviado para a zona não confiável.

Note-se que, no caso de as *row keys* estarem cifradas com esta nova técnica criptográfica baseada numa cifra AES, os criptogramas vão apresentar propriedades não determinísticas e por isso não revelam igualdade nem ordem. Posto isto, as chaves irão, obrigatoriamente, ser enviadas na sua totalidade para o *TrustProcessing*. Considerando a necessidade de fazer uma consulta com todas as chaves da base de dados e que as chaves são únicas, o resultado de um *Scan* sobre uma chave será uma única linha, correspondente ao resultado de um *Get*. Assim, assumindo o pedido *Get* sobre uma chave z, podemos transformá-lo num pedido *Scan* com um filtro sobre as chaves da base de dados (*RowFilter*) com a operação de igualdade (*EQUAL*).

Para todos os pedidos *Scan* provenientes do *TrustClient*, o processo de cifragem do pedido é semelhante ao explicado anteriormente para o *Get*, com a necessidade de executar o mesmo processo no caso de o *Scan* apresentar uma *start key* ou na presença de filtros, em que o valor de comparação também tem de ser cifrado da mesma forma.

5.4.2 *TrustProxy*

De forma a permitir a um módulo a reação a eventos, despoletada pelos pedidos provenientes do cliente, foi criado o componente *TrustProxy*, baseado na instanciação de um *RegionObserver*. Assim, o *TrustProxy* quando recebe um pedido do *TrustClient* é dotado da

capacidade de escolher quais os dados a ser consultados e a forma como esta consulta é realizada de acordo com o pedido. O *TrustProxy* é o componente que interceta os pedidos e controla tudo o que acontece do lado *Untrusted*. De forma a melhor explicar o que acontece neste módulo, na figura 11 é ilustrado o fluxo dos pedidos dentro do *TrustProxy*.

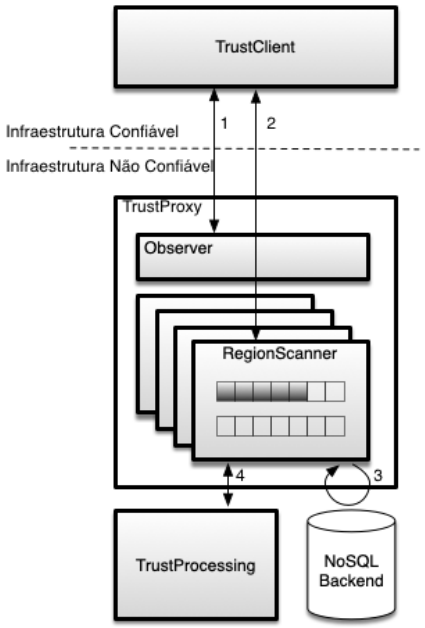


Figura 11: Fluxo de pedidos no *TrustProxy*.

Para melhor entender este componente, é necessário compreender que o coprocessador *Observer* representado na figura tem a capacidade de intercetar os pedidos do cliente, através de primitivas tais como o *postScannerOpen()*. Esta primitiva reage ao pedido *Scan*, efetuado ao *HBase* e pode ser sobrescrito de forma a ser controlado todo o processo. Quando o cliente envia um pedido *Scan* ao *HBase*, este cria objetos *RegionScanner*, um para cada região da base de dados, responsáveis por tratar dos pedidos feitos à sua região de dados. Estes *RegionScanner* são criados com as propriedades de ambiente do *Scan* original e são retornados para o cliente para que este possa operar sobre ele (1). Assim, o cliente tem a capacidade de fazer os pedidos e receber resultados diretamente do *RegionScanner* (2) através do método *next()* do *HBase*.

O *TrustProxy* percorre a base de dados por *batches*, enviando esse conjunto de dados para o *TrustProcessing* para ser processado. Assim, cada vez que é efetuado um pedido ao *hardware* confiável, é processado um número predefinido de linhas da base de dados, para evitar que o *RegionScanner* vá ao *NoSQL Backend* cada vez que for invocado o método *next()* sobre o *Scanner* no cliente. Para isso, o módulo *RegionScanner* possui uma *cache* de dados que lhe permite guardar os resultados retornados do *TrustProcessing*. Assim, enquanto

existirem resultados na *cache*, o *next()* vai retornar esses valores, ao invés de executar um novo pedido à base de dados e ao *hardware* confiável. Pelo contrário, quando a *cache* de resultados estiver vazia, o *RegionScanner* tem que ir ao *NoSQL Backend* buscar novas linhas (3) e enviá-las para o *TrustProcessing*, que por sua vez vai colocar os resultados na *cache* de modo a serem disponibilizados para o cliente (4).

O *TrustProxy* é também a unidade lógica que transforma os pedidos do cliente numa sequência de invocações ao *hardware* confiável. Posto isto, consideremos dois pedidos diferentes, um primeiro *Scan* com uma *start key* e um segundo com um *SingleColumnValueFilter* sobre uma coluna *x*.

Em ambos os casos, o *RegionScanner* começa por retirar toda a informação do *Scan* original, testando não só a presença de uma *start* e/ou *stop Key*, mas também de filtros, assim como o seu tipo (*RowFilter* ou *SingleColumnValueFilter*). Estas verificações revelam qual a operação (primitiva) pretendida para executar no *TrustProcessing*.

No primeiro caso, perante a presença de uma *start key*, o *RegionScanner* reconhece que se trata de uma operação sobre as chaves da base de dados. Posto isto, para cada *batch* de dados proveniente do *NoSQL Backend*, o *TrustProxy* envia o respetivo pedido para o *TrustProcessing*. Após a reposta deste, o *RegionScanner* coloca todos os resultados na *cache* para que o cliente tenha acesso através no método *next()*.

No segundo caso, com a presença de um filtro *SingleColumnValueFilter*, o *RegionScanner* identifica o pedido como uma operação sobre os dados de uma coluna. Este tipo de filtro contém qual a família e qualificador, de forma a saber qual é a coluna sobre a qual o cliente quer operar. Posto isto, o *worker* vai à base de dados buscar os valores posicionados nessa coluna, envia-os para o *TrustProcessing* e, à semelhança do exemplo anterior, coloca os resultados na *cache* para serem disponibilizados para o cliente.

5.4.3 *CryptoBoxes*

5.4.3.1 *CryptoTechnique*

O sistema *SafeNoSQL* contém várias *CryptoBoxes* com os vários esquemas criptográficos já implementados (STD, DET, OPE e FPE). Este conjunto foi estendido com a *CryptoBox SGX*, capaz de disponibilizar a cifra compatível com a cifra utilizada dentro do *TrustProcessing*. Esta *CryptoBox* implementa uma cifra de *block AES* em modo *CTR*. Para isso utiliza um contador, incrementado a cada cifragem, para gerar aleatoriedade. Esta cifra utiliza as primitivas *ippsAESEncryptCTR* [13] e *ippsAESDecryptCTR* [12] da *Intel Integrated Performance Primitives Cryptography Library* (IPPCP). As operações utilizadas do lado *Untrusted* que requerem a cifragem e decifragem de dados respeitam a interface 5.1, composta pelos métodos *init*, *destroy*, *gen*, *encrypt* e *decrypt*.

```
public void init();
```

```
public void destroy();
public byte[] gen();
public byte[] encrypt(byte[] plaintext, byte[] key);
public byte[] decrypt(byte[] ciphertext, byte[] key);
```

Interface 5.1: Interface do módulo *CryptoTechnique*

5.4.3.2 *TrustProcessing*

A unidade de processamento seguro integrada no *TrustNoSQL* é o *Intel SGX*. Como mencionado anteriormente na subsecção 2.7, o conceito central do *Intel* SGX é o *Enclave*, uma zona de memória protegida por um conjunto de instruções incorporados nos processadores, criando assim um ambiente seguro que contém código e dados relativos a uma computação sensível. Os mecanismos de isolamento do SGX destinam-se a proteger a confidencialidade e a integridade da computação realizada dentro de um *Enclave*, de ataques provenientes de *software* malicioso, que executam no mesmo computador, bem como de um conjunto limitado de ataques físicos.

As primitivas utilizadas que permitiram tomar partido do *hardware* confiável foram desenvolvidas na linguagem de programação C. Com o objetivo de abranger todo o leque de operações que o *SafeNoSQL* [28] oferece, tirando partido do *Intel SGX*, foram desenvolvidas várias primitivas dentro do *Enclave*, desde o processamento necessário para pedidos sem ou com os vários filtros (*RowFilter* e *SingleColumnValueFilter*), com ou sem limites de ordem e com os respetivos tipos de operações desejadas.

O *Enclave* está preparado para receber um conjunto de valores cifrados provenientes da base de dados, o valor de pesquisa do pedido original, um eventual filtro num *Scan* com a operação de comparação que pretende fazer, e retornar todos os valores que respeitam estas condições. Para isso, o *Enclave* é capaz de decifrar os valores que recebe, com uma primitiva de decifragem e chave previamente acordada com o cliente, tratar de todo o processamento necessário com os valores decifrados dentro do *Enclave* e por fim, retornar a informação de todos os valores que respeitam as condições do pedido.

Considerando que o processamento no *TrustProcessing* está a ser executado apenas sobre uma parte da base de dados de cada vez, e não sobre a sua totalidade e de forma a otimizar o processo de migração de resultados para fora do *Enclave*, os resultados são retornados na forma de um *array* de inteiros e não o *array* de *bytes* que o *Enclave* recebeu. Este *array* de inteiros corresponde às posições no *array* de *bytes* fornecido pelo *TrustProxy* ao *Enclave*. Isto deve-se ao facto de, na maioria dos casos, o *array* de *bytes* correspondente à informação de cada linha da base de dados ser maior do que o espaço ocupado pelo inteiro (4 *bytes*) correspondente ao seu índice no *batch*. Outra condição que possibilitou esta otimização foi o facto de estes valores poderem ser retornados em *plaintext*, uma vez que apenas revelam

a posição dos dados cifrados num *array* criado para processamento no *TrustProxy*, e não valores sensíveis guardados na base de dados. O *TrustProxy* vai receber então os índices das *rows* que correspondem às soluções e fazer a correspondência para enviar o resultado para o cliente.

As operações sensíveis que necessitam de ser executadas de forma segura e do lado do servidor no *TrustProcessing*, respeitam a interface 5.2, composta pelos métodos *init_enclave*, *destroy_enclave* e *enclave_process*.

```
public int init_enclave();
public int destroy_enclave();
public Pair<Integer, byte[]> enclave_process(byte[] array_encrypted,
   int[] array_encrypted_sizes, int numberOfElements, byte[][]
   value_encrypted, int operation, int type_of_process);
```

Interface 5.2: Interface do módulo *TrustProcessing*

A operação executada do lado *Untrusted*, que permite a criação de um *Enclave* é a *init_enclave*. Esta operação é responsável por, no primeiro pedido que necessitar do *Intel* SGX, criar um novo *Enclave* e guardar o seu *ID*, para que, em pedidos futuros seja utilizado esse mesmo *Enclave* e não forçar a criação de um novo por cada pedido.

A operação que permite a destruição do *Enclave* é a *destroy_enclave*, responsável por libertar a memória utilizada pelo *Enclave* no final da execução do programa.

A operação que permite a execução de código dentro do *Enclave* é a *enclave_process*. Esta primitiva recebe o conjunto de valores que pretendemos comparar dentro do *Enclave* (*byte[] array_encrypted*), um conjunto de inteiros respetivo aos tamanhos dos valores, uma vez que vamos processar estes valores em Linguagem C (*int[] array_encrypted_sizes*), o número de valores que queremos processar (*int numberOfElements*), o(s) valor(es) com que pretendemos comparar o nosso conjunto de elementos (*byte[][] value_encrypted*), um inteiro respetivo à operação que pretendemos caso tenhamos um filtro (*maior*, *maior ou igual*, *igual*, *diferente*, *menor*, *menor ou igual*) (*int operation*) e por fim um inteiro correspondente ao tipo de processamento que quero fazer dentro do enclave, que varia dependendo da presença ou não de um filtro ou de *Start* e *StopKeys* (*int type_of_process*).

Devido a restrições de código dentro do *Enclave*, a *CryptoBox TrustProcessing* foi desenvolvida na Linguagem de Programação *C*. Todos os outros componentes como a *CryptoBox CryptoTechnique*, o *TrustProxy*, *TrustClient*, entre outros, foram desenvolvidos em *Java*. Posto isto, a integração do projeto *Java* com as primitivas SGX desenvolvidas em Linguagem *C* é feita através da utilização do *Java Native Interface*.

O Java Native Interface (JNI) fornece um conjunto de funções em linguagem *C* capazes de interagir com objetos *Java*. Estas funções são compostas por funções genéricas que são utilizadas pelo código nativo, neste caso as primitivas em Linguagem C relativas ao SGX, tornando possível a integração com o código *Java* do resto do sistema.

Por se tratar de um conceito não fulcral nos objetivos do sistema desenvolvido nesta dissertação, as primitivas apresentadas no capítulo 4.2 da arquitetura do *TrustNoSQL* relacionadas com a certificação de *software* não foram implementadas.

6

AVALIAÇÃO EXPERIMENTAL

Com o objetivo de validar e avaliar o sistema *TrustNoSQL* foi realizada uma extensa avaliação experimental. Esta avaliação experimental mede o impacto de processamento seguro em *hardware* confiável com o *TrustNoSQL* e contrasta com vários tipos de sistemas distintos. O primeiro sistema estabelece o desempenho padrão de uma base de dados NoSQL sem processamento seguro, *Baseline*, o segundo sistema estabelece o desempenho de um base de dados NoSQL com processamento seguro assente em esquemas criptográficos que preservam a ordem ou igualdade de criptogramas, e os restantes establecem o desepenho da conjugação de técnicas criptográficas com *hardware* confiável de modo a obter soluções com resultados mais flexíveis. Deste modo, são apresentados neste capítulo os resultados obtidos de vários testes que medem o desempenho dos sistemas em ambientes de teste isolados para cada tipo de pedido NoSQL (micro testes) e em ambientes mais realistas (macro testes), conjugando diferentes tipos de pedidos.

6.1 METODOLOGIA

O *TrustNoSQL* suporta a API NoSQL completa da base de dados *HBase* utilizando a primitiva de processamento segura *enclave_process*. Esta operação, executada em *hardware* confiável, juntamente com o processamento executado pelo *TrustProxy* tem o maior impacto no desempenho do sistema. De modo a avaliar o custo de processamento seguro na API NoSQL foram realizados micro testes que medem de forma isolada o débito e latência de cada operação (*Put*, *Get* e *Scan*).

Os micro testes consideram o custo total das operações, desde cifrar os dados no cliente, o envio das interrogações seguras pela rede, o processamento seguro executado pelo *SafeProxy* até à resposta ao cliente. Portanto, cada teste contém a interação de todos os componentes da arquitetura *TrustNoSQL* e permite contrastar os resultados com o *Baseline* sem considerar o impacto que interrogações distintas podem ter nos sistemas.

Um dos parâmetros críticos para o desempenho do *TrustNoSQL* é a quantidade de pares chaves-valores (*batch*) lidos da base de dados e processados pelo *hardware* confiável em

cada iteração do *TrustProxy*. Na realidade, cada chamada ao *TrustProcessing* tem um determinado custo, relacionado não só com as comparações entre valores, mas também com a transformação e cópia de informação até esta chegar ao *Enclave*. Este valor é também limitado pela tecnologia atual da *Intel* SGX que restringe o tamanho de espaço que pode ser alocado a um *Enclave*. Com o objetivo de definir o número ótimo de linhas que deve ser lido do *NoSQL Backend* e processado no *hardware* confiável em cada pedido, os micro testes mediram o débito do sistema com vários tamanhos de *batch*.

Por fim foram realizados os macro testes que avaliam o desempenho do*TrustNoSQL* num ambiente realista com diferentes distribuições de pedidos (*workloads*). Há semelhança dos micro testes, estes testes consideram a interação entre todos os componentes da arquitetura, mas também medem o impacto das diferentes operações num só teste. Adicionalmente, os testes também demonstram e avaliam a combinação de diferentes abordagens para processamento seguro.

De modo a ter uma avaliação consistente entre todos os testes e sistemas avaliados foram utilizadas as mesmas configurações experimentais, a mesma ferramenta de avaliação e o mesmo esquema de base de dados que simula um caso de estudo real. Estes detalhes são apresentados nas próximas secções.

6.2 CONFIGURAÇÕES EXPERIMENTAIS

A nível de *hardware*, os testes foram realizados utilizando duas máquinas idênticas, com um CPU *dual core* 3.90GHz (*Intel Core i3-7100 CPU*), com 8GB de Random Access Memory (RAM) e um disco Solid-State Drive (SSD) (*SK hynix SC311 SATA*) de 128GB. Ambas as máquinas executam uma distribuição do *Ubuntu 16.04.5 LTS* de 64-bit, a versão de *hardware* do SGX corresponde ao SGX 1.0 e comunicam entre si através de um *switch* de 1 *gigabit*.

As duas máquinas correspondem uma à zona confiável e outra à zona não confiável do *TrustNoSQL*. A máquina correspondente à zona confiável foi utilizada como cliente da base de dados que avaliou o desempenho do sistema. Para isso foi utilizada a plataforma de avaliação do desempenho de base de dados YCSB [23]. A segunda máquina, correspondente à zona não confiável, alojou os componentes *TrustProxy* e *TrustProcessing*, responsáveis por armazenar os dados protegidos e processar as interrogações do cliente.

Para uma avaliação mais precisa, todos os testes executados (micro e macro testes) correram sobre a base de dados previamente povoada com 1 milhão de entradas, durante 20 minutos e repetidos 5 vezes para calcular a média e o desvio padrão de cada execução.

6.3 YAHOO! CLOUD SERVING BENCHMARK

O YCSB [23] é uma plataforma de *benchmarking*. Esta plataforma é *open-source* e foi desenvolvida para avaliar e comparar o desempenho de vários tipos de sistemas de bases de dados. O YCSB é composto por um cliente capaz de gerar pedidos e enviá-los para a base de dados, e pelas *workloads*, um conjunto de cenários de carga de trabalho para testar o desempenho do sistema.

Esta plataforma disponibiliza vários comandos e *workloads* parametrizáveis que permitem simular múltiplos ambientes reais. A parametrização das *workloads* permite dar uma maior flexibilidade ao ambiente de teste, de forma a avaliar aspetos do sistema que não são cobertos pelas *workloads* fornecidas. Cada sistema desenvolvido tem o seu foco, por isso, é necessário criar *workloads* que vão de encontro a uma avaliação específica para cada sistema. Posto isto, cada *workload* é composta por diferentes tipos de operações (*Read*, *Write*, *Filter*, *Scan* entre outras), em que o utilizador pode escolher qual a proporção que pretende atribuir a cada uma, de forma a construir vários cenários de avaliação distintos. Um dos parâmetros de configuração das *workloads* é o tipo de distribuição a aplicar sobre a geração de pares chave-valor e a ordem das operações a serem executadas. O tipo de distribuição utilizada nos testes do *TrustNoSQL* é a distribuição uniforme, que gera valores aleatórios, em que a probabilidade de escolha é igual para todos os valores.

Cada *workload* tem o objetivo de focar uma certa particularidade num sistema. A plataforma disponibiliza um conjunto de *workloads* padrão (A, B, C, D e E) que simulam casos de uso distintos. Por exemplo, a *workload* A simula uma aplicação intensiva em escritas enquanto que a *worload* B e D simulam *workloads* intensivas em leituras. De modo a avaliar todas as funcionalidades do *TrustNoSQL* o conjunto base de *workloads* da plataforma foi estendida para suportar duas *workloads* adicionais que se focam em medir o *overhead* do processamento seguro em aplicações intensivas em leitura. Estas *workloads* também suportam filtros HBase que não é avaliado pela plataforma original.

Após a execução de uma *workload*, as métricas devolvidas pelo YCSB são o débito em operações por segundo (ops/s) e a latência em milissegundos (ms).

6.4 ESQUEMAS DE BASE DE DADOS

De forma a avaliar o sistema *TrustNoSQL*, foi utilizado um esquema de base de dados que simula um caso de estudo real no setor da Saúde [5]. Este caso de estudo é relevante na medida em que é um setor que gere grandes quantidades de informação sensível, como os dados pessoais dos pacientes, dos médicos, informação sobre consultas e infraestruturas de saúde.

A tabela *Consultas* armazena os dados relativos à informação de consultas hospitalares, para um determinado médico e paciente. O esquema é composto por um identificador de linha gerado aleatoriamente pela camada aplicacional, o conjunto de *column families* Médico, Paciente, Consulta e Instituição e as *column qualifiers* Identificador do Médico, Identificação do Paciente, Data, Tipo e Observações sobre a Consulta e Nome e Morada da Instituição.

De forma a avaliar o impacto da adição dos novos componentes relativamente ao sistema base, consideramos dois esquemas criptográficos para a tabela Consultas: o *Baseline* em que as chaves e todas as colunas não têm qualquer tipo de proteção criptográfica, ou seja, estão em texto na base de dados, e um segundo esquema criptográfico *SGX*, em que toda a informação está protegida com a técnica criptográfica implementada no *CryptoTechnique* e, por isso, tem de ser tratada dentro do *Enclave*.

Para os testes direcionados a ambientes mais realistas (macro testes) são propostos mais dois esquemas criptográficos, um primeiro esquema *OPE*, em que o identificador de linha e a coluna da Data da consulta sobre a qual é executado o filtro estão protegidos com a técnica criptográfica *OPE*, sendo que o resto das colunas estão protegidas com *STD*, e um segundo esquema *OPE+SGX*, em que o identificador de linha está protegido com *OPE*, a coluna Data protegida com *SGX* e todas as outras com *STD*. O esquema baseado em *OPE* permite comparar o *TrustNosQL* em relação às bases de dados com processamento seguro assente em cifras, enquanto que o segundo esquema demonstra não só a flexibilidade do *TrustNoSQL* como também tirar proveito do melhor dos dois tipos de sistemas.

Analisando o fluxo de pedidos descrito na secção 4.3, o maior custo introduzido pelo *hardware* confiável revela-se nas operações *Read*, causado pela necessidade de percorrer todas as chaves no *TrustProcessing*. De forma a diminuir este impacto, existe a possibilidade de tratar a cifra *SGX* como uma cifra determinística. A técnica criptográfica *DET* não foi utilizada por não permitir pesquisas de ordem, uma vez que o seu criptograma apenas revela igualdade e não ordem. Assim, não admitindo a hipótese de decifrar os valores para os comparar, foi proposta e desenvolvida uma solução utilizando a técnica criptográfica *SGX*, o esquema *SGX-DET*. Se retirarmos a aleatoriedade introduzida na cifra *SGX* utilizando um parâmetro estático na cifragem, a cifra torna-se determinística, adotando as mesmas propriedades de segurança da cifra *DET*, sendo que, no caso de querermos comparar a ordem de dois criptogramas, podemos executar uma chamada ao *hardware* confiável e processar sobre os dados decifrados no *Enclave*. Este processo permite dotar a cifra *SGX* das mesmas propriedades de segurança da cifra *DET*, mantendo as capacidades de pesquisa da cifra *OPE*, eliminando o esforço de cifragem e decifragem do *OPE*. Este esquema apenas vai ser testado nas *workloads* que contêm a operação *Get*, uma vez que é o único tipo de operações afetadas por esta solução.

Os esquemas criptográficos definidos para a tabela Consultas e o respetivo tamanho de cada coluna em bytes estão apresentados na tabela 2.

	Chave	Médico	Paciente	Consulta			Instituição	
		ID do Médico	ID do Paciente	Data	Tipo	Observações	Nome	Morada
Baseline	7 PLT	10 PLT	10 PLT	12 PLT	10 PLT	10 PLT	10 PLT	10 PLT
SGX	7 SGX	10 SGX	10 SGX	12 SGX	10 SGX	10 SGX	10 SGX	10 SGX
OPE	7 OPE	16 STD	16 STD	14 OPE	64 STD	1024 STD	128 STD	256 STD
OPE+SGX	7 OPE	16 STD	16 STD	12 SGX	64 STD	1024 STD	128 STD	256 STD
SGX-DET	7 DET	16 STD	16 STD	12 SGX	64 STD	1024 STD	128 STD	256 STD

ID - Identificação

Tabela 2: Esquemas da base de dados utilizados.

Como é possível verificar na tabela 2, nos esquemas *OPE* e *OPE+SGX* todos os dados sensíveis e que estão diretamente relacionados com a identidade do paciente ou do médico (identificadores, nome, morada) estão protegidos com *STD*. O identificador de linha e a data da consulta estão protegidos com esquemas criptográficos com garantias de segurança mais relaxadas, de modo a permitir uma computação mais rápida e eficiente sem colocar em risco a informação.

6.5 MICRO TESTES

6.5.1 *Análise do tamanho do batch*

O processamento seguro realizado pelo *TrustNoSQL* requer a leitura, processamento e cópia de dados entre o *TrustProcessing*, o *TrustProxy* e o *NoSQLBackend*. Considerando uma base de dados com x entradas, se estes dados forem enviados linha a linha para o *TrustProcessing*, este custo de chamada ao *Enclave* é introduzido x vezes. Posto isto, e tendo em consideração as restrições de memória impostas pelo *Intel* SGX [24], cada chamada efetuada ao *Enclave* envia um conjunto de entrada y da base de dados, de forma a mitigar o custo do envio de dados para o *hardware* confiável.

Com o objetivo de compreender qual o número de linhas que deve ser lido do *NoSQL Backend* e processado no *hardware* confiável em cada pedido, foram executados testes com a operação *Scan* para vários tamanhos de conjunto de dados (*batch*) enviados para o *Enclave* numa escala logarítmica base 10. O limite máximo suportado pelo *Enclave* para o esquema clínico apresentado são 15000 pares chave-valores. Os resultados estão apresentados na figura 12.

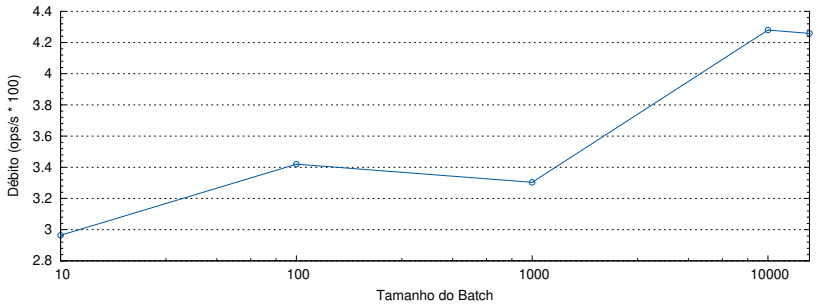


Figura 12: Débito para vários tamanhos de *batch*.

No que diz respeito aos resultados podemos ver que, à medida que aumentamos o tamanho do *batch* o débito em operações por segundo aumenta consideravelmente. Como podemos observar no gráfico, quando aumentamos o tamanho do *batch* de 10 para 100, observamos um aumento de aproximadamente 15% no débito. Observamos ainda um aumento de aproximadamente 30% quando aumentamos o valor para 10000. Todos os testes subsequentes, micro e macro testes, utilizam um *batch* de tamanho 15000, com o objetivo de transferir o maior conjunto de dados possíveis em cada iteração. Adicionalmente, este tamanho de *batch* apresenta um débito de operações por segundo elevado. Contudo, é observada uma descida de débito de um tamanho de 10000 para 15000. Esta descida é atribuída à aleatoriedade dos valores armazenados e filtrados pela base de dados.

6.5.2 *Análise do tempo de execução do Enclave*

Com a introdução do *hardware* confiável no sistema *TrustNoSQL*, é importante perceber qual o impacto isolado da adição deste componente comparado com a globalidade do sistema. Para isso, foi medido o tempo de execução de uma interrogação *Scan* para uma *start key* 500000, de forma a que o resultado tenha uma quantidade de resultados considerável. Assim, foram medidos os tempos de execução em quatro partes distintas código. Em primeiro no *benchmark* YCSB, quando o pedido é transmitido ao *TrustClient*, envolvendo todo o processamento, desde a preparação e cifragem do pedido, acessos à base de dados, tratamento da interrogação no *TrustProxy* e o processamento no *Enclave*. Em segundo foi medido o tempo no coprocessador (*TrustProxy*) quando é preparado o pedido para o *hardware* confiável, eliminando o tempo dos acessos à base de dados comparativamente ao primeiro tempo. De forma a medir o impacto isolado do transporte de dados entre os componentes, chamadas ao *Enclave* e utilização do JNI, foi medido um terceiro tempo aquando da chamada ao *Enclave* por parte do *TrustProxy*. Por fim, foi medido o tempo da execução apenas dentro do *Enclave*.

Posto isto, considerando o tempo medido na plataforma de *benchamrking* YCSB como tempo total da execução, o segundo tempo medido representa apenas 2,47% do tempo

total. Este baixo valor significa que as cifragens do pedido no *TrustClient*, acessos à base de dados e por fim o retorno e decifragem dos resultados no cliente representam a maior parte do tempo total (97,53%) de execução da interrogação. Em seguida, o valor medido na terceira fase do código representa 1,75% do valor total, o que revela o baixo impacto da recolha dos dados e preparação dos pedidos no *TrustProxy*. Por fim, o valor medido apenas na execução dentro do *Enclave* representa 1,74% do tempo total de execução, revelando o baixo custo do JNI uma vez que esta é a única diferença entre estes dois últimos valores medidos. Adicionalmente, podemos concluir que o custo de chamada ao *Enclave* tem um peso mínimo no tempo total de execução, uma vez que o quarto valor medido apresenta um valor muito reduzido quando comparado com o valor medido da execução total da interrogação.

6.5.3 *Análise de operadores NoSQL*

Os micro testes foram efetuados sobre o esquema Consultas, apresentado na tabela 2, de modo a perceber qual o impacto da introdução do *hardware* confiável, comparado com o sistema *SafeNoSQL* sem qualquer alteração. O esquema criptográfico *Baseline* representa o sistema *SafeNoSQL* com um esquema criptográfico em que todos os dados estão em texto na base de dados.

Para a avaliação foram utilizadas cinco *workloads*: *Read*, *Scan*, *Scan* com o filtro *SingleColumnValueFilter*, tendo este duas variantes, uma com o operador do filtro *Equal* e outro com *Greater*, e por fim a *workload Write*. A *workload Read* corresponde a pedidos *Get* dada uma determinada chave, já a *workload Scan* contém sempre uma *start key*, isto é, pesquisa sobre os identificadores de linha e retorna todos os que são maiores do que um determinado valor proveniente do cliente. Adicionando um filtro *SingleColumnValueFilter* à *workload Scan*, temos mais duas *workloads*, uma com filtro de igualdade e outra de ordem de valores. Ambas, à semelhança da *workload* anterior, têm uma *start key* e executam exatamente o mesmo processamento sendo, nestes casos, feita uma segunda chamada ao *hardware* confiável para processar o filtro com o operador respetivo. Por fim, a *workload Write* procede à inserção de linhas na base de dados. Uma vez que os dados para ser inseridos na base de dados têm de ser cifrados, esta *workload* mede o impacto da adição do processo de cifragem dos dados relativamente ao *Baseline*.

Foram executados testes para estas *workloads* com dois tipos de argumentos: em primeiro com o valor de pesquisa (no caso do *Read* e *Scan*) ou de escrita (no caso do *Write*) gerados aleatoriamente pelo YCSB, e uma segunda, com o valor de pesquisa estático a 500000. Este valor está a meio da tabela, o que obriga o *Scan* a devolver para todos os testes não só o mesmo número mas também um valor considerável de resultados. Este segundo teste não foi executado para a *Worlkoad Write*, uma vez que isso significaria inserir sempre o mesmo

		READ	SCAN	SCVF EQUAL	SCVF GREATER	WRITE
Baseline	Débito (ops/s)	354,515 ± 0,366	0,046 ± 0,001	1,056 ± 0	0,132 ± 0,004	508,228 ± 1,952
	Latência (ms)	2,812 ± 0,003	21648,850 ± 353,778	946,682 ± 0,376	7576,193 ± 259,826	1,893 ± 0,007
SGX	Débito (ops/s)	0,214 ± 0,001	0,043 ± 0	0,313 ± 0,001	0,092 ± 0,003	496,152 ± 1,93
	Latência (ms)	4678,964 ± 11,119	22998,147 ± 56,682	3192,958 ± 30,5	10836,187 ± 307,652	1,941 ± 0,008
OPE	Débito (ops/s)	145,372 ± 0,578	0,007 ± 0,001	0,541 ± 0,006	0,013 ± 0,001	77,996 ± 0,161
	Latência (ms)	6,859 ± 0,027	138881,848 ± 25462,417	1846,458 ± 21,281	78202,765 ± 7063,845	12,767 ± 0,027
OPE+SGX	Débito (ops/s)	144,597 ± 0,251	0,008 ± 0,002	0,347 ± 0,052	0,031 ± 0,003	283,513 ± 1,502
	Latência (ms)	6,895 ± 0,012	144099,12 ± 28257,032	2986,205 ± 501,713	32182,856 ± 2990,378	3,388 ± 0,018
SGX-DET	Débito (ops/s)	353,561 ± 1,37	-	-	-	-
	Latência (ms)	2,785 ± 0,01	-	-	-	-

Tabela 3: Resultados do débito e da latência para os micro testes com valores gerados pelo YCSB.

resultado. O primeiro conjunto de testes, com valor de pesquisa e inserção aleatório, é o padrão do YCSB. Contudo, a aleatoriedade dos valores causa operações que requerem *Scans* a transferir quantias diferentes de valores para o cliente o que coloca um custo adicional no sistema. O segundo conjunto de testes, com valores de pesquisa estáticos, remove esta aleatoriedade e permite obter uma comparação idêntica entre sistemas e testes. Os resultados, débito e da latência, para os respetivos testes são apresentados nas figuras 13 e 14 e nas tabelas 3 e 4.

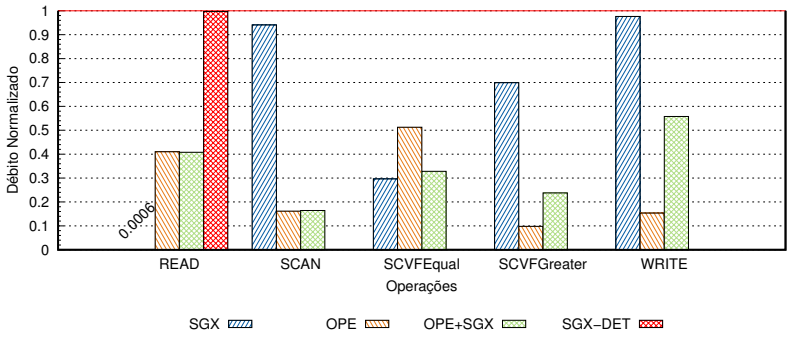


Figura 13: Débito Normalizado para os micro testes com valores gerados pelo YCSB.

		READ	SCAN	SCVF EQUAL	SCVF GREATER	WRITE
Baseline	Débito (ops/s)	363,522 ± 1,106	0,049 ± 0	1,014 ± 0	0,111 ± 0	-
	Latência (ms)	2,742 ± 0,008	20453,134 ± 126,471	985,976 ± 0,175	9004,686 ± 16,204	-
SGX	Débito (ops/s)	0,328 ± 0,002	0,041 ± 0	0,295 ± 0,001	0,148 ± 0	-
	Latência (ms)	3050,492 ± 15,626	24245,371 ± 29,491	3385,457 ± 16,233	6744,845 ± 9,344	-
OPE	Débito (ops/s)	269,432 ± 1,03	0,007 ± 0	0,478 ± 0	0,012 ± 0	-
	Latência (ms)	3,668 ± 0,014	136094,97 ± 859,832	2092,068 ± 1,08	81916,942 ± 74,099	-
OPE+SGX	Débito (ops/s)	267,848 ± 1,312	0,007 ± 0	0,39 ± 0,001	0,03 ± 0	-
	Latência (ms)	3,69 ± 0,018	133796,841 ± 140,101	2561,378 ± 7,212	32878,26 ± 82,717	-

Tabela 4: Resultados do débito e da latência para os micro testes com valores pré-definidos.

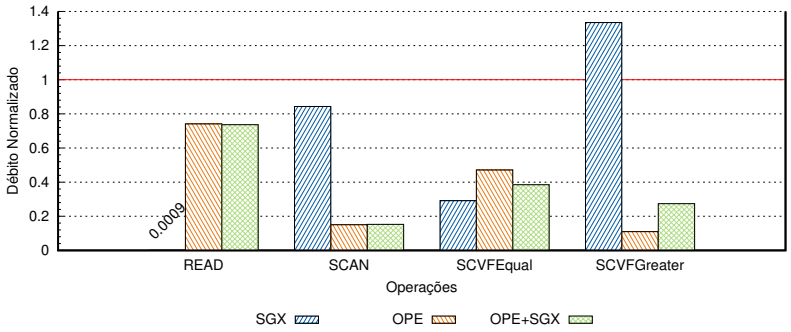


Figura 14: Débito Normalizado para os micro testes com valores pré-definidos.

No que diz respeito aos resultados, é necessário realçar os resultados do comportamento do sistema quando o esquema utilizado tem todas as colunas protegidas com *SGX*, em relação com o *Baseline*, de modo a avaliar o impacto da adição das novas *CryptoBoxes*.

Posto isto, as operações de leitura (*Read*) do *SGX* trazem um custo elevado comparado com os outros três esquemas criptográficos, apresentando um valor de aproximadamente 0,06% do débito do *Baseline* e 0,15% em relação aos esquemas *OPE* e *OPE+SGX*. Este elevado custo deve-se ao facto de o sistema ter de enviar todas as chaves para o *hardware* confiável, decifrá-las e fazer uma pesquisa sobre todas as linhas, contrariamente ao *Get* no *Baseline*, que, tendo acesso às chaves em *plaintext*, faz o acesso direto à linha correspondente e ao *OPE* e *OPE+SGX*, em que são comparados os criptogramas. Os esquemas *OPE* e *OPE+SGX* apresentam débitos de aproximadamente 41% do débito do *Baseline*. Com o objetivo de diminuir este impacto, consideramos o esquema *SGX-DET*, onde o esforço de cifragem e decifragem do *OPE* é eliminado. Este esquema apresenta um débito de cerca de 99,9% do

débito relativamente ao *Baseline*, uma vez que o princípio de pesquisa é o mesmo, em que não é necessário decifrar qualquer tipo de dados nem percorrer toda a tabela para fazer a pesquisa.

As operações *Scan* e *Write* do *SGX*, trazem um custo mínimo comparado com as mesmas operações no *Baseline*, apresentando um débito de 93,5% do *Baseline*. Este custo mínimo contabiliza já o tempo necessário para transferir dados entres os componentes, o processamento seguro dentro do *Hardware* confiável e as operações de cifragem e decifragem da técnica criptográfica. Nos esquemas que utilizam a cifra *OPE* (*OPE* e *OPE+SGX*), o débito é muito inferior, cerca de 15-17%, devido ao custo de cifragem e decifragem da cifra *OPE*.

Por outro lado, no *SGX* as operações *Scan* com o filtro *SingleColumnValueFilter* apresentam um custo significativo comparado com o *Baseline*, cerca de 30% do débito no caso do operador *Equal* e cerca de 70% para o operador *Greater*, causado pelas duas chamadas ao *hardware* confiável. Este custo é agravado pelo carregamento para memória de todas as linhas resultantes do primeiro pedido ao *hardware* confiável, para poderem ser enviadas para a segunda chamada. O *Scan* com o filtro com o operador *Equal*, à semelhança do *Baseline*, embora não revelado pelo gráfico, no *SGX* tem um débito superior, na gama dos 340%, em comparação com o filtro com o operador *Greater*, uma vez que são testadas menos condições para a igualdade de valores do que para a ordem, por exemplo, para a igualdade podemos testar os tamanhos dos valores, se estes forem diferentes podemos imediatamente descartar a entrada. Os esquemas *OPE* e *OPE+SGX* apresentam um débito superior ao *SGX* na operação de filtro com o operador *Equal* (173% e 111% respetivamente) devido à rápida comparação de igualdade entre os criptogramas, uma vez que a comparação é feita sobre os dados cifrados. Contudo, quando é utilizado o operador *Greater* no filtro, os esquemas *OPE* e *OPE+SGX* apresentam um débito inferior (14% e 34% respetivamente) comparado com o *SGX*.

As diferenças entre os testes com os valores de pesquisa variados e estáticos mais evidentes nos *Scan*s e no *SingleColumnValueFilter* com o operador *Greater*, deve-se ao número de entradas da base de dados retornadas pelo *Scan*.

Com a análise da avaliação dos micro testes, estabelecemos que o *SGX* apresenta um débito mínimo de cerca de 0,06% nas operações *Read*, causado pela necessidade de ler todas as entradas da tabela da base de dados. Em relação às operações *Scan* e *Write*, o *SGX* apresenta débitos muito parecidos com o *Baseline* na ordem dos 90%, em comparação com os esquemas com *OPE*, que apresentam débitos na gama dos 15%.

6.6 MACRO TESTES

A execução dos micro testes permitiu perceber os compromissos de desempenho do *CryptoTechnique* introduzido no *TrustNoSQL* isolado, relativamente ao *Baseline*. Contudo, estes

testes não avaliam o sistema como um todo, sendo necessário proceder à avaliação de testes mais realistas, os macro testes. Estes testes foram elaborados para avaliar o impacto no desempenho na combinação de várias técnicas criptográficas para o esquema de base de dados Consultas.

De forma a avaliar o desempenho do *TrustNoSQL* foram executadas as *workloads* com as respetivas proporções, apresentadas na tabela 5.

Workloads	Get	Update	Insert	Scan	SCVF-Equal	SCVF-Greater
A	50%	50%	-	-	-	-
B	95%	5%	-	-	-	-
D	95%	-	5%	-	-	-
E1	-	-	5%	75%	10%	10%
E2	-	-	5%	75%	20%	-

Tabela 5: Proporção das operações *YCSB* em cada *workload*.

Estas *workloads* correspondem às configurações padrão do YCSB e são, tipicamente utilizadas para avaliar bases de dados *NoSQL*. A *workload* E originalmente contém 95% de operações *Scan* e 5% de operações *Insert*, contudo foi modificada de forma a avaliar dois tipos de filtros à semelhança dos micro testes. Para isso, esta *workload* foi modificada, resultando em duas variantes E1 e E2. Ambas as variantes reduzem a proporção da operação *Scan* para 75%, atribuindo os 20% restantes a *Scan*s com o filtro *SingleColumnValueFilter*. Na primeira esses 20% são divididos em metade para as operações *Equal* e *Greater* dentro do filtro. A segunda atribui os 20% à operação *Equal*. Estes *Scan*s com filtro ocorrem sobre a coluna Data do esquema Consultas.

Em relação à *workload-a* e *workload-b*, as operações testadas são *Read* e *Update*. Como explicado na secção 6.5 dos micro testes, a operação *Read* com o esquema *SGX* tem um custo muito elevado, apresentando uma latência cerca de 900-1000 vezes superior ao *Baseline* e *SGX-DET* e 400 vezes superior ao *OPE* e *OPE+SGX*. Estes valores devem-se ao facto de o *SGX* ser o único esquema em que a chave se encontra protegida com *SGX* e não revela informação sobre os dados cifrados. Isto implica uma pesquisa sobre toda a tabela, em que todas as chaves têm de ser decifradas dentro do *Enclave*, ao invés do *Baseline*, em que a chave não tem qualquer tipo de proteção e é acedida diretamente, e dos esquemas *OPE*, *OPE+SGX* e *SGX-DET*, nos quais é possível comparar a igualdade dos criptogramas protegidos com *OPE* e *SGX* no caso da solução *SGX-DET*.

Em particular para a *workload-d*, além das operações *Read* temos as operações *Insert*. O impacto de cerca de 200% relativamente ao débito destas operações no *SGX* em comparação com o *Baseline*, deve-se ao processo de cifragem de cada técnica criptográfica. Como visto na secção 6.5, a técnica criptográfica *OPE* apresenta uma custo de cifragem e decifragem bastante elevado, assim, é expectável que o esquema *OPE*, uma vez que contém não só o identificador de linha mas também a coluna Data protegidas com *OPE*, apresente uma

latência superior ao restantes esquemas, de cerca de 600% relativamente ao *Baseline*, e de cerca de 400% comparando com o *OPE+SGX*, em que apenas o identificador de linha está protegido com *OPE*.

Os resultados destas *workloads* são apresentados nas figuras 15, 16 e 17.

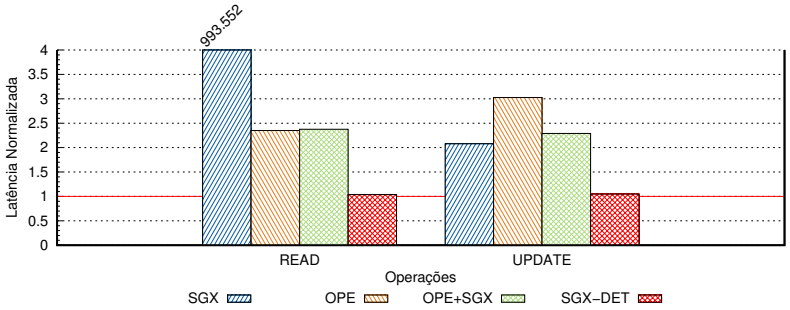


Figura 15: Latência das operações da *workload-a*.

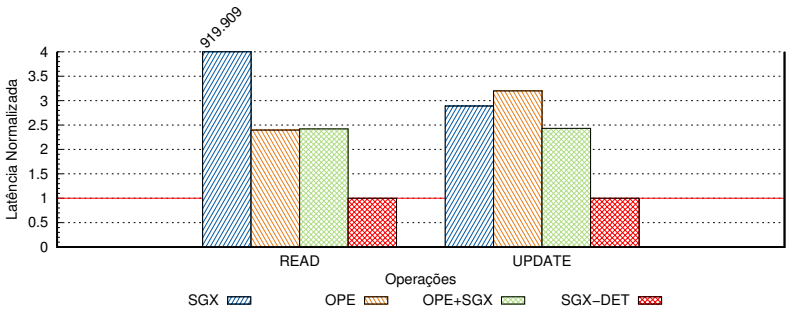


Figura 16: Latência das operações da *workload-b*.

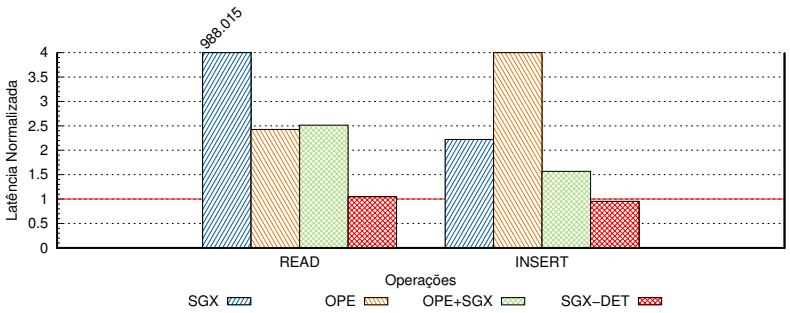


Figura 17: Latência das operações da *workload-d*.

Por sua vez, as *workloads e1* e *e2* apresentam não só operações *Insert* (analisadas no parágrafo anterior), mas também operações de *Filter* e *Scan*. Como visto na secção 6.5, os esquemas criptográficos *OPE* e *OPE+SGX* têm um débito superior quando é utilizado o

operador *Equal* no *Filter*. Ainda assim, concluímos que o esquema *OPE* tem vantagem em relação ao *OPE+SGX* quando se trata do operador *Equal*, e o contrário quando se trata do operador *Greater*, em que o esquema *OPE+SGX* tem um débito superior ao *OPE*. Posto isto, é espectável que na *workload-e2*, em que o operador do filtro é *Equal* para todos os pedidos, o esquema *OPE* tenha uma latência inferior ao *OPE+SGX*, o que, de facto, se demonstra, apresentando uma latência de cerca de 84% comparativamente com o *OPE+SGX*. Contrariamente, na *workload-e1*, uma vez que o operador do filtro se divide entre *Equal* e *Greater*, o esquema criptográfico *OPE* demonstra uma latência de cerca de 240% comparativamente ao *OPE+SGX*, uma vez que nos micro testes, o esquema *OPE+SGX* apresenta um débito superior quando estamos perante um *Filter* com o operador *Greater*. Os resultados destas duas *workloads* são apresentados nas figuras 18 e 19.

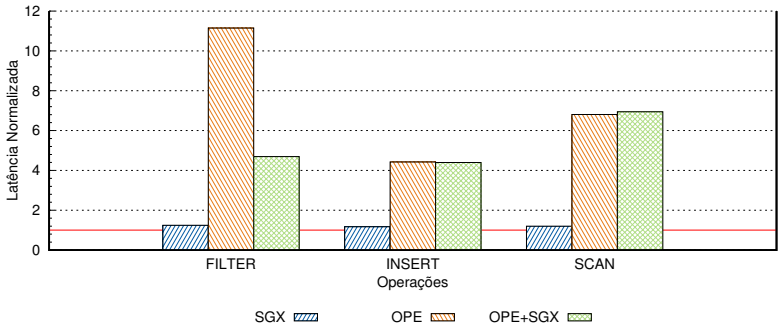


Figura 18: Latência das operações da *workload-e1*.

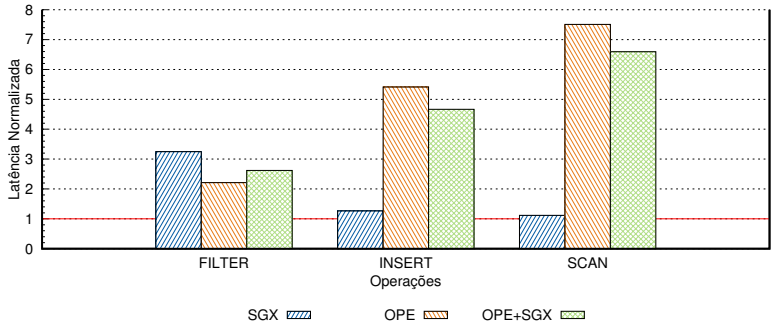


Figura 19: Latência das operações da *workload-e2*.

Em resumo, a solução proposta no esquema criptográfico *OPE+SGX* relativamente ao *OPE*, mostra melhorias nas operações *Insert*, em que a latência é cerca de 4 vezes inferior, e *Filter* quando são utilizados de forma dividida ambos os operadores *Greater* e *Equal*, com latências cerca de 2,4 vezes inferior. Esta diferença deve-se ao facto de que na inserção de uma nova linha, não ter os custos de cifragem e decifragem da técnica *OPE* na coluna Data, sendo que é esta a coluna utilizada no filtro. Contudo, na situação em que são uti-

lizados apenas filtros com o operador *Equal*, o esquema *OPE* apresenta latências inferiores ao *OPE+SGX*, na ordem dos 80%. No que diz respeito à proposta *SGX-DET*, notam-se melhorias na operação *Read*, uma vez que os resultados se apresentam idênticos aos obtidos com o *Baseline*, o que torna esta solução viável na possibilidade de atribuir propriedades determinísticas à cifra utilizada para proteger o identificador de linha. O desempenho do sistema *TrustNoSQL* para todas as *workloads* está apresentado na tabela 6.

		Workload-a		Workload-b		Workload-d		Workload-e1			Workload-e2		
Baseline	Débito (ops/s)	416,39 ± 9,795		341,626 ± 0,392		350,622 ± 1,717		0,059 ± 0,01			0,066 ± 0,002		
	Latência (ms)	Read	Update	Read	Update	Read	Insert	Filter	Insert	Scan	Filter	Insert	Scan
		2,725 ± 0,046	1,984 ± 0,066	2,921 ± 0,003	2,134 ± 0,008	2,823 ± 0,017	2,468 ± 0,01	3233,203 ± 457,424	18,242 ± 1,414	20631,135 ± 1538,814	982,547 ± 121,171	16,130 ± 1,388	20918,562 ± 1011,707
SGX	Débito (ops/s)	0,731 ± 0,002		0,394 ± 0,002		0,378 ± 0,006		0,053 ± 0,003			0,057 ± 0,002		
	Latência (ms)	Read	Update	Read	Update	Read	Insert	Filter	Insert	Scan	Filter	Insert	Scan
		2707,227 ± 12,129	4,125 ± 0,1	2686,891 ± 18,322	6,165 ± 0,105	2788,952 ± 48,744	5,483 ± 0,095	4023,006 ± 100,819	21,334 ± 2,281	24760,333 ± 1455,068	3190,398 ± 19,735	20,417 ± 1,638	23351,931 ± 872,102
OPE	Débito (ops/s)	160,636 ± 0,25		142,631 ± 0,736		137,164 ± 0,154		0,011 ± 0,001			0,01 ± 0,002		
	Latência (ms)	Read	Update	Read	Update	Read	Insert	Filter	Insert	Scan	Filter	Insert	Scan
		6,403 ± 0,014	6,003 ± 0,012	6,998 ± 0,036	6,827 ± 0,044	6,847 ± 0,008	15,2 ± 0,053	36047,989 ± 6336,132	80,672 ± 5,547	140461,071 ± 12246,443	2171,806 ± 461,835	87,328 ± 0	157092,952 ± 18397,699
OPE+SGX	Débito (ops/s)	180,731 ± 0,95		142,848 ± 0,881		143,741 ± 0,782		0,011 ± 0,002			0,012 ± 0,001		
	Latência (ms)	Read	Update	Read	Update	Read	Insert	Filter	Insert	Scan	Filter	Insert	Scan
		6,476 ± 0,034	4,543 ± 0,027	7,074 ± 0,044	5,189 ± 0,035	7,096 ± 0,039	3,869 ± 0,039	15170,81 ± 3486,616	80,192 ± 9,44	143269,093 ± 20651,055	2575,06 ± 623,625	75,264 ± 8,48	137964,955 ± 13040,088
DET-SGX	Débito (ops/s)	399,503 ± 1,13		326,948 ± 1,585		336,628 ± 2,334		-			-		
	Latência (ms)	Read	Update	Read	Update	Read	Insert	Filter	Insert	Scan	Filter	Insert	Scan
		2,826 ± 0,03	2,081 ± 0,019	3,053 ± 0,015	2,256 ± 0,011	2,954 ± 0,021	2,355 ± 0,017	-	-	-	-	-	-

Tabela 6: Resultados dos Macro-Testes.

6.7 DISCUSSÃO

A micro e macro avaliação do *TrustNoSQL* demonstra que é possível processar proteger e processar dados com *hardware* confiável sem libertar informação sensível significativa e com um custo aceitável de performance. O custo de cifrar e inserir pares chave-valor com *STD* é no máximo 0.1% em comparação com o *Baseline*. O processamento de operações NoSQL de pesquisa, *Scans*, tem um custo mais elevado sendo no pior caso 3 vezes mais lento que o *Baseline*. Contudo, este custo é significativamente menor que o custo de soluções protegidas com OPE que podem ser no máximo 11 vezes mais lentas que o *Baseline*. O maior custo do *TrustNoSQL* encontra-se na operação de acesso a um único par chave valor (*Get*) pois necessita de fazer uma pesquisa completa na tabela. Contudo, como as chaves são tipicamente identificadores únicos que não revelam informação, é plausível que este custo não seja crítico, uma vez que pode ser deixado em texto claro ou protegido com uma técnica com garantias de segurança relaxadas como *OPE* ou *DET*. Ainda assim, retirando a aleatoriedade da cifra *SGX* de forma a tratá-la como uma cifra determinística, podemos executar operações *Get* da mesma forma que a cifra *DET*, eliminando a pesquisa sobre toda a tabela, eliminando o custo apresentado sobre o esquema *SGX*.

7

CONCLUSÃO

Esta dissertação apresenta o sistema *TrustNoSQL*, uma base de dados NoSQL com garantias de privacidade assentes em *hardware* confiável. Esta solução recorre à tecnologia *Intel* SGX para garantir o processamento de interrogações num ambiente de execução seguro em infraestruturas não confiáveis, tirando partido de novas instruções incorporadas nestes processadores. Para o desenvolvimento deste sistema, tornou-se essencial a análise do estado da arte atual sobre o *hardware* confiável, incluindo várias tecnologias com propriedades comparáveis com o *Intel SGX*, e sobre bases de dados seguras, englobando sistemas desenvolvidos em bases de dados SQL e NoSQL, assim como algumas soluções assentes em *hardware* confiável. Assim, pela análise do estado da arte, foi possível concluir que existem soluções propostas para a computação segura sobre bases de dados SQL assentes em *hardware* confiável, no entanto, num mundo onde o desenvolvimento das bases de dados está a tomar um rumo a escalabilidade horizontal com processamento seguro é um aspeto essencial, nenhuma solução atual explorou *hardware* confiável.

O objetivo do sistema exposto nesta dissertação consiste em fornecer uma solução que utiliza *hardware* confiável para bases de dados NoSQL, de forma a permitir a integração de soluções em *hardware* confiável. Assim, é apresentada uma arquitetura em que o objetivo é utilizar *hardware* confiável para proteger código capaz de processar dados decifrados de uma forma confidencial, mantendo o suporte à API NoSQL completa da base de dados *HBase*.

Desta forma, a arquitetura do sistema está dividida em duas zonas, confiável e não confiável. A zona confiável representa uma infraestrutura segura do lado do cliente, onde os dados podem ser processados como texto. A esta zona foram adicionados os componentes *CryptoTechnique*, capaz de abstrair as cifras e os respetivos modos necessários ao funcionamento do sistema, e *TrustClient*, um módulo responsável por transformar pedidos inseguros (ou sobre texto desprotegido) em pedidos seguros. A segunda zona está sujeita a ataques e por isso não é confiável, obrigando a que os dados que circulam sejam mantidos no *hardware* confiável ou circulem cifrados. Nesta zona foram adicionados os componentes *TrustProxy*, um módulo capaz de comunicar com o *NoSQL Backend* e encarregue de identificar e tratar os pedidos do cliente, e o *TrustProcessing*, que corresponde à entidade capaz de

processar interrogações no *hardware* confiável de forma confidencial. A modularidade do componente *TrustClient* permite a combinação de técnicas criptográficas como *Standard Encryption*, *Deterministic Encryption*, *Format-Preserving Encryption* e *Order-Preserving Encryption* com a técnica definida no *CryptoTechnique*, de forma a dinamizar a construção de esquemas criptográficos com a finalidade de atribuir garantias de segurança diferentes, de acordo com a necessidade de cada coluna da base de dados.

Sobre este sistema, foi realizada uma extensa avaliação experimental, considerando os micro testes, que revelaram o impacto da introdução do *hardware* confiável em ambientes de teste isolados para cada tipo de pedido NoSQL e os macro testes, apropriados para avaliar o sistema em ambientes mais realistas e conjugando diferentes tipos de pedidos.

Para testes relacionados com a operação *Read*, o *SGX* apresenta um débito mínimo de cerca de 0,06% comparado com o *Baseline*, causado pela necessidade de ler todas as entradas da tabela da base de dados. Contudo, na possibilidade de atribuir propriedades determinísticas à cifra utilizada no *SGX* (esquema *SGX-DET*), este débito aumenta para valores idênticos ao *Baseline*. Em relação às operações *Scan* e *Write*, o *SGX* apresenta débitos muito parecidos com o *Baseline* na ordem dos 90%. Relativamente à solução proposta no esquema criptográfico *OPE+SGX*, relativamente ao *OPE*, o *OPE+SGX* apresenta melhorias nas operações *Insert*, em que a latência é cerca de 4 vezes inferior, e *Filter* com ambos os operadores *Greater* e *Equal*, com latências cerca de 2,4 vezes inferior. Contudo, na situação em que são utilizados apenas filtros com o operador *Equal*, o esquema *OPE* apresenta latências inferiores ao *OPE+SGX*, na ordem dos 80%.

Para concluir, esta dissertação apresenta uma base de dados dotada de SGX que permite a execução das interrogações necessárias a uma base de dados NoSQL, permitindo a utilização de esquemas criptográficos com garantias de privacidade assentes no *hardware* confiável.

7.1 Trabalho Futuro

Com esta dissertação, o sistema *TrustNoSQL* vem adicionar a uma base de dados NoSQL, a utilização de um componente que permite o processamento de dados seguro em *hardware* confiável. Este componente é capaz de processar interrogações de forma a garantir confidencialidade e integridade dentro do *Enclave*. Contudo o sistema *TrustNoSQL* não apresenta garantias de integridade no resto dos componentes da zona não confiável, uma vez que a cifra utilizada não garante a integridade da informação.

Outro tópico interessante seria tomar partido de novas versões do SGX, com o intuito de utilizar novas instruções do CPU introduzidas com o SGX 2.0, possibilitando alocar dinamicamente a memória do *Enclave* [29]. Desta forma, seria possível manter dados armazenados dentro do *Enclave*, de forma a fazer pesquisas mais eficientes.

BIBLIOGRAFIA

[1] Amazon web services (aws) serviços de computação em nuvem. URL https://aws.amazon.com/pt/.

[2] Amd secure technology. URL http://www.amd.com/en-us/innovations/software-technologies/security.

[3] Relatório de segurança de infraestrutura mundial anual - relatório especial da arbor networks. URL http://br.arbornetworks.com/wp-content/uploads/2017/01/12th_WISR_SUMMARY_POR.pdf.

[4] Manage massive amounts of data, fast, without losing sleep. URL http://cassandra.apache.org/.

[5] Clinical database schema. URL https://dcm4che.atlassian.net/wiki/spaces/ee2/pages/2556012/Database+Table+Descriptions.

[6] Ataques ddos se multiplicam nos serviços na nuvem e data centers. URL http://sis-publique.convergenciadigital.com.br/cgi/cgilua.exe/sys/start.htm?from_info_index=81&infoid=41580&sid=97.

[7] Dropbox. URL http://www.dropbox.com/.

[8] Amazon dynamodb – nosql cloud database service. URL https://aws.amazon.com/dynamodb/.

[9] Google drive - cloud storage and file backup for photos, docs and more. URL https://www.google.com/drive/.

[10] Apache hbase – apache hbase™ home. URL https://hbase.apache.org/.

[11] Cloudant. URL https://www.ibm.com/cloud/cloudant.

[12] Aesencryptctr, . URL https://software.intel.com/en-us/ipp-crypto-reference-aesencryptctr.

[13] Aesencryptctr, . URL https://software.intel.com/en-us/ipp-crypto-reference-aesencryptctr.

[14] Welcome to microsoft onedrive. URL https://onedrive.live.com/about.

[15] Yelp case study - amazon web services (aws). URL https://aws.amazon.com/pt/solutions/case-studies/yelp/.

[16] I. H. Akin and B. Sunar. On the difficulty of securing web applications using cryptdb. *IACR Cryptology ePrint Archive*, 2015:82, 2015. URL http://eprint.iacr.org/2015/082.

[17] A. ARM. Security technology building a secure system using trustzone technology (white paper). *ARM Limited*, 2009.

[18] T. W. Arnold, C. U. Buscaglia, F. Chan, V. Condorelli, J. C. Dayka, W. Santiago-Fernandez, N. Hadzic, M. D. Hocker, M. Jordan, T. E. Morris, and K. Werner. IBM 4765 cryptographic coprocessor. *IBM Journal of Research and Development*, 56(1):10, 2012. doi: 10.1147/JRD.2011.2178736. URL https://doi.org/10.1147/JRD.2011.2178736.

[19] S. Bajaj and R. Sion. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *IEEE Trans. Knowl. Data Eng.*, 26(3):752–765, 2014. doi: 10.1109/TKDE.2013.38. URL https://doi.org/10.1109/TKDE.2013.38.

[20] D. Bellin. Review of "php5 and mysql bible by tim converse and joyce park". *ACM Queue*, 3(2):58, 2005. doi: 10.1145/1053331.1053350. URL http://doi.acm.org/10.1145/1053331.1053350.

[21] D. Champagne and R. B. Lee. Scalable architectural support for trusted software. In *16th International Conference on High-Performance Computer Architecture (HPCA-16 2010), 9-14 January 2010, Bangalore, India*, pages 1–12, 2010. doi: 10.1109/HPCA.2010.5416657. URL https://doi.org/10.1109/HPCA.2010.5416657.

[22] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008. doi: 10.1145/1365815.1365816. URL http://doi.acm.org/10.1145/1365815.1365816.

[23] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, pages 143–154, 2010.

[24] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016. URL http://eprint.iacr.org/2016/086.

[25] V. Costan, I. A. Lebedev, and S. Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 857–874,

2016. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[26] F. Lardinois. Google updates drive with a focus on its business users, Mar 2017. URL https://techcrunch.com/2017/03/09/google-drive-now-has-800m-users-and-gets-a-big-update-for-the-enterprise/.

[27] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong. L-encdb: A lightweight framework for privacy-preserving data queries in cloud computing. *Knowledge-Based Systems*, 79:18–26, 2015.

[28] R. Macedo, J. Paulo, R. Pontes, B. Portela, T. Oliveira, M. Matos, and R. Oliveira. A practical framework for privacy-preserving nosql databases. In *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*, pages 11–20, 2017. doi: 10.1109/SRDS.2017.10. URL https://doi.org/10.1109/SRDS.2017.10.

[29] 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.

[30] B. Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[31] B. Pinkas and T. Reinman. Oblivious RAM revisited. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 502–519, 2010. doi: 10.1007/978-3-642-14623-7_27. URL https://doi.org/10.1007/978-3-642-14623-7_27.

[32] R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591, 2016. URL http://eprint.iacr.org/2016/591.

[33] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012. doi: 10.1145/2330667.2330691. URL http://doi.acm.org/10.1145/2330667.2330691.

[34] R. Price. Google drive now hosts more than 2 trillion files, May 2017. URL http://uk.businessinsider.com/2-trillion-files-google-drive-exec-prabhakar-raghavan-2017-5.

[35] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM trustzone to build a trusted language runtime for mobile applications. In *Architectural Support for Programming*

Languages and Operating Systems, ASPLOS '14, Salt Lake City, UT, USA, March 1-5, 2014, pages 67–80, 2014. doi: 10.1145/2541940.2541949. URL http://doi.acm.org/10.1145/2541940.2541949.

[36] D. X. Song, D. A. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000. doi: 10.1109/SECPRI.2000.848445. URL https://doi.org/10.1109/SECPRI.2000.848445.

[37] G. E. Suh, C. W. O'Donnell, and S. Devadas. Aegis: A single-chip secure processor. *IEEE Design & Test of Computers*, 24(6):570–580, 2007. doi: 10.1109/MDT.2007.179. URL https://doi.org/10.1109/MDT.2007.179.

[38] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013. URL http://www.vldb.org/pvldb/vol6/p289-tu.pdf.

[39] A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982. doi: 10.1109/SFCS.1982.38. URL https://doi.org/10.1109/SFCS.1982.38.

[40] 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 *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 283–298, 2017. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zheng.

[41] W. Zheng, F. H. Li, R. A. Popa, I. Stoica, and R. Agarwal. Minicrypt: Reconciling encryption and compression for big data stores. In *Proceedings of the Twelfth European Conference on Computer Systems, EuroSys 2017, Belgrade, Serbia, April 23-26, 2017*, pages 191–204, 2017. doi: 10.1145/3064176.3064184. URL http://doi.acm.org/10.1145/3064176.3064184.