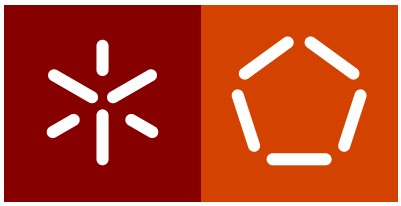
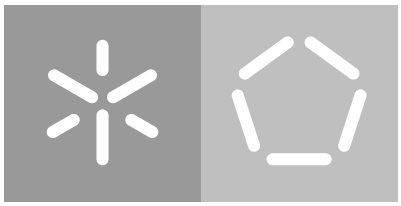


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Gonçalves Macedo

Computação Segura em Bases de Dados NoSQL

Julho 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ricardo Gonçalves Macedo

Computação Segura em Bases de Dados NoSQL

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sob a orientação de
Professor Doutor Rui Carlos Mendes Oliveira
Doutor João Tiago Medeiros Paulo

Julho 2017

AGRADECIMENTOS

Esta dissertação não seria possível sem a ajuda de várias pessoas, as quais eu gostaria de exprimir os meus agradecimentos.

Ao meu orientador, Professor Doutor Rui Carlos Oliveira, pelo apoio e disponibilidade prestados e pela oportunidade que me deu em integrar no grupo de investigação *HASLab*.

Ao meu co-orientador, Doutor João Paulo, por me guiar e motivar durante todo o projeto, mostrando-se sempre disponível para ajudar e pelas múltiplas revisões da minha dissertação. Fico também muito agradecido por toda a paciência dedicada sempre que a frase *"Tive uma ideia"* era pronunciada e por todos os ensinamentos transmitidos durante o projeto.

Ao Rogério Pontes, que desde o início demonstrou um grande interesse no meu trabalho, pela ajuda e disponibilidade em múltiplas ocasiões e por todas as discussões relacionadas e não relacionadas com o trabalho.

Ao Bernardo Portela, Miguel Matos e Tiago Oliveira, pela colaboração na realização do artigo científico resultante desta dissertação.

Aos meus colegas do Grupo de Sistemas Distribuídos do *HASLab*, por se mostrarem sempre disponíveis em ajudar e pelo fantástico ambiente de trabalho.

Aos meus pais, pela educação e valores que me deram durante toda a vida, e por todo o apoio prestado durante todo o meu percurso académico.

Finalmente, quero deixar um agradecimento especial à minha namorada Isabel Barbosa, por me acompanhar e aturar durante todos estes anos, por partilhar comigo todos os bons momentos e nunca me abandonar nos maus.

ABSTRACT

Companies are increasingly dependent on storage and processing solutions that can handle massive amounts of data. Cloud database services are a good fit for this challenge as they provide both capabilities in a remote cloud infrastructure, thus alleviating enterprises from buying and managing their own private data center.

However, cloud services are now a popular target for malicious attacks (*e.g., Sony Pictures Entertainment* hack, *Dropbox* leak of 68 million passwords, *iCloud* leak of celebrity photos) in which large amounts of private and sensitive information, from either large companies or end-users, have been leaked. Also, cloud providers (*e.g., Google*) have access to clients' data and are able to process it to extract meaningful information that may be sold to companies for publicity purposes. Cloud data privacy is thus a main concern that needs to be addressed.

This thesis aims to explore and apply secure computation schemes over *NoSQL* databases while delivering secure data storage, processing and transmission. In detail, the following contributions are presented: a detailed state-of-the-art revision was made for the current secure *SQL* and *NoSQL* database systems, as well as, for the most relevant cryptographic techniques adopted by these systems. Moreover, this review proposes a generic architectural abstraction and a taxonomy for classifying all these solutions. As a second contribution, a generic framework called *SafeNoSQL* is proposed and provides a modular and extensible design that enables secure data processing over *NoSQL* databases. Also, a prototype of *SafeNoSQL*, based on *Apache HBase*, is implemented along with a set of four distinct cryptographic libraries that leverage secure storage and computation over sensible data stored on untrusted third-party infrastructures. The prototype is validated with an extensive experimental evaluation resorting to both micro, macro and multi-client experiments. These experiments validate not only the performance of the solution but also its resource usage (*e.g., CPU, RAM*). The results show that it is possible to achieve a practical and secure solution for realistic workloads with a database throughput and latency overhead inferior to 15%. As a final contribution, we propose the integration of *SafeNoSQL* with a query engine in order to further extend the secure computation and storage guarantees to traditional *SQL* applications.

RESUMO

Hoje em dia é usual as empresas necessitarem de analisar e processar quantidades massivas de dados, levando-as a optar pela utilização de serviços de bases de dados na nuvem, tirando proveito do poder de armazenamento e computação da nuvem reduzindo, desta forma, os custos de aquisição e administração de uma infraestrutura privada.

No entanto, os serviços de nuvem têm vindo a sofrer ataques (*p.e.*, ataque à *Sony Pictures Entertainment*, fuga de 68 milhões de passwords da *Dropbox*, fuga de fotos de celebridades da *iCloud*), onde grandes quantidades de informação crítica e privada, seja de grandes empresas ou do utilizador final, são comprometidas e muitas vezes divulgadas à comunidade geral. Da mesma forma, as empresas que fornecem estes serviços (*p.e.*, *Google*), efetuam análises sobre os dados armazenados com o objetivo de capturar o máximo de informação do utilizador de modo a traçar um perfil, e por fim vender estas informações a empresas por motivos de publicidade.

Dada esta falha de privacidade, esta dissertação visa explorar e implementar mecanismos de computação segura sobre bases de dados *NoSQL* fornecendo armazenamento, processamento e transmissão de dados de forma segura e transparente. Em detalhe, esta dissertação apresenta as seguintes contribuições: uma revisão extensa e detalhada do estado da arte atual dos sistemas de computação segura sobre bases de dados *SQL* e *NoSQL*, bem como sobre os esquemas criptográficos mais frequentes e relevantes suportados por estes sistemas seguros. Deste estudo é ainda proposta uma arquitetura genérica e taxonomia dos sistemas de computação segura atuais. Como segunda contribuição é proposto um sistema denominado *SafeNoSQL* que apresenta uma arquitetura modular e extensível de computação segura sobre bases de dados *NoSQL*. Ainda, esta arquitetura é concretizada num protótipo que suporta a base de dados *Apache HBase* e inclui quatro técnicas criptográficas que permitem o armazenamento e processamento seguro sobre informação sensível armazenada em serviços de terceiros não confiáveis. De forma a validar o protótipo é efetuada uma avaliação experimental detalhada com micro e macro testes e testes de carga com múltiplos clientes, sendo feita não só a avaliação do desempenho mas também da utilização de recursos computacionais (*p.e.*, *CPU*, memória) para cada um dos testes. Os resultados mostram que é possível atingir uma solução segura e funcional para *workloads* realistas, obtendo um custo no desempenho da base de dados inferior a 15%. Como contribuição final, é proposta a integração do sistema *SafeNoSQL* com um componente de tradução de *SQL* para *NoSQL* de forma a extender as vantagens de segurança a aplicações tradicionais que apenas suportam o modelo e interface *SQL*.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE INTERROGAÇÕES E FRAGMENTOS DE CÓDIGO

LISTA DE ACRÓNIMOS

GLSSYMBOLS

2PC Two-Party Computation.

A

AES Advanced Encryption Standard.

API Application Programming Interface.

C

CBC Cipher Block Chaining.

CFB Cipher FeedBack.

CMC CBC-Mask-CBC.

CPA Chosen Plaintext Attack.

CTR Counter Mode.

D

DBMS Database Management System.

DES Data Encryption Standard.

DQE Distributed Query Engine.

E

ECB Electronic Code Book.

F

FBI Federal Bureau of Investigation.

FFSEM Feistel Finite Set Encryption Mode.

FFX Format-Preserving Feistel-based Encryption.

FPE Format-Preserving Encryption.

FPGA Field-Programmable Gate Array.

H

HDFS Hadoop Distributed FileSystem.

I

IAAS Infrastructure as a Service.

IDEA International Data Encryption Algorithm.

IND-CPA Indistinguishability against Chosen-Plaintext Attack.

IND-OCPA Indistinguishability under Ordered Chosen-Plaintext Attack.

IP Internet Protocol.

IV Vetor de Inicialização.

J

JDBC Java Database Connectivity.

M

MAC Message Authentication Code.

MPC Multi-Party Computation.

MR-FPE Multi-Radix Format-Preserving Encryption.

N

NIST National Institute of Standards and Technology.

NSA National Security Agency.

O

OFB Output FeedBack.

OPE Order-Preserving Encryption.

ORE Order-Revealing Encryption.

P

POPF-CCA Pseudorandom Order-Preserving Function advantage under Chosen Ciphertext Attack.

R

RC5 Rivest Cipher 5.

RSA Rivest, Shamir and Adleman cryptosystem.

S

SE Searchable Encryption.

SGX Software Guard Extensions.

SQL Structured Query Language.

T

TPC-C TPC Benchmark ™C.

TPC-H TPC Benchmark ™H.

U

UDF User-Defined Function.

X

XOR Exclusive or.

Y

YCSB Yahoo! Cloud Serving Benchmark.

1

INTRODUÇÃO

Com o decorrer dos anos e a evolução da tecnologia, tornou-se regular o recurso diário a aplicações e serviços *online*, de forma a facilitar a execução de tarefas diárias comuns, aumentando assim a produtividade do indivíduo (*p.e.*, aplicações de mensagens instantâneas, redes sociais, *streaming* de música e vídeo, armazenamento de documentos e multimédia, motores de pesquisa).

Num estudo feito pela *Brandwatch*, estimou-se que no ano de 2015 foram feitas em média 100 mil milhões de pesquisas mensais no motor de pesquisa *Google*, foram carregadas 300 horas de vídeo a cada minuto no *Youtube* e 80 milhões de fotos para o *Instagram* por dia. Relativamente ao *Facebook*, foram registados 500 mil utilizadores por dia. Sobre esta enorme quantidade de dados, são realizadas análises exaustivas, levadas a cabo por organizações como a *Google* e *Facebook*, gerando diariamente Terabytes de informação útil [1][2].

Em muitos casos, esta enorme quantidade de dados é guardada e processada numa base de dados, seja relacional ou não relacional. À medida que os dados armazenados e as interrogações sobre os mesmos aumenta, as empresas que fornecem estes serviços carecem de recursos de computação e armazenamento, o que leva as mesmas a optar pela utilização de serviços de computação de nuvem de terceiros, que seguem um modelo *Infrastructure as a Service (IaaS)* [3][4]. A adoção deste modelo, reflete-se no facto de que para além de proporcionar recursos de armazenamento e computação flexíveis, sobre a forma de uma base de dados tradicional, o custo associado é diretamente proporcional à sua utilização, suprimindo os custos de aquisição e administração de uma infraestrutura privada.

Com a adoção de serviços de computação em nuvem de terceiros, a segurança e privacidade dos dados passa a ser uma preocupação central, na medida em que estes serviços têm vindo a ser alvos de ataques de agentes externos, como foram os casos do ataque à *Sony Pictures Entertainment*, a fuga de 68 milhões de passwords da *Dropbox*, e ainda a fuga de fotos de celebridades da *iCloud*, onde grandes quantidades de informação sensível e privada foi comprometida e divulgada à comunidade geral [5] [6] [7]. Ainda, apesar dos fornecedores destes serviços aplicarem técnicas e medidas de segurança para negar o acesso a entidades não autorizadas, a privacidade dos dados continua vulnerável a quem tem acesso direto às bases de dados subjacentes, como administradores maliciosos e ações de *subpoena*, como

foi o caso *San Bernardino*, em que o *Federal Bureau of Investigation (FBI)* fez uma intimação à *Apple* para decifrar e divulgar toda a informação sensível de um *iPhone 5C* [8] [9].

Para além disso, as empresas fornecedoras dos serviços de nuvem, com auxílio de ferramentas de *data mining* e *data analytics*, efetuam análises intensívas sobre os dados armazenados sem consentimento dos utilizadores, com o objetivo de agregar o máximo de informação dos clientes, de modo a traçar um perfil, com o fim de vender esses resultados a empresas de publicidade (*p.e.*, *Groupon*) ou mesmo agências governamentais, tal como *Snowden* referiu em 2013 sobre as atividades ilegais de vigilância global da *National Security Agency (NSA)* [10].

Esta invasão da privacidade e falta de segurança nos serviços de computação em nuvem atuais são os principais fatores pelos quais muitas empresas continuam reticentes quanto à sua adoção. Um exemplo prático disso aconteceu em março de 2011, quando a seguradora americana *HealthNet* anunciou uma violação de privacidade de quase 2 milhões dos seus clientes, tendo sido expostos os seus nomes, moradas, números de segurança social, problemas de saúde e ainda dados financeiros. Estes dados estavam armazenados nas infraestruturas de dados da *IBM* e não estavam cifrados [11].

1.1 Problema e Objetivos

Hoje em dia as bases de dados, relacionais e não relacionais, são uma componente fundamental para um enorme número de empresas. No entanto, muitas destas empresas estão ainda reticentes quanto à utilização de soluções de nuvem para as suas bases de dados devido à migração do controlo dos dados para uma terceira entidade, a nuvem. Este fator leva, em muitos casos, ao compromisso da privacidade dos dados armazenados e processados nestas infraestruturas.

Como tal, um dos meios usados para proteger esta informação sensível é recorrer a técnicas criptográficas e de computação segura [12]. No estado da arte atual existem já diversas propostas que recorrem a técnicas criptográficas e de computação segura seja para bases de dados relacionais como não relacionais [13][14][15]. Contudo é possível identificar alguns problemas nestas soluções, devido à estaticidade e fraca modularidade da arquitetura do sistema devido às técnicas criptográficas sobre ele assentes. Ainda, este problema conta com um estado da arte diminuto no que diz respeito à computação segura em bases de dados não relacionais. Estas estão cada vez mais populares devido à sua flexibilidade e escalabilidade e são, hoje em dia, utilizadas por diversas empresas e utilizadores.

Com a escolha de esquemas criptográficos que fornecem garantias de segurança fortes a informação é armazenada de forma segura e privada, mas o processamento na nuvem sobre essa informação protegida passa a ser diminuto devido à falta de preservação das

propriedades do conteúdo original. Assim há a necessidade de trazer toda a informação armazenada para a infraestrutura do cliente de forma a ser decifrada e processada.

Para ultrapassar este desafio, há a possibilidade de recorrer a esquemas criptográficos que preservam parte das propriedades do conteúdo original. Desta forma, ao fornecer garantias de segurança mais relaxadas, o processamento da informação protegida na nuvem torna-se possível, contudo a possibilidade de ataques bem sucedidos sobre os dados é acrescida.

Podemos concluir assim que quantas mais propriedades do conteúdo original são preservadas pelos esquemas criptográficos, mais susceptíveis estão os criptogramas a ataques, havendo assim um compromisso entre a segurança, desempenho e funcionalidades permitidas sobre os dados.

De forma a garantir privacidade da informação sensível num ambiente de computação em nuvem prentende-se com esta dissertação dotar bases de dados *NoSQL* com diversas combinações de técnicas criptográficas (*p.e.*, *Standard Encryption*, *Deterministic Encryption*, *Order-Preserving Encryption* [16][17][18]) através de uma *framework* genérica, modular e extensível. A genericidade permite adaptar a *framework* a bases de dados *NoSQL* do tipo *Key-Value Store*. Já a modularidade e extensibilidade da *framework* permitem agilizar a combinação de técnicas criptográficas a integrar no sistema.

Ainda, é impreterível a avaliação extensa do sistema desenvolvido por forma a analisar os compromissos entre desempenho, operações de bases de dados permitidas e garantias de segurança fornecidas.

1.2 CONTRIBUIÇÕES

Apresentados os objetivos, esta dissertação apresenta um conjunto de contribuições distribuídas em três fases distintas: revisão detalhada e classificação do estado da arte; desenho, desenvolvimento e avaliação de um sistema de computação segura sobre bases de dados *NoSQL*; integração do protótipo desenvolvido com um tradutor de interrogações *SQL*. Em maior detalhe:

- Foi feita uma revisão detalhada da literatura dos esquemas criptográficos mais frequentes e relevantes nos sistemas de computação segura atuais, sendo o foco principal o reconhecimento das garantias de segurança assentes, principais características das cifras e que tipo de propriedades do conteúdo original são preservadas. Foi feita ainda uma análise extensa do estado da arte atual dos sistemas de computação segura para bases de dados *SQL* e *NoSQL*. Deste estudo foi possível abstraír uma arquitetura genérica destes sistemas de computação segura atuais, identificando os principais módulos constituintes, bem como o tipo de computação que é feita sobre os mesmos. Ainda, foi feita uma taxonomia que permite classificar as soluções atuais pelo tipo de bases de dados, processamento e técnicas criptográficas utilizadas.

- Como segunda grande contribuição, esta dissertação propõe um novo desenho de uma arquitetura modular e extensível sobre bases de dados *NoSQL* do tipo *Key-Value Store* de forma a colmatar a estaticidade e fraca modularidade dos sistemas atuais. Este sistema denomina-se *SafeNoSQL* e foi desenvolvido um protótipo sobre a base de dados *Apache HBase*. Neste protótipo estão incluídos quatro esquemas criptográficos: o esquema *Format-Preserving Encryption* corresponde a uma contribuição desta dissertação, ao passo que os esquemas *Standard Encryption*, *Deterministic Encryption* e *Order-Preserving Encryption* foram desenvolvidos por investigadores do *HASLab INESC TEC*. Ainda, foi realizada uma avaliação experimental detalhada do protótipo do sistema *SafeNoSQL* através de micro e macro testes que contemplam a medição do desempenho das técnicas criptográficas, gestão dos recursos consumidos e testes de carga com múltiplos clientes.

- Como terceira contribuição, o sistema *SafeNoSQL* foi integrado com um tradutor de interrogações *SQL*, com a finalidade de permitir extender as garantias de computação segura a aplicações *SQL*.

Estas contribuições permitiram chegar a um sistema de computação segura sobre bases de dados *NoSQL* e *SQL* completamente modular, funcional e escalável que permite o uso flexível de técnicas criptográficas, levando a um balanceamento ideal entre o desempenho e a segurança do sistema nunca antes registado nas soluções precedentes.

Como resultado desta dissertação foi realizada uma publicação com o título *"A Practical Framework for Privacy-Preserving NoSQL Databases"* na conferência *36th IEEE International Symposium on Reliable Distributed Systems (SRDS 2017)*.

1.3 Estrutura da dissertação

No capítulo 2, é feita um revisão da literatura sobre esquemas criptográficos relevantes para a computação segura sobre dados cifrados. Este capítulo define como se comportam os vários esquemas em termos de segurança e complexidade. No capítulo 3 refere-se o estado da arte atual sobre computação segura sobre bases de dados relacionais e não relacionais, decrevendo as arquiteturas dos vários sistemas, bem como os esquemas criptográficos assentes, as funcionalidades oferecidas e o desempenho dos sistemas. O capítulo 4 descreve o desenho da arquitetura modular e extensível do sistema *SafeNoSQL*, descrevendo o papel de cada um dos módulos criados. No capítulo 5 é feita uma descrição da implementação da arquitetura proposta sobre uma base de dados *NoSQL*, descrevendo os tipos de técnicas criptográficas assentes e funcionalidades permitidas. No capítulo 6 é feita uma avaliação detalhada sobre o protótipo produzido de forma a avaliar o desempenho das técnicas criptográficas adotadas e das operações seguras através de um *benchmark* com *workloads*

realistas. No capítulo 7 é feita uma descrição da arquitetura, integração e avaliação do *SafeNoSQL* com um tradutor de interrogações *SQL*. Por último, no capítulo 8 são feitas algumas considerações finais sobre o trabalho desenvolvido e é ainda discutido o trabalho futuro.

2

ESQUEMAS CRIPTOGRÁFICOS DE COMPUTAÇÃO SEGURA

Com o decorrer dos anos, surgiu a necessidade do desenvolvimento de técnicas criptográficas fortes e seguras que visam assegurar a proteção de informação sensível. Por sua vez, foi observado que existia também a necessidade de preservar algumas das propriedades do conteúdo original, relaxando as garantias de segurança dos criptossistemas. Assim, a combinação destes esquemas criptográficos possibilita a implementação de sistemas de computação segura sobre bases de dados relacionais e não relacionais. Neste capítulo é feito um levantamento dos principais esquemas criptográficos que estão presentes nos vários sistemas de computação segura atuais.

2.1 GARANTIAS DE SEGURANÇA

A criação de um esquema criptográfico surge com a necessidade de assegurar a segurança e privacidade de um determinado tipo de informação. Por vezes, a sua criação visa restabelecer um esquema precedente ou fornecer uma nova opção de proteção, devido à sua fraca adoção, segurança relaxada ou impossibilidade de computação sobre os dados protegidos. Como tal é necessário fornecer garantias de segurança sobre os esquemas criptográficos de modo a que os utilizadores tenham perceção do grau de segurança de uma cifra ou que tipo de operações podem ser efetuadas sobre os criptogramas.

2.1.1 *IND-CPA*

Indistinguishability against Chosen-Plaintext Attack (IND-CPA) é uma definição formal de privacidade sobre a dificuldade de uma entidade A conseguir decifrar a informação protegida por uma entidade D [19]. Nesta definição, A não tem acesso à chave criptográfica (uma chave criptográfica é uma *string* de *bits* utilizada por um algoritmo criptográfico para cifrar dados e decifrar criptogramas) e escolhe duas mensagens em *plaintext* arbitrárias do mesmo comprimento. A fornece essas mensagens a D que irá cifrar aleatoriamente uma das mensagens. Intuitivamente, fornecendo o criptograma resultante a A, o esquema criptográfico diz-se seguro se este tem dificuldade em distinguir qual das duas mensagens foi cifrada.

Assim, pode-se dizer que um esquema criptográfico é seguro contra ataques IND-CPA se um adversário não consegue obter vantagem significativa (*i.e.*, não consegue decifrar um criptograma em tempo útil), considerando que este tem recursos computacionais limitados. De notar, que dado o estado da arte atual, não existem recursos computacionais suficientes capazes de quebrar a cifra num período de tempo aceitável.

2.1.2 *PRIV*

A garantia de segurança IND-CPA apenas consegue ser assegurada em algoritmos criptográficos aleatórios/probabilísticos, que geram criptogramas diferentes para conteúdos iguais. Como tal, esquemas criptográficos capazes de preservar propriedades do conteúdo original não conseguem garantir este requisito, em particular *Deterministic Encryption*, que gera sempre o mesmo criptograma para o mesmo conteúdo. Assim, há a necessidade de definir uma garantia de segurança mais relaxada capaz de assegurar privacidade em esquemas determinísticos.

Deste modo, surge o *PRIV*, uma definição formal de privacidade para esquemas criptográficos determinísticos [20]. Esta garantia define que um esquema criptográfico com propriedades determinísticas considera-se seguro se não revelar mais que a igualdade entre dois criptogramas.

2.1.3 *IND-OCPA*

De forma a comparar a ordem entre dois criptogramas, foram criados os esquemas *Order-Preserving Encryption (OPE)*, contudo não conseguem cumprir os requisitos das garantias de segurança IND-CPA e *PRIV* por revelarem a igualdade e ordem entre valores. Desta forma *Boldyreva et al.* propôs o *Indistinguishability under Ordered Chosen-Plaintext Attack (IND-OCPA)*, um modelo de segurança mais relaxado que o IND-CPA e o *PRIV* [18].

Esta garantia define que um esquema criptográfico do tipo OPE considera-se seguro se não revelar mais do que a igualdade e a ordem entre dois criptogramas. Contudo, *Boldyreva* considerou que a garantia de segurança imposta era impraticável para contextos reais, visto que era necessário gerar criptogramas com tamanhos impraticáveis.

Assim, o mesmo autor propôs uma outra garantia para esquemas OPE, *Pseudorandom Order-Preserving Function advantage under Chosen Ciphertext Attack (POPF-CCA)*, apresentando garantias mais relaxadas que IND-OCPA, sendo mais tarde verificado que não apresenta propriedades suficientemente fortes para proteger informação sensível [21][22].

2.2 SYMMETRIC ENCRYPTION

Os algoritmos que utilizam a mesma chave criptográfica para cifrar o conteúdo original dos dados (*plaintext*) e decifrar o criptograma, denominam-se por algoritmos de chave simétrica. Estes algoritmos dividem-se em dois tipos, cifras sequenciais e cifras por blocos.

2.2.1 *Cifras Sequenciais*

As cifras sequenciais processam o *plaintext* caractere a caractere por intermédio de um gerador de chaves pseudo-aleatório, tendendo a ser muito eficientes. Para a sua construção é necessário que: o tamanho da chave de cifragem seja o maior possível (sempre maior do que a mensagem a transmitir), visto que esta chave servirá de semente do gerador e a sequência de cifragem é cíclica; a sequência da chave deve ser pseudo-aleatória e imprevisível, dado que a segurança da cifra reside na dificuldade em prever a sequência de valores gerados; a chave de cifragem nunca deve ser usada mais que uma vez, devido à forte possibilidade de reconhecimento de padrões [23].

2.2.2 *Cifras por Blocos*

Devido às vulnerabilidades das cifras sequenciais surgiram as cifras por blocos, que como o nome indica, processam o *plaintext* em blocos de bits de tamanho fixo (tipicamente 128 ou 256 bits). Para produzir blocos de tamanho apropriado, são associadas à função de cifragem unidades de partição e *padding*. O *padding* é um mecanismo necessário para assegurar o correto funcionamento de alguns modos de operação destas cifras, visto que obrigam a que o *plaintext* tenha tamanho múltiplo do tamanho do bloco. Assim este mecanismo introduz caracteres ao *plaintext* até que os blocos em que se insere fiquem completamente preenchidos. Para decifrar aplica-se a função inversa da função de cifragem aos blocos do criptograma e revertem-se os processos de *padding* e partição, de forma a obter a informação original correta.

As cifras por blocos para além de garantirem a confidencialidade e autenticidade, permitem auxiliar a construção de outros sistemas criptográficos, como é o caso de funções de *hash*, geradores pseudo-aleatórios e *Message Authentication Code (MAC)* [24]. Um MAC pode ser visto como uma função de *hash* cujo resultado depende de uma chave e da mensagem a cifrar; tipicamente é utilizado para garantir a autenticidade e integridade de uma mensagem, permitindo detetar ataques de manipulação de blocos.

Relativamente à sua construção, as cifras por blocos focam-se em características interessantes como as propriedades de confusão e difusão de *Shannon* [25]. A confusão define que cada bit do criptograma deve ser uma função complexa dos bits do *plaintext*, tornando

difícil encontrar propriedades estatísticas entre o criptograma e o *plaintext*. Já a difusão define que cada bit do *plaintext* deve afetar o maior número de bits do criptograma, escondendo assim as propriedades estatísticas da mensagem.

Este tipo de cifras permite ainda a utilização de vários modos, em que cada um assume um balanceamento entre os critérios segurança, eficiência, propagação e tolerância a erros, e ainda a variação do débito. Os modos mais comuns são o *Electronic Code Book (ECB)*, o *Cipher Block Chaining (CBC)*, o *Cipher FeedBack (CFB)*, o *Output FeedBack (OFB)* e o *Counter Mode (CTR)* [26].

O ECB não prevê o uso de um *Vetor de Inicialização (IV)*, logo um bloco cifrado duas vezes com a mesma chave resulta em criptogramas iguais. Um IV é um vetor de *bits* que serve para dar aleatoriedade ou pseudo-aleatoriedade a algoritmos criptográficos. No que diz respeito à segurança, os padrões do *plaintext* não são disfarçados, fazendo com que a repetição de blocos seja detetável (*code book attack*), tornando assim este modo vulnerável a ataques por repetição e substituição. Quanto à eficiência, este modo permite que qualquer bloco possa ser decifrado independentemente, para além disso, permite também fazer o processamento paralelo da informação. Quando existe a ocorrência de um erro de um bit num bloco do criptograma, não há propagação do erro, sendo que é apenas afetado um só bloco após a decifragem.

O CBC é o modo de funcionamento recomendado para aplicações genéricas, sendo muito utilizado para cifrar ficheiros, onde os erros são pouco frequentes. Este modo usa um IV (tipicamente aleatório) no primeiro bloco para tornar cada mensagem única, dando assim início ao processo. Este IV será necessário na decifragem, pelo que pode ser enviado em claro com o criptograma, ou preferencialmente, cifrado com ECB. Na cifragem, os padrões do *plaintext* são mascarados pelo *Exclusive or (XOR)* e este esquema de cifragem, pode ou não assumir propriedades determinísticas quanto à geração de cifras iguais para conteúdos em plaintext iguais. Este modo prevê ainda a utilização de um MAC, permitindo detetar ataques por manipulação de blocos.

Quanto à eficiência, é possível paralelizar a decifragem mas não a cifragem, devido ao encadeamento do processo desta operação, que faz depender a cifragem de um bloco de todos os que o antecedem. Este encadeamento irá também permitir aquando um erro, a sua propagação, sendo que quando ocorre um erro num bit do criptograma, irá ser afetado o bloco correspondente e um bit do bloco seguinte.

O CFB corresponde a uma cifra sequencial, visto que a sequência de bits da chave é retirada da saída da função de cifragem. A função de cifragem deste modo opera sobre blocos de um tamanho pré-determinado, sendo necessário o auxílio de um *shift-register*. Um *shift-register* é um vetor com tamanho arbitrário preenchido tipicamente com um IV. A cada execução do processo ocorre um *shift* de uma posição do vetor de bits, entrando um novo bit para o início do vetor e saindo o último. Este *shift-register* está inicialmente preenchido

com um IV e com o decorrer das execuções, os bits do criptograma vão substituindo os bits do IV. A alteração do IV mostra-se determinante, visto que como em todas as cifras sequenciais, a repetição de uma sequência de chaves torna a cifra vulnerável a ataques.

Quanto à propagação de erros, um erro no criptograma tem como efeito imediato uma decifragem errada do bit do *plaintext* correspondente. Para além disso, enquanto o bit errado estiver no *shift-register*, todo o *plaintext* estará errado.

O OFB tal como o CFB, corresponde a uma cifra sequencial. O *shift-register* e o IV cumprem a mesma função que no CFB, contudo o *shift-register* é alimentado pelos bits da chave já utilizados, fazendo com que a chave seja independente da mensagem. Neste seguimento, os padrões de *plaintext* são mascarados pela pseudo-aleatoriedade da sequência de chaves, e tal como no CFB, a alteração do IV é essencial. Relativamente à eficiência, não faz sentido falar em paralelismo das operações de cifragem e decifragem na medida em que a sequência de chaves geradas não depende do criptograma. Contudo é possível antecipar a geração de chaves, tornando a cifragem bastante eficiente. Neste modo não há propagação de erros, visto que um erro num bit do criptograma afeta apenas o bit do *plaintext* respetivo.

O CTR tal como o OFB, simula uma cifra sequencial, desta vez com auxílio de um contador. À função de cifragem é-lhe aplicado um IV (tipicamente aleatório) e é conjugado com um contador. A cada cifragem, o contador é incrementado, permitindo assim produzir valores distintos para todos os blocos. Este modo permite paralelismo na cifragem e decifragem dos blocos, visto não haver dependência entre o processamento dos mesmos. Para além disso, como não há dependência de blocos também não há propagação de erros, sendo que quando ocorre um erro num bit do criptograma, apenas o bit respetivo do *plaintext* estará errado.

Por fim, existem diversos algoritmos que contemplam cifras por blocos tais como: *Advanced Encryption Standard (AES)*, *Data Encryption Standard (DES)*, *International Data Encryption Algorithm (IDEA)*, *Rivest Cipher 5 (RC5)* e *Blowfish* [27][28][29][30][31]. Atualmente o algoritmo mais usado é o AES que opera com blocos de tamanho compreendido entre 128 e 256 bits e chaves de tamanho 128, 192 ou 256 bits.

De modo a uniformizar os termos usados quanto às técnicas criptográficas, por omissão, chamar-se-á *Standard Encryption (STD)* a uma cifra aleatória (*p.e.,* AES-128 CBC com IV aleatório) e *Deterministic Encryption (DET)* a uma cifra que assume propriedades determinísticas (*p.e.,* AES-128 CBC com IV fixo).

2.3 FORMAT-PRESERVING ENCRYPTION

O recurso à criptografia tradicional é principalmente útil para garantir a privacidade dos dados num contexto de armazenamento em computação de nuvem, cifrando os dados sensíveis dos utilizadores numa base de dados. Contudo este tipo de criptografia altera

a estrutura original da base de dados e dos valores inseridos na mesma, restringindo na maioria dos casos o poder de computação do motor de base de dados. Exemplos desta limitação são a impossibilidade de restringir o tamanho de um *input* específico, bem como de preservar o formato de um valor, como é o caso de datas.

Dado este problema, *Black et al.* propôs o primeiro esquema de *Format-Preserving Encryption (FPE)*. Este esquema criptográfico é capaz de proteger a informação sensível e simultaneamente manter a estrutura de dados original [32].

A noção de FPE foi proposta com o sentido de gerar criptogramas com o mesmo formato e domínio que o *plaintext*. Mais especificamente, este esquema permite manter o tipo de dados e o comprimento do conteúdo original no criptograma gerado, tornando-se bastante útil quando há uma restrição do comprimento ou do formato dos dados. Esta característica permite que a privacidade dos dados seja garantida de forma transparente à base de dados, *p.e.*, se os dados do *plaintext* forem do tipo *Integer* o formato do respetivo criptograma será também do tipo *Integer*.

Tipicamente as técnicas FPE tem como base a cifra de *Feistel*, também conhecida como circuito de *Feistel*. Esta cifra, com auxílio de uma cifra simétrica, permite efetuar uma série de rondas sobre o conteúdo original, em que em cada uma das rondas são feitas permutações e cifragem de metade dos bits. Ao fim das rondas é obtido o criptograma resultante.

Com o decorrer dos anos foram propostas várias técnicas criptográficas de FPE, permitindo cifrar a informação sensível e mantendo o seu formato, como é o caso do *Feistel Finite Set Encryption Mode (FFSEM)* que permite preservar as características dos tipos numéricos; o *Format-Preserving Feistel-based Encryption (FFX)* que cifrar caracteres com comprimento finito (*p.e.*, *nchar*, *nvarchar*); o *Multi-Radix Format-Preserving Encryption (MR-FPE)* que visa preservar caracteres de tamanho finito (*p.e.*, *char*, *varchar*); e ainda uma abordagem que preserva as características de datas (*p.e.*, o tipo *datetime* das bases de dados relacionais) [33].

Quando comparada com uma cifra AES-*128* em *CBC*, torna-se bastante vantajosa em termos de comprimento do criptograma. Assumindo que se pretende proteger o número de um cartão de crédito (16 dígitos), ao utilizar a abordagem AES o resultado será um criptograma com blocos de 128 bits (mais *padding*), enquanto que cifrando com FPE o criptograma será um número com 16 dígitos.

Contudo para o formato do *plaintext* ser preservado, há um compromisso na segurança. O primeiro ataque a tomar em consideração é por força bruta se o domínio do criptograma for muito pequeno. Outro tipo de ataque, são os ataques por dedução. Considerando o exemplo anterior, após cifrar o número do cartão de crédito, o criptograma, apesar de cifrado, continua a preservar as mesmas características que um número de um cartão de crédito normal. O mesmo se pode verificar para datas e endereços *Internet Protocol (IP)*.

2.4 ORDER-PRESERVING ENCRYPTION

Tanto os esquemas criptográficos tradicionais (*p.e.*, *Standard* e *Deterministic Encryption*) como o esquema *Format-Preseving Encryption* têm a limitação de não preservarem a ordem entre dois criptogramas. Num contexto prático, um exemplo desta limitação é se pretendermos pesquisar numa base de dados quais os utilizadores que têm uma idade superior a x anos. Com este tipo de esquemas criptográficos há a necessidade transportar a computação para o cliente, enviando-lhe todos os dados necessários para serem decifrados e comparados pelo grau de grandeza. Este procedimento resulta num baixo desempenho da base de dados e por vezes torna a mesma inaplicável em contextos de aplicações de tempo real. Devido a este fator, surgiu o conceito de OPE.

O primeiro esquema de OPE foi proposto por *Agrawal et al.* em que o seu objetivo é possibilitar operações de pesquisa sobre dados cifrados numa base de dados relacional sem ter que os decifrar, aumentando assim o desempenho destas operações quando comparadas com uma cifra aleatória ou determinística [34].

Como já referido, a principal propriedade deste esquema é preservar a ordem dos dados em *plaintext* após a cifragem, ou seja, a ordem numérica de uma mensagem em *plaintext* corresponde à ordem numérica dos criptogramas.

$$Enc_{key}(P_1) > Enc_{key}(P_2) \text{ sse } P_1 > P_2$$

Desta forma, na abordagem de *Agrawal*, é possível processar eficientemente sobre conteúdo cifrado as operações *SQL* dependentes da ordem dos valores (*p.e., RANGE*, *COUNT*, *MAX*, *MIN*, *GROUP BY* e *ORDER BY*). No caso das operações *SUM* ou *AVG*, há a necessidade de decifrar os criptogramas, visto que este esquema não possibilita a executação operações algébricas sobre conteúdo protegido como é o caso do criptossistema de *Paillier* [35].

Com o decorrer dos anos, foram propostas novas implementações de OPE, contudo são soluções mais orientadas ao desempenho do sistema e menos à segurança, ou seja, de forma a obter esquemas com um melhor desempenho, apresentam modelos de segurança mais fracos (inexistentes em alguns casos) [34] [36].

A adopção de garantias de segurança mais relaxadas ou da sua inexistência resulta em fugas de informação para além da ordem. Em alguns casos, verifica-se a aprendizagem por parte do atacante de pelo menos metade dos bits do *plaintext* [21].

2.4.1 *Order-Revealing Encryption*

Recentemente, *Boneh et al.* propôs uma generalização do esquema OPE denominado de *Order-Revealing Encryption (ORE)* [37]. Em contraste ao OPE tradicional, os criptogramas do ORE não necessitam de ser elementos de um conjunto ordenado de valores. Em vez disso,

a ordem dos criptogramas rege-se por uma função (pública) que permite verificar qual dos criptogramas é maior, ou seja, não há a necessidade de decifrar os valores para determinar a sua relação de ordem. Contudo, tal como nos esquemas OPE, as instâncias de ORE são apenas provadas seguras sobre garantias de segurança relaxadas, como o IND-OCPA [38][39].

2.5 SEARCHABLE ENCRYPTION

Muitos sistemas de computação sobre bases de dados relacionais e não relacionais fazem uso de várias técnicas criptográficas apresentadas previamente, permitindo aumentar a privacidade dos dados e simultaneamente tirar partido do processamento dos serviços de computação em nuvem. Contudo nenhuma das técnicas anteriores permite a pesquisa de prefixos/sufixos sobre uma tabela da base de dados.

Para resolver este problema, surgiu o esquema criptográfico *Searchable Encryption (SE)*, que permite efetuar a procura de palavras-chave (prefixos e/ou sufixos) em criptogramas [40]. Originalmente estes criptogramas eram direcionados à proteção (e respetiva pesquisa segura) de frases e documentos (*p.e., e-mails*).

Na primeira abordagem do esquema SE, foram propostas duas formas de efetuar a pesquisa sobre criptogramas: (i) realizar uma pesquisa à base de dados sequencialmente; (ii) construir um índice de palavras-chave, em que para cada palavra-chave ser-lhe-á associada uma lista das localizações dos criptogramas que contém a mesma.

Na primeira proposta, para cada documento a ser armazenado no servidor, cada palavra que o compõe será cifrada com uma cifra determinística, possibilitando a igualdade de valores entre criptogramas. Para o cliente procurar com sucesso todos os documentos que contém uma palavra-chave W, esta será cifrada (com a mesma chave que cifra os documentos), resultando em W_{Enc} e enviada para o servidor. Após a recepção da palavra-chave, o servidor efetua uma pesquisa sequencial por todos os elementos da base de dados, comparando cada uma das palavras dos documentos cifrados com W_{Enc}. Por fim, todos os documentos que contém a palavra-chave, serão enviados para o cliente e finalmente decifrados. Devido à grande quantidade de comparações a ser feita, esta abordagem ajusta-se mais a bases de dados de pequenas dimensões [40].

Na segunda proposta, para cada documento a ser armazenado no servidor, o cliente irá identificar palavras-chave associadas a este. O documento é cifrado e armazenado no servidor, e cada uma das palvras-chave irá guardar a localização do mesmo. Estas palavras-chave serão também cifradas e armazenadas num índice, que por sua vez será guardado no servidor. Quando o cliente efetuar uma pesquisa sobre uma palavra-chave W, esta será cifrada resultando em W_{Enc} e enviada para o servidor. Após a recepção da palavra-chave, o servidor irá pesquisar no índice a ocorrência desta e irá enviar todos os documentos que a

contém. Finalmente, serão decifrados pelo cliente. Esta solução traz uma grande otimização no tempo de pesquisa dos documentos, face à proposta anterior. Contudo, sempre que é adicionado, atualizado ou removido um documento há uma sobrecarga na atualização do índice [40].

A garantia de segurança assente sobre o esquema SE é igual ao esquema de *Deterministic Encryption* (*PRIV*), na medida em que as palavras-chave são cifradas com o mesmo tipo de cifra. Sendo assim, o esquema apresenta fugas de informação quanto à igualdade de valores aquando a comparação de dois criptogramas.

2.6 HOMOMORPHIC ENCRYPTION

A utilização de técnicas criptográficas como *Deterministic Encryption* e *Order-preserving Encryption* permitem o processamento seguro de informação sensível. Contudo, com este tipo de técnicas, torna-se impossível realizar operações algébricas sobre criptogramas, limitando o desempenho da base de dados para interrogações analíticas. Assim, para ultrapassar esta limitação, surgiu o esquema criptográfico de *Homomorphic Encryption* que possibilita o processamento de operações algébricas sobre informação cifrada [41].

Esquemas criptográficos de *Homomorphic Encryption* assumem propriedades probabilísticas e com garantia de segurança IND-CPA. Este tipo de esquemas permite a realização de operações algébricas sobre criptogramas, permitindo ao servidor efetuar computação sobre informação cifrada sendo o resultado final decifrado pelo cliente. Este tipo de esquemas é dividído em duas categorias: *Fully-Homomorphic Encryption* (*FHE*) e *Partial-Homomorphic Encryption* (*PHE*). *Fully-Homomorphic Encryption* considera esquemas criptográficos probabilísticos que permitem efetuar múltiplas operações algébricas *p.e.*, a adição, subtração, *xoring*, multiplicação [41][42]. Contudo por permitir um domínio de operações mais abrangente, o desempenho da cifra é severamente penalizado devido aos modelos de computação e segurança [43]. O modelo de segurança indica que nada pode ser revelado na computação sobre a informação cifrada. O modelo de computação assume a geração de circuitos boleanos de grandes dimensões para cada tipo de computação (*p.e.*, algumas implementações desta cifra implicam a transação de 73TB de informação protegida para processar 4MB de *plaintext* [44]). Desta forma, *FHE* não é o esquema criptográfico mais apropriado para assegurar a privacidade da informação sensível em bases de dados ou aplicações de tempo real.

Na segunda categoria, *Partial-Homomorphic Encryption* relaxa o modelo de computação apresentado em *FHE* ao concentar a computação do esquema criptográfico a um conjunto bastante mais limitado de operações algébricas *p.e.*, XOR, adições ou multiplicações [42]. Apesar deste tipo de operações algébricas não suportar análises estatísticas complexas, a

sua aplicabilidade revela-se útil no processamento de agregações da bases de dados (*p.e.*, *SUM*, *AVG*).

2.6.1 *Paillier Encryption*

Paillier Encryption é um esquema critográfico do tipo *Partial-Homomorphic Encryption* que permite efetuar a adição de duas mensagens cifradas fornecendo garantias de segurança IND-CPA [35]. Dados dois valores x e y e a chave criptográfica k, este esquema criptográfico assume a seguinte propriedade:

$$\begin{gathered} x = HOM_{Dec}(k, HOM_{Enc}(x)) \\ y = HOM_{Dec}(k, HOM_{Enc}(y)) \\ x + y = HOM_{Dec}(k, HOM_{Add}(k, HOM_{Enc}(x)) + HOM_{Add}(k, HOM_{Enc}(x))) \end{gathered}$$

em que HOM_{Enc}, decifragem HOM_{Dec} e adição HOM_{Add} correspondem às funções de cifragem, decifragem e adição, respetivamente.

Com a necessidade de proteger a informação sensível das bases de dados, este esquema criptográfico tem sido cada vez mais adotado por sistemas de computação segura [13][14].

2.7 MULTI-PARTY COMPUTATION

Uma solução alternativa às técnicas criptográficas anteriores é o *Secret Sharing* [45]. Esta técnica foi inicialmente proposta por *Shamir* que consiste em dividir um valor D em n partes, D_1, D_2, ..., D_n, de tal forma que tendo conhecimento de k partes, D é facilmente computável. Por outro lado, mesmo que se tenha conhecimento de $k-1$ partes, não é possível obter informação sobre D.

O esquema *Secret Sharing* é também conhecido por esquema de *threshold* (k, n) em que n é o número de partes cujo valor D será dividido e k é o número de partes necessárias para obter o valor original. Por exemplo, num esquema *threshold* $(3, 5)$ é possível obter D combinando D_1, D_2, D_3 ou D_2, D_3, D_5 ou qualquer outro subconjunto de D desde que $k = 3$.

O *threshold* (k, n) assume algumas propriedades interessantes: o tamanho de cada parte não excede o tamanho dos dados originais; as partes D_i são facilmente trocadas sem alterar o conteúdo original D, levando a uma aumento da segurança dos dados sensíveis. Este *threshold* permite ainda que seja possível construir um esquema hierárquico sobre os tuplos, fornecendo diferentes níveis de acesso a quem possui as chaves.

Dadas estas características, os esquemas de *Secret Sharing* permitem garantir a privacidade dos dados, dados estes que poderão vir a ser armazenados em serviços de terceiros. Contudo, este tipo de esquemas não foi formulado para fornecer processamento seguro sobre os segredos.

Devido a esta limitação surgiram os protocolos de *Multi-Party Computation (MPC)*. Estes protocolos permitem fazer computação distribuída sobre informação privada, recorrendo a um número arbitrário de *parties* que comunicam entre si.

Dentro das soluções de MPC existem duas componentes principais: *Two-Party Computation (2PC)*, para um ambiente de duas *parties* e MPC para um ambiente *multi-party*.

Uma das primeiras e principais soluções de 2PC é o protocolo de *Yao*, também conhecido por *Yao's Garbled Circuits* [46]. Trata-se de um protocolo eficiente, visto que o número de iterações é independente da função a ser avaliada, e seguro contra adversários passivos, ou seja, é seguro contra adversários que apenas querem ler a informação. Este protocolo é visto como um circuito boleano com *inputs* binários de tamanho fixo. Tipicamente, em protocolos como o de *Yao*, cada *party* tem um papel, levando a que a computação em cada uma seja diferente e que em termos de segurança, se uma *party* for comprometida, apenas os dados dessa *party* ficam comprometidos.

Num ambiente *multi-party* é seguida a solução MPC. Nestes protocolos cada *party* exerce o mesmo tipo de computação, comunicando entre si para gerar o *output* correto. Maioritariamente, estes protocolos são construídos sobre esquemas de *Secret Sharing*, fazendo com que seja possível atingir a privacidade dos dados armazenados e o processamento privado da informação sensível.

Durante a computação da informação sensível, nenhuma *party* deverá aprender qualquer informação acerca do *input* de outra *party* para além das informações que possam ser inferidas através do *output*. Esta propriedade permite que as fugas de informação sejam inexistentes, a não ser que mais que uma *party* seja comprometida. Esta técnica criptográfica assume assim IND-CPA como garantia de segurança.

No estado da arte atual de MPC, *Bogdanov* propôs o sistema *Sharemind* que permite fazer computação segura sobre bases de dados relacionais, mais especificamente, num contexto de análise de dados [47]. Ainda, um sistema baseado em algumas propriedades do *Sharemind*, é o *SafeRegions* proposto por *Pontes* que permite fazer computação segura sobre bases de dados não relacionais [48].

2.8 DISCUSSÃO

Feito um levantamento dos esquemas criptográficos mais relevantes nos presentes sistemas de computação segura, torna-se possível verificar que são esquemas com características variadas, com garantias de segurança diferentes, e consequentemente, as propriedades preservadas dos criptogramas diferem face ao conteúdo original. A tabela 1 descreve as propriedades preservadas do conteúdo original em cada uma das técnicas criptográficas. As propriedades apresentadas são a igualdade, ordem, formato, pesquisa e operações algébricas. A igualdade e a ordem referem-se à verificação da igualdade e ordem entre dois criptogramas, respetivamente. O formato rege-se à preservação do tamanho e/ou do tipo de dados gerado do criptograma. A pesquisa entende-se a técnicas que permitem a pesquisa de um prefixo e/ou sufixo de uma palavra. Por fim, operações algébricas entende-se como a propriedade que permite fazer adições sobre valores cifrados.

	PLT	STD	DET	OPE	FPE	SE	PLR	MPC
Igualdade	✓		✓	✓	✓			✓
Ordem	✓			✓		✓		✓
Formato	✓				✓			
Pesquisa	✓					✓		
Ops. Algébricas	✓						✓	

Tabela 1: Propriedades preservadas nas várias técnicas criptográficas.

PLT refere-se ao *plaintext* ou seja, o conteúdo original. *STD* e *DET* referem-se a *Standard Encryption* e *Deterministic Encryption* respetivamente. Ambas as técnicas pertencem ao esquema criptográfico *Symmetric Encryption*. *OPE* refere-se a *Order-Preserving Encryption*, *FPE* a *Format-Preserving Encryption* e *SE* a *Searchable Encryption*. Por sua vez, *PLR* refere-se a *Paillier Encryption* e *MPC* a *Multi-Party Computation*.

3

COMPUTAÇÃO SEGURA EM BASES DE DADOS

Existem já várias propostas que visam assegurar a computação segura em bases de dados relacionais e não relacionais. O principal propósito deste capítulo é mostrar o estado da arte atual deste tipo de sistemas em ambientes relacionais e não relacionais, descrevendo as noções gerais de cada sistema, a sua arquitetura, o tipo de computação efetuada e ainda os esquemas criptográficos assentes.

Tipicamente as soluções de computação segura em bases de dados (*SQL* ou *NoSQL*) seguem uma arquitetura genérica composta por duas entidades: uma entidade segura (*trusted site*) e uma entidade não confiável (*unstrusted site*). O *trusted site* corresponde ao ponto de acesso dos clientes da base de dados que podem ser simples computadores pessoais ou um *cluster* confiável controlado pelo proprietário dos dados ou por alguém da confiança dos proprietários dos dados. O *unstrusted site* é onde a maioria dos dados são processados, geralmente constituído por um ou mais fornecedores de serviços de nuvem, em que os dados não são controlados pelos seus possuídores e é necessário ter em conta as vulnerabilidades de segurança assentes. Na figura 1 é apresentada uma visão abstrata desta arquitetura genérica.

Na camada não confiável estarão alojados o *backend* da base de dados e o sistema de ficheiros onde irão ser armazenados e processados os dados. Nesta camada assume-se o modelo de segurança para atacantes *honest-but-curious*, sendo que toda a informação sensível deverá estar devidamente cifrada, bem como as interrogações a processar [24]. Este modelo assume que os adversários como administradores de nuvem maliciosos, *hackers* ou agentes de *subpoena* conseguem visualizar a informação armazenada mas não a corrompem.

Em algumas bases de dados é ainda possível extender o comportamento do motor de base de dados sem modificar o código fonte, através de *Extended Behavior Functions*.

Já a camada segura está dividida em duas partes: a camada aplicacional e o *database client*. Na camada aplicacional encontra-se toda a lógica da aplicação e uma *key store* criptográfica que faz toda a gestão das chaves criptográficas de todos os clientes. Na camada *database client* é onde todo o processo de segurança ocorre através dos módulos de computação *Query Translator*, *Crypto Layer*, *Mapping Layer* e *Result Digester*.

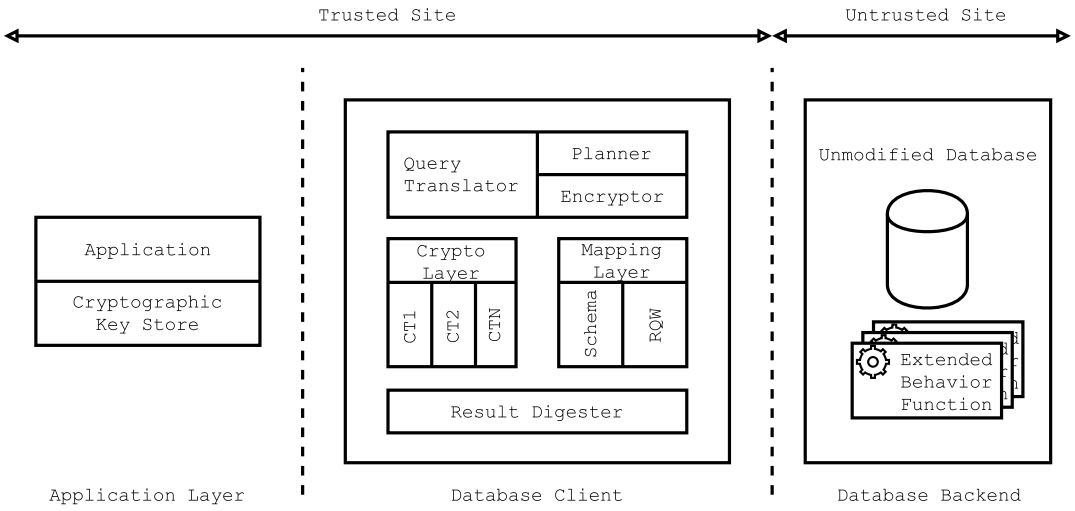


Figura 1: Arquitetura genérica dos sistemas de computação segura.

O *Query Translator* intercepta as interrogações da camada aplicacional e gera uma nova interrogação através do *Query Planner* e do *Query Encryptor*. O *Query Planner* faz *parsing* à interrogação original e transforma-a em uma ou mais interrogações, distribuíndo a computação entre as entidades *Trusted Site* e *Untrusted Site* devido à falta de interoperabilidade das técnicas criptográficas, ou então, são totalmente executadas pelo servidor. O *Query Encryptor* cifra os valores que compõem a interrogação por forma a processa-los de forma segura.

O módulo *Crypto Layer* está responsável por fazer o *encoding* e *decoding* dos dados sensíveis de uma dada técnica criptográfica, *CT*. A *Mapping Layer* visa fornecer um mapeamento do esquema (*Schema*) da base de dados e um *RQW (Representative Query Workload)*. O primeiro, permite identificar as colunas da base de dados e as técnicas criptográficas que as protegem. O último pretende dar conhecimento ao sistema das interrogações mais frequentes e como dividir a computação entre as duas entidades de forma a ter o melhor desempenho possível. Estas interrogações auxiliam ainda na construção do esquema anterior influenciando diretamente o tipo da técnica criptográfica que protege uma determinada coluna devido ao tipo de computação a ser exercído sobre a mesma.

Por fim, o módulo *Result Digester* compila o resultado das interrogações processadas no servidor, decifrando-os e efetuando a computação necessária para retornar o resultado correto à camada aplicacional.

3.1 COMPUTAÇÃO SEGURA EM BASES DE DADOS SQL

As bases de dados relacionais, também conhecidas como base de dados *SQL*, são uma coleção de objetos com relações pré-definidas entre si [49]. Estes objetos, formalmente 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 múltiplas tabelas através de chaves estrangeiras, criando assim o conceito de relação da informação. Estes dados podem ser acedidos e interrogados de várias formas sem haver a necessidade de reorganizar as tabelas da base de dados.

Bases de dados deste tipo como *MySQL* e *Oracle Database* estão presentes na grande maioria das aplicações que usamos diariamente [50][51][52][53][54]. Motivo disso é o uso de interrogações *Structured Query Language (SQL)*, a integridade dos dados, as transações e as propriedades *ACID*.

A linguagem SQL é a interface primária de comunicação com as bases de dados relacionais, sendo usada para gerir todos os aspetos da base de dados, adicionar, ler, atualizar e eliminar dados e ainda efetuar computação atómica sobre a base de dados.

Para assegurar a integridade, precisão e coerência dos dados, as bases de dados relacionais dispõem de um conjunto de restrições (*contraints*) para reforçar a integridade dos dados armazenados. Estas restrições incluem as chaves primárias, chaves estrangeiras e as restrições *Not Null*, *Unique*, *Default* e *Check*.

As transações das bases de dados são uma ou mais declarações *SQL* que são executadas como uma sequência de operações à base de dados que formam uma única unidade de trabalho lógica. As transações seguem um modelo *"tudo ou nada"*, em que uma transação é bem sucedida se todas as declarações forem corretamente executadas resultando num *COMMIT*, caso contrário, se uma ou mais declarações não for bem sucedida a transação é falhada e é feita uma operação de *ROLLBACK* sobre as declarações previamente executadas.

Estas transações têm de seguir impreterivelmente as propriedades *ACID*, por outras palavras, têm de respeitar as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade. A atomicidade necessita que uma transação seja totalmente bem sucedida ou, no caso de uma parte falhar, toda a transação tem de ser inválida. A consistência exige que os dados escritos na base de dados como parte de uma transação têm de respeitar todas as regras e restrições definidas. O isolamento é uma propriedade crítica para atingir o controlo de concorrência e garante que cada transação é independente. Finalmente, a durabilidade necessita que todas as modificações feitas à base de dados sejam permanentes assim que a transação for completada com sucesso.

Contudo, uma das limitações das bases de dados relacionais (e não relacionais) é a falta de privacidade e segurança da informação sensível.

3.1.1 *CryptDB*

Para resolver as limitações de segurança e privacidade nas bases de dados *SQL*, surgiu o *CryptDB* , um dos principais e mais conhecidos sistemas que permite efetuar computação segura sobre bases de dados relacionais [13].

O *CryptDB* é um sistema que fornece privacidade a aplicações que recorrem a um *Database Management System (DBMS)* alojado em servidores não confiáveis. Para isso, tal como demonstra a figura 2 este sistema divide-se numa camada *middleware* (*proxy*), alojada na parte segura, que interceta as interrogações submetidas pela camada aplicacional, analisando e reescrevendo as mesmas nas respetivas operações sobre dados cifrados e num DBMS não modificado, alojado na camada não confiável [13].

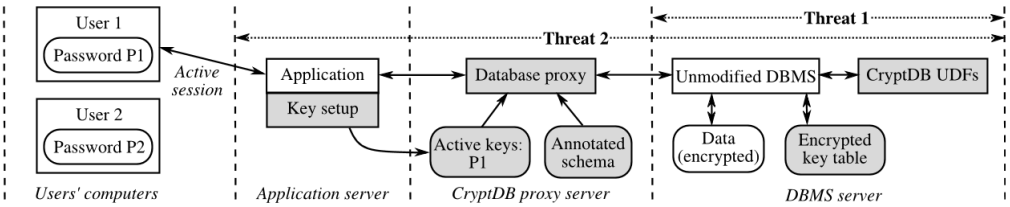


Figura 2: Arquitetura do sistema *CryptDB*.

O *CryptDB* assume dois tipos de ameaças. A primeira ameaça são os administradores da base de dados curiosos ou outros adversários externos que tenham acesso total à camada DBMS. Estes adversários tentam aprender informação sensível sobre os utilizadores (*p.e.*, registos de saúde, movimentos bancários). Para prevenir que estes adversários possam então ganhar conhecimento sobre a identidade dos utilizadores, na medida em que estes têm apenas controlo sobre a camada DBMS, o *CryptDB* executa interrogações *SQL* sobre dados cifrados no servidor DBMS. Desta forma, os adversários apenas conseguem tomar conhecimento da informação sensível através da fuga de informação assente nas várias técnicas criptográficas. Contudo apenas é garantida a privacidade da informação, não sendo dadas quaisquer garantias quanto à integridade dos dados. Por outras palavras, estes adversários podem editar ou remover dados armazenados no servidor.

A segunda ameaça são os adversários que ganham acesso total às camadas aplicacional, *middleware* e DBMS. Para este tipo de adversários, o *CryptDB* não fornece nenhuma garantia aos utilizadores com a sessão iniciada no sistema, podendo ver todos os dados submetidos à camada aplicacional em *plaintext*, bem como os dados armazenados no DBMS por terem acesso às chaves de cifragem e decifragem. Já para utilizadores sem sessão iniciada no

sistema, são dadas as mesmas garantias que a primeira ameaça, já que os adversários não têm acesso às chaves de cifragem e decifragem destes utilizadores.

Para combater estas ameaças existem dois desafios principais. O primeiro desafio passa pela escolha da técnica criptográfica indicada para cifrar os dados, de forma a que o compromisso entre a quantidade de informação sensível revelada ao DBMS e a habilidade de executar eficientemente uma grande variedade de interrogações seja minimizada. É necessário considerar que recorrendo a um esquema seguro de cifragem aleatória (*p.e., Standard Encryption*), os dados iriam efetivamente estar seguros, contudo iria impossibilitar a execução de grande parte das interrogações ao DBMS.

O segundo desafio é minimizar a quantidade de fuga de informação quando um adversário compromete a camada aplicacional para além do servidor DBMS.

Para resolver estes desafios o *CryptDB* define três ideias chave: *estratégia de cifragem SQL-aware*, *cifragem ajustável à interrogação* e *chaves de cifragem em cadeia sobre as passwords dos utilizadores*.

A primeira ideia, *estratégia de cifragem SQL-aware*, visa executar interrogações *SQL* sobre dados cifrados. Sabe-se originalmente que as interrogações *SQL* são compostas por operadores primitivos, como é o caso das comparações de igualdade e de ordem e adições. Adaptando criptossistemas conhecidos a este tipo de operadores, o *CryptDB* cifra cada um dos dados recebidos de tal forma que seja possível o servidor DBMS executar operações sobre os mesmos.

Para processar interrogações o *proxy* do *CryptDB* armazena uma *master key* secreta *MK*, o esquema da base de dados e as camadas de cifragem atual para todas as colunas. Por sua vez, o servidor DBMS contém um esquema anónimo, dados cifrados, tabelas auxiliares a ser usadas pelo *CryptDB* e ainda *User-Defined Function (UDF)*, que são funções que permitem ao servidor extender o seu comportamento. Neste caso, permitem fazer computação sobre criptogramas de forma segura.

A execução de uma interrogação no *CryptDB* passa por quatro passos. Primeiro a aplicação envia uma interrogação *SQL*, que é intercetada e reescrita pelo *proxy*. Os identificadores da tabela e das colunas são anonimizados, e usando a *master key MK*, todos os campos da interrogação são cifrados de acordo com o esquema de cifragem que melhor lhe assenta. No segundo passo, o *proxy* verifica se há a necessidade de ajustar as camadas de cifragem no servidor, antes de executar a interrogação. Se se confirmar, o *proxy* envia uma interrogação de atualização ao servidor, que invocará uma UDF para decifrar a primeira camada de segura de determinadas colunas. No terceiro passo, o *proxy* reencaminha a interrogação cifrada para o DBMS, que será executada seguindo o padrão de *SQL*. Por último, o DBMS retorna o resultado cifrado, ao *proxy*, que por sua vez o irá decifrar e enviar à camada aplicacional.

Relativamente aos esquemas criptográficos, o *CryptDB* recorre a *Standard Encryption* (STD) (representado na figura 3 como *RND*), *Deterministic Encryption* (DET), *Order-Preserving Encryption* (OPE), *Homomorphic Encryption* (HOM), *Join* (JOIN e OPE-JOIN) e *Searchable Encryption* (SE) (representado na figura 3 como *SEARCH*).

Para o STD pode ser usada uma cifra por blocos AES ou *Blowfish*, a operar no modo CBC com um IV aleatório. Por este esquema não permitir exercer computação eficiente sobre os dados cifrados, é utilizado como camada mais externa das *onions* de igualdade e ordenação, visto que apresenta garantias de segurança muito fortes, IND-CPA. Uma *onion* pode ser vista como um grupo de esquemas criptográficos que estão agrupados de forma hierárquica segundo as garantias de segurança assentes.

Para o esquema DET é usada uma cifra por blocos AES ou *Blowfish*, a operar no modo *CBC-Mask-CBC (CMC)* com um IV fixo [55]. O modo CMC envolve aplicar duas rondas de CBC, sendo a primeira um CBC normal e na segunda ronda aplica-se o CBC aos blocos por ordem inversa. Estes esquema é utilizado numa camada interna da *onion* de igualdade, e permite fazer operações como igualdade de valores, *GROUP BY* ou *COUNT*.

O esquema OPE será usado numa camada interna da *onion* de ordenação e possibilita que sejam feitas operações do tipo *range query*, como por exemplo *ORDER BY*, *MIN* ou *MAX*.

Os esquemas de *Homomorphic Encryption* são esquemas de cifragem probabilística, que fornecem garantias de segurança IND-CPA. Estes esquemas são tipicamente utilizados para realizar operações aritméticas sobre dados cifrados. No esquema HOM, é utilizada a cifra de *Paillier* para realizar operações *SUM* [35]. O HOM é utilizado como camada única na *onion* de adição.

O esquema JOIN possibilita a realização de *joins* de igualdade entre duas colunas. Este esquema assume propriedades determinísticas e encontra-se na camada mais interna da *onion* de igualdade. Já o esquema *OPE-JOIN* permite realizar *joins* por relações de ordem. Este esquema, encontra-se na camada mais interna da *onion* de ordenação.

Por fim, para realizar operações de pesquisa com o operador *LIKE*, é utilizado o esquema SE, que está assente como camada única na *onion* de pesquisa. Contudo, a operação de pesquisa apenas funciona corretamente para palavras completas e não para expressões regulares, prefixos ou sufixos.

A segunda ideia, *cifragem ajustável à interrogação*, visa ajustar dinâmicamente a estratégia de cifragem através de *onions* de igualdade, ordem, pesquisa e adição, como demonstra a figura 3. A ideia é cifrar cada um dos dados recebidos com uma ou mais *onions*.

Cada uma destas *onions* recorre a camadas de cifragem que correspondem a técnicas criptográficas. Estas camadas são incrementalmente mais fortes, isto é, a camada mais externa assume um grau de segurança mais elevado e por sua vez possibilita poucas ou nenhumas funcionalidades computacionais, *p.e., Standard Encryption*. As camadas mais internas

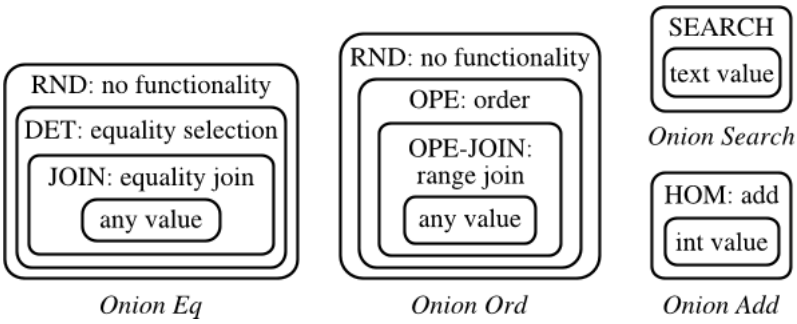


Figura 3: *Onions* criptográficas com as respetivas camadas de cifragem.

já possibilitam uma maior computação sobre os dados mas apresentam um compromisso na segurança, *p.e.,* OPE.

No processo de cifragem e decifragem, para cada camada em cada *onion*, o *proxy* usa a mesma chave criptográfica para cada coluna, e chaves criptográficas diferentes para anonimizar as tabelas, colunas, *onions* e camadas. Isto porque ao usar a mesma chave para todos os valores na mesma coluna, permite ao *proxy* operar sobre uma coluna sem ter de gerar chaves individuais para cada linha a ser manipulada. Todas as chaves dependem de uma *master key* MK, bem como de uma tabela t, uma coluna c, uma *onion* o e uma camada de cifragem l:

$$K_{t,c,o,l} = P_{MK}(t, c, o, l),$$

onde P é uma permutação pseudo-aleatória (p.e., AES). Para cifrar, começa-se pela camada de cifragem mais interna até à mais externa. Assumindo a *onion* de igualdade, implementa-se primeiro a layer *JOIN*, depois a *DET* e por fim a *STD*. Para decifrar, realizar-se-á o processo inverso, contudo, numa próxima execução (numa mesma sessão), apenas se procederá à decifragem direta do *DET* e do *JOIN*.

Por fim, a terceira ideia, *chaves de cifragem em cadeia sobre as passwords dos utilizadores*, que define que cada item da base de dados apenas pode ser decifrado com uma cadeia de chaves enraizadas na palavra-chave de um utilizador. Como resultado, se o utilizador não estiver ligado ao sistema e se um adversário não tiver conhecimento da sua password, este não consegue decifrar os dados do utilizador, mesmo que a camada aplicacional e o servidor DBMS sejam comprometidos.

Relativamente ao desempenho do sistema, quando comparado com a base de dados relacional sem alterações, o *CryptDB* apresenta um custo no desempenho de 14.5% numa plataforma *online* de fóruns e 26% na execução de interrogações da plataforma de *TPC Benchmark* ^{TM}*C (TPC-C)*.

Relativamente à segurança, como demonstrado em *Akin*, o *CryptDB* está susceptível a ataques de criptoanálise *p.e.,* ataques por análise de frequência, e ainda a ataques por

compilação de interrogações [56]. Este último visa identificar a identidade dos utilizadores, colecionar palavras de interesse e modificar interrogações de pesquisa.

3.1.2 *Monomi*

Um outro sistema capaz de processar interrogações sobre dados cifrados é o *Monomi* [14]. Este sistema é baseado no *CryptDB*, recorrendo também a vários esquemas criptográficos para executar operações *SQL* sobre dados cifrados, como comparações de igualdade e ordem de valores e agregações. Contudo o *CryptDB* apenas suporta interrogações sobre colunas protegidas por um único esquema criptográfico, fazendo com que apenas execute corretamente 4 de 22 interrogações do *TPC Benchmark* ^{TM}*H (TPC-H)*. O *benchmark* TPC-H é usado sobretudo para testar operações de análise num contexto empresarial sobre grandes quantidades de informação.

Para colmatar estas limitações, o *Monomi* segue uma abordagem diferente do *CryptDB*, fazendo um pré-processamento das interrogações de forma a particiona-las entre um servidor não confiável e um cliente confiável, levando a uma execução mais eficiente dos pedidos efetuados (modelo de execução *split client-server*), tal como apresentado na figura 4 [14]. Ainda, introduz várias otimizações para aumentar o desempenho das interrogações, permitindo que execute 19 das 22 interrogações do TPC-H.

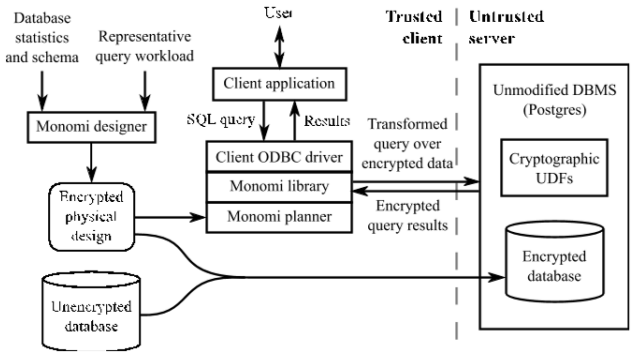


Figura 4: Arquitetura do sistema *Monomi*.

Contudo, há três desafios inerentes a este sistema. Primeiro, o processamento de quantidades massivas de dados é por vezes limitado pelo *I/O*, sendo que esquemas criptográficos que aumentam significantemente o tamanho dos dados armazenados irão atrasar o processamento das interrogações. Segundo, interrogações analíticas podem envolver um processamento complexo, pelo que podem ser bastante ineficientes ou até impossíveis, quando executadas sobre conteúdo cifrado. O desafio assente sobre este problema passa por fazer uma partição correta das interrogações de tal forma que possam ser eficientemente execu-

tadas. Por último, após a aplicação de técnicas criptográficas, o processamento dos dados cifrados pode ser rápido para algumas interrogações e lento para outras.

Para resolver estes desafios, o sistema *Monomi* propõe três soluções. Primeiro, seguir um modelo de execução *split client-server*, permitindo que o processamento das interrogações seja dividido entre as duas partes do sistema. Numa primeira fase, uma interrogação é executada no servidor e numa segunda fase, as partes da interrogação que o servidor não consegue executar ou cujo processamento é mais eficiente do lado do cliente, são executadas pelo cliente.

```
SELECT SUM(price) AS total
FROM orders
GROUP BY order_id
HAVING total > 100
```

Interrogação 3.1: Exemplo de uma interrogação *SQL* sobre o sistema *Monomi*.

Assumindo então este modelo, para executar interrogação 3.1, o *Monomi* cifra o *order_id* com *Deterministic Encryption*, gerando o criptograma *order_id_det* sendo este dotado de propriedades determinísticas que irão permitir pesquisas de igualdade de modo a encontrar todas as linhas com o mesmo *order_id* e executar então a agregação *GROUP BY*. Já o valor *price*, é cifrado com *Homomorphic Encryption*, com o criptossistema de *Paillier*, *price_paillier*, permitindo ao sistema fazer a adição de valores cifrados. Contudo, por os valores numéricos serem cifrados com *Paillier Encryption*, o sistema não consegue efetuar a comparação *total* $>$ 100 na medida em que a ordem dos criptogramas não é preservada. Assim, o *Monomi* irá reescrever a interrogação 3.1 e na interrogação 3.2, que será executada do lado do servidor. De notar que *PAILLIER_SUM()* é uma UDF que assume o comportamento do criptossistema de *Paillier* [35].

```
SELECT PAILLIER_SUM(price_paillier) AS total
FROM orders
GROUP BY order_id_det
```

Interrogação 3.2: Interrogação gerada pelo *Monomi* após a partição da interrogação 3.1.

Após o processamento no lado do servidor, o resultado da interrogação é enviado para o cliente, que irá decifrar os dados e executar a parte restante da interrogação original, *HAVING total* $>$ *100*.

Outra solução proposta é a adoção de técnicas para aumentar o desempenho do sistema como pré-processamento por linha, criptogramas eficientes quanto ao espaço dos dados, adições homomorficas em grupos e pré-filtragem.

Para otimizar a execução de algumas interrogações, o *Monomi* adiciona colunas auxiliares cujo valor depende de operações entre outras colunas. Por exemplo, assumindo que temos as colunas C_A e C_B e pretende-se executar a operação $C_A * C_B$, os resultados estarão pré-processados na coluna C_C.

Para minimizar o espaço adicional gerado pelos criptogramas quando comparado com o *plaintext* e evitar problemas de desempenho derivado do *I/O*, o *Monomi* usa esquemas criptográficos orientados ao formato e espaço dos criptogramas, como é o caso de esquemas de *Format-Preserving Encryption*. Para além destes esquemas, o *Monomi* usa também para cifrar a informação sensível, *Standard Encryption*, *Deterministic Encryption*, *Order-Preserving Encryption*, *Homomorphic Encryption (Paillier Encryption)* e ainda *Searchable Encryption*.

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. Esta decisão é baseada segundo exemplos de possíveis interrogações, que estão armazenadas numa base de dados do lado do cliente.

Comparativamente ao *CryptDB*, a nível de segurança do sistema, o *Monomi* assume as mesmas propriedades, uma vez que a sua construção é baseada no *CryptDB*. Já no desempenho, devido às otimizações propostas e o modelo de execução *split client-server* o *Monomi* é mais eficiente [14]. Contudo, por o processamento das interrogações ser dividido, obriga a que o cliente possua poder computacional significativo.

3.1.3 *L-EncDB*

Com a adoção de sistemas de computação segura sobre *SQL* como os sistemas apresentados anteriormente, apesar dos dados permanecerem privados a sua estrutura original é modificada perdendo algumas das suas propriedades como o tipo (*p.e.,* string, inteiro) e o seu comprimento, resultando por vezes numa impraticabilidade de algumas operações *SQL*. Exemplo disso é a impossibilidade de restringir o comprimento de dados, visto que estes sistemas utilizam esquemas criptográficos cujo comprimento do criptograma é sempre superior ao comprimento do conteúdo original.

Para ultrapassar esta limitação, *Li* propôs o *L-EncDB*, um sistema capaz de preservar o formato dos dados sensíveis através de esquemas FPE [57]. Ao usar este tipo de esquemas, o *L-EncDB* permite preservar as propriedades dos conteúdos originais após serem cifrados, permitindo que possam ser armazenados na base de dados sem alterar a sua estrutura original, bem como suportar as operações base de *SQL* (*Insert*, *Update*, *Delete*) e algumas mais avançadas, como *range queries* e *fuzzy queries* (similar a operações de pesquisa mas com um fator de granularidade adicional, de modo a pesquisar não só as palavras iguais mas também palavras semelhantes) [58]. Esta última operação visa suportar o operador

SQL LIKE, que permite a pesquisa de palavras na base de dados por prefixos e/ou sufixos ou pela sua totalidade.

Relativamente à arquitetura do sistema, o *L-EncDB* divide-se em duas camadas, a aplicacional e a base de dados. Na camada aplicacional estará uma interface de interpretação *SQL* (*SQL Translator*) que irá interpretar todas as interrogações que forem submetidas, reescrevendo-as com a informação sensível cifrada.

A camada da base de dados irá processar e armazenar a informação sensível. Os campos originais serão usados para preservar os criptogramas dos dados originais, e serão adicionados novos campos para armazenar criptogramas adicionais que possibilitem a execução de *range* e *fuzzy queries*.

Para executar as interrogações, o *L-EncDB* divide as operações *SQL* em dois tipos: as operações base e as operações complexas. Para as operações base, cada uma das constantes da interrogação será cifrada com FPE. Se por ventura estas constantes pertencerem a campos em que seja necessário fazer *range* ou *fuzzy queries*, serão gerados os respetivos criptogramas. Tome-se como exemplo a interrogação 3.3, em que *Table1* é a tabela destino, *Field1* e *Field2* campos que necessitam respetivamente de *range* e *fuzzy queries*, e *Str1* e *Str2* os dados sensíveis a armazenar.

```
INSERT INTO Table1 (Field1, Field2)
VALUES (Str1, Str2)
```

Interrogação 3.3: Exemplo de uma interrogação *SQL Insert* sobre o sistema *L-EncDB*.

A camada aplicacional envia a interrogação 3.3 para a interface *SQL*, que por sua vez irá reescrever a mesma na interrogação 3.4.

```
INSERT INTO Table1
(Field1, Field1Extra, Field2, Field2Extra)
VALUES
(FPE_k(Str1), OPE_k(Str1), FPE_k(Str2), FQE_k(Str2))
```

Interrogação 3.4: Interrogação gerada pelo *L-EncDB* a partir da interrogação 3.3

em que *Field1Extra* será o campo adicional para exercer *range queries*, sendo usado para esse efeito o esquema OPE. *Field2Extra* será o campo adicional para exercer *fuzzy queries*, sendo usado para esse efeito um esquema criptográfico de *Fuzzy Query Encryption* (*FQE*) [59].

Para as operações complexas, o procedimento é semelhante. Tome-se como exemplo a seguinte interrogação em que *Table1* é a tabela destino, *Field1* é o campo a executar a pesquisa e *key1* o valor.

```
SELECT *
FROM Table 1
WHERE Field1 > key1
```

Interrogação 3.5: Exemplo de uma interrogação *SQL Select* sobre o sistema *L-EncDB*.

A camada aplicacional envia a interrogação 3.5 para a interface *SQL*, que por sua vez irá reescrever a mesma na interrogação 3.6.

```
SELECT *
FROM Table 1
WHERE Field1Extra > OPE_k(key1)
```

Interrogação 3.6: Interrogação gerada pelo *L-EncDB* a partir da interrogação 3.5

em que *Field1Extra* será o campo que permite executar *range queries*.

Relativamente à segurança assente sobre o sistema *L-EncDB* existem dois tipos de adversários. O primeiro tipo de adversários tem acesso à camada da base de dados, tendo acesso a todos os dados cifrados e à estrutura da base de dados. Assume-se que este adversário assume um modelo *honest-but-curious*. O segundo tipo tem acesso total ao sistema (camada aplicacional e base de dados), tendo acesso à interface de interpretação e construção das interrogações *SQL* e às chaves de cifragem e decifragem. Neste caso o adversário assume um comportamento ativo.

3.1.4 *SDB*

Uma das limitações dos sistemas apresentados anteriormente é o facto das operações sobre dados cifrados não serem interoperáveis, isto é, o *output* de uma operação não pode ser usado como *input* de outra, por se tratarem de esquemas criptográficos incompatíveis. Como resultado, estas soluções fornecem desempenhos limitados aquando a execução de interrogações complexas que envolvam múltiplos operadores.

Para ultrapassar estas limitações, *Wong* propôs o sistema *SDB*, que permite a interoperabilidade dos dados cifrados. Esta abordagem permite que todas as interrogações do benchmark TPC-H sejam eficientemente executadas [60]. Tal como nos sistemas anteriores, o *SDB*

adota o uso de UDF's no servidor, permitindo que o processamento das interrogações seja feito em qualquer base de dados relacional que suporte UDF's.

Este sistema baseia-se no protocolo 2PC com *secret sharing*, e recorre a duas *parties*: o cliente e o servidor. Ambas as *parties* necessitam de capacidade de armazenamento e processamento para efetuar corretamente o processamento seguro de dados cifrados. A figura 5 representa a arquitetura deste sistema [60].

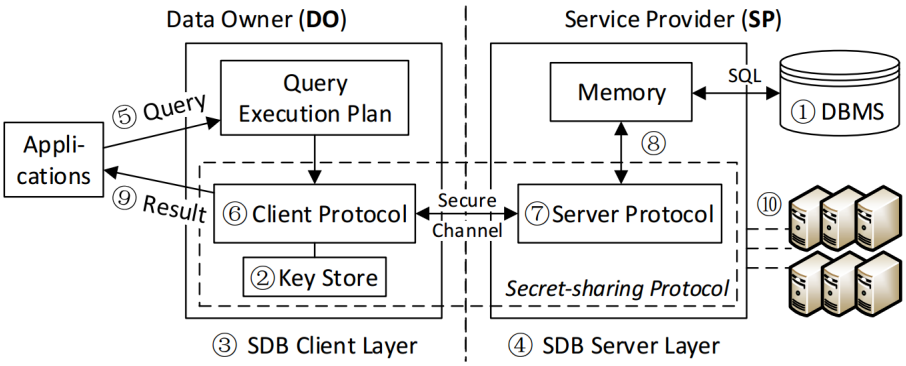


Figura 5: Arquitetura do sistema *SDB*.

O cliente contém um *proxy SDB*, que será responsável por receber as interrogações *SQL* da camada aplicacional. Essas interrogações serão reescritas de acordo com os UDF's correspondentes, sendo depois enviadas para o servidor. Para além disso, o *proxy* é responsável por cifrar e decifrar os dados com *secret sharing*. Para auxiliar na construção dos criptogramas, o cliente irá possuir um número secreto g e uma chave pública n. Esta chave pública é gerada através do criptossistema *Rivest, Shamir and Adleman cryptosystem (RSA)*.

O servidor irá conter a camada de dados, bem como um conjunto de UDF's que permitem fazer o processamento seguro sobre dados cifrados. O servidor está também responsável por armazenar toda a informação, dados sensíveis e não sensíveis, e processar as interrogações reescritas pelo *proxy*.

Represente-se $[[v]]$ como informação sensível a ser armazenada e v_k e v_e as partes resultantes quando aplicado o esquema *secret sharing* sobre $[[v]]$. v_k corresponde à *item key* e é gerada através da chave de uma coluna A, $ck_A =< m, x >$, em que m e x são um par de valores aleatórios. v_e corresponde ao valor cifrado.

Num exemplo mais prático, considere-se duas colunas sensíveis A e B de uma tabela T, cujas chaves da coluna correspondem $ck_A =< m_A, x_A >$ e $ck_B =< m_B, x_B >$, respetivamente. Suponha-se que se pretende processar a interrogação 3.7.

```
SELECT (A*B) AS C
FROM Table T
```

Interrogação 3.7: Exemplo de uma interrogação *SQL* sobre o sistema *SDB*.

A interrogação 3.7 será enviada ao *proxy SDB* que a reescreverá na interrogação 3.8.

```
SELECT row_id,sdb-multiply(A_e,B_e,n) AS C_e
FROM Table T
```

Interrogação 3.8: Interrogação gerada pelo *SDB* a partir da interrogação 3.7.

em que row_{id} corresponde ao identificador da linha, A_e, B_e e C_e correspondem aos valores cifrados das colunas A, B e C, respetivamente, e o valor n corresponde à chave pública armazenada no cliente. Por fim, o protocolo de mutiplicação *sdb-multiply* corresponde a uma UDF, capaz de efetuar a computação dos dados de forma segura. O resultado desta computação será enviada para o *proxy SDB* que irá decifrar a informação e enviar ao cliente.

Relativamente à segurança, o sistema *SDB* considera dois tipos de ameaças. A primeira ameaça corresponde a adversários que têm controlo sobre o servidor e que conseguem obter conhecimento por ataques *Chosen Plaintext Attack (CPA)*. A segunda ameaça corresponde a adversários que têm controlo sobre o servidor e conseguem obter conhecimento das interrogações seguras. Para ambas as ameaças, o *SDB* considera que não são adversários de risco, visto que a computação é feita num ambiente distribuído. Este ambiente torna difícil que as informações de utilizadores específicos sejam comprometidas devido ao fluxo das operações, bem como a sua ordem.

Quando comparado com os sistemas anteriores, *CryptDB* e *Monomi*, o sistema *SDB* assume vantagens na interoperabilidade dos dados, tornando o sistema menos complexo por assumir apenas um esquema criptográfico e permite ainda o processamento eficiente de múltiplas interrogações (consegue processar todas as interrogações do TPC-H). Contudo, o sistema necessita que ambas as partes envolvidas, cliente e servidor, tenham capacidade de armazenamento e processamento.

3.1.5 *Sharemind*

Uma proposta diferente dos sistemas previamente abordados é o *Sharemind* proposto por *Bogdanov* que tira partido das propriedades de MPC e *Secret Sharing* para fazer computação segura sobre dados cifrados [47]. Este sistema foi originalmente desenvolvido para possibilitar o cruzamento de dados, *p.e.*, operações de agregação de dados e estatísticas, entre

diversas entidades independentes que possuem a sua base de dados e não querem revelar dados sensíveis a outras entidades.

Este sistema é composto por três entidades: as *input parties*, as *computation parties* e as *result parties*. As *input parties* são as entidades que irão os fornecer os dados do sistema. Cada um dos dados será cifrado com *Secret Sharing* e cada uma das partes resultantes será armazenada numa *computation party*. As *computation parties* são totalmente dedicadas ao MPC e são necessárias três *parties*. Por conseguinte, o esquema *threshold* a utilizar será $(3,3)$. Por fim, as *result parties* serão as entidades que irão invocar as interrogações sobre a base de dados.

Num exemplo prático do sistema *Sharemind*, assume-se que um conjunto de clínicas e hospitais disponibilizam temporariamente a informação dos pacientes numa infraestrutura de serviços na nuvem. Pretende-se também permitir que uma entidade certificada efetue interrogações à base de dados para recolher dados estatísticos acerca das clínicas e hospitais, *p.e., quantos pacientes sofrem de diabetes?*.

Neste exemplo, as clínicas e hospitais assumem o papel de *input parties*, a infraestrutura de serviços em nuvem será a *computation party* e a entidade certificada a *result party*. Como o sistema usado é o *Sharemind*, a entidade certificada consegue saber todos os dados estatísticos que pretende sem que a informação sensível dos utilizadores seja comprometida.

O modelo original do sistema *Sharemind* permite realizar operações MPC sobre inteiros. Para isso foram propostos vários protocolos que permitem executar várias operações aritméticas.

Relativamente à segurança do sistema, o *Sharemind* é seguro contra adversários passivos, assumindo o modelo *honest-but-curious*. Este modelo assume que um adversário é capaz de ganhar acesso de, no máximo uma *party*, antes da execução de qualquer protocolo. Estes adversários podem ver todos os segredos armazenados na *party*, bem como as mensagens que estão a ser transmitidas com a mesma.

3.1.6 *Soluções baseadas em Hardware*

As soluções apresentadas previamente são baseadas somente em *software*, contudo, surgiram recentemente abordagens que recorrem a *hardware* para efetuar computação segura sobre bases de dados relacionais. Exemplos dessas abordagens são os sistemas *TrustedDB* e *Cipherbase*.

O *TrustedDB* recorre a um coprocessador criptográfico em *hardware* da *IBM* (*IBM 4764*) [61]. Por o coprocessador apresentar limitações quanto à capacidade de armazenamento e memória, os dados serão guardados no servidor e a computação será dividida entre o servidor e o coprocessador.

Os atributos da base de dados são classificados como sendo públicos ou privados, fazendo referência à informação não sensível e sensível, respetivamente. Os dados públicos são processados pelo servidor. Já os dados privados podem ser cifrados e decifrados pelo cliente e pelo coprocessador e o processamento das interrogações sobre os dados cifrados é totalmente feito pelo coprocessador.

Para executar uma interrogação sobre este sistema, esta é submetida à camada aplicacional que a reescreve, cifrando todos os dados sensíveis com *Standard Encryption* e envia para o servidor. Por sua vez, o servidor interpreta a interrogação recebida e divide a computação a ser feita sobre o servidor e o coprocessador. No caso da computação a ser executada sobre o coprocessador, juntamente com a interrogação, são enviados os dados cifrados.

Ao contrário dos sistemas baseados em *software*, quer o cliente quer o coprocessador têm acesso às mesmas chaves de cifragem. Estas chaves são transmitidas entre as duas entidades através de mensagens protegidas com criptografia pública. O coprocessador permite que estas chaves sejam guardadas de forma segura pois é seguro contra adulteração de informação [62].

O sistema *Cipherbase*, recorre a *hardware* programável, *Field-Programmable Gate Array (FPGA)* [63]. Esta tecnologia consegue cifrar, decifrar e processar dados de forma segura.

O processamento das interrogações assume um modelo semelhante ao sistema *TrustedDB*, contudo, o *Cipherbase* suporta uma maior variedade de esquemas criptográficos. Para *Deterministic Encryption* é usado o AES a operar no modo ECB e o processamento dos dados pode ser feito pelo servidor. Para realizar *range queries* é usado o esquema OPE e o processamento é feito também pelo servidor. Já agregações ou qualquer tipo de operações aritméticas, sendo neste último caso usado o esquema de *Paillier*, os dados têm de ser enviados para o FPGA para este os poder processar. Por último, os dados mais críticos são cifrados com *Standard Encryption*, usando o AES a operar no modo CBC sem IV e são processados pelo FPGA.

Uma grande vantagem apresentada pelo *Cipherbase* é a possibilidade de configurar a segurança do sistema, permitindo ao programador da aplicação escolher o esquema criptográfico que melhor se adequa a cada tipo de dados.

Uma solução que se baseia simultâneamente em *software* e *hardware* é oferecida pela Intel através do *instruction set Intel Software Guard Extensions (SGX)* [64]. O SGX é uma extensão à arquitetura da *Intel* que permite fornecer confidencialidade e integridade através de *Enclaves*. *Enclave* é uma zona protegida da memória que serve para processar dados seguros com vários esquemas criptográficos.

Para concluir, ambos os sistemas apresentam soluções interessantes e provam que é possível tirar partido do *hardware* para permitir computação segura sobre dados cifrados, sendo um ramo que continua em expansão. Contudo, pelo mesmo motivo, estes sistemas

apresentam uma restrição tecnológica, obrigando a que os fornecedores dos serviços de nuvem invistam em *hardware* específico. Além disso, o uso de *hardware* seguro introduz novos modelos de segurança que podem não ser apropriados a todos os cenários [65].

3.2 COMPUTAÇÃO SEGURA EM BASES DE DADOS NOSQL

As bases de dados não relacionais, também conhecidas como bases de dados *NoSQL* relaxam as garantias de coerência de dados e a complexidade de interrogações fornecidas pelas bases de dados relacionais de forma a obter uma melhor disponibilidade e escalabilidade em ambientes distribuídos [66][67].

Estas bases de dados são por isso bastante populares para grandes quantidades de dados não estruturados que têm de estar sempre disponíveis e são acedidos por milhões de utilizadores em paralelo. Este é o caso de redes sociais, serviços de vendas *online*, entre outros [68][69].

As bases de dados *SQL* necessitam de saber em antemão o tipo e a estrutura dos dados a armazenar, sendo assim definido um esquema estático antes de adicionar qualquer tipo de dados à base de dados. A principal desvantagem de um esquema estático é a necessidade de fazer a migração de toda a base de dados quando se pretende adicionar/modificar colunas, por forma a respeitar o novo esquema. Se a base de dados for demasiado grande, este processo vai ser demorado e envolve um tempo de inatividade considerável. Já as bases de dados *NoSQL* são construídas de forma a permitir a inserção dos dados sem ter um esquema predefinido, tornado a base de dados mais ágil a modificações em tempo real, sem haver preocupações com interrupções do serviço.

Quanto à escalabilidade dos sistemas, as bases de dados relacionais escalam verticalmente, em que toda a bases de dados está alojada num único servidor de forma a garantir um desempenho aceitável. Esta abordagem torna-se rapidamente dispendiosa e limita a escalabilidade. Assim, a solução passa por escalar horizontalmente, adicionando servidores extra em vez de concentrar toda a informação num único servidor. Este processo é denominado de *Sharding*.

Originalmente, as bases de dados *SQL*, não previam o uso nativo de *Sharding*, havendo a necessidade de fazer este processo manualmente. Por outro lado, as bases de dados *NoSQL*, normalmente suportam *Sharding* automático, fazendo com que nativamente, transparentemente e automaticamente os dados sejam distribuídos por um número arbitrário de servidores.

Um dos tipos mais conhecidos de bases de dados *NoSQL* são as *Key-Value Stores* [70][68][69]. Este tipo de base de dados assume uma estrutura multi-dimensional em que cada coluna é composta por um conjunto de sub-colunas. Cada entrada na base de dados constitui uma linha que é composta por um par chave-valor em que para cada chave está-lhe associada

um conjunto de valores (colunas). Estas bases de dados disponibilizam uma *Application Programming Interface (API)* com as operações *PUT*, *GET*, *DELETE* e *SCAN*. O *PUT* insere uma nova linha (chave e valores correspondentes) na base de dados, o *GET* devolve uma linha dada uma determinada chave, o *DELETE* remove uma chave e os valores associados (linha) da base de dados e o *SCAN* devolve um conjunto de linhas para um determinado conjunto contíguo de chaves.

No entanto, as propostas originais destas bases de dados não integram soluções que garantam a privacidade dos dados. Assim, esta secção visa identificar alguns sistemas capazes ultrapassar estas limitações, permitindo a computação segura sobre bases de dados *NoSQL*.

Implementar mecanismos capazes de garantir a forte privacidade de informação sensível, como *Standard Encryption*, num contexto de bases de dados, é possível mas tem um impacto significante no desempenho do sistema. Esta técnica restringe grande parte do poder computacional das bases de dados, levando a uma contradição do propósito dos serviços de computação em nuvem de terceiros. Visto o modelo *NoSQL* reduzir a complexidade das interrogações feitas à base de dados, é também possível reduzir a complexidade de introduzir computação segura nestes motores de bases de dados e, desta forma, chegar a soluções mais eficientes e com um menor custo de desempenho.

3.2.1 *SafeRegions*

O *SafeRegions* trata-se de um novo sistema que possibilita computação segura e distribuída sobre bases de dados *NoSQL*, mais especificamente, sobre o *Apache HBase* [48]. Este sistema fornece garantias fortes quanto à privacidade dos dados, e em simultâneo, suporta parte das operações base do sistema *HBase*.

Este sistema baseia-se na solução *Sharemind* (secção 3.1.5), assumindo um esquema seguro de MPC com *Secret Sharing* ($threshold(3,3)$). Assim, são necessárias três *computation parties* para efetuar corretamente a computação. Neste caso, os clientes da base de dados funcionam tanto como *input* e *result parties*.

No que diz respeito às funcionalidades deste sistema, na versão mais atual, é disponibilizada uma API segura com as operações *store* e *search*, que dizem respeito às operações *PUT* e *GET* do *HBase*, respetivamente.

O sistema é constituído por três tipos de entidades, o cliente, a *client machine* e as *parties*. A *client machine* corresponde a um ou mais servidores a correr o cliente de base de dados do *HBase* numa infraestrutura confiável. As *parties* correspondem a um ou mais servidores a correr um *cluster HBase* independente numa infraestrutura não confiável.

Para inserir informação sensível no sistema, é executada a operação *store*, que compreende os passos apresentados na figura 6.

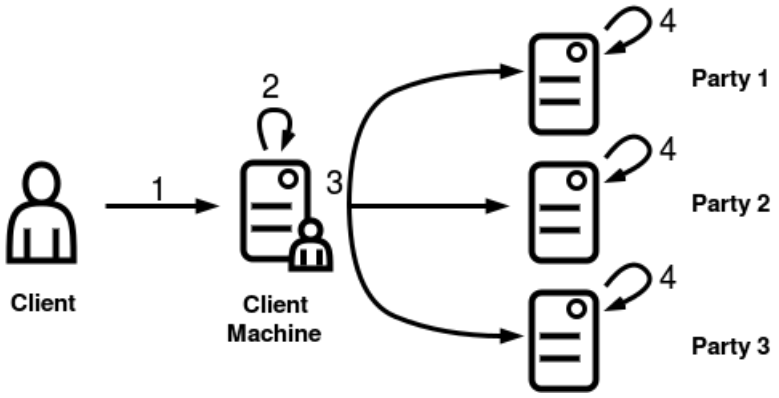


Figura 6: Inserção de dados no sistema *SafeRegions*.

O cliente envia para a *client machine* os dados a armazenar (1); a *client machine*, para cada valor de *input* recebido, irá aplicar o esquema de *Secret Sharing*, dividindo em três segredos (2); cada segredo será enviado para uma *party* (3); cada uma das *parties* irá armazenar o segredo que recebeu (4).

Relativamente à operação *search*, o seu comportamento é descrito na figura 7. Neste exemplo, assume-se que se pretende verificar se um determinado valor v existe no sistema.

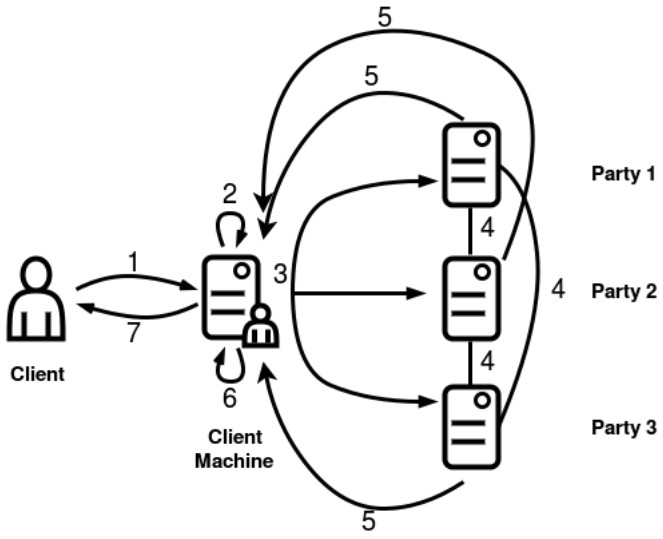


Figura 7: Pesquisa de dados no sistema *SafeRegions*.

O cliente submete a operação de pesquisa para a *client machine* (1); a *client machine* irá gerar com *Secret Sharing* novos segredos a partir de v (2); depois de gerados os segredos, envia um deles para todas as *parties*, pedindo para executar um protocolo de comparação (3); após a recepção, cada *party* executa um conjunto de geração e troca de segredos entre *parties*, de forma a poder executar o protocolo de comparação (4); ao fim do processamento

e troca de mensagens, cada *party* irá enviar o resultado obtido à *client machine* (5); por sua vez, a *client machine* irá juntar os três segredos, decifrando o resultado (6); finalmente, o resultado é enviado para o cliente (7).

De notar que em ambas as operações, todo o processo de segurança e computação é tranparente ao cliente.

No que diz respeito ao desempenho do sistema, quando comparado com o *HBase* original, haverá um decréscimo no desempenho das interrogações por vários motivos: todas as interrogações submetidas, irão ser processadas pela *client machine* que irá cifrar/decifrar os dados com *Secret Sharing*. A isto, será ainda acrescentado o tempo de transmissão dos segredos para as *parties*. Adicionalmente, nas operações de pesquisa, haverá um custo na geração e posterior transmissão de segredos entre *parties*, por fim a executar os protocolos corretamente.

Relativamente à segurança, as garantias são análogas ao sistema *Sharemind*.

3.2.2 *BlindDB*

Numa abordagem diferente, *Yuan* propôs uma base de dados não relacional do tipo *Key-Value Store*, segura e distribuída [71] [72]. Para garantir a privacidade dos dados é usado um esquema de SE, permitindo preservar grande parte das propriedades do conteúdo original. Simultaneamente, a *Key-Value Store* não perde as propriedades típicas destes sistemas, como a baixa latência, balanceamento de carga, escalabilidade e disponibilidade.

Relativamente à segurança, o sistema é seguro contra adversários passivos (*Honest-but-curious*), que têm acesso total à camada do servidor, que por sua vez está alojado nos serviços de nuvem de terceiros. Estes adversários estão interessados nos dados armazenados no sistema, os atributos das interrogações questionadas e os resultados das mesmas. Para se proteger destes adversários, o sistema cifra toda a informação relevante na camada aplicacional, fazendo com que os dados que os adversários poderão ter acesso estejam protegidos.

A arquitetura do sistema, descrita na figura 8, divide-se em três entidades: o cliente, o *dispatcher* e um conjunto de nodos distribuídos. As duas últimas entidades estão alojadas nos serviços em nuvem.

O cliente efetua pedidos à camada aplicacional que irá efetuar o pedido das interrogações ao sistema. Nesta camada, está alojada uma base de dados *Key-Value* que contém múltiplas colunas e os identificadores das linhas. Desta forma, esta base de dados permite fazer o mapeamento com os dados armazenados na *Key-Value Store*. Toda a informação submetida pelo cliente, será cifrada na camada aplicacional antes de ser enviada para o *dispatcher*.

O *dispatcher* é responsável por gerir e traduzir todas as interrogações submetidas pelo cliente, bem como distribuir todos os dados cifrados por todos os nodos uniformemente.

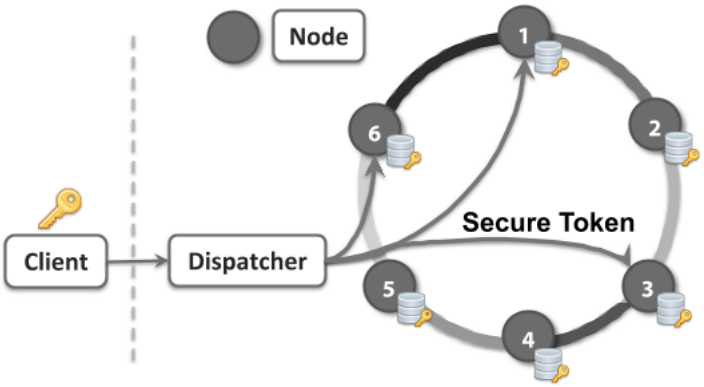


Figura 8: Arquitetura do sistema *BlindDB*.

Esta distribuição é feita seguindo algoritmos de *consistent hashing*, em que cada um dos nodos do sistema irá estar associado a um valor aleatório. Aplicando uma função de *hash* sobre a chave do par chave-valor, irá determinar a localização do valor a armazenar.

A confiabilidade e a disponibilidade do sistema são asseguradas através de réplicas. Todos os dados inseridos no sistema irão ser armazenados num conjunto de réplicas a definir pelo cliente. Para assegurar o processamento intensivo do sistema, novos nodos vão sendo adicionados que irão permitir fazer um balanceamento da carga do sistema.

O armazenamento dos dados na base de dados segue um modelo *Key-Value Store*, sendo representado pelo par $< l, v >$. As chaves l são determinadas através de uma função de *hash*, resultando em l^*, que indicam em que posição está armazenada a informação sensível. Os valores são protegidos com *Symmetric Encryption* e resultam em v^*. A implementação destas técnicas resulta assim no par $< l^*, v^* >$.

Esta abordagem, descrita na Figura 9, permite que seja possível fazer um mapeamento de todos os valores armazenados na base de dados, bem como visa facilitar uma distribuição da informação uniforme por todos os nodos.

A nível de funcionalidades, o sistema disponibiliza uma API segura para as operações *PUT* e *GET*. Dado um pedido *GET*, o cliente gera o valor de l^* e envia para o *dispatcher* que irá determinar em que nodo será feita a computação. Encontrado o nodo, ser-lhe-á feito o pedido $get(l^*)$ assim como para as n réplicas. Para a operação ser bem sucedida, os valores obtidos de todos os nodos deverão ser iguais.

Dado um pedido *PUT*, o processamento é semelhante à operação *GET* com exceção do processo de armazenamento. Quando encontrado o nodo de computação, para as n réplicas definidas pelo cliente, será feito o pedido $put(l^*, v^*)$. A operação é bem sucedida se todas as réplicas executarem corretamente o pedido.

Este sistema, para além das operações seguras definidas, permite ainda realizar operações de pesquisa, do tipo *range queries*, e também agregações.

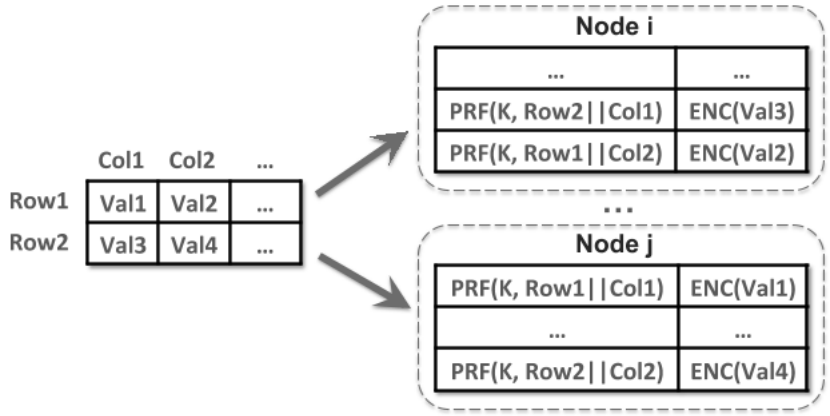


Figura 9: Mapeamento dos valores no sistema.

3.2.3 *BigSecret*

As propostas anteriores de computação segura sobre bases de dados não relacionais, propõe soluções mais estáticas quanto à sua segurança, não permitindo flexibilizar a segurança dos dados em diferentes contextos.

Para ultrapassar isso, surge o sistema *BigSecret* e visa fornecer, de uma forma mais flexível, computação segura sobre bases de dados não relacionais do tipo *Key-Value Store* [15]. Neste caso, é proposta uma solução sobre a base de dados *Apache HBase*, disponibilizando uma API segura sobre as funcionalidades base.

Esta flexibilidade é conseguida através de três modelos criptográficos que permitem definir o grau de privacidade que se pretende sobre os valores sensíveis. Este sistema é seguro contra adversários passivos (*Honest-but-curious*), tendo apenas controlo sobre os serviços de nuvem de terceiros onde se encontram os nodos que armazenam a *Key-Value Store*.

Este sistema é composto por três entidades: o cliente, um *proxy* e os nodos de computação. O cliente é a entidade que irá fazer interrogações ao sistema. O *proxy* interceta os pedidos do cliente, reescrenvendo-os de acordo com os modelos de segurança e enviado-os para os nodos de computação. Esta entidade corresponde à parte central do sistema *BigSecret* e contém ainda informações sobre os nodos de computação e sobre os *Buckets*, que correspondem a grupos que contém os criptogramas. O *proxy* é considerado seguro, permitindo assim que os dados sensíveis sejam então cifrados e decifrados nesta entidade. Por último, os nodos de computação irão armazenar a informação sensível.

O *Big Secret* utiliza *Bucketization*, funções de *Hash* e cifras simétricas sobre três modelos criptográficos. Como primeiro modelo, a informação é protegida através de *Bucketization*. *Bucketization* é um conceito baseado nas propriedade dos esquemas de SE e OPE, em que essencialmente, os valores sensíveis são armazenados por grupos (*Buckets*). Para identificar

em que *bucket* um valor está armazenado, é gerada uma chave através do mapeamento do identificador de linha, da coluna e da sub-coluna.

As funções de *Hash* são utilizadas para gerar índices criptográficos que determinam em que nodo está o respetivo valor cifrado. O recurso a funções de *Hash* deve ser adotado por sistemas cujas *workloads* não envolvam operações *Scan*, visto que este esquema não assume propriedades que preservem a ordem dos valores. Este esquema é usado nos modelos criptográficos 2 e 3. A diferença é que o modelo criptográfico 3 apenas cifra o identificador da linha para gerar uma maior aleatoriedade sobre os valores.

As cifras simétricas são utilizadas para cifrar os valores sensíveis de todos os modelos, visto que assumem garantias fortes de segurança e privacidade.

Para processar interrogações, o cliente envia uma interrogação para o *proxy*. O *proxy*, por sua vez, irá reescrevê-la consoante o modelo criptográfico definido e irá gerar um índice para determinar em que nodo irá ser armazenado o valor cifrado. Depois de obtido o índice, o *proxy* envia o valor para o nodo. Caso seja uma pesquisa sobre a base de dados, o nodo irá realizar a computação e irá enviar todos os resultados para o *proxy*. Este, por sua vez, irá decifrar e filtrar se necessário, e irá retornar os resultados ao cliente.

3.2.4 *Arx*

A grande maioria dos sistemas de computação segura atuais recorrem a técnicas criptográficas com garantias de segurança relaxadas (*p.e., PRIV*, IND-OCPA) de forma a preservar as propriedades de igualdade e ordem da informação sensível. A solução ideal deveria proteger o conteúdo com IND-CPA e possibilitar o processamento das interrogações de forma eficiente (*Server-side computation*).

Com esta visão em mente, surgiu o *Arx*, um sistema que recorre a esquemas criptográficos com segurança IND-CPA e simultaneamente suporta um conjunto alargado de interrogações, como operações de igualdade de valores, *range queries* e agregações [73].

Como representado na figura 10, este sistema é composto por um cliente seguro (*client proxy*), alojado no servidor aplicacional e um servidor não confiável (*server proxy*) alojado em simultâneo com a base de dados. O *client proxy* exporta a mesma API que o cliente de base de dados, tornando a sua integração transparente para as aplicações. O seu papel é intercetar as interrogações e traduzi-las em operações seguras com ajuda do *planner* e do *query rewritter*. Essas interrogações são submetidas diretamente à base de dados se não compreenderem informação sensível ou são reencaminhadas para o *server proxy* no caso de se tratarem de operações seguras. O cliente armazena ainda metadados (informação relativa ao esquema da base de dados) de forma a fazer um mapeamento do tipo de operações que vão ser executadas sobre os dados e qual a cifra a usar.

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 à base de dados. Esta entidade preserva numa estrutura as informações relativas às propriedades do *plaintext*, enquanto que a base de dados (não modificada), armazena a informação com um esquema criptográfico forte.

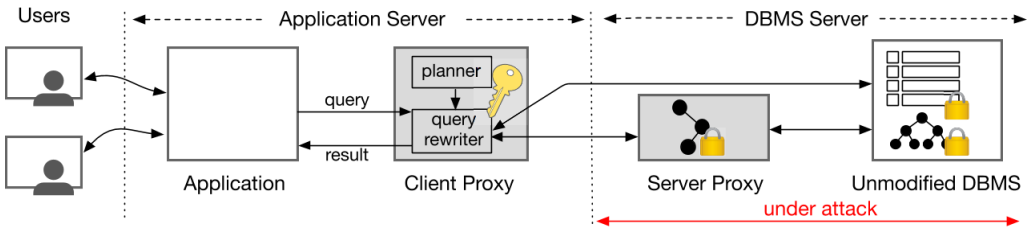


Figura 10: Arquitetura do sistema *Arx*.

De forma a possibilitar o processamento da informação com técnicas criptográficas fortes, o *Arx* introduz quatro novos índices de base de dados: *Arx-Range* que permite fazer pesquisas e comparações de ordem entre valores, *Arx-Eq* que permite fazer comparações de igualdade de valores, *Arx-Agg* que permite fazer operações aritméticas sobre os dados, como somas (*SUM*) e médias (*AVG*) e ainda, o *Arx-Join* que permite efetuar *joins* sobre duas tabelas.

Considere-se uma estrutura de dados de índices, como uma árvore binária, onde cada um dos nodos está protegido com uma cifra segura (*p.e.*, AES). Para atravessar a árvore, a primeira solução em mente seria o *server proxy* contactar o *client proxy* para este decifrar o conteúdo dos nodos e saber qual a localização da informação que necessitamos, mas tal seria extraordinariamente ineficiente. Assim, o *Arx-Range* armazena em cada nodo um programa obfuscado (mais propriamente um *Garbled Circuit*) que efetua a comparação de forma segura [46]. Assim, dado um criptograma $x = Enc_k(a)$, o servidor pode encontrar o *plaintext* correspondente a a ao avaliar a função de obfuscação em cada um dos nodos. Esta avaliação irá retornar o caminho a seguir na árvore. Ao fim da travessia da árvore, o índice (todos os nodos afetados) deverá ser reconstruído de forma a fornecer um novo *Garbled Circuit*.

O índice do *Arx-Range* apesar de possibilitar a comparação de igualdade de valores, não é de todo o mais eficiente para este fim. Assim, o *Arx-Eq* 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: (1) quando os valores são únicos (*p.e.*, identificadores de linha), estes são protegidos com *Deterministic Encryption* e armazenados diretamente na base de dados; (2) quando os valores são repetidos, o *client proxy* conta os valores repetidos e guarda o par valor/quantidade ($v/counter[v]$). Por sua vez, os valores são armazenados na base de

dados, sendo protegidos por $Enc(v) = H(EQunique(v), counter[v])$, em que $EQunique$ é a função de cifragem de um esquema de *Deterministic Encryption* e H é uma função de *hash* sobre o valor cifrado e o respetivo contador, gerando assim um valor pseudo-aleatório. Para pesquisar valores do segundo tipo, o *client proxy* gera todos os índices possíveis e envia para o servidor para efetuar a pesquisa respetiva.

Já os índices *Arx-Agg* e *Arx-Join* seguem abordagens semelhantes aos índices anteriores. No *Arx-Agg*, de forma similar ao *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 seguro. Quanto ao *Arx-Join* há duas operações a considerar: (1) comparação de igualdade de valores e (2) comparação da ordem dos valores. Assumindo que queremos unir duas tabelas, T_1 e T_2, em que T_2 possui uma chave estrangeira que aponta para a chave primária de T_1. Para o primeiro caso, o *Arx* segue a metodologia seguida no índice *Arx-Eq*, sendo para cada chave estrangeira 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 metodologia seguida no índice *Arx-Range*, sendo armazenados na árvore binária ambos os valores das chaves primárias de T_1 e das chaves estrangeiras de T_2. Para gerar os resultados da interrogação, o *server proxy* atravessa 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 irá fazer o mapeamento correto dos conteúdos cifrados com as chaves primárias.

O sistema *Arx* fornece assim uma solução promissora na computação segura sobre bases de dados, fornecendo garantias de segurança fortes (IND-CPA) através de um modelo de computação *split client-server*, sendo a maioria da computação efetuada pelo servidor. Contudo há a necessidade de reconstruir vários nodos dos índices do *server-proxy* a cada operação efetuada, bem como o respetivo balanceamento das árvores. Para além disso, a persitência dos metadados é imprescindível quer no *server-proxy* quer no *client-proxy*.

3.2.5 *MiniCrypt*

Um dos principais motivos da adoção das bases de dados não relacionais é o constante crescimento dos dados a armazenar gerado todos os dias. Para obter processamento eficiente sobre esta quantidade massiva de informação, as bases de dados *NoSQL* distribuem o seu armazenamento e processamento por múltiplas máquinas, paralelizando as interrogações submetidas, e aplicam ainda algoritmos de compressão eficientes. Ao recorrer à compressão da informação, o espaço ocupado e a largura de banda utilizada diminuem, enquanto que a eficiência das interrogações aumenta por possibilitar manter mais informação em memória.

Contudo, no âmbito da computação segura, incluir no mesmo sistema compressão e proteção dos dados não é uma tarefa trivial por duas razões: (1) a aleatoriedade dos criptogramas gerados pelos esquemas criptográficos limitam a capacidade dos algoritmos de compressão; (2) a compressão do conteúdo da base de dados apresenta compromissos, em que podemos ter um maior controlo sobre a informação da base de dados e menores rácios de compressão, ou ter rácios de compressão mais elevados e ter funcionalidades limitadas da base de dados.

Para colmatar estas dificuldades foi proposto o *MiniCrypt*, o primeiro sistema que integra computação segura e compressão sobre bases de dados *NoSQL* [74].

A ideia principal deste sistema é agrupar dinâmicamente os pares chave-valor em conjuntos pequenos, denominados de *packs*, de forma encontrar o melhor rácio entre a compressão da informação e a facilidade em processar interrogações sobre eles. Gerado o *pack*, este será cifrado (com uma chave criptográfica partilhada) e armazenado na base de dados. A figura 11 compara um modelo de cifragem sobre bases de dados típico (esquerda) e o modelo do *MiniCrypt* (direita).

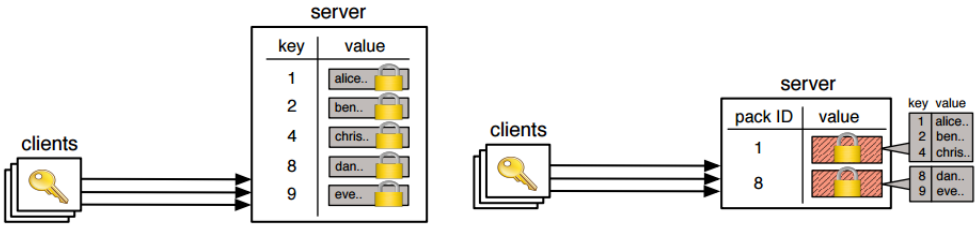


Figura 11: Modelo de cifragem do *MiniCrypt*.

O *MiniCrypt* é composto por um cliente seguro onde é feito o processo de cifragem/decifragem e compressão/descompressão da informação e por um servidor não confiável que contém um DBMS inalterado.

Desta forma, o processamento sobre a informação passa a ser feito sobre *packs* e não sobre linhas. Assumindo o exemplo presente na figura 11, se quisermos consultar a informação da linha 4, a interrogação deverá ser feita ao *pack* 1. Recebido o *pack*, o cliente decifra-o, descomprime-o e verifica qual o valor da linha 4. Para efetuar alterações sobre os valores, o pacote deverá ser trazido para o cliente e este irá gerar um novo pacote com a informação atualizada. À medida que o tamanho dos pacotes vai aumentando, o *MiniCrypt* une ou divide os pacotes de forma a manter um tamanho constante, estabilizando o desempenho do sistema.

No que diz respeito à segurança, o *MiniCrypt* considera adversários *honest-but-curious* e protege os *packs* com um esquema criptográfico de *Standard Encryption* (*AES-256 CBC*). 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*.

3.3 DISCUSSÃO

Como visto neste capítulo, existem várias abordagens seguras para as bases de dados relacionais e não relacionais que fornecem garantias de segurança e desempenhos variáveis. Com a revisão da literatura é possível verificar que existem sempre compromissos entre a segurança do sistema, as funcionalidades disponibilizadas e o desempenho do mesmo.

			Técnicas Criptográficas										C. S.	
			STD	DET	FPE	OPE	SE	PLR	SS	MPC	TPC	HW	Client	Server
Soluções de Computação Segura	SQL	CryptDB	✓	✓		✓	✓	✓						✓
		Monomi	✓	✓	✓	✓	✓	✓					✓	✓
		L-EncDB			✓	✓	✓							✓
		SDB							✓		✓		✓	✓
		Sharemind							✓	✓				✓
		Cipherbase	✓	✓		✓		✓				✓		✓
		TrustedDB	✓									✓		✓
	NoSQL	SafeRegions							✓	✓				✓
		BlindDB	✓	✓			✓							✓
		BigSecret	✓	✓			✓						✓	✓
		Arx	✓	✓									✓	✓
		MiniCrypt	✓	✓										✓

C. S. - Computation Side; PLR - Paillier Encryption; SS - Secret Sharing;
TPC - Two-Party Computation; HW - Soluções Hardware;

Tabela 2: Propriedades das soluções de computação segura do estado da arte atual.

Nesta dissertação, o foco é o modelo *NoSQL* por ser uma área da computação segura ainda em exploração, devido às propriedades interessantes fornecidas, como a alta disponibilidade e escalabilidade e ainda, fornece uma API menos complexa (comparativamente a *SQL*) que permite introduzir diferentes técnicas criptográficas com um menor impacto no desempenho do sistema.

Na tabela 2 é feito um balanço dos sistemas de computação segura existentes em *SQL* e *NoSQL* e as técnicas criptográficas presentes no mesmo, bem como o tipo de computação efetuado sobre os mesmos. Como é possível observar, não existe atualmente uma solução suficientemente genérica que permita a integração flexível de técnicas criptográficas, de modo a fornecer um balanceamento dos compromissos entre a segurança fornecida e as

funcionalidades permitidas. A elevada estaticidade e baixa modularidade destas soluções faz com que o domínio das aplicações a integrar não escale.

4

BASE DE DADOS SAFENOSQL

Conforme descrito no capítulo anterior, as soluções de computação segura sobre bases de dados *SQL* e *NoSQL* tipicamente fornecem um conjunto restrito de técnicas criptográficas, levando a soluções estáticas e pouco flexíveis. Ainda, o modelo de computação que apresentam revela fraca modularidade, extensibilidade e flexibilidade, tornando-as orientadas a aplicações específicas. Desta forma, a solução passa por desenvolver um modelo de computação sobre bases de dados modular e extensível que permita criar bases de dados altamente escaláveis com mecanismos de segurança e funcionalidades configuráveis, maximizando o desempenho e débito do sistema, garantindo simultaneamente um nível adequado de privacidade que permita armazenar os dados na nuvem de forma segura e privada. Assim, neste capítulo é apresentada uma *framework* denominada de *SafeNoSQL*, que propõe uma arquitetura modular e extensível para bases de dados *NoSQL* do tipo *Key-Value Store*, que permite ir de encontro com os requisitos de privacidade e desempenho impostos por diferentes aplicações, permitindo o processamento seguro com múltiplas técnicas criptográficas.

O *SafeNoSQL* é um sistema modular e extensível que permite fornecer transparentemente privacidade e segurança a bases de dados *NoSQL*. A sua arquitetura, apresentada na figura 12, distribuí-se por um *Trusted Site* (o lado seguro que contém o cliente de base de dados) e um *Untrusted Site* (o lado não confiável que compreende os fornecedores dos serviços de nuvem). A *framework* pretende ser genérica, de forma a ser compatível com a maioria das bases de dados *NoSQL* do tipo chave-valor (*Key-Value Store*) [68][69][70]. Para isso, o sistema é composto por quatro módulos principais: *CryptoWorker*, *CryptoBox*, *Handler* e *SafeMapper*.

Esta *framework* possibilita extender o comportamento das bases de dados *NoSQL* (mais precisamente os mecanismos de segurança) através de entidades que abstraem a integração de múltiplas técnicas criptográficas no sistema. Estas entidades denominam-se de *CryptoWorkers*, estando presentes em ambos os lados, *Trusted* e *Untrusted*, e fornecem uma API *NoSQL* transparente, privada e segura, permitindo uma integração simples com as bases de dados *NoSQL* atuais.

Cada pedido efetuado à base de dados é intercetado pelo *CryptoWorker* e traduzido numa operação com a mesma semântica mas com garantias de segurança. Dependendo da garan-

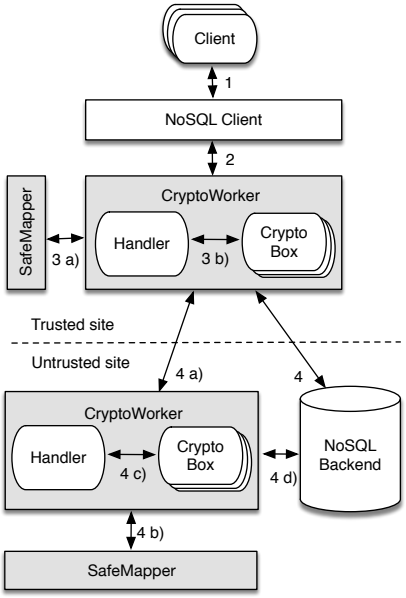


Figura 12: Arquitetura do sistema *SafeNoSQL*.

tia de segurança definida e da técnica criptográfica pretendida, a operação é traduzida de forma diferente.

Esta transformação de interrogações originais em interrogações seguras é conseguida através do módulo *Handler*. Contudo, o processo de tradução é abstraído em três operações distintas:

- *Codificação (Encode)* é uma operação executada no lado seguro, que permite proteger a interrogação antes de ser transmitida para o *NoSQL Backend*. Por exemplo, codificando uma operação *Put* com *Standard Encryption* necessita simplesmente que os dados sensíveis sejam cifrados antes de serem enviados para o lado *Untrusted*. Contudo, mecanismos de obfuscação podem também ser implementados, transformando assim a interrogação original em múltiplas interrogações codificadas do mesmo tipo.

- *Processamento (Process)* é uma operação executada no lado *Untrusted* e permite processar interrogações codificadas. Esta operação permite abstrair o processamento sobre informação cifrada.

- *Descodificação (Decode)* é uma operação executada no lado seguro e permite descodificar a informação recebida do *NoSQL Backend*, retornando à camada que antecede o *SafeNoSQL* os resultados em *plaintext*.

Estas operações são suportadas por componentes modulares e permutáveis denominadas *CryptoBoxes*. Estas componentes recorrem a bibliotecas independentes que implementam técnicas criptográficas que respeitam a interface 4.1 composta pelos métodos *init*, *gen*, *encrypt* e *decrypt*.

```
void init();
byte[] gen();
byte[] encrypt(byte[] key, byte[] value);
byte[] decrypt(byte[] key, byte[] value);
```

Interface 4.1: Interface das *CryptoBoxes*.

O método *init* permite inicializar uma *CryptoBox*, bem como os respetivos parâmetros do seu esquema criptográfico; o método *gen* cria uma chave criptográfica aleatória do tipo do esquema criptográfico; o método *encrypt* permite cifrar conteúdo sensível (*value*) dada uma chave criptográfica (*key*), originando um criptograma; o método *decrypt* permite decifrar um criptograma (*value*) dada uma chave criptográfica (*key*), retornando o conteúdo original, previamente cifrado.

Relativamente à extensão de novas *CryptoBoxes* no sistema é conseguida através de dois passos lógicos: primeiro será feita a implementação da técnica criptográfica, respeitando a interface 4.1; segundo, será adicionado às operações *NoSQL* no *CryptoWorker* o suporte da nova *CryptoBox* de forma a possibilitar operações seguras com a técnica assente.

Assim, a arquitetura demonstra-se modular em dois sentidos: num primeiro nível, do ponto de vista das *CryptoBoxes*, é possível implementar múltiplas técnicas criptográficas que permitem níveis de desempenho e garantias de segurança variáveis, permitindo criar um ecossistema de segurança e privacidade configuráveis; ao nível da própria implementação da *CryptoBox*, na necessidade de atualizar uma técnica já existente, apenas é necessário trocar a implementação ao nível da *CryptoBox*, sem comprometer o *CryptoWorker* original.

Por fim, o módulo *SafeMapper* permite fazer o mapeamento entre o esquema que compõe a base de dados original e os respetivos metadados que definem como irão ser protegidos os pares chave-valor. Desta forma, os compromissos entre desempenho, privacidade e funcionalidades permitidas conseguem ser diretamente configurados e balanceados através da especificação do tipo de proteção (*CryptoBox*) assente sobre cada coluna da base de dados.

O mapeamento dos metadados é realizado durante duas fases: numa primeira fase, antes do processamento de operações, o mapeamento é feito através de um ficheiro de configuração que contêm a informação relevante da estrutura da base de dados (tabelas e colunas) e dos metadados (*CryptoBoxes* que protegem as colunas, número de bytes máximo de cada valor). Por sua vez, este ficheiro define dois tipos de configuração: uma configuração inicial que define metadados padrão para todas as colunas de uma tabela da bases de dados *p.e.*, em vez de especificar todas as colunas de uma tabela, pretendo que estejam todas protegidas com *Order-Preserving Encryption*; uma segunda configuração possibilita definir os metadados para colunas específicas, substituindo o valor dos metadados padrão. No lim-

ite, o ficheiro de configuração irá conter a informação de todas as tabelas da base de dados. Usando a opção de metadados padrão, o cliente de base de dados irá ter uma visão parcial do esquema, sendo apenas conhecidas as colunas que lhe são importantes (nomeadamente, as colunas que irá proteger com esquemas criptográficos diferentes do padrão). Para além disso, a adição/remoção de colunas é transparente (apenas nas colunas com os metadados padrão). No que diz respeito à alteração ou atualização da *CryptoBox* de uma coluna, há a necessidade de ativar o modo de manutenção da base de dados para migrar os valores da coluna em questão para o cliente de base de dados, que por sua vez os irá decifrar e cifrar novamente com a nova *CryptoBox*.

Em termos de escalabilidade e disponibilidade, a arquitetura proposta não afeta as propriedades de replicação e *sharding* do *NoSQL Backend*. Contudo, de forma a suportar múltiplos clientes em simultâneo, é necessário haver um serviço de gestão de chaves criptográficas para que todos os *CryptoWorkers* do lado seguro consigam ter acesso às chaves necessárias para fazer a codificação e descodificação correta da informação sensível para cada interrogação.

Comparativamente à arquitetura genérica proposta na figura 1, a arquitetura do *SafeNoSQL* corresponde às entidades *Database Client* e *Database Backend*. O *CryptoWorker* pode ser visto como o *Database Client*, em que o *Handler* corresponde aos módulos *Query Translator* e *Result Digester*, visto que interceta os pedidos que chegam da camada aplicacional e os resultados do *NoSQL Backend*, efetuando sobre estes as operações de *Encode* e *Decode*. Já o conjunto que compõe as *CryptoBoxes* corresponde à *CryptoLayer*. Por fim, verifica-se que o módulo *SafeMapper* corresponde ao módulo *Mapping Layer*.

Já o *CryptoWorker* presente no lado não confiável do *SafeNoSQL*, faz parte das *Extended Behavior Functions*.

4.1 Fluxo das operações no SafeNoSQL

Para exemplificar o comportamento da *framework* proposta e verificar como cada um dos módulos interage com os outros, assume-se que existe um esquema *NoSQL* em que as chaves estão protegidas com *Deterministic Encryption*, uma primeira coluna de valores com *Standard Encryption* e uma segunda coluna de valores com *Deterministic Encryption*. Nesta descrição passo a passo, os números a ser indicados fazem referência a uma ação entre um ou mais módulos, descrita na figura 12.

Dado um pedido *Put* para um determinado par chave-valor (1), o módulo *CryptoWorker* interceta o pedido (2) e executa a respetiva operação de codificação. Esta operação irá cifrar o par chave-valor com as técnicas definidas no esquema (sendo verificadas na camada de mapeamento), recorrendo às várias *CryptoBoxes* (3a)(3b). Depois, a operação *Put* segura

será reencaminhada para o *NoSQL Backend*, onde a operação de processamento irá apenas armazenar os dados cifrados no servidor não confiável (4).

Dado um pedido *Get* sobre uma determinada chave (1), o módulo *CryptoWorker* irá intercetar o pedido (2) e executar a respetiva operação de codificação, cifrando a chave com a *CryptoBox* de *Deterministic Encryption* (3a) (3b). Esta nova operação *Get* será então enviada para o *NoSQL Backend* (4), que por sua vez irá executar a interrogação. O resultado é retornado para o *CryptoWorker* do lado seguro (4) para descodificar a informação protegida, recorrendo novamente às *CryptoBoxes* e à camada de mapeamento, que decifrarão o conteúdo protegido do par chave-valor, resultando nos valores originais (3a) (3b), e sendo finalmente retornados ao cliente original (2,1).

4.2 PROCESSAMENTO REMOTO

As técnicas criptográficas usadas nesta dissertação (5.3) apenas necessitam do processamento do *CryptoWorker* alojado na infraestrutura segura, realizando as operações de *Encode* e *Decode*. Contudo, para a *framework SafeNoSQL* ser extensível a otimizações ou outras técnicas criptográficas discutidas no capítulo 2, como é o caso de *Searchable Encryption*, deverá ser possível aplicar estruturas adicionais (*p.e.,* índices protegidos) e realizar alguma computação adicional no lado *Untrusted*. Para isso recorre-se ao módulo *CryptoWorker* do lado não confiável que permitirá realizar operações complexas através da operação de processamento (*Process*).

Para exemplificar, considere-se uma operação de pesquisa de prefixos do tipo *Like query* sobre dados protegidos com *Searchable Encryption* (1). O *CryptoWorker* do lado seguro interceta o pedido (2) e executa a operação de codificação para produzir um *token* (3a)(3b)[40]. Este *token* será enviado para o *CryptoWorker* do lado não confiável (4a) para executar a operação de processamento. O processamento remoto necessita agora de aceder à *CryptoBox* de *searchable encryption* para processar o *token* sobre os dados armazenados e gerar uma nova interrogação para processar no *NoSQL Backend* (4b, 4c, 4d). O resultado desta nova interrogação será retornado ao *CryptoWorker* do lado seguro para ser descodificado pela respetiva *CryptoBox* (3a)(3b) e finalmente, retornar o resultado final ao cliente (2,1).

5

IMPLEMENTAÇÃO DO SAFENOSQL

Como visto no capítulo 4, a *framework SafeNoSQL* permite dotar as bases de dados *NoSQL* de armazenamento e processamento seguros, garantindo a privacidade da informação sensível das mais variadas aplicações. Neste capítulo será feita uma descrição detalhada da implementação dessa arquitetura, descrevendo a base de dados *NoSQL* abordada, as técnicas criptográficas adotadas e as funcionalidades implementadas tendo em conta os fatores privacidade e segurança.

5.1 VISÃO GERAL DA IMPLEMENTAÇÃO DO SAFENOSQL

O protótipo *SafeNoSQL*, produzido em *Java*, está implementado sobre a base de dados *Apache HBase* (*CryptoWorker*), estando presente no lado seguro como um *Secure HBase Client* e no lado não confiável como um cluster *HBase*.

Como *CryptoBoxes* foram adotadas as técnicas criptográficas *Standard Encryption (STD)* que fornece garantias fortes de segurança, *Deterministic Encryption (DET)* que dispõe de propriedades determinísticas, *Order-Preserving Encryption (OPE)* que preserva as propriedades determinísticas e ordem entre valores (tal como *plaintext*) e ainda *Format-Preserving Encryption (FPE)* que assume propriedades determinísticas e preserva o formato dos valores (mais especificamente de inteiros). Ainda, o módulo *SafeMapper* foi instânciado de forma a fornecer um mapeamento direto com as tabelas da base de dados *HBase* e os seus metadados. Na figura 13 está representada a implementação do sistema *SafeNoSQL* sobre a base de dados *HBase*. As caixas cinzentas representam os novos componentes adicionados à solução original do *HBase* de forma a possibilitar uma implementação de *NoSQL* seguro.

5.2 APACHE HBASE

Como caso de estudo para esta dissertação, foi usada a base de dados *NoSQL Apache HBase*, uma base de dados não relacional, distribuída, que oferece grande eficiência e escalabilidade [70]. O seu desenvolvimento foi inspirado na base de dados *BigTable* da *Google*,

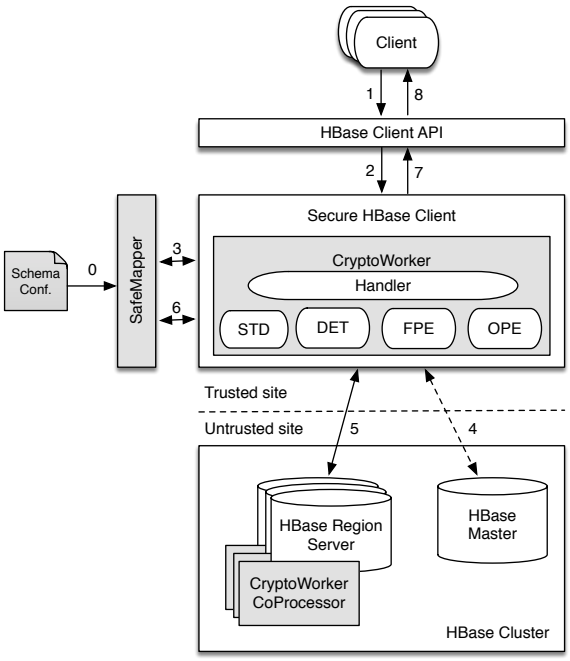


Figura 13: Implementação do sistema *SafeNoSQL*.

proposto por *Chang* [75]. A escolha desta base de dados deve-se ao facto da mesma ser *open-source*, o que é fundamental para integrar soluções de computação segura, e também de ser uma base de dados com bastante visibilidade e utilizada em vários aspetos pelas comunidades ciêntifica e empresarial [70].

Esta base de dados assume um modelo orientado à coluna e tal como a bases de dados relacionais, os dados são guardados em tabelas. As tabelas são composta pelos identificadores das linhas, colunas e *timestamps*. O identificador da linha ou *row key* é único e corresponde a um vetor de *bytes*. As colunas são compostas por dois níveis: as *column families* e os *column qualifiers*. As primeiras são definidas aquando a criação da tabela e são compostas por múltiplos *column qualifiers*. As últimas, irão conter os valores da base de dados e tal como os *timestamps*, são criados dinâmicamente, sendo definidos aquando a inserção de um novo par chave-valor (linha). A tabela 3 descreve um exemplo da estrutura das tabelas no *HBase*.

Identifier (Key)	Name (CF)		Contacts (CF)		
	First(CQ)	Last(CQ)	Phone Number(CQ)	Mobile(CQ)	Email(CQ)
1627	John	Doe	(800) 609-2233	(800) 420-1372	jdoe@gmail.com
1821	Anna	Far	(202) 513-4280	(202) 698-3281	far_a@gmail.com

Key - Row key, CF - Column Family, CQ - Column Qualifier

Tabela 3: Exemplo de uma tabela no *HBase*.

A figura 14 ilustra uma visão lógica da arquitetura do *HBase*, sendo esta composta por várias camadas, cada uma com a sua função. A *Application Layer* é onde a aplicação corre e interceta todas as ações dos clientes, convertendo essas ações em interrogações à *HBase Client API*. Essas interrogações serão enviadas para a camada *HBase Client*, onde serão digeridas e enviadas para o *HBase Backend*.

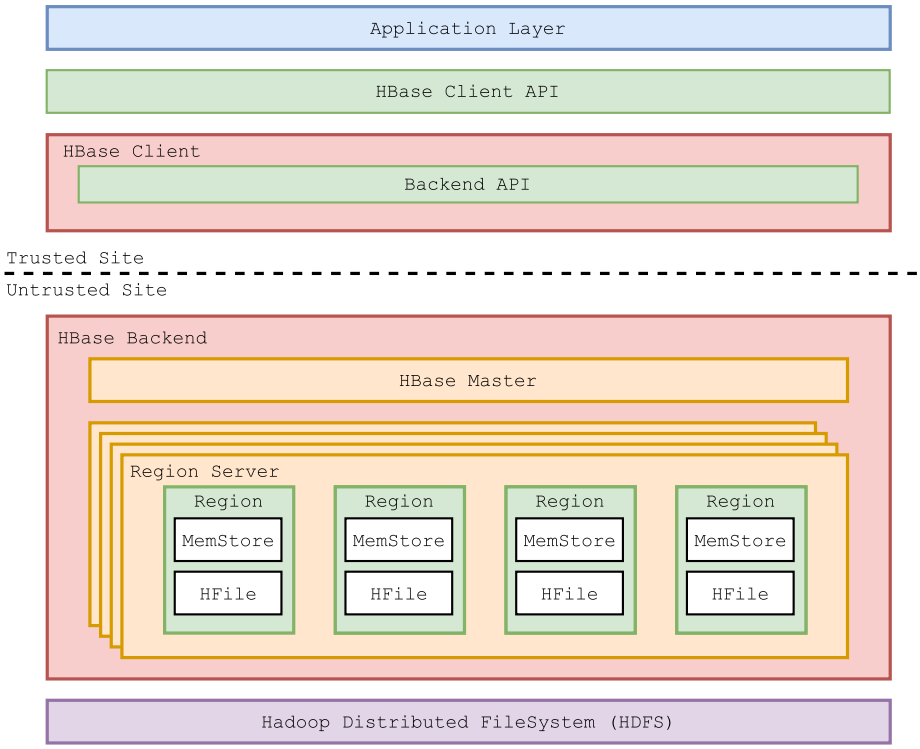


Figura 14: Visão lógica da arquitetura do *HBase*.

O *HBase Backend* é composto por um componente denominado *HBase Master*, que recebe interrogações e é responsável por distribuir as mesmas por múltiplos *Regions Servers*. Cada *Region Server* irá guardar um subconjunto de linhas que serão armazenadas em várias *Regions*. Cada *Region* é servida apenas por um *Region Server*, mas cada *Region Server* pode ser responsável por várias *Regions*. De notar que os *Region Servers* podem correr em servidores independentes, escalando a solução horizontalmente, e é nestes que a maioria da computação da base de dados ocorre.

A informação armazenada numa *Region* passa por duas fases. Numa primeira fase, os dados são armazenados em memória na camada *MemStore*, podendo ser acedidos e/ou modificados mais facilmente. Numa segunda fase, quando a *MemStore* não tem mais espaço ou lhe é invocada a operação *flush*, os dados atuais são escritos para um ou mais ficheiros denominados de *HFiles*. Cada *HFile* irá armazenar os valores originalmente criados pelas camadas anteriores sobre o formato chave/valores. Por sua vez, os *HFiles* são armazenados

no *Hadoop Distributed FileSystem (HDFS)*, o sistema de armazenamento primário usado pelas aplicações *Hadoop* [76].

Para além de fazer a distribuição uniforme da carga alocando regiões aos *Region Servers* com menor taxa de utilização, o *HBase Master* gere onde a computação é efetuada. Esta entidade faz a administração do *cluster* num modelo *master/slave* e é o ponto de entrada do sistema. Assim, todas as operações que o cliente envia ao sistema, têm de interagir primeiro com o *Master*, que indicar-lhe-á em que *Region Server(s)* terá de efetuar a computação.

No que diz respeito às operações típicas de uma bases de dados, o *HBase* disponibiliza à camada aplicacional através da *HBase Client API* as operações *PUT*, *GET*, *DELETE* e *SCAN*.

A operação *PUT* é usada para as inserções na base de dados. Esta operação necessita de uma *row key*, uma *column family* e um *column qualifier* para poder inserir com sucesso o par chave-valor pretendido. Para remover dados, é usada a operação *DELETE* e indicando apenas a *row id*, todos os dados dessa linha são removidos ou então, para remover valores mais específicos, adiciona-se também a *column family* e/ou *column qualifier*.

A operação *GET* permite ir buscar à base de dados todos os valores armazenados numa determinada linha com um certo *row id*. A operação *SCAN* permite ir buscar os pares chave-valor associados a um conjunto contíguo de chaves. Ainda, estas operações possibilitam o uso de operadores de filtro que permitem retornar um subconjunto de resultados ao cliente. Apesar de não diminuirem o processamento do servidor, os filtros reduzem simultaneamente a largura de banda e o volume de dados a ser processado pelo cliente. No âmbito desta tese foram abordados os filtros *RowFilter*, *SingleColumnValueFilter*, *FilterList* e *WhileMatchFilter*.

O *RowFilter* é utilizado para filtrar identificadores de linha, ao passo que o *SingleColumnValueFilter* filtra sobre os valores das colunas. Os filtros *WhileMatchFilter* e *FilterList* são na realidade *wrappers* de filtros que recebem outros filtros como argumento. Estes permitem dotar os seus argumentos com propriedades de filtragem adicionais. No caso do *WhileMatchFilter*, retorna o subconjunto de resultados até não se verificar a condição especificada. O *FilterList* permite agrupar filtros, interoperando os seus resultados através dos operadores *AND* e *OR*.

Para além das operações base, o *HBase* dispõe também as operações *checkAndPut*, *incrementColumnValue*, *getRegionLocation* e *getRowOrBefore*.

O *checkAndPut* verifica atomicamente se um valor armazenado em *row-key/family/qualifier* corresponde a um valor esperado. Se corresponder, adiciona um objeto *Put* enviado como parâmetro. O *incrementColumnValue* incrementa atomicamente o valor de uma coluna específica. O *getRegionLocation* devolve a localização da região em que se situa um identificador de linha dado como parâmetro. Por fim, na operação *getRowOrBefore*, dado um identificador de linha é devolvida a linha cujo identificador é igual ou, na sua inexistência, o identificador precedente.

5.3 CRYPTOBOX

Como referido na arquitetura do *SafeNoSQL* (4), as *CryptoBoxes* são entidades modulares que contém técnicas criptográficas para proteger a informação sensível. Relativamente aos esquemas criptográficos, para além de possibilitar o armazenamento do conteúdo no seu estado original *(PLT)*, foram abordados os esquemas *Standard (STD)*, *Deterministic (DET)*, *Order-Preserving (OPE)* e *Format-Preserving Encryption (FPE)*. Todas as *CryptoBoxes* estão implementadas em *C++*, com exceção do esquema *FPE* que está implementado em *Java*.

5.3.1 *Standard Encryption (STD)*

Uma das soluções integradas no sistema é um esquema de *Standard Encryption*. Este esquema está assente sobre uma cifra AES com uma chave de 128 bits a operar no modo CBC com um IV aleatório, assumindo propriedades não determinísticas. Esta abordagem, construída com recurso à biblioteca criptográfica *OpenSSL*, é rápida, eficiente e segura contra ataques por força bruta, visto que no estado da arte atual diz-se que uma cifra é segura quando a chave é maior ou igual a 80 bits, pois não existe computação suficiente para fazer 2^{80} substituições em tempo útil [77]. Para além disso esta cifra apresenta como garantias de segurança IND-CPA, não havendo assim fuga de informação do criptograma [16].

Contudo por não se tratar de um esquema determinístico nem preservar a ordem entre criptogramas, a computação será maioritariamente feita do lado do cliente (*Client-side computation*). Assim, dadas as circunstâncias da cifra, apesar da *framework SafeNoSQL* disponibilizar a sua utilização nas funcionalidades *HBase* implementadas, é aconselhável apenas usar esta cifra como proteção de informação extremamente sensível e que não sejam efetuadas interrogações diretas sobre estes dados. Caso contrário, há a necessidade de trazer toda a informação armazenada na base de dados para o lado seguro, de forma a poder ser decifrada e processada, levando a elevados gastos de recursos do cliente (largura de banda, processamento, memória) e consequentemente a uma perda de desempenho acentuada.

Assim, as operações que envolvem processamento sobre conteúdo cifrado como *Get*, *Delete*, *Update* e *Scan* não são eficientemente processadas.

5.3.2 *Deterministic Encryption (DET)*

Outra abordagem integrada no sistema é um esquema de *Deterministic Encryption* [17]. Este esquema, construído com recurso à biblioteca *OpenSSL*, está assente sobre uma cifra AES com o tamanho da chave e o modo de operação iguais à *CryptoBox* anterior (128 bits, CBC) [77]. Contudo, para assumir propriedades determinísticas o esquema terá um IV fixo, mantendo desempenho semelhante ao caso anterior, porém menos seguro. Relativamente a

ataques por força bruta, as garantias são iguais ao esquema anterior. Já a ataques por CPA, o esquema não é seguro por haver uma maior facilidade por parte de atacantes em encontrar padrões conhecidos no criptograma.

Pela mesma razão, comparativamente à *CryptoBox STD* já é possível efetuar mais operações eficientemente (*Server-side Computation*), *p.e.*, *Get*, *Update*, *Delete* ou mesmo filtros de igualdade de valores (*SingleColumnValueFilter(EQ)*).

Por não preservar a ordem entre os criptogramas, continua a não ser possível efetuar eficientemente operações *Scan* ou filtros de ordem de valores (*SingleColumnValueFilter(GR)*).

Comparativamente ao *STD*, verifica-se aqui um excelente exemplo dos compromissos de desempenho, funcionalidades e segurança. Visto que na *CryptoBox STD* temos uma garantia de segurança superior e funcionalidades reduzidas (em *Server-side Computation*), em *DET* temos uma garantia de segurança mais relaxada, *PRIV*, mas maior possibilidade de operações. Em termos teóricos, o desempenho da cifra *STD* é ligeiramente inferior por ter de gerar um IV em cada cifragem, ao passo que em *DET* o IV é fixo.

5.3.3 *Order-Preserving Encryption (OPE)*

Com as *CryptoBoxes STD* e *DET* temos uma solução aleatória e uma solução determinística, assim, surgindo a necessidade de integrar no sistema um esquema de *Order-Preserving Encryption*. Para esse efeito, foi integrada a implementação de *OPE* de *Boldyreva* com recursos às bibliotecas *OpenSSL* e *MPFR* (*Multiple-Precision Floating-Point library*) [18][77][78]. Esta solução segue um algoritmo de distribuição hipergeométrica que assume propriedades determinísticas e possibilita a preservação da ordem dos dados após a cifragem [79]. Contudo, pelas mesmas razões, este esquema torna impraticável a assunção de garantias de segurança fortes (IND-CPA ou mesmo IND-OCPA), assumindo assim garantias de segurança mais relaxadas, POPF-CCA. Mais tarde, demonstrou-se que o esquema de *Boldyreva* apresenta fuga de informação sensível em metade dos *bits* do *plaintext* [22].

Comparativamente à *CryptoBox DET*, para além das operações permitidas pela mesma, é também possível realizar operações *Scan* e filtros de ordem sobre criptogramas (que sejam identificadores de linha ou valores). Desta forma a *CryptoBox OPE* possibilita a cobertura de grande parte das operações do *Apache HBase*, possibilitando a computação segura sobre informação sensível.

Apesar da *CryptoBox OPE* atual apresentar fugas de informação de cerca de metade dos *bits* do *plaintext*, pretende-se acima de tudo apresentar uma solução OPE funcional num contexto de aplicações de tempo real. Além disso, como descrito na secção 4, o sistema apresenta uma arquitetura modular tornando possível uma troca transparente por técnicas OPE mais seguras e/ou eficientes [37][38].

5.3.4 *Format-Preserving Encryption (FPE)*

A última *CryptoBox* implementada no sistema é um esquema de *Format-Preserving Encryption* (*FPE*), baseado nos métodos apresentados no guião de referência do *National Institute of Standards and Technology (NIST) "Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption"* [80]. Em maior detalhe, esta *CryptoBox* segue implementação da técnica criptográfica *FF1* proposta por *Bellare et al.* [81].

Esta solução funciona sobre circuitos de *Feistel* e tem uma cifra por blocos como base de cifragem dos dados. Como argumentos das funções de cifragem e decifragem, para além do conteúdo sensível, são necessários um *radix* e um *tweak*. O *radix* representa a base dos valores a processar, *p.e.*, binário (base 2), décimal (base 10). O *tweak* tem a mesma função que um IV.

A tabela 4 identifica as principais propriedades do esquema *FF1* da *CryptoBox FPE*, sendo as rondas o número de iterações no circuito de *Feistel*, o *Tweak.length* o comprimento do *tweak* em bits e *CC* o comprimento dos criptogramas gerados.

Radix	$radix \in [2 \,..\, 2^{16}]$
Tweak	$tweak.length \in [0..2^{32}]$
Cifra	AES-128 CBC
Rondas	10
CC	$x \in [2 \,..\, 2^{32}]$

Tabela 4: Propriedades do esquema *FF1* da *CryptoBox FPE*.

Relativamente a garantias de segurança, por se tratar de um esquema determinístico está sujeito aos ataques modelo desta propriedade. Para além disso, por usar circuitos de *Feistel* e assumir propriedades de preservação do formato dos valores, está também sujeito a ataques por inferência e ataques ao circuito de *Feistel* [82][83].

Finalmente, em termos de funcionalidades *HBase*, como a *CryptoBox FPE* assume propriedades determinísticas, são garantidas as mesmas funcionalidades que a *CryptoBox DET*. Para além disso, por preservar o formato do conteúdo original, é um excelente caso de estudo para a preservação do mesmo tamanho de armazenamento que em *plaintext*.

Em resumo, as *CryptoBoxes* implementadas na *framework SafeNoSQL* possibilitam segurança configurável e cobrem grande parte das funcionalidades *HBase*. A figura 15 demonstra uma visão abstrata dos compromissos entre a quantidade de funcionalidades *HBase* permitidas e as garantias de segurança assentes sobre as *CryptoBoxes*.

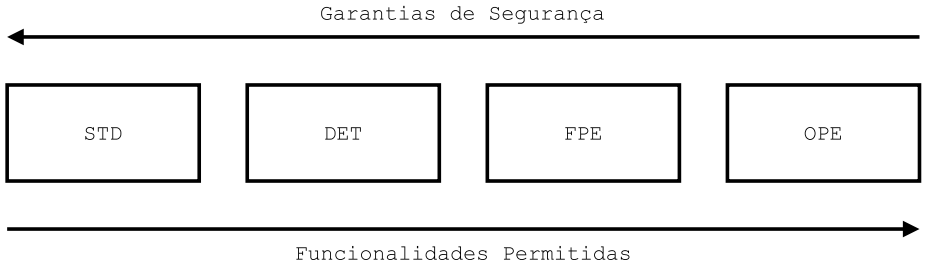


Figura 15: Compromissos entre funcionalidades permitidas e garantias de segurança das *CryptoBoxes*.

5.4 CRYPTOWORKER

Conforme descrito na arquitetura do sistema *SafeNoSQL* (4) o papel do módulo *CryptoWorker* é intercetar de forma transparente os pedidos da base de dados *NoSQL* e traduzi-los em operações seguras e privadas. Desta forma, tal como apresentado na figura 16, o *SafeNoSQL* será integrado com a base de dados *HBase* na camada *HBase Client*, assegurando então a computação segura da informação sensível no lado confiável, sem haver a necessidade de modificar o *HBase Backend*.

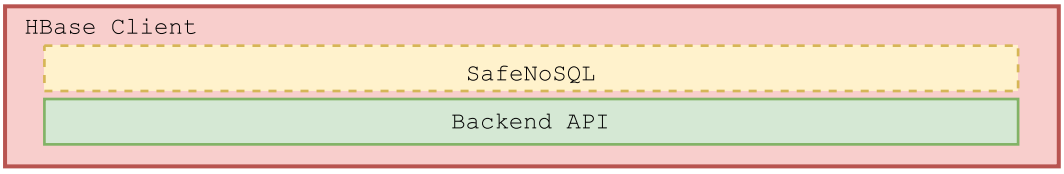


Figura 16: Visão da integração do *SafeNoSQL* com o *HBase*.

Na implementação atual do *SafeNoSQL*, de forma a suportar computação segura sobre informação sensível, foram modificadas as operações descritas em 5.2. Os novas operações incluem o fator segurança como mais uma variável do sistema.

A tabela 5 representa a relação entre as operações modificadas e as *CryptoBoxes* suportadas pelas mesmas.

Na prática, todas as operações suportam todas as *CryptoBoxes*, contudo estão apenas assinaladas na tabela as operações cujo processamento é totalmente efetuado pelo lado *Untrusted* (*Server-side Computation*).

De notar que relativamente às operações base *Put*, *Get* e *Delete* para além de serem suportadas as interrogações individuais (*Put(Put p)*, *Get(Get g)*, *Delete(Delete d)*), para efeitos de desempenho são também suportadas interrogações em *batch* (*Put(List<Put> p)*, *Get(List<Get> g)*, *Delete(List<Delete> d)*).

Ainda, é possível verificar que todas as operações são suportadas por pelo menos uma *CryptoBox*, com excepção do *incrementColumnValue*. Neste caso, haveria a necessidade de

Operação *HBase*	**PLT**	**STD**	**DET**	**OPE**	**FPE**
Put	✓	✓	✓	✓	✓
Get	✓		✓	✓	✓
Delete	✓		✓	✓	✓
Scan	✓			✓	
RowFilter*	✓			✓	
QEF*	✓		✓	✓	✓
QRF*	✓			✓	
CheckAndPut	✓		✓	✓	✓
IncrementColumnValue	✓				
GetRegionLocation	✓		✓	✓	✓
GetRowOrBefore	✓			✓	

**Operações de Filtro; QEF = Qualifier Equality Filter; QRF = Qualifier Range Filter; (QEF ∩ QRF) ∈ SingleColumnValueFilter.*

Tabela 5: Operações *HBase* e sua relação com as *CryptoBoxes*.

usar uma *CryptoBox* com um esquema de *Paillier Encryption* para possibilitar a adição de valores cifrados.

Relativamente a otimizações no sistema, surgiu a necessidade de modificar a forma com que os dados protegidos com a *CryptoBox OPE* são decifrados, devido ao elevado custo de decifragem desta técnica criptográfica (6.2.1). Para colmatar esta limitação, durante o processo de leitura e construção do esquema da base de dados, para cada *column qualifier* protegida com a *CryptoBox OPE* será adicionalmente criada uma *column qualifier* que armazena o mesmo conteúdo mas protegido com a *CryptoBox STD*. Desta forma, é possível tirar partido das propriedades determinísticas e de ordem preservadas pelo *OPE* e da eficiência da operação de decifragem da técnica *STD*.

5.5 SAFEMAPPER

Como referido na secção 4, o módulo *SafeMapper* é uma camada que permite fazer o mapeamento entre o esquema de uma base de dados *NoSQL* e as respetivas propriedades de segurança assentes sobre cada coluna (*p.e., CryptoBox* que protege cada coluna) descritas num ficheiro de configuração, sendo gerada uma representação em memória da união do esquema da base de dados e as respetivas propriedades.

No âmbito da *framework SafeNoSQL* e da base de dados *HBase*, a camada *SafeMapper* divide-se em cinco entidades: o *Schema Parser*, o *Table Schema*, a *Key*, a *Family* e o *Qualifier*. O *Schema Parser* é responsável por analisar ficheiros de esquemas de bases de dados. Com esta análise serão retiradas as informações de *column families* e *column qualifiers* específicas, bem como os respetivos metadados que asseguram o tipo de segurança assente sobre as mesmas. Toda a informação e metadados recolhidos, são mapeados para os objetos *Family*

e *Qualifier*, respetivamente. Relativamente aos identificadores de linha, os metadados são mapeados para o objeto *Key*. Por sua vez, estes objetos estão contidos num *Table Schema* que permite fazer o mapeamento de uma tabela *HBase*. A figura 17 descreve a visão lógica da camada *SafeMapper*.

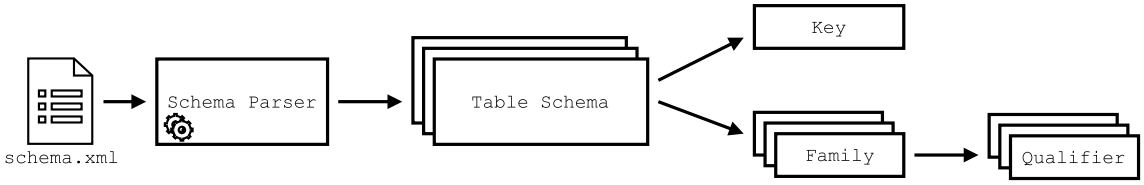


Figura 17: Visão lógica da camada de *mapping* do esquema da bases de dados.

Durante o processamento das operações, a camada *SafeMapper* auxilia na tradução segura das interrogações. Considerando como exemplo uma interrogação *put(Put p)*, o processo de tradução é efetuado em cinco passos: (1) o *SafeNoSQL* interceta a operação original *put*; (2) o objeto *Put p* é analisado e são extraídas as células que contém a informação relativa às *column families (byte[] family)* e *column qualifiers (byte[] qualifier)* e o respetivo valor a proteger (*byte[] value*); (3) retirada a informação útil de cada célula, esta é enviada para o objeto *TableSchema* que irá pesquisar os objetos *Family* e *Qualifier* relativos aos valores *family (byte[])* e *qualifier (byte[])* respetivamente; (4) encontrados os objetos, o valor *value* é cifrado de acordo com os metadados previamente especificados; (5) este processo é feito para todas as células do objeto *Put*, e resulta num novo objeto *Put*, seguro, que por sua vez será processado no *HBase Backend*.

No caso de operações *Get* o procedimento é dividido em dois processos: cifragem do identificador de linha e decifragem do resultado retornado. Na cifragem, o processo é semelhante ao procedimento especificado para a operação *Put*. Na decifragem do resultado retornado pelo *HBase Backend*, os passos (1), (2), (3) e (4) são semelhantes aos especificados acima. Já nos passos (5) e (6), os valores de todas as colunas são decifrados (5) e reconstruídos num novo objeto *Get* (6), que por sua vez será retornado ao cliente.

De notar que esta estrutura de mapeamento é armazenada em memória na máquina do cliente de base de dados, sendo pesquisada e atualizada eficientemente.

6

AVALIAÇÃO EXPERIMENTAL

Definida a arquitetura e implementação de um protótipo do sistema *SafeNoSQL* sobre a base de dados *HBase*, é necessário efetuar uma avaliação precisa e extensa sobre o sistema, de forma a perceber o impacto da computação segura em bases de dados *NoSQL*. Assim, neste capítulo é feita uma apreciação dos resultados obtidos para um conjunto extenso de testes, sendo avaliado o desempenho do *SafeNoSQL* em ambientes de teste mais isolados (micro testes), ambientes mais realistas (macro testes) e ainda com múltiplos clientes (teste de carga). Desta forma, é possível determinar a aplicabilidade da *framework*, percebendo quais os pontos de contenção do sistema, facilitando a identificação e resolução de problemas de desempenho.

6.1 CONFIGURAÇÕES EXPERIMENTAIS

A avaliação do sistema *SafeNoSQL* é constituída por dois níveis de configurações, a configuração de *hardware* e a configuração da base de dados (*HBase*).

A nível de *hardware*, a avaliação foi feita num *cluster* composto por seis servidores, cada um deles equipado com um processador *Intel i3* com quatro *cores* de 3.7 GHz, 8GB de memória e um disco SSD de 128GB. A conexão da rede partilhada entre as máquinas foi feita usando um *switch* de um gigabit.

O *cluster HBase* foi distribuído em cinco nodos: um *HBase Master* a correr num servidor isolado e quatro *HBase RegionServers* a correr nas restantes máquinas. Para cada *RegionServer* foram atribuídos 4GB de *heap* (memória Java), sendo 55% do espaço dedicado à *Memstore* e 10% à *Block Cache*. Os restantes 35% são dedicados à JVM para chamadas *RPC* e operações internas do *HBase* e *HDFS*.

O último nodo foi usado como cliente de base de dados que avaliou o desempenho da base de dados *NoSQL*. Para esse efeito, foi usada a plataforma de avaliação do desempenho de bases de dados, YCSB [84].

Para uma avaliação mais precisa dos micro testes (6.2), macro testes (6.3) e testes com múltiplos clientes (6.4), todos os testes correram sobre uma base de dados pré-populada

com 10 milhões de linhas durante 20 minutos com um período de *"aquecimento"* prévio de 3 minutos. Ainda, cada um dos testes foi repetido 5 vezes para calcular a média e o desvio padrão de cada execução. De notar ainda que as avaliações dos micro testes, macro testes e com múltiplos clientes demoraram aproximadamente 2 dias e 13 horas, 2 dias e 10 dias, respetivamente.

6.1.1 *Yahoo! Cloud Serving Benchmark*

O YCSB é uma plataforma de avaliação de desempenho de bases de dados *NoSQL* alojadas em nuvem, incluindo *HBase*, que permite a avaliação destes sistemas com *workloads* realistas [84]. A motivação da criação desta plataforma rege-se pela uniformização do processo de avaliação de bases de dados, possibilitando a avaliação de sistemas diferentes com *workloads* iguais. Uma *workload* dispõem de um conjunto de propriedades parametrizáveis que ditam o ambiente de execução e a carga de trabalho a ser exercida sobre a base de dados.

Cada propriedade presente numa *workload* influencia diretamente o ambiente de testes. As propriedades mais comuns do YCSB são: o *RecordCount* que corresponde ao número de linhas a inserir na fase de população da base de dados (de notar que nesta fase não são realizadas operações para além da população da base de dados); o *OperationCount* que corresponde ao número total de operações a efetuar durante a execução dos testes de avaliação; *MaxExecutionTime* que corresponde ao tempo máximo de cada execução (a execução termina quando o tempo decorrido do teste é igual ao *MaxExecutionTime* especificado ou quando o número total de operações foi atingido); o *ThreadCount* corresponde ao número de clientes (*threads*) de base de dados a operar em simultâneo; o *RequestDistribution* corresponde ao tipo de distribuição a aplicar sobre a geração de pares chave-valor e a ordem das operações a serem executadas. Os dois tipos mais comuns de distribuição são a uniforme (*Uniform Distribution*) que gera e distribui conteúdo aleatório e a *zipfian* (*Zipfian Distribution*) que gera e distribui o conteúdo segundo uma distribuição em que certos elementos da base de dados são acedidos mais frequentemente (mais populares). Por fim, o *OperationProportion* corresponde à percentagem do tipo de operações a efetuar na execução da *workload*. Esta propriedade é composta por um conjunto de 6 operações: *InsertProportion* corresponde à proporção de operações de inserção (operação *HBase Put*); *ReadProportion* corresponde à proporção de operações de leitura (operação *HBase Get*); *DeleteProportion* corresponde à proporção da operações de eliminação (operação *HBase Delete*); *UpdateProportion* corresponde à porporção de operações de atualização de valores (operação *HBase Put*); *ScanProportion* corresponde à proporção de operações de pesquisa (operação *HBase Scan*) e *ReadModifyWriteProportion* corresponde à proporção de operações de leitura de uma linha, modificação de um valor, e respetiva atualização desse valor (operações *HBase Get-Put*). A soma das proporções das operações a realizar deve ser igual a 100%.

Contudo, o YCSB nativo não fornece algumas propriedades úteis para a avaliação do sistema *SafeNoSQL*, que é o caso dos operadores de filtro, geração de *workloads* pseudo-aleatórias e ainda *"tempo de aquecimento"* prévio da base de dados. Desta forma, o módulo YCSB para a base de dados *HBase* foi modificado de forma a possibilitar estas propriedades. Ainda, o conjunto de propriedades disponibilizado pelo YCSB foi extendido com as propriedades: *FilterProportion* que corresponde à proporção de operações de filtro; *FilterType* que corresponde ao tipo de filtro a ser executado (atualmente os filtros suportados são o *RowFilter* e *SingleColumnValueFilter*); *CompareValue* que corresponde ao valor a ser comparado pelo filtro; *FilterColumnDescriptors* que indica a que *column family* e *column qualifier* o filtro será executado; *ComparisonProportion* que corresponde à proporção do comparador a usar sobre o filtro ($=, >, <, >=, <=$); *Seed* que corresponde ao valor a inserir nos geradores aleatórios que determinam a próxima operação e os valores sobre os quais irá ser aplicada, de modo a gerar valores pseudo-aleatórios e garantir sempre o mesmo ambiente de execução; *RampUpTime* que corresponde ao *"tempo de aquecimento"* da base de dados.

6.1.2 *Esquemas de base de dados*

De modo a avaliar de forma realista o sistema *SafeNoSQL*, foram criadas especificamente dois esquemas de bases de dados, tabelas 6 e 7, que simulam casos de estudo reais no ambiente do setor da saúde [85]. Este caso de estudo é relevante pois, visto ser um setor que gere grandes quantidades de informação sensível acerca de pacientes, médicos, auxiliares de saúde e infraestruturas de saúde. Não obstante, o armazenamento desta informação deve cumprir vários regulamentos legais relativos à privacidade de dados [86]. Desta forma, é possível extrair resultados mais relevantes, levando consequentemente a conclusões mais significativas acerca dos custos de desempenho induzidos pelo uso de diferentes técnicas criptográficas.

O primeiro esquema, *Patients*, é descrito na tabela 6 e complementa um subconjunto de dados tipicamente encontrados numa base de dados hospitalar que dizem respeito à informação pessoal dos pacientes. Este esquema é composto por um identificador de linha para cada paciente, aleatoriamente gerado pela camada aplicacional, e por um conjunto de *column families* (*Identificação*, *Contactos*, *Obs.* e *Cons.*) que agrupam um conjunto distinto de *column qualifiers* que dizem respeito à informação sensível dos pacientes (*MainID*, *Nome*, *Apelido*, *D. Nasc.*, *Nac.*, *C.C.*, *Morada*, *Contacto* e *Obs.*). A *column family Cons.* pode ter um número dinâmico de *column qualifiers*, cada um deles fazendo referência ao identificador de uma consulta médica de um paciente em questão (este identificador diz respeito ao identificador de linha do esquema da tabela 7).

A tabela 6 indica ainda o tamanho, em bytes, de cada *column qualifier*, bem como uma proposta das técnicas criptográficas a serem aplicadas sobre cada campo, de forma a obter o

melhor compromisso entre as funcionalidades a processar na base de dados e a privacidade da informação sensível dos pacientes.

	Identificação						**Contactos**		**Obs.**	**Cons.**
Key	*MainID*	*Apelido*	*Nome*	*D. Nasc.*	*Nac.*	*C.C.*	*Morada*	*Contacto*	*Obs.*	*[1-*]*
8	64	64	64	14	4	9	256	13	1024	8
DET	DET	STD	STD	STD	STD	STD	STD	STD	STD	STD

Cons. - Consultas; MainID - Identificador principal; D. Nasc. - Data de Nascimento; Nac. Nacionalidade; C.C. - Cartão de Cidadão; Obs. - Observações.

Tabela 6: Esquema de base de dados *NoSQL* relativo a pacientes hospitalares.

Como é possível verificar, os campos diretamente associados à informação pessoal dos pacientes são protegidos com *Standard Encryption*. Contudo, de forma a possibilitar computação sobre a informação dos pacientes, o *column qualifier MainID* foi protegido com *Deterministic Encryption*. Este identificador é construído através do primeiro e último nome e data de nascimento do paciente, sendo frequentemente usado para identificar os pacientes em sistemas da área da saúde.

De notar ainda que o espaço de armazenamento de uma linha em *plaintext* é 1552 bytes e para o esquema seguro proposto é 1888 bytes. Este aspeto revela-se importante por representar novos compromissos em sistemas de computação segura sobre bases de dados *NoSQL*, que é o caso do espaço de armazenamento dos criptogramas e a largura de banda no processamento dos mesmos.

O segundo esquema, *Appointments*, é apresentado na tabela 7 e armazena as consultas médicas de um hospital para um dado médico e um paciente. O esquema é composto por um identificador de linha para cada consulta, gerado aleatoriamente pela camada aplicacional, e por um conjunto de *column families* (*Médico*, *Paciente*, *Consulta* e *Instituição*) que agrupam um conjunto distinto de *column qualifiers* que dizem respeito à informação sensível das consultas médicas (*Méd. ID*, *Pac. ID*, *Data*, *Tipo*, *Obs.*, *Nome* e *Morada*).

Nesta tabela, é apresentada uma proposta para um possível esquema de base de dados em que os *column qualifiers Méd. ID* é cifrado com *DET* e a *Data* com *OPE*, enquanto que os restantes *column qualifiers* são cifrado com *STD*. Desta forma, o esquema permite realizar as operações mais comuns sobre consultas médicas, *p.e.*, quais as consultas agendadas para um determinado médico para o mês x.

De acordo com a otimização discutida na secção 5.4, o *column qualifier Data-STD* é criado dinâmicamente e permite reduzir o custo de decifrar dados protegidos com *OPE*. Relativamente ao espaço de armazenamento, uma linha em *plaintext* tem um tamanho de 1552 bytes, ao passo que uma linha do esquema seguro tem 1756 bytes.

	Médico	**Paciente**	**Consulta**				**Instituição**	
Key	*Méd. ID*	*Pac. ID*	*Data*	*Data-STD*	*Tipo*	*Obs.*	*Nome*	*Morada*
8 DET	16 DET	16 STD	14 OPE	14 STD	64 STD	1024 STD	128 STD	256 STD

Méd. ID. - Identificador do Médico; Pac. ID - Identificador do Paciente; Obs. - Observações.

Tabela 7: Esquema de base de dados *NoSQL* relativo a consultas hospitalares.

Em resumo, foram especificados dois esquemas sobre a base de dados, *Patients* e *Appointments*, contendo respetivamente a informação relativa a pacientes e consultas médicas de um hospital. Como é possível verificar, toda a informação considerada crítica (dados sensíveis que podem comprometer a identidade do paciente) é protegida com *STD*. Identificadores de linha, identificadores de pacientes e médicos e datas de consultas estão protegidos esquemas criptográficos com garantias de segurança mais relaxados, de forma a permitir computação eficiente sobre a informação.

6.2 MICRO-BENCHMARK

De forma a perceber o impacto das técnicas criptográficas quando aplicadas num contexto de bases de dados, foi feito um conjunto de micro testes que pretende avaliar o desempenho das operações de cifragem e decifragem de cada técnica criptográfica para conteúdos de tamanho variável. Ainda, estes testes permitem perceber qual o desempenho do sistema *SafeNoSQL* perante operações *NoSQL* com os identificadores de linha protegidos com as diferentes *CryptoBoxes* (técnicas criptográficas).

6.2.1 *Avaliação isolada das CryptoBoxes*

Numa primeira fase é necessário perceber o custo de armazenamento (e por ventura, de processamento e largura de banda) associado à geração dos criptogramas face ao conteúdo original. Assim, a tabela 8 descreve como se comporta o espaço gerado dos criptogramas, segundo as *CryptoBoxes* atuais, *STD*, *DET*, *FPE* e *OPE*. Como referido na secção 5.3.4, a técnica *FPE* preserva o tamanho do *plaintext*. Já a *CryptoBox OPE* gera um criptograma com o dobro do tamanho do conteúdo original devido ao domínio da fórmula hipergeométrica sobre ela assente [79]. Por fim, devido ao *padding* associado, as técnicas criptográficas *STD* e *DET* geram criptogramas múltiplos de 32 bytes e 16 bytes, respetivamente.

STD	$plt\_len + (32 - (plt\_len \bmod 32))$
DET	$plt\_len + (16 - (plt\_len \bmod 16))$
OPE	$plt\_len * 2$
FPE	plt_len

plt_len - tamanho em bytes do conteúdo original.

Tabela 8: Fórmulas que determinam o tamanho dos criptogramas gerados.

O incremento do tamanho dos criptogramas face ao conteúdo original, acrescenta um custo no desempenho das cifras, tal como descrito na tabela 9, e também nos recursos computacionais fornecidos pelo cliente e pelo servidor. No lado do cliente, quanto maior o tamanho dos criptogramas, maior a carga do processador para cifrar e decifrar informação. Ainda, recebidos os resultados de operações de pesquisa e filtragem, há uma maior percentagem de ocupação da memória. Já no lado do servidor, há um menor aproveitamento da *cache* da base de dados, na medida em que é necessário mais espaço para armazenar a *"mesma informação"* na *Memstore* e na *Block Cache*. O mesmo se verifica na persistência dos dados. Relativamente ao processamento, há um acréscimo no tempo gasto (e na carga do processador) em operações de igualdade ou ordem de grandeza de valores (*p.e.*, comparações de igualdade entre valores de tamanho de 80 bytes face a valores 64 bytes, revelam um custo significativo para bases de dados com muitas entradas). Este tópico é discutido em maior detalhe nas secções 6.3.1 e 6.4.

Numa segunda fase, é interessante avaliar a velocidade de execução das operações de cifragem e decifragem das *CryptoBoxes* implementadas para conteúdos de tamanho variável com o intuito de perceber o desempenho de cada cifra. A tabela 9 descreve o desempenho das *CryptoBoxes STD*, *DET*, *OPE* e *FPE*, para *plaintexts* de tamanho variável.

		1	2	4	8	16	32	64	128	256
STD	C	18.327 μs	18.513 μs	20.954 μs	21.125 μs	21.797 μs	22.158 μs	23.292 μs	26.930 μs	30.970 μs
	D	5.674 μs	5.694 μs	5.844 μs	6.094 μs	5.533 μs	6.351 μs	7.095 μs	7.848 μs	8.028 μs
DET	C	10.818 μs	12.705 μs	13.218 μs	14.198 μs	16.870 μs	17.876 μs	17.653 μs	18.258 μs	22.507 μs
	D	3.549 μs	3.848 μs	4.165 μs	5.431 μs	5.932 μs	6.262 μs	5.963 μs	6.024 μs	6.591 μs
OPE	C	1.832 ms	1.485 ms	3.681 ms	9.088 ms	25.970 ms	58.317 ms	214.503 ms	615.889 ms	2,985.567 ms
	D	60.970 μs	85.155 μs	133.815 μs	251.909 μs	629.152 μs	1.159 ms	4.553 ms	438.575 ms	2,861.170 ms
FPE	C	-	-	15.150 ms	-	-	-	-	-	-
	D	-	-	1.159 ms	-	-	-	-	-	-

C - Operação de cifragem; D - Operação de decifragem.

Tabela 9: Desempenho das *CryptoBoxes* do sistema *SafeNoSQL*.

Como é possível verificar, as operações de cifragem e decifragem das *CryptoBoxes STD* e *DET* são efetuadas na ordem dos microssegundos (μs), tornando-as especialmente úteis para aumentar o desempenho das operações sobre bases de dados seguras. De notar que a técnica *DET* revela-se mais eficiente que *STD*. Tal acontece porque a última necessita de gerar um IV a cada cifragem, ao passo que o IV da técnica *DET* é estático.

Relativamente à *CryptoBox OPE*, tal como referido na secção 5.3.3, apresenta uma custo de cifragem e decifragem bastante elevado, devido às propriedades do algoritmo hipergeométrico assente sobre a cifra. Como tal, aquando a utilização desta técnica é necessário ter atenção ao domínio de valores a proteger.

Por fim, na medida em que a *CryptoBox FPE* atual só opera sobre inteiros, apenas foi verificado o desempenho para inteiros (4 bytes).

6.2.2 *Micro testes SafeNoSQL*

Os micro testes foram efetuados sobre o esquema *Appointments* (tabela 7) de forma a perceber isoladamente o impacto das técnicas criptográficas implementadas no sistemas *SafeNoSQL*. O testes executaram operações YCSB isoladas, isto é, em cada *workload* definida era apenas executada um tipo de operação sobre a base de dados. Todos os *column qualifiers* foram armazenados em *plaintext*, ao passo que os identificadores de linha foram protegidos com as diferentes *CryptoBoxes* (*STD*, *DET* e *OPE*).

Para esta avaliação foram usadas seis *workloads*: *INS* (inserção de linhas), *READ* (leitura de linhas), *RMW* (atualização de valores, implicando uma leitura prévia da linha especificada, a modificação do valor pretendido e a escrita desse valor), *SCAN* (pesquisa sobre os identificadores de linha da base de dados), *QEF* (*Qualifier Equality Filter*, filtro de igualdade de valores) e *QRF* (*Qualifier Range Filter*, filtro de ordem de valores). Em *INS* todos os campos são escritos e em *READ* todos os campos são lidos. Para *SCAN*, *QEF* e *QRF*, todas as operações foram iniciadas numa linha aleatória (*start row*) até ao final da tabela. De notar que as *workloads INS*, *READ*, *RMW* e *SCAN* operam diretamente sobre os identificadores de linha, que por sua vez estão protegidos com as várias *CryptoBoxes*. Já as *workloads QEF* e *QRF* operam sobre *column qualifiers*, sendo que o desempenho das *CryptoBoxes* destas *workloads* apenas influenciará o processo de decifragem dos resultados retornados pelo servidor.

Esta aproximação foi usada para fornecer um cenário de testes controlado onde o único fator de mudança é a maneira com que um único tipo de informação é cifrado. Desta forma, é possível fazer uma comparação mais precisa relativamente ao custo de cada técnica criptográfica quando comparada com a versão *HBase* sem qualquer proteção. Para obter resultados mais precisos para as operações de filtragem, foi garantido que os valores a serem filtrados existiam sempre na base de dados. Ainda, para verificar como variava o desempenho do *SafeNoSQL*, foram usadas as distribuições *Uniform* e *Zipfian* para todos os testes.

O desempenho do sistema é descrito nas figuras 18 e 19 e a tabela 10, através do débito e latência de cada *CryptoBox* para cada operação *YCSB*, para ambas as distribuições, *Uniform* e *Zipfian*. A linha vermelha com valor $y = 1$ corresponde ao valor normalizado das operações em *plaintext*.

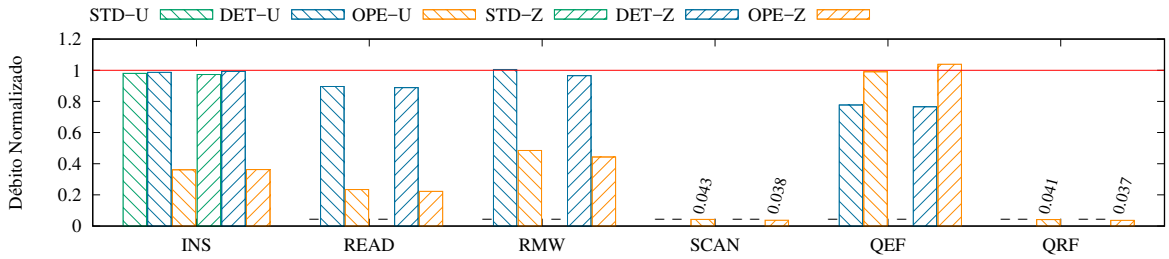


Figura 18: Débito normalizado para os micro testes.

Para *Standard Encryption* foram apenas feitos testes de escrita, enquanto que para *Deterministic Encryption* não foram feitas operações de pesquisa (do tipo *range query*). Tal como explicado na secção 5.3, a aplicabilidade de *Standard Encryption* é limitada a cenários de proteção de dados onde não são efetuadas diretamente operações de processamento sobre os mesmos. No caso de *Deterministic Encryption*, a ordem dos valores não é preservada pelos criptogramas, levando a que se efetuar-mos uma operação de pesquisa sobre valores armazenados com esta técnica, é necessário retornar toda a base de dados para o cliente para efetuar a decifragem dos dados e a respetiva comparação em *plaintext*, o que é inaplicável em qualquer base de dados que fornece serviços de tempo real e que contêm um elevado número de entradas.

No que diz respeito aos resultados, tal como esperado, operações de escrita (*INS*) sobre dados protegidos com *STD* e operações de escrita (*INS*), leitura (*READ*), atualizações (*RMW*) e pesquisas de igualdade (*QEF*) sobre dados protegidos com *DET* trazem um custo mínimo quando comparada com as mesmas operações em *plaintext*, devido à velocidade das operações de cifragem e decifragem destas técnicas criptográficas. Por outro lado, as interrogações efetuadas com a *CrytoBox OPE* têm um custo significativo devido ao desempenho desta cifra, correspondendo ao expectável, tal como apresentado na secção 6.2.1. Isto significa que o desempenho para operações que envolvem a *CryptoBox OPE* pode ser melhorado com a permutação por *CryptoBoxes* mais eficientes *p.e., Order-Revealing Encryption* ou *Searchable Encryption*.

Na avaliação das operações sobre *OPE*, a única exceção é a operação de pesquisa de igualdade (*QEF*), onde o desempenho da técnica é semelhante à operação análoga em *plaintext*. Nesta operação de filtro, a maior parte do tempo é gasto na pesquisa do valor especificado no *HBase Backend*, sendo os pares chave-valor correspondentes agrupados num objeto (*ResultScanner*), que por sua vez será retornado ao cliente de base de dados para os decifrar. Por outro lado, na medida em que o subconjunto de resultados retornados ao cliente de base de dados é significativamente superior nas operações *Scan* e *QRF*, é notável o custo elevado na decifragem dos identificadores de linha, protegidos com *OPE*.

Relativamente ao desempenho das operações com as diferentes distribuições, *Uniform* e *Zipfian*, pode-se verificar que a distribuição *Zipfian* apresenta um débito superior para a

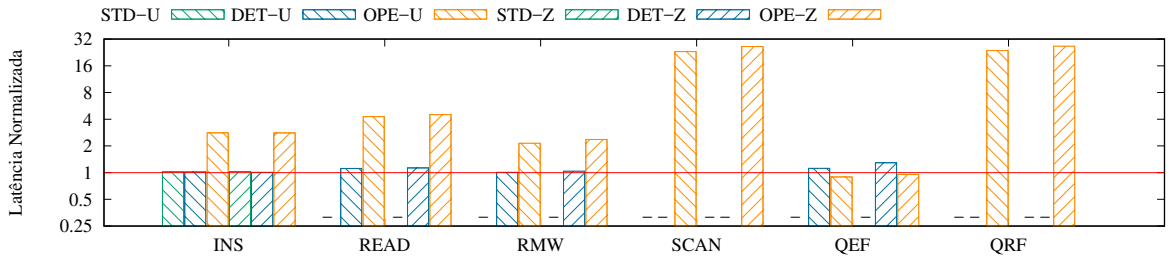


Figura 19: Latência normalizada para os micro testes.

maioria dos testes, devido à distribuição das operações por *hotspots*, permitindo tirar partido dos mecanismos de *caching* do *HBase* (*Memstore* para operações de escrita e atualização e *Block Cache* para as operações de leitura e pesquisa).

Estes resultados revelam que cada técnica criptográfica apresenta compromissos diferentes nos termos do desempenho da base de dados e das funcionalidades suportadas. Desta forma, num cenário de execução real, recorrer a uma única técnica criptográfica para proteger todo o conteúdo da mesma base de dados não é a melhor solução.

		INS		READ		RMW		SCAN		QEF		QRF	
		U	Z	U	Z	U	Z	U	Z	U	Z	U	Z
PLT	D	333.835 ±7.107	334.255 ±5.291	532.071 ±10.689	614.828 ±11.542	183.432 ±7.082	204.391 ±4.873	69.576 ±7.331	79.095 ±7.301	0.043 ±0.006	0.043 ±0.004	70.234 ±11.020	78.537 ±11.519
	L	2.033 ± 0.064	2.928 ±0.047	1.874 ±0.038	1.624 ±0.031	5.582 ±0.240	5.105 ±0.165	14.447 ±1.463	12.666 ±1.134	26675.757 ±3234.64	23404.915 ±2063.02	14.502 ±2.440	12.912 ± 2.019
STD	D	327.228 ± 5.783	325.061 ±5.746	-	-	-	-	-	-	-	-	-	-
	L	2.993 ± 0.054	3.015 ±0.056	-	-	-	-	-	-	-	-	-	-
DET	D	329.315 ± 4.954	331.907 ±7.559	476.395 ±5.631	546.04 ±22.883	184.304 ±6.208	197.209 ±8.222	-	-	0.033 ±0.002	0.032 ±0.002	-	-
	L	2.973 ± 0.046	2.951 ±0.068	2.093 ±0.025	1.828 ±0.079	5.630 ±0.137	5.313 ±0.156	-	-	29821.832 ±1850.85	30228.441 ±1951.79	-	-
OPE	D	120.438 ± 0.862	121.082 ±0.927	124.83 ±1.382	136.988 ±0.440	89.02 ±1.892	90.579 ±1.641	2.994 ±0.026	2.998 ±0.039	0.042 ±0.006	0.045 ±0.002	2.893 ±0.049	2.906 ±0.092
	L	8.234 ± 0.061	8.191 ±0.064	8.001 ±0.089	7.288 ±0.023	11.998 ±0.798	12.056 ±0.449	333.716 ±2.952	333.176 ±4.441	23816.373 ±3787.09	22355.588 ±1134.39	345.251 ±5.888	343.964 ±11.362

U - Distribuição Uniform; Z - Distribuição Zipfian; D - Débito (ops/s); L - Latência (ms).

Tabela 10: Resultados do débito e latência para os micro testes.

6.2.3 *Avaliação de Format-Preserving Encryption*

Tal como apresentado na secção 6.2.2, foi realizada uma avaliação dos micro testes das *CryptoBoxes STD*, *DET* e *OPE*, revelando os compromissos de cada técnica criptográfica (desempenho, segurança assente e funcionalidades da base de dados permitidas). Contudo, a *CryptoBox FPE* não foi integrada nesta avaliação por apresentar um domínio de cifragem incompatível, visto que a implementação atual apenas cifra (e decifra) conteúdos com tamanho de 4 bytes (inteiros), e na avaliação dos micro testes foram usados identificadores de linha com 8 bytes.

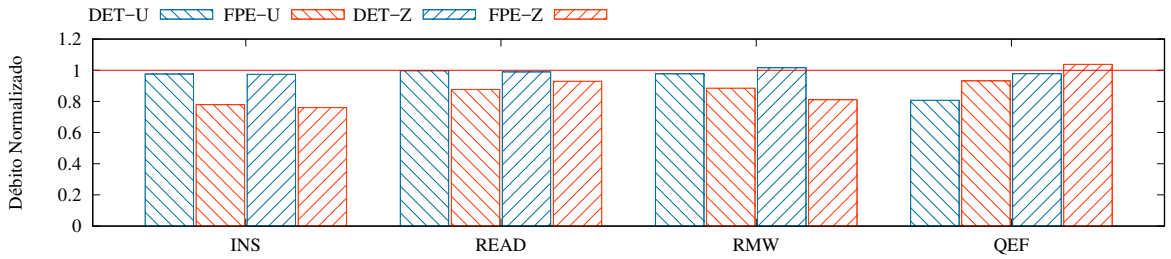


Figura 20: Débito normalizado para os micro testes com *FPE*.

Desta forma, foi realizada uma segunda avaliação dos micro testes contemplando o *HBase* sem segurança e os esquemas *DET* e *FPE*, visto fornecerem as mesmas funcionalidades de base de dados por serem ambos esquemas determinísticos. À semelhança dos micro testes anteriores, foi usado o esquema de base de dados *Appointments* presente na tabela 7, sendo apenas modificado o tamanho do identificador de linha de 8 bytes para 4 bytes.

O desempenho do sistema é descrito nas figuras 20 e 21 e na tabela 11, através do débito e latência de cada *CryptoBox* para as operações YCSB *INS*, *READ*, *RMW* e *QEF*, para ambas as distribuições *Uniform* e *Zipfian*. A linha vermelha com valor $y = 1$ corresponde ao valor normalizado das operações em *plaintext*.

Como se pode observar, ambas as técnicas seguras aproximam-se do desempenho das operações sem qualquer segurança. O custo de desempenho presente, reflete-se no desempenho das cifras criptográficas, apresentado na tabela 9. Por sua vez, a *CryptoBox DET*, comparativamente a *FPE*, assume melhor desempenho em todas as operações YCSB com exceção da operação de filtragem *QEF*. Esta última operação revela-se mais eficiente com *FPE* devido às suas propriedades de preservação do tamanho do criptograma gerado, face ao conteúdo em *plaintext* (os criptogramas gerados pelas *CryptoBoxes DET* e *FPE*, relativos aos identificadores de linha, assumem 16 bytes e 4 bytes de comprimento, respetivamente). Assim, há um maior aproveitamento da *cache* da base de dados, menor espaço de armazenamento e menor transmissão de informação entre o cliente de base de dados e o servidor. Relativamente ao uso do *CPU*, o processamento é semelhante na versão do *HBase* sem segurança, com o esquema *DET* e com o esquema *FPE*.

À semelhança dos micro testes anteriores, estes resultados revelam que cada técnica apresenta compromissos diferentes em termos do desempenho da base de dados e das funcionalidades. Ainda, apesar de fornecer pior desempenho na maioria dos testes e fornecer garantias de segurança mais relaxadas comparativamente a *DET*, o esquema *FPE* permite manter o formato e tamanho dos dados originais, sendo assim mais eficiente em termos de gestão de memória e largura de banda, conforme observado nas operações de filtros ao backend NoSQL.

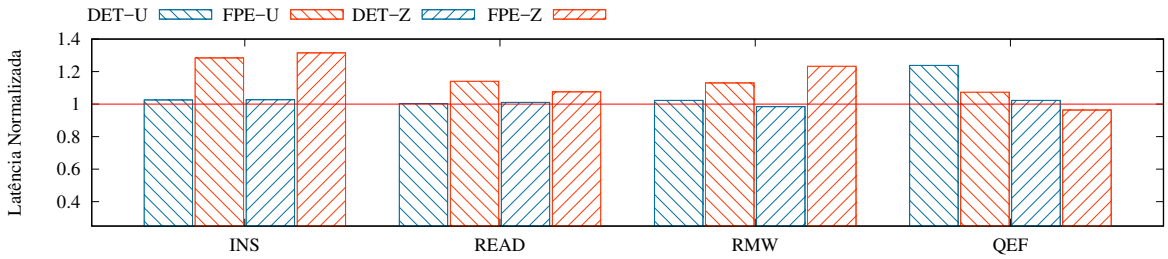


Figura 21: Latência normalizada para os micro testes com *FPE*.

		INS		READ		RMW		QEF	
		U	Z	U	Z	U	Z	U	Z
PLT	D	421.009 ± 0.988	421.168 ± 3.271	520.167 ± 23.368	550.881 ± 11.901	164.332 ± 16.741	185.661 ± 11.629	0.039 ± 0.00004	0.033 ± 0.00044
	L	2375.243 ± 0.0044	2374.346 ± 0.0179	1922.457 ± 0.185	1815.272 ± 0.0483	6085.237 ± 0.586	5386.153 ± 0.636	25298922.971 ± 244147	30184026.587 ± 445434
DET	D	410.603 ± 1.383	410.161 ± 0.882	518.533 ± 2.488	544.759 ± 10.216	160.584 ± 5.326	188.694 ± 5.446	0.031 ± 0.00074	0.0323 ± 0.00062
	L	2435.439 ± 0.0087	2438.068 ± 0.0052	1928.515 ± 0.0092	1835.673 ± 0.029	6227.263 ± 0.181	5299.575 ± 0.299	31328396.842 ± 479149	30878714.336 ± 479156
FPE	D	327.874 ± 3.308	320.247 ± 3.399	456.072 ± 1.008	512.231 ± 10.003	145.392 ± 4.401	150.525 ± 2.451	0.0368 ± 0.00002	0.0343 ± 0.00097
	L	3049.948 ± 0.031	3122.587 ± 0.033	2192.633 ± 0.0038	1952.243 ± 0.031	6877.916 ± 0.166	6643.397 ± 0.043	27157567.331 ± 11846	29073561.124 ± 848714

U - Distribuição Uniform; Z - Distribuição Zipfian; D - Débito (ops/s); L - Latência (ms).

Tabela 11: Resultado do débito e latência para os micro testes com *FPE*

6.3 MACRO-BENCHMARK

Como descrito na secção 6.2, a avaliação dos micro testes permitiu perceber os compromissos entre o desempenho da base de dados e as funcionalidades suportadas das *CryptoBoxes* implementadas sobre o sistema *SafeNoSQL*. Contudo, estes testes não se mostram suficientes para avaliar o sistema como um todo, sendo necessário realizar uma análise extensa para ambientes mais realistas. Desta forma, esta secção foca-se na avaliação dos macro testes onde diferentes técnicas criptográficas são combinadas para fornecer uma base de dados segura e funcional.

Os macro testes foram elaborados para avaliar o impacto da combinação de diferentes técnicas criptográficas para os esquemas de base de dados *Patients* (tabela 6) e *Appointments* (tabela 7). Tal como apresentado em 6.1.2, os identificadores de linha e os *column qualifiers* foram protegidos com as *CryptoBoxes* apresentadas nas tabelas 6 e 7 de forma a assegurar que interrogações úteis e importantes possam continuar a ser processadas sobre informação cifrada.

Para avaliar o desempenho do sistema *SafeNoSQL* perante ambientes de computação e processamento de informação mais realistas, foram executadas as *workloads* apresentadas na tabela 12. Estas *workloads* (*A* a *F*) correspondem às configurações padrão já disponibilizadas pelo YCSB, e são tipicamente usadas para avaliar o desempenho de bases de dados *NoSQL* [87]. A *workload E*, originalmente com 95% de operações *Scan* e 5% de operações *Insert*, foi

Workload	*Insert*	*Update*	*RMW*	*Read*	*Scan*	*QEF*	*QRF*
A	-	50%	-	50%	-	-	-
B	-	5%	-	95%	-	-	-
E_1	5%	-	-	-	75%	10%	10%
E_2	5%	-	-	-	75%	20%	-
F	-	-	50%	50%	-	-	-
G	50%	-	15%	15%	-	10%	10%
H	10%	-	45%	30%	-	15%	-

Tabela 12: Proporção das operações YCSB em cada *workload*.

modificada, da qual resultaram as variantes E_1 e E_2. Na primeira, parte da proporção das operações *Scan* foi dividida com as operações de filtro *QEF* (filtro de igualdade de valores) e *QRF* (filtro de ordem de valores), que por sua vez serão aplicados sobre o *column qualifier Data* do esquema *Appointments*. Na segunda variante, parte da proporção da operação *Scan* é partilhada com a operação *QEF*, que por sua vez irá iterar sobre o *column qualifier MainID* do esquema *Patients*. Além disso, as *workloads G* e *H* foram especificamente criadas para esta avaliação, de forma a reproduzir um ambiente de computação e processamento típico para o caso de estudo médico definido.

A *workload G* pretende simular um ambiente de consultas médicas (esquema *Appointments*), contendo uma parte significativa de inserções (registo de novas consultas) e o restante processamento é dividido por atualizações (modificação do estado das consultas), leituras (verificação de consultas) e filtros (pesquisa das consultas do dia ou do mês). A *workload H* pretende reproduzir a tabela dos pacientes de um hospital (esquema *Patients*), em que 75% das operações são dedicadas à atualização e leitura das informações dos pacientes e a restante percentagem refere-se à inserção de novos pacientes e pesquisa pela *column family MainID*.

No que diz respeito às configurações dos esquemas definidos, para o esquema *Appointments*, o *column qualifier Data* foi populado com valores aleatórios compreendidos entre 2015 e 2020. Para o esquema *Patients*, o *column qualifier MainID* foi gerado aleatoriamente. Contudo, um subconjunto destes valores é guardado em memória a cada execução do YCSB, para garantir que as pesquisas sobre estes valores (*QEF*) não retornem resultados vazios. De referir que todos os testes foram executados com a distribuição *Zipfian*.

O desempenho do sistema *SafeNoSQL* é descrito nas figuras 22 e 23 e na tabela 13, através do débito e da latência para cada *workload* especificada. A linha vermelha com valor $y = 1$ corresponde ao valor normalizado das *workloads* em *plaintext*. Como era expectável, as *workloads* executadas sobre o esquema *Patients* apresentam na sua maioria um custo reduzido, já que não recorrem à *CryptoBox OPE* para proteger informação sensível. Em média, quando comparado com o *HBase* sem proteção, as *workloads* executadas sobre o esquema *Patients* apresentam um custo adicional de 12.29%, tendo como pior caso a *workload* E_2 com um

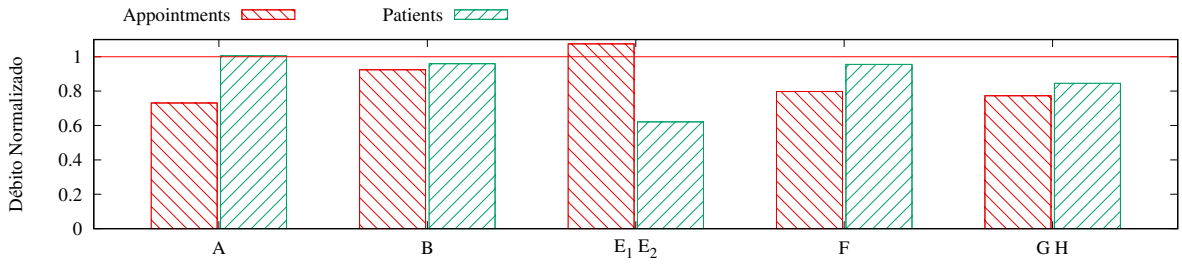


Figura 22: Débito normalizado para os macro testes.

custo adicional de 37.9%. Apesar das operações *QEF* apenas corresponderem a 20% da carga total da *workload* E_2, como se pode verificar na figura 18 e como descrito na secção 6.2.1, esta perda de desempenho deve-se ao tamanho do criptograma gerado (80 bytes) face ao tamanho do conteúdo original (64 bytes). Por outro lado, as *workloads* executadas sobre o esquema *Appointments* apresentam um custo adicional de 14.03%, tendo como pior caso a *workload A* com um custo adicional de aproximadamente 27%, maioritariamente devido ao desempenho da *CryptoBox OPE*.

Para as *workloads G* e *H*, quando comparadas com o *HBase* sem proteção, é observada uma perda de desempenho de 22.76% e 15.42%, respetivamente. Na *workload G*, uma parte considerável deste custo deve-se ao desempenho da cifragem da *CryptoBox OPE*, já que 50% da carga total deste teste são inserções. Na *workload H* (e E_2), para além do tamanho dos criptogramas gerados, parte da perda de desempenho deve-se ao facto do identificador de linha não estar protegido com uma técnica que preserva a sua ordem, levando a que para as operações *Scan* e *QEF* que começam a partir de uma linha específica, seja feita uma pesquisa completa sobre tabela (*full table scan*). Retornado o subconjunto de resultados ao cliente, estes serão decifrados e novamente filtrados de acordo com o identificador de linha inicial (*start row*).

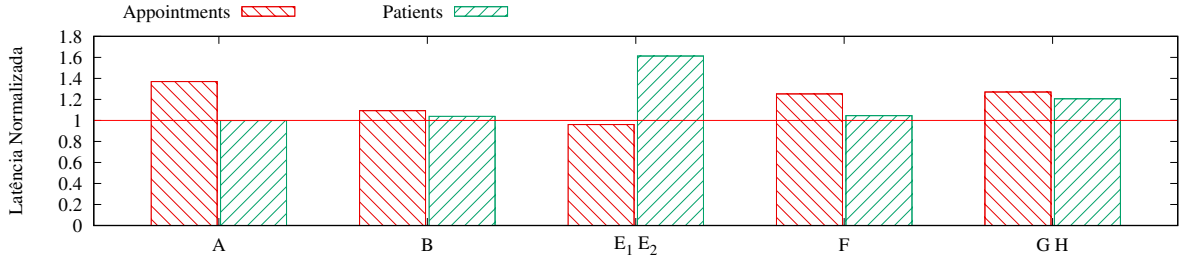


Figura 23: Latência normalizada macro testes.

De notar ainda que a *workload* E_1 (esquema *Appointments*) apresenta resultados interessantes, sendo que os valores do débito e da latência são similares aos resultados em *plaintext*. Isto acontece devido à otimização referida em 5.4, através do *column qualifier* adicional *Data-STD*, protegido com *Standard Encryption*. Desta forma, é possível efetuar computação

segura sobre o *column qualifier Data* e em simultâneo, obter custos de decifragem reduzidos devido ao desempenho da *CryptoBox STD*.

Esq.	Sistema	Métrica	A	B	E_1	E_2	F	G	H
AP	s/ Seg.	Déb. (ops/s)	400.36 ± 16.191	421.183 ± 11.920	6.370 ± 0.430	-	277.306 ± 9.689	8.567 ± 1.897	-
		Lat. (ms)	2.502 ± 0.097	2.377 ± 0.081	157.703 ± 10.578	-	3.10 ± 0.123	125.278 ± 39.02	
	SafeNoSQL	Déb. (ops/s)	292.396 ± 10.986	389.056 ± 41.218	6.843 ± 1.352	-	221.253 ± 3.725	6.617 ± 1.595	-
		Lat. (ms)	3.425 ± 0.127	2.597 ± 0.255	151.533 ± 27.342	-	4.521 ± 0.078	159.074 ± 33.503	-
PA	s/ Seg.	Déb. (ops/s)	331.014 ± 10.075	326.441 ± 23.309	-	0.089 ± 0.003	221.154 ± 11.286	-	0.066 ± 0.002
		Lat. (ms)	3.024 ± 0.092	3.078 ± 0.210	-	11,284.588 ±395.137	4.534 ± 0.232	-	15,219.327 ±533.208
	SafeNoSQL	Déb. (ops/s)	332.467 ± 23.298	313.042 ± 11.189	-	0.055 ± 0.003	211.308 ± 8.300	-	0.056 ± 0.008
		Lat. (ms)	3.022 ± 0.195	3.199 ± 0.114	-	18,196.153 ±925.209	4.740 ± 0.186	-	18,354.071 ±2 801.68

Esq. - Esquema de base de dados; AP - Esquema Appointments; PA - Esquema Patients; Déb. - Débito; Lat. - Latência.

Tabela 13: Resultados do débito e latência para os macro testes.

Os resultados dos macro testes mostram que a combinação de diferentes técnicas criptográficas é fundamental para suportar um domínio alargado de aplicações, que executam diferentes operações *NoSQL*, fornecendo um desempenho da base de dados praticável. Além disso, ainda há espaço para melhorias, o que justifica a importância de uma *framework* modular e flexível como o *SafeNoSQL*, permitindo a fácil integração de novas técnicas criptográficas com diferentes compromissos entre o desempenho da base de dados, funcionalidades suportadas e segurança da informação sensível.

6.3.1 *Análise da gestão dos recursos*

Com o objetivo de compreender em pormenor o comportamento do sistema *SafeNoSQL* face à versão sem qualquer proteção, foram retiradas algumas estatísticas dos recursos utilizados durante o processamento das várias *workloads*, com recurso à ferramenta *dstat* [88]. Esta ferramenta permite verificar em tempo real o comportamento dos recursos (*p.e. CPU*, memória, disco, largura de banda) do cliente e do *cluster*.

Como a implementação atual do *SafeNoSQL* não inclui técnicas criptográficas que recorram a *coprocessors*, o uso do *CPU* no *HBase Backend* assume um comportamento similar em ambos os casos, havendo apenas um acréscimo de 1.5 a 5% na versão segura. Relativamente à memória, enquanto houver espaço disponível, os blocos protegidos vão sendo guardados na *Memstore* e na *Block Cache*. Contudo, por apresentarem um maior comprimento, a base de dados efetua operações de *flush* dos dados para os *HFiles* com maior frequência. No que diz respeito à largura de banda, a informação transferida do cliente para o servidor (e vice-versa), há um acréscimo de aproximadamente 14% para o esquema *Appointments* e de 22%

para o esquema *Patients*, face à versão sem segurança. Este acréscimo vai de encontro com a diferença especificada dos tamanhos dos esquema, descritos na secção 6.1.2. De forma análoga, o mesmo se verifica relativamente à persistência dos dados.

No cliente de base de dados, a execução das *workloads* para o esquema *Appointments*, na versão sem proteção, levou ao uso do *CPU* entre os 0.8% e 2.3% e a memória usada rondou entre os 650 MB e os 1200 MB. Na versão protegida, o *CPU* correu entre 1.5% e 7.5% da sua capacidade, e a memória usada rondou os 815 e 1540 MB. Para o esquema *Patients*, na versão original, o *CPU* correu entre os 0.2% e 2.2% da sua capacidade e a memória utilizada variou entre os 562 MB e os 1065 MB. Na versão protegida, o *CPU* correu entre 0.3% e os 3.2%, e a memória usada rondou os 580 e 1100 MB.

6.4 AVALIAÇÃO COM MÚLTIPLOS CLIENTES

Depois de avaliados os micro e macro testes, procedemos à avaliação e análise do desempenho do sistema *SafeNoSQL* num ambiente de carga, com múltiplos clientes a operar sobre a base de dados em simultâneo. À semelhança dos macro testes (6.3), foram executadas as *workloads* apresentadas na tabela 12 para os esquemas *Appointments* e *Patients*. De forma a perceber o comportamento incremental do sistema, as *workloads* foram executadas para 2, 4, 8, 16, 32 e 64 clientes.

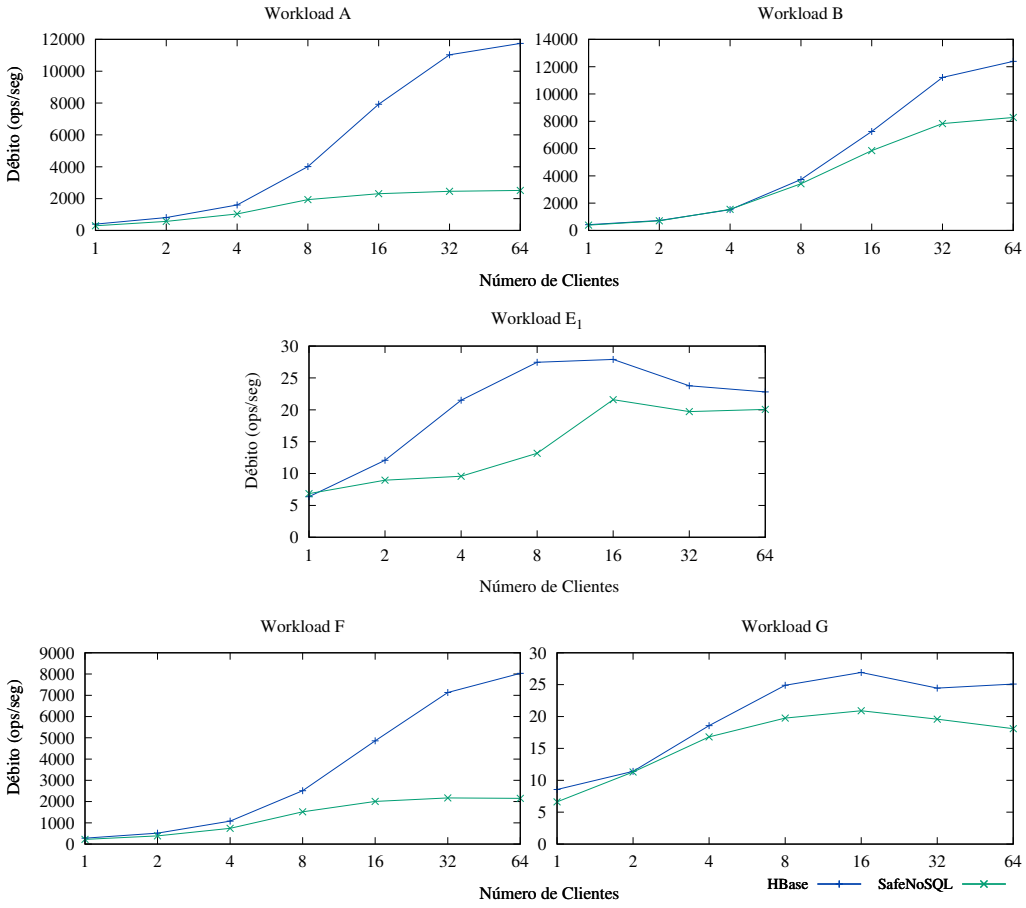


Figura 24: Débito das *workloads* do esquema *Appointments* com múltiplos clientes.

A figura 24 e a tabela 14 descrevem o desempenho do *SafeNoSQL* e do sistema sem qualquer proteção para o esquema *Appointments*, sendo apresentado o débito para cada *workload*. Como se pode verificar nas *workloads A*, *B* e *F*, o débito do sistema seguro escala sensivelmente até 32 clientes, começando a estabilizar a partir daí devido ao custo associado à cifragem dos valores protegidos com a *CryptoBox OPE*. Em maior detalhe, através do *dstat* foi possível verificar a carga de trabalho na máquina do cliente *SafeNoSQL*. Assumindo a

notação $[CPU_{HBase} - CPU_{SafeNoSQL}]$, sendo o valor CPU_{HBase} a percentagem de utilização do *CPU* sem operações seguras e o valor $CPU_{SafeNoSQL}$ a percentagem de utilização do *CPU* com operações seguras, para as *workloads A*, *B* e *F*, foram obtidos os pares [45% - 98%], [47% - 68%] e [44% - 92%], respetivamente. Esta diferença de carga do processador justifica a diferença de desempenho entre a versão *HBase* sem proteção e o sistema *SafeNoSQL*. Relativamente às *workloads* E_1 e *G*, estas são as que mais se aproximam da linha de desempenho do sistema sem operações seguras, sendo possível verificar que o ponto máximo do desempenho é com 16 clientes. Isto acontece devido à elevada sobrecarga das operações de pesquisa (*Scan*) e filtros (*QEF* e *QRF*) a ocorrer em simultâneo no *HBase Backend*, levando à exaustão dos recursos existentes. No que diz respeito aos recursos do cliente de base de dados, seguindo a mesma notação apresentada acima, as *workloads* E_1 e *G* apresentam uma utilização de *CPU* de [3% - 6%] e [5% - 8%], respetivamente. Estes valores revelam-se baixos na medida a maioria do tempo é gasto em operações mais demoradas no lado do servidor (*Scan* e *QEF*). Contudo, quando o valor destas operações é retornado, são registados picos relativos à carga do *CPU*, respetivamente [25% - 55%] e [20% - 98%].

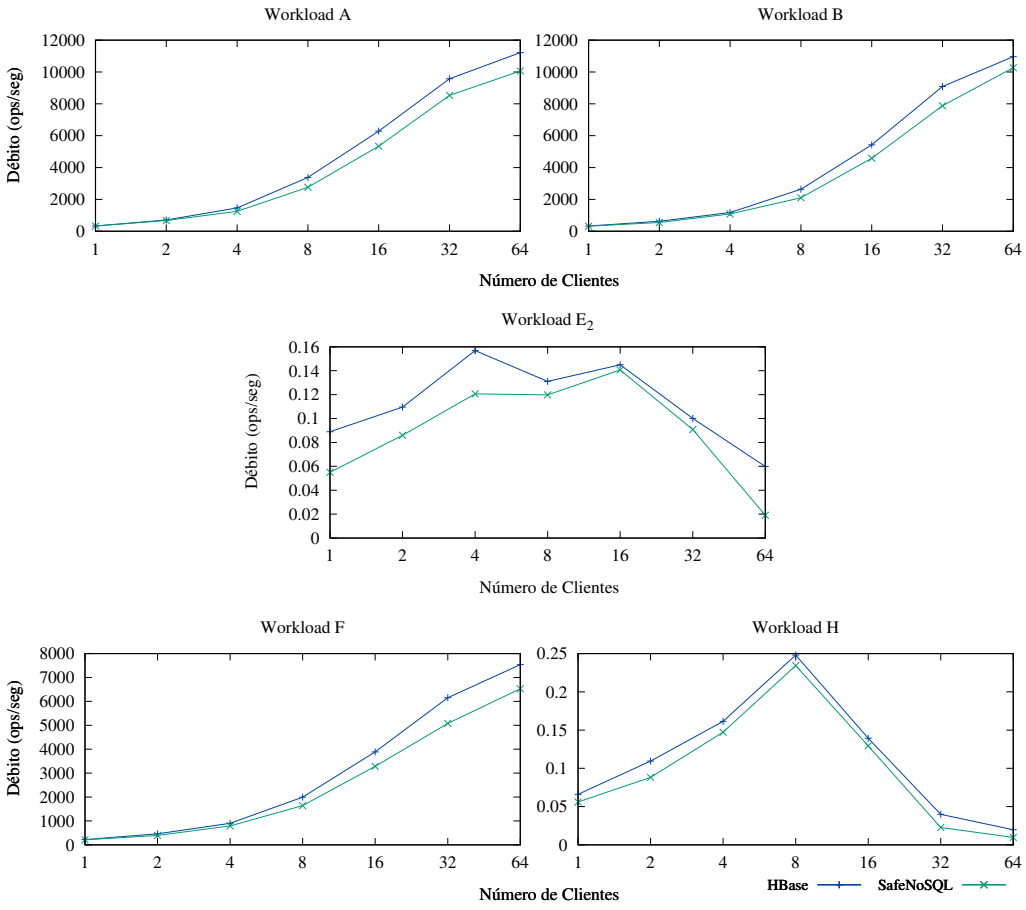


Figura 25: Débito das *workloads* do esquema *Patients* com múltiplos clientes.

As figuras 25 e a tabela 15 descrevem o desempenho do *SafeNoSQL* e do sistema sem qualquer proteção para o esquema *Patients*, sendo apresentado o débito para cada *workload*. Como se pode verificar em todas as *workloads*, o sistema *SafeNoSQL* segue a mesma linha de desempenho e apresenta um custo mínimo comparativamente às operações sem segurança. Este desempenho, deve-se ao uso de técnicas criptográficas eficientes como *STD* e *DET*, que apresentam operações de cifragem e decifragem com custo reduzido (6.2.1). Assumindo a notação $[CPU_{HBase} - CPU_{SafeNoSQL}]$, as *workloads A*, *B* e *F* apresentam os pares [43% - 46%], [49% - 51%] e [45% - 47%], respetivamente. À semelhança das *workloads* E_1 e *G*, as *workloads* E_2 e *H* apresentam um ponto máximo de desempenho de 16 e 8 clientes, respetivamente, devido à sobrecarga de operações de pesquisa (*Scan*) e filtro (*QEF*) a ocorrer em simultâneo, refletindo-se em cargas de processamento reduzidas na máquina do cliente de base de dados, respetivamente, [14% - 16%] e [24% - 30%].

Cli.	Sistema	A	B	E_1	F	G
1	s/ Seg.	400.36 ± 16.101	421.183 ± 11.92	6.37 ± 0.43	277.306 ± 9.689	8.567 ± 1.897
	SafeNoSQL	292.396 ± 10.986	389.056 ± 41.218	6.843 ± 1.352	221.253 ± 3.725	6.617 ± 1.595
2	s/ Seg.	808.825 ± 29.636	726.206 ± 19.583	12.082 ± 1.686	516.387 ± 7.754	11.407 ± 1.726
	SafeNoSQL	567.295 ± 39.939	706.082 ± 41.978	8.956 ± 0.853	384.612 ± 17.143	11.31 ± 0.095
4	s/ Seg.	1602.266 ± 76.022	1519.965 ± 61.194	21.502 ± 1.509	1079.824 ± 38.543	18.578 ± 1.168
	SafeNoSQL	1036.958 ± 32.425	1543.996 ± 18.067	9.58 ± 1.946	739.9761 ± 18.335	16.803 ± 0.599
8	s/ Seg.	4008.938 ± 29.504	3735.214 ± 120.697	27.455 ± 1.341	2510.843 ± 77.949	24.912 ± 1.273
	SafeNoSQL	1939.726 ± 63.028	3420.117 ± 277.341	13.176 ± 0.437	1518.523 ± 15.302	19.767 ± 0.855
16	s/ Seg.	7926.486 ± 122.909	7255.445 ± 242.258	27.9 ± 1.774	4858.565 ± 165.486	26.909 ± 0.701
	SafeNoSQL	2308.051 ± 86.0671	5850.221 ± 91.523	21.585 ± 0.661	2004.382 ± 39.823	20.908 ± 0.441
32	s/ Seg.	11029.609 ± 47.151	11210.867 ± 400.556	23.767 ± 0.935	7132.283 ± 205.201	24.456 ± 0.284
	SafeNoSQL	2460.986 ± 74.467	7831.497 ± 284.977	19.723 ± 1.306	2171.552 ± 90.255	19.594 ± 1.306
64	s/ Seg.	11746.377 ± 180.736	12388.908 ± 79.064	22.816 ± 0.186	8031.393 ± 67.374	25.086 ± 0.914
	SafeNoSQL	2510.723 ± 57.099	8277.213 ± 200.879	20.057 ± 0.162	2152.754 ± 60.221	18.095 ± 0.185

Cli. - Número de clientes; s/ Seg. - HBase sem garantias de segurança;
Cada célula representa o débito (ops/s) e o respetivo desvio padrão.

Tabela 14: Resultados do débito do esquema *Appointments* para os testes com múltiplos clientes.

À semelhança dos resultados obtidos nos micro e macro testes, os resultados desta avaliação, mostram mais uma vez que a combinação de diferentes técnicas criptográficas possibilita um balanceamento aceitável entre os compromissos de desempenho, funcionalidades permitidas e segurança assente sobre o sistema.

Cli.	Sistema	A	B	E_2	F	H
1	s/ Seg.	331.014 ± 10.075	326.441 ± 23.309	0.089 ± 0.003	221.154 ± 11.286	0.066 ± 0.002
	SafeNoSQL	332.467 ± 23.298	313.042 ± 11.189	0.055 ± 0.003	211.308 ± 8.301	0.056 ± 0.008
2	s/ Seg.	704.833 ± 42.051	628.421 ± 22.58	0.109 ± 0.0055	460.182 ± 13.041	0.109 ± 0.011
	SafeNoSQL	670.455 ± 34.249	548.534 ± 24.767	0.085 ± 0.0027	392.839 ± 10.672	0.088 ± 0.002
4	s/ Seg.	1465.552 ± 10.942	1172.758 ± 45.127	0.156 ± 0.018	902.595 ± 43.702	0.161 ± 0.002
	SafeNoSQL	1247.813 ± 34.289	1086.761 ± 59.431	0.121 ± 0.0025	792.279 ± 12.428	0.147 ± 0.0039
8	s/ Seg.	3380.42 ± 40.378	2635.322 ± 58.646	0.131 ± 0.0016	1993.575 ± 65.057	0.248 ± 0.0037
	SafeNoSQL	2763.881 ± 26.826	2104.088 ± 32.011	0.119 ± 0.0011	1642.741 ± 50.681	0.234 ± 0.0082
16	s/ Seg.	6281.83 ± 306.27	5427.03 ± 76.369	0.145 ± 0.068	3888.923 ± 184.83	0.139 ± 0.064
	SafeNoSQL	5333.951 ± 127.045	4580.804 ± 56.0322	0.141 ± 0.091	3283.524 ± 33.2076	0.129 ± 0.0018
32	s/ Seg.	9572.634 ± 213.794	9087.902 ± 404.998	0.101 ± 0.0102	6154.347 ± 145.503	0.039 ± 0.004
	SafeNoSQL	8530.385 ± 169.267	7881.957 ± 186.565	0.091 ± 0.0098	5076.693 ± 71.785	0.023 ± 0.019
64	s/ Seg.	11215.38 ± 643.414	10965.483 ± 157.019	0.059 ± 0.0008	7536.767 ± 45.919	0.019 ± 0.0011
	SafeNoSQL	10065.883 ± 132.458	10265.702 ± 97.516	0.019 ± 0.001	6525.267 ± 3.765	0.009 ± 0.0015

Cli. - Número de clientes; s/ Seg. - HBase sem garantias de segurança;
Cada célula representa o débito (ops/s) e o respetivo desvio padrão.

Tabela 15: Resultados do débito do esquema *Patients* para os testes com múltiplos clientes.

7

SUPORTE DE INTERROGAÇÕES SQL NO SAFENOSQL

O projeto europeu *SafeCloud* pretende reestruturar o paradigma de computação em nuvem ao garantir a segurança e privacidade do armazenamento, processamento e transmissão de informação sensível na nuvem [89]. Neste contexto, o projeto *SafeCloud* propõe a introdução de mecanismos de computação segura em bases de dados *NoSQL* e *SQL* que irão ser instaladas totalmente ou parcialmente em serviços de nuvem. Desta forma, o sistema *SafeNoSQL* vai de encontro com os objetivos deste projeto no ambiente não relacional, contudo surge a necessidade de assegurar a computação segura no modelo relacional que pode ser feita de duas formas: criar uma *framework* (como o *SafeNoSQL*) para bases de dados relacionais ou integrar um tradutor de interrogações *SQL* para *NoSQL* com o sistema *SafeNoSQL*. Assim, neste capítulo é feita uma análise, decrição da arquitetura e da implementação de um tradutor de interrogações *SQL* para *NoSQL* e da respetiva integração com o sistema *SafeNoSQL*.

7.1 VISÃO LÓGICA DO QUERY ENGINE

As bases de dados *NoSQL* do tipo *Key-Value Store* como *HBase*, *DynamoDB* ou *Cassandra* tornaram-se bastante atrativas para um conjunto vasto e diversificado de aplicações [70] [69][68]. Contudo, a falta de suporte da linguagem *SQL* por parte destas bases de dados representa um grande obstáculo para uma adoção ainda maior.

No ponto de vista desta dissertação, a *framework SafeNoSQL* rege-se a bases de dados não relacionais, excluíndo assim aplicações que utilizam o modelo relacional como base de dados. Assim, seguindo a mesma visão proposta no projeto *SafeCloud*, a solução passa por traduzir, de forma transparente, interrogações *SQL* em *NoSQL* através de um tradutor de interrogações (*Query Engine*), preservando assim as propriedades típicas das bases de dados *NoSQL* como a escalabilidade horizontal, *sharding* e a flexibilidade do esquema, mantendo ao mesmo tempo a expressividade e características de um DBMS relacional. A figura 26 descreve a visão abstrata de um sistema integrado com um tradutor de interrogações.

A arquitetura é composta por três módulos principais: a camada aplicacional onde correm aplicações que comunicam sobre uma API *SQL*, um *Query Engine* que transforma as

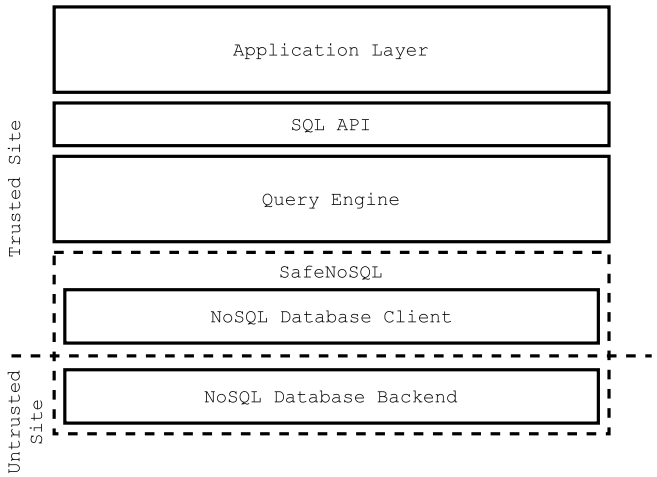


Figura 26: Visão abstrata de um sistema com *Query Engine*.

interrogações *SQL* da camada aplicacional em interrogações *NoSQL* para serem executadas sobre o terceiro módulo, a base de dados *NoSQL* (*NoSQL Database Backend*). Para além disso, o tradutor tem também como papel transformar os resultados obtidos da base de dados *NoSQL* em respostas do tipo *SQL*.

À semelhança dos sistemas de computação segura presentes no capítulo 3, estes módulos estão compreendidos em duas entidade, segura (*Trusted Site*) e não confiável (*Untrusted Site*). A entidade segura compreende a camada aplicacional, o *Query Engine* e o cliente de base de dados (*NoSQL Database Client*). De notar que estes componentes não requerem estado persistente, com excepção de alguma informação de configuração e de *caching* para optimizar o desempenho da base de dados. Desta forma, é possível instalar estes componentes em diversas máquinas obtendo assim uma solução escalável horizontalmente.

Na camada não confiável é compreendido o *NoSQL Database Backend*. Desta forma, a *framework SafeNoSQL* é aplicada sobre o cliente e o servidor de base de dados *NoSQL*, assegurando o armazenamento, processamento e transmissão de dados protegidos.

7.2 Implementação do SafeNoSQL sobre um Query Engine

Definida a arquitetura da solução, segue-se a integração do *SafeNoSQL* com o tradutor de interrogações *Distributed Query Engine (DQE)* [90]. Trata-se de um tradutor distribuído que permite fazer a tradução eficiente de interrogações *SQL* para *NoSQL*. Essa tradução passa por criar múltiplas tabelas de dados e de índice, na base de dados *NoSQL*, e efetuar múltiplas operações *NoSQL* por cada operação *SQL* instanciada, de forma a atualizar e retornar os resultados corretos sobre as múltiplas tabelas criadas.

Alavancando conhecimento de projeto prévios onde este tradutor foi produzido (*CumuloNimbo*, *CoherentPaaS* e *LeanBigData*) é possível integrar ambos os sistemas, DQE e *SafeNoSQL* para obter um modelo *SQL* seguro [91][92][93].

A implementação atual do DQE, está neste momento implementada sobre o motor *SQL* *Apache Derby* suportando como *backend NoSQL* o sistema *Apache HBase* [94]. Contudo devido à arquitetura modular do sistema, torna-se possível a integração de qualquer base de dados *SQL* e *NoSQL* (esta última aplica-se a bases de dados do tipo *Key-Value Store*).

O *Apache Derby* é uma base de dados relacional *open-source* desenvolvida pela *Apache Foundation* que permite ser embebida com aplicações *Java* e usada para processamento de transações *online*. Por sua vez, o motor da base de dados tem suporte absoluto sobre as API's de *Java Database Connectivity (JDBC)* e *SQL*.

De notar que a implementação atual não permite transações, não se tratando então de uma solução que assume as propriedades *ACID*. Futuramente esta limitação será colmatada com o incremento de uma camada transacional no *Query Engine*, com recurso a tecnologias que possibilitam transações *ACID* em bases de dados *NoSQL* como o *Apache Omid* [95].

7.3 Integração da solução com aplicações médicas

Um dos objetivos do projeto europeu *SafeCloud*, para além da implementação do sistema *SafeNoSQL* com o DQE, passa por integrar a mesma numa aplicação real da área da saúde pertencente a um dos parceiros do projeto, a *Maxdata Software, S. A.*. A *Maxdata Software, S. A.* é uma empresa que desenvolve e mantém software para a área da saúde, sendo o seu produto principal o CLINIdATA® (aplicação em causa). O software CLINIdATA® está presente em grande parte dos hospitais portugueses em várias áreas e laboratórios, nomeadamente, requisição eletrónica, patologia clínica, anatomia patológica, imunohemoterapia (banco de sangue e transfusões), vigilância epidemiológica (controlo de infeções hospitalares) e gestão de termos de responsabilidade. Por usa vez, esta aplicação armazena e processa milhares de dados sensíveis em *plaintext* numa base de dados *SQL* (*Derby*). Simultaneamente, a aplicação é usada em paralelo em várias clínicas de saúde, laboratórios e hospitais, levando ao armazenamento e processamento concorrente.

Desta forma, tal como especificado na figura 27, a aplicação CLINIdATA® assume o lugar do módulo aplicacional, comunicando transparentemente com o DQE através de um cliente de base de dados *Derby*. Por sua vez, o DQE traduz as interrogações *SQL* em operações *NoSQL* sem segurança, sendo enviadas o cliente de base de dados (*HBase*), onde o *SafeNoSQL* interceta e gera as respetivas operações seguras e processa-as sobre a base de dados (*HBase*).

De modo a avaliar a solução especificada foi feita uma avaliação qualitativa, através de uma *workload SQL* fornecida pela *Maxdata Software, S. A.*, que simula um ambiente típico

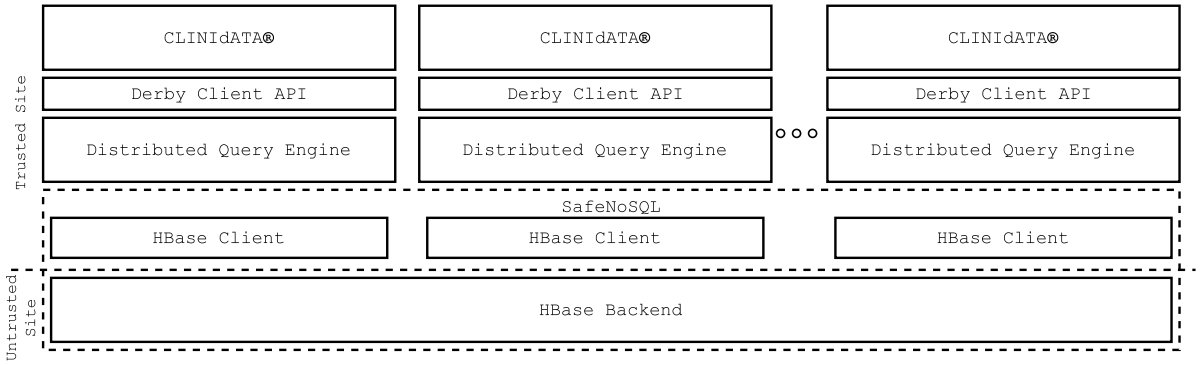


Figura 27: Modelo distribuído da aplicação CLINIdATA® sobre o sistema seguro.

da utilização do software CLINIdATA®. Esta *workload* é executada sobre o DQE e todas as operações sobre ele executadas são desconhecidas (assumindo o comportamento de uma *black box*), sendo apenas conhecidas as interrogações traduzidas e o esquema original da base de dados *SQL*, sendo este composto por 611 tabelas. Todas as operações executadas sobre o DQE são traduzidas e enviadas ao sistema *SafeNoSQL*. Por sua vez, o processo de tradução resulta na geração de 2 304 tabelas e 818 736 operações *NoSQL*, excluíndo as operações de administração da base de dados (*p.e.,* CREATE_TABLE, ALTER_TABLE, DROP_TABLE). Em detalhe, foram executadas 220 109 operações *Put*, 370 248 operações *Get* e 225 950 operações de pesquisa e filtros e 2 429 operações distribuídas pelas restantes operações descritas na secção 5.4.

Relativamente ao processamento e armazenamento seguro da informação, foram protegidas no total cinco colunas de três tabelas que dizem respeito à identidade de pacientes e dos respetivos resultados de exames e análises médicas. Com a *CryptoBox STD* foram protegidas as colunas que não envolvem processamento direto; com *DET* foram protegidas as colunas que necessitam de operações *Get* e *QEF*; com a *CryptoBox OPE* foram protegidas as colunas que necessitam de operações *QRF*. A operação original correspondente a estas operações é o *SQL Select*. De notar que estas colunas foram definidas num ficheiro de configuração de forma ao sistema *SafeNoSQL* efetuar o mapeamento da base de dados.

Desta integração consegue-se provar a fiabilidade e a integrabilidade de sistemas seguros e aplicações *SQL*. Como trabalho futuro seria importante avaliar a solução qualitativamente, percebendo o desempenho da mesma. Ainda, seria interessante fazer um *profiling* das operações realizadas de modo a analisar os pontos de contenção do sistema. Eventualmente, seria interessante proteger mais campos da base de dados e analisar o desempenho do sistema como um todo.

8

CONCLUSÃO

Com a realização desta dissertação, criou-se o sistema *SafeNoSQL*, um sistema de computação segura sobre bases de dados *NoSQL* que introduz uma nova solução para melhorar a privacidade dos dados armazenados em serviços de computação em nuvem, colmatando as falhas de privacidade e falta de segurança presentes nos atuais serviços. Para este efeito, a análise extensa do estado da arte atual dos sistemas de computação segura para bases *SQL* e *NoSQL* mostrou-se essencial. A análise arquitetural e das técnicas criptográficas assentes sobre estes sistemas, bem como o modelo de computação usado sobre as interrogações para otimizar a segurança e desempenho das bases de dados, permitiu concluir que a sua elevada estaticidade e fraca modularidade causam a contenção do desempenho da base de dados, tornando-as soluções monolíticas e de aplicabilidade real limitada. Ainda, esta análise permitiu também abstraír uma arquitetura genérica destes sistemas de computação segura e uma taxonomia que permite classificar as soluções atuais pelo tipo de base de dados, processamento e técnicas criptográficas utilizadas. Além desta análise, a revisão detalhada da literatura dos esquemas criptográficos mais frequentes e relevantes nos sistemas de computação segura atuais permitiu avaliar qual o ambiente a que cada esquema melhor se adequa face às garantias de segurança fornecidas e o tipo de características de cada cifra.

Com o conhecimento obtido do estudo realizado, foi desenhada a solução do sistema *SafeNoSQL*, sendo formalizada uma nova arquitetura modular e extensível sobre bases de dados *NoSQL* que resolve a elevada estaticidade e fraca modularidade presentes nos sistemas atuais. A genericidade do sistema *SafeNoSQL* permite a adaptatibilidade com bases de dados *NoSQL* do tipo *Key-Value Store*, enquanto que a modularidade e extensibilidade do sistema permitem agilizar a combinação de técnicas criptográficas a integrar no sistema, por fim a fornecer um sistema funcional e de segurança configurável. Ainda, foi implementado um protótipo do sistema *SafeNoSQL* sobre a base de dados *Apache HBase* que inclui quatro esquemas criptográficos, *Standard Encryption*, *Deterministic Encryption*, *Format-Preserving Encryption* e *Order-Preserving Encryption*, que fornecem garantias de segurança distintas e possibilitam o armazenamento, processamento e transmissão seguro de informação sensível. Sobre o sistema, foi ainda realizada uma avaliação experimental detalhada do protótipo do sistema *SafeNoSQL*. Esta avaliação contempla micro testes, macro testes e testes de carga

com múltiplos clientes. Com os micro testes foi possível concluir o desempenho do sistema num ambiente de teste mais controlado, permitindo avaliar o desempenho e comportamento dos vários esquemas criptográficos face a um conjunto de operações da ferramenta de análise de desempenho YCSB. Por sua vez, os macro testes permitiram avaliar a solução em ambientes mais realistas, através de um conjunto de *workloads* típico que contempla casos de estudo da área da saúde, sendo obtido um custo no desempenho da base de dados inferior a 15%. Os resultados obtidos desta avaliação experimental mostram que a combinação de diferentes técnicas criptográficas é fundamental para atingir uma solução funcional, com um balanceamento aceitável entre os compromissos de desempenho, funcionalidades permitidas e segurança assente sobre o sistema.

Finalmente, no âmbito do projeto *SafeCloud*, foi feita a integração do sistema *SafeNoSQL* com um componente de tradução de *SQL* para *NoSQL* que permitiu extender as garantias de segurança presentes neste sistema, a aplicações tradicionais que apenas suportam o modelo *SQL*. Como caso de estudo, a solução integrada foi validada com a aplicação da área da saúde CLINIdATA®, de forma a mostrar que é possível extender as garantias de segurança a aplicações SQL reais.

Para concluir, a privacidade dos dados de cada índividuo é um tema fundamental na sociedade atual. Desta forma é esperado que cada vez mais surjam novas técnicas criptográficas e esquemas de computação segura, o que justifica a importância de uma *framework* modular e flexível como o *SafeNoSQL*, permitindo a fácil integração de novas técnicas criptográficas com diferentes compromissos entre o desempenho da base de dados, funcionalidades suportadas e segurança da informação sensível.

8.1 TRABALHO FUTURO

Com esta dissertação, o sistema *SafeNoSQL* vem colmatar falhas de privacidade e segurança presentes nos serviços de nuvem e bases de dados atuais, através de uma nova solução de computação segura sobre bases de dados *NoSQL*. A plataforma tem ainda espaço para ser extendida com a implementação de novas técnicas criptográficas como *Searchable Encryption* e *Paillier Encryption*, que recorrem ao uso do *backend* da base de dados para processar informação protegida. Esta extensão revela-se interessante para analisar e avaliar novas soluções de computação segura, nomeadamente a forma como estas podem aumentar o desempenho e segurança da solução *SafeNoSQL* Ainda, a integração e avaliação com mais aplicações reais *SQL* e *NoSQL*, bem como a implementação do sistema *SafeNoSQL* sobre mais bases de dados, é impreterível para alavancar o paradigma de computação segura.

Como trabalho futuro numa outra vertente, esta solução pode ser extendida a ferramentas de processamento analítico como *Apache Spark* e *Hadoop MapReduce*, de forma a tirar partido do poder de processamento distribuído destes sistemas e simultaneamente fornecer privacidade e segurança sobre a informação sensível [96][97]. No estado da arte atual, os sistemas de processamento analítico de dados seguro, como o Opaque, fornecem garantias de segurança fortes com um custo no desempenho do sistema entre 1.6 a 46 vezes, sendo impraticável em grande parte dos sistemas de tempo real [98]. À semelhança dos sistemas de computação segura sobre bases de dados, a solução poderá passar por dotar estes sistemas com múltiplas técnicas criptográficas e avaliar os diferentes compromissos entre o desempenho do sistema, as funcionalidades suportadas e a segurança assente.

9

ANEXOS

9.1 PUBLICAÇÕES

R. Macedo, J. Paulo, R. Pontes, B. Portela, T. Oliveira, M. Matos e R. Oliveira, *"A Practical Framework for Privacy-Preserving NoSQL Databases"*, 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), Hong Kong, 2017.

BIBLIOGRAFIA

[1] "Brandwatch: Social media statistics and facts for 2016." [Online]. Available: https://www.brandwatch.com/blog/96-amazing-social-media-statistics-and-facts-for-2016/

[2] M. Chen, S. Mao, and Y. Liu., "Big data: A survey." 2014. [Online]. Available: http://link.springer.com/article/10.1007%2Fs11036-013-0489-0

[3] "Google cloud platform." [Online]. Available: https://cloud.google.com/compute/

[4] "Amazon web services." [Online]. Available: https://aws.amazon.com/

[5] "Sony pictures entertainment hack." 2014. [Online]. Available: https://www.washingtonpost.com/news/the-switch/wp/2014/12/18/the-sony-pictures-hack-explained

[6] "Dropbox hack," 2012. [Online]. Available: https://www.theguardian.com/technology/2016/aug/31/dropbox-hack-passwords-68m-data-breach

[7] "icloud leaks of celebrity photos." 2014. [Online]. Available: https://en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos

[8] B. JAMES, "Security and privacy challenges in cloud computing environments," 2010.

[9] "Apple-fbi encryption dispute," 2016. [Online]. Available: http://www.theverge.com/2016/3/28/11317396/apple-fbi-encryption-vacate-iphone-order-san-bernardino

[10] "Edward snowden: the whistleblower behind the nsa surveillance revelation," 2013. [Online]. Available: https://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance

[11] "Healthnet: two million accounts exposure," 2011. [Online]. Available: https://cdt.org/blog/hhs-should-require-the-encryption-of-portable-devices-to-curb-health-data-breaches/

[12] B. Fuller, M. Varia, A. Yerukhimovich, E. Shen, A. Hamlin, V. Gadepally, R. Shay, J. D. Mitchell, and R. K. Cunningham, "Sok: Cryptographically protected database search," *CoRR*, vol. abs/1703.02014, 2017.

[13] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 85–100.

[14] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 289–300.

[15] E. Pattuk, M. Kantarcioglu, V. Khadilkar, H. Ulusoy, and S. Mehrotra, "Bigsecret: A secure data management framework for key-value stores." in *IEEE CLOUD*, 2013, pp. 147–154.

[16] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2014.

[17] P. Rogaway and T. Shrimpton, "Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem," *IACR Cryptology ePrint Archive*, vol. 2006, p. 221, 2006.

[18] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill, "Order-preserving symmetric encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009, pp. 224–241.

[19] M. Bellare, P. Rogaway, A. Barroso, K. Bell, K. Bimpikis, D. Boswell, B. Buesker, M. Burton, C. Calabro, S. Davis, A. Gantman, B. Huffaker, H. M. Kang, V. Manpuria, C. Namprempre, A. Palacio, and W. Rao, "Introduction to modern cryptography," 2001.

[20] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," *IACR Cryptology ePrint Archive*, vol. 2006, p. 186, 2006.

[21] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 463–477.

[22] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Annual Cryptology Conference*. Springer, 2011, pp. 578–595.

[23] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[24] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[25] C. Shannon, "A mathematical theory of cryptography (no. 20878). new jersey: Bell labs," 1945.

[26] M. Dworkin, "Recommendation for block cipher modes of operation. methods and techniques," DTIC Document, Tech. Rep., 2001.

[27] J. Daemen and V. Rijmen, "The design of rijndael: Aes - the advanced encryption standard," in *Information Security and Cryptography*, 2002.

[28] W. Diffie and M. E. Hellman, "Special feature exhaustive cryptanalysis of the nbs data encryption standard," *Computer*, vol. 10, pp. 74–84, 1977.

[29] X. Lai and J. L. Massey, "A proposal for a new block encryption standard," in *EUROCRYPT*, 1990.

[30] R. L. Rivest, "The rc5 encryption algorithm," in *FSE*, 1994.

[31] B. Schneier, "Description of a new variable-length key, 64-bit block cipher (blowfish)," in *FSE*, 1993.

[32] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *Cryptographers' Track at the RSA Conference*. Springer, 2002, pp. 114–130.

[33] Z. Liu, C. Jia, J. Li, and X. Cheng, "Format-preserving encryption for datetime," in *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, vol. 2. IEEE, 2010, pp. 201–205.

[34] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004, pp. 563–574.

[35] P. Paillier, *Public-Key Cryptosystems Based on Composite Degree Residuosity Classes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238. [Online]. Available: http://dx.doi.org/10.1007/3-540-48910-X_16

[36] L. Seungmin, P. Tae-Jun, L. Donghyeok, N. Taekyong, and K. Sehun, "Chaotic order preserving encryption for efficient and secure queries on databases," *IEICE transactions on information and systems*, vol. 92, no. 11, pp. 2207–2217, 2009.

[37] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," *IACR Cryptology ePrint Archive*, vol. 2014, p. 834, 2014.

[38] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *ACM Conference on Computer and Communications Security*, 2016.

[39] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, "Practical order-revealing encryption with limited leakage," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1125, 2015.

[40] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches over encrypted data," in *IEEE S & P Symposium*, 2000.

[41] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009.

[42] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *EURASIP Journal on Information Security*, vol. 2007, no. 1, p. 013801, 2007.

[43] "Ibm touts encryption innovation: New technology performs calculations on encrypted data without decrypting it." [Online]. Available: http://www.computerworld.com/article/2526031/securityo/ibm-touts-encryption-innovation.html

[44] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes fv and yashe," *IACR Cryptology ePrint Archive*, vol. 2014, p. 62, 2014.

[45] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[46] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'o8. 23rd Annual Symposium on*. IEEE, 1982, pp. 160–164.

[47] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[48] R. Pontes, F. Maia, J. Paulo, and R. M. P. Vilaça, "Saferegions: Performance evaluation of multi-party protocols on hbase," in *SRDS Workshop*, 2016.

[49] E. F. Codd, "A relational model of data for large shared data banks (reprint)," *M.D. computing : computers in medical practice*, vol. 15 3, pp. 162–6, 1970.

[50] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *SIGMOD Conference*, 2017.

[51] "Mysql." [Online]. Available: https://www.mysql.com/

[52] M. Stonebraker, L. A. Rowe, and M. Hirohama, "The implementation of postgres," *IEEE Trans. Knowl. Data Eng.*, vol. 2, pp. 125–142, 1990.

[53] "Mariadb." [Online]. Available: https://mariadb.org/

[54] "Oracle database." [Online]. Available: https://www.oracle.com/database/index.html

[55] S. Halevi and P. Rogaway, "A tweakable enciphering mode," in *Annual International Cryptology Conference*. Springer, 2003, pp. 482–499.

[56] I. H. Akin and B. Sunar, "On the difficulty of securing web applications using cryptdb," in *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*. IEEE, 2014, pp. 745–752.

[57] 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*, vol. 79, pp. 18–26, 2015.

[58] R. Fagin, "Fuzzy queries in multimedia database systems," in *PODS*, 1998.

[59] Z. Liu, H. Ma, J. Li, C. Jia, J. Li, and K. Yuan, "Secure storage and fuzzy query over encrypted databases," in *NSS*, 2013.

[60] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1395–1406.

[61] S. Bajaj and R. Sion, "Trusteddb: a trusted hardware based database with privacy and data confidentiality," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 2011, pp. 205–216.

[62] "Ibm 4764 coprocessor." [Online]. Available: https://www.ibm.com/support/knowledgecenter/POWER7/p7hcd/fc4764.htm

[63] A. Arasu, S. Blanas, K. Eguro, M. Joglekar, R. Kaushik, D. Kossmann, R. Ramamurthy, P. Upadhyaya, and R. Venkatesan, "Engineering security and performance with cipherbase," 2012.

[64] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *HASP@ISCA*, 2013.

[65] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[66] J. Han, E. Haihong, G. Le, and J. Du, "Survey on nosql database," in *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. IEEE, 2011, pp. 363–366.

[67] R. Cattell, "Scalable sql and nosql data stores," *SIGMOD Record*, vol. 39, pp. 12–27, 2010.

[68] "Apache cassandra." [Online]. Available: http://cassandra.apache.org/

[69] "Amazon dynamodb." [Online]. Available: https://aws.amazon.com/dynamodb/

[70] "Apache hbase." [Online]. Available: https://hbase.apache.org/

[71] X. Yuan, X. Wang, C. Wang, C. Qian, and J. Lin, "Building an encrypted, distributed, and searchable key-value store," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM, 2016, pp. 547–558.

[72] X. Yuan, X. Wang, J. Lin, C. Wang, and C. Qian, "Blinddb: an encrypted, distributed, and searchable key-value store," 2016.

[73] R. Poddar, T. Boelter, and R. A. Popa, "Arx: A strongly encrypted database system," *IACR Cryptology ePrint Archive*, vol. 2016, p. 591, 2016.

[74] W. Zheng, F. 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*, ser. EuroSys '17. New York, NY, USA: ACM, 2017, pp. 191–204. [Online]. Available: http://doi.acm.org/10.1145/3064176.3064184

[75] 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 Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[76] "Hadoop distributed filesystem (hdfs)." [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[77] "Openssl." [Online]. Available: http://www.openssl.com/

[78] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "Mpfr: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, p. 13, 2007.

[79] V. Kachitvichyanukul and B. Schmeiser, "Computer generation of hypergeometric random variates†," *Journal of Statistical Computation and Simulation*, vol. 22, no. 2, pp. 127–145, 1985.

[80] M. Dworkin, "Recommendation for block cipher modes of operation: Methods for format-preserving encryption," 2013.

[81] M. Bellare, P. Rogaway, and T. Spies, "The ffx mode of operation for format-preserving encryption," 2010.

[82] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *ACM Conference on Computer and Communications Security*, 2015.

[83] M. Bellare, V. T. Hoang, and S. Tessaro, "Message-recovery attacks on feistel-based format preserving encryption," *IACR Cryptology ePrint Archive*, vol. 2016, p. 794, 2016.

[84] 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*. ACM, 2010, pp. 143–154.

[85] "Clinical database schema." [Online]. Available: https://dcm4che.atlassian.net/wiki/display/ee2/Database+Table+Descriptions

[86] "Eu general data protection regulation." [Online]. Available: http://www.eugdpr.org/

[87] R. Escriva, B. Wong, and E. G. Sirer, "Hyperdex: a distributed, searchable key-value store," in *SIGCOMM*, 2012.

[88] "Dstat - versatile tool for generating systems resource statistics." [Online]. Available: https://linux.die.net/man/1/dstat

[89] "Safecloud project." [Online]. Available: http://www.safecloud-project.eu/

[90] R. Vilaça, F. Cruz, J. Pereira, and R. Oliveira, "An effective scalable sql engine for nosql databases," in *Distributed Applications and Interoperable Systems: 13th IFIP WG 6.1 International Conference, DAIS 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, 2013, pp. 155–168.

[91] "Cumulonimbo." [Online]. Available: http://www.cumulonimbo.eu/

[92] "Coherentpaas." [Online]. Available: http://coherentpaas.eu/

[93] "Lean big data." [Online]. Available: http://leanbigdata.eu/

[94] "Apache derby." [Online]. Available: https://db.apache.org/derby/

[95] "Apache omid." [Online]. Available: https://omid.incubator.apache.org/

[96] "Apache spark." [Online]. Available: https://spark.apache.org/

[97] "Hadoop mapreduce." [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[98] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *NSDI*, 2017.

Este projeto foi financiado pela European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, sobre o acordo de subvenção No. 653884 (SafeCloud).