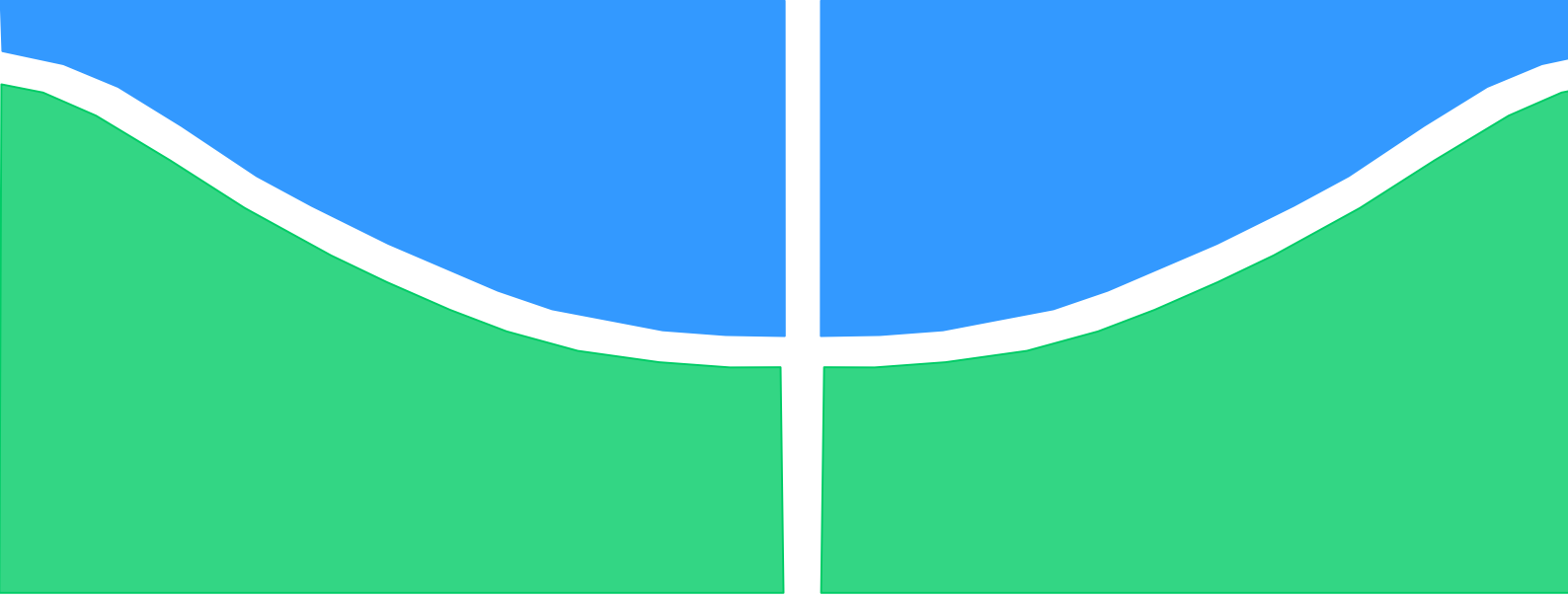

Universidade de Brasília - UnB
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

ARMAZENAMENTO DISTRIBUÍDO DE DADOS SEGUROS COM ESQUEMA DE CONSULTAS DIRETAS

Autor: Renata Elisa Medeiros Jordão
Orientador: Flávio Elias Gomes de Deus

Brasília, DF
2014

RENATA ELISA MEDEIROS JORDÃO

ARMAZENAMENTO DISTRIBUÍDO DE DADOS SEGUROS COM ESQUEMA DE CONSULTAS DIRETAS

Trabalho de Conclusão de Curso submetido ao curso de graduação em Engenharia de Redes de Comunicação da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Orientador: Prof. Dr. Flávio Elias Gomes de Deus

Brasília, DF
2014

CIP – Catalogação Internacional da Publicação

Medeiros Jordão, Renata Elisa.
Armazenamento distribuído de dados seguros com esquema de consultas diretas / Renata Elisa Medeiros Jordão. Brasília: UnB, 2014. Xii, 71 p., 210 x 297 mm

Trabalho de Conclusão de Curso (Graduação)
Universidade de Brasília,
Faculdade de Tecnologia.
Departamento de Engenharia Elétrica.
Orientação: Prof. Dr. Flávio Elias Gomes de Deus

1. Distribuído. 2. Dados seguros. 3. Consulta I. Gomes de Deus, Flávio Elias. II. Doutor.

CDU Classificação

REGULAMENTO E NORMA PARA REDAÇÃO DE RELATÓRIOS DE PROJETOS DE GRADUAÇÃO DEPARTAMENTO DE ENGENHARIA ELÉTRICA, UNIVERSIDADE DE BRASÍLIA-UNB

Renata Elisa Medeiros Jordão

Trabalho de Conclusão de Curso submetido ao departamento de Engenharia Elétrica da Faculdade de Tecnologia da Universidade de Brasília como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação, em 15/08/2014 apresentado e aprovado pela banca examinadora abaixo assinada:

FLAVIO ELIAS GOMES DE DEUS, Doutor, UnB
Orientador

FÁBIO MESQUITA BUIATI
Membro Convidado

VALÉRIO AYMORÉ MARTINS
Membro Convidado

Brasília, DF
2014

AGRADECIMENTOS

Ao meu orientador Prof. Dr. Flávio Elias Gomes de Deus, pelo apoio, incentivo, essenciais para o desenvolvimento deste trabalho e para o meu desenvolvimento como profissional.

Ao Prof. Valério Aymoré Martins e Fábio Mesquita Buiati pelo apoio, incentivo, dedicação e amizade. Pelas discussões sobre o tema e compartilhamento do grande conhecimento que possuem. Pela enorme ajuda e disponibilidade que demonstraram e também foram vitais para a realização deste trabalho.

RESUMO

O roubo de dados confidenciais é um quesito de extrema relevância para os dias atuais. Na maioria das aplicações, os dados confidenciais são armazenados em servidores. Assim, os sistemas existentes naturalmente tentam impedir os adversários de comprometer esses servidores por meio de controles de acesso. No entanto, estudos têm demonstrado que os adversários ainda assim tentam encontrar uma maneira para invadir e roubar os dados. Com o intuito de impedir a invasão e descoberta de dados sigilosos, uma nova abordagem é apresentada para proteger a confidencialidade dos dados mesmo quando os atacantes tenham acesso a todos os dados do servidor: a construção de sistemas práticos e seguros que realizem consultas e inserções em dados criptografados sem acesso a chave de decodificação. Este trabalho propõe um modelo de implantação que permita um meio de acesso comprovadamente seguro, tanto na atualização quanto na consulta a dados, com perda de eficiência tolerada em suas transações, isto é, com menor interferência em seu desempenho possível. O modelo previsto é aplicado em um cenário real: um sistema distribuído hospedado pelo Google (Encrypted BigQuery), que abarca o uso de criptografia em dados que são armazenados de forma distribuída em bancos de dados não-relacionais desenvolvido para o tratamento massivo de dados. Como resultado, é mostrado que estes sistemas suportam uma variedade de aplicações com baixo custo operacional.

Palavras-chave: Confidencialidade, consulta dados criptografados, Encrypted BigQuery, desempenho.

ABSTRACT

Theft of confidential information is a question of extreme relevance to the present day. In most applications, sensitive data are stored on servers. Thus, existing systems naturally try to prevent adversaries from compromising these servers through access controls. However, studies have shown that adversaries still find a way to break in and steal data. In order to prevent the invasion and discovery of sensitive data, a new approach is presented to protecting the confidentiality of data even when the attackers have access to all data from the server: building practical and reliable systems that perform queries and inserts on encrypted data without the decryption key. This work proposes a deployment model that provide for a proven safe means of access, both in the update query as the data, with efficiency loss tolerated in their transactions, ie, with as less interference on your performance as possible. The model set is applied in a real scenario: a distributed system hosted by Google (Encrypted BigQuery), which includes the use of encryption in data stored in non-relational databases developed for massive data processing. As a result, it is shown that these systems support a variety of applications with low overhead.

Keywords: Confidenciality, query on encrypted data, Encrypted BigQuery, performance.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

3DES	*Triple Data Encryption Standard*
ABE	*Attributed Based Encryption*
ACID	Atomicidade, Correspondência, Isolamento e Durabilidade
AES	*Advanced Encryption Standard*
BASE	*Basically Available, Soft-State, Eventually consistent*
CAP	*Consistency, Avaiability, Partition tolerance*
CBC	*Cipher Blocking Chaining*
CFB	*Cipher Feedback Mode*
DER	Diagramas de Entidade-Relacionamento
DES	*Data Encryption Standard*
EBQ	*Encrypted BigQuery*
ECB	*Eletronic Code Book*
FPGA	*Field-Programmable Gate Array*
HDFS	*Hadoop Distributed File System*
IND-CCA	Indistinguishability a-priori Chosen-Cipher text Attack
IND-CCA2	Indistinguishability posteriori Chosen-Cipher text Attack
IND-CPA	*Indistinguishability under Chosen Plain text Attack*
JSON	*JavaScript Object Notation* - Notação de Objetos JavaScript

MD4	*Message Digest Algorithm*
MD5	*Message Digest Algorithm* RDA-MD5
NoSQL	*Not Only SQL*
OFB	*Output Feedback Mode*
ORDMS	*Object-Relacional Database Management System*
SGBD	Sistemas Gerenciadores de Banco de Dados
SHA-1	*Secure Hash Algorithm*
SQL	*Structured Query Language*

SUMÁRIO

1. INTRODUÇÃO

Segundo Le Coadic (2004), dados são uma representação composta de valores com significado e codificados de uma forma a permitir colocá-los sob processamento eletrônico. Esse conjunto de dados implícitos é geralmente agrupado na forma de coleções que tem uma lógica e coerência, visto que uma organização de dados randômica não pode ser corretamente interpretada.

Essas coleções são chamadas de bancos de dados e surgem da necessidade de relacionar dados com origem e volume variados, de forma que haja certa organização e o tratamento seja facilitado. Assim, um banco de dados é projetado, construído e povoado por dados, atendendo a uma proposta específica.

Para atender a sua proposta de uso, os bancos de dados possuem grupos de usuários de interesse definido, e são associados a aplicações pré-concebidas de acordo com esse interesse distinto. Neste sentido, um banco de dados é um conjunto de dados integrados que tem por objetivo atender a uma comunidade de usuários (Heuser, 2001), e estão presentes em várias áreas tais como comércio, mercado financeiro e uso militar, tornando-se essenciais para o funcionamento, coordenação e controle de uma organização.

A segurança desses dados armazenados e das informações transmitidas no seu acesso através de qualquer sistema é extremamente importante, pois é ela que garante que o sistema conseguirá fazer aquilo pelo qual foi projetado, de maneira correta, e disponibilizada apenas para usuários autorizados. Muitas informações mantidas são sensíveis e sigilosas, portanto, essa proteção é fundamental, e consiste em atender aspectos de confiabilidade, integridade, disponibilidade, autenticidade e irretratabilidade. Na transmissão de informações, a criptografia de canal vem sendo o mecanismo mais utilizado e, dependendo do sigilo imposto às chaves utilizadas, é eficiente e eficaz.

Porém, a segurança no armazenamento dos dados herda as mesmas dificuldades que a segurança da informação transmitida e ainda enfrenta dificuldades adicionais, pois deve fornecer mecanismos que auxiliem o SGBD na tarefa de cumprir os aspectos fundamentais comprometendo ao mínimo seu tempo de resposta. A preocupação com a criação e manutenção de ambientes seguros se tornou a ocupação principal de administradores de redes, de sistemas operacionais e de bancos de dados. Esta preocupação é reforçada pois estudos comprovam que a

maioria dos ataques relatados, de roubos de informações e acessos não autorizados, são feitos diretamente a origem dos dados (Unisys, 2014), isto é, junto aos seus bancos de dados, ou por pessoas pertencentes a organização ou pela falta de mecanismos que evitem a leitura direta das bases de dados remotamente por terceiros. Por esse motivo, as organizações se esforçam para criar e usar artifícios com a finalidade de eliminar os acessos não-autorizados ou diminuir as chances de sucesso das tentativas de invasão (internas ou externas).

É então proposta deste trabalho a implementação de armazenamento distribuído de dados seguros para ambientes aonde haja risco de acesso direto ao dado ou ao uso do SGBD por outros acessos não autorizados.

1.1. PROBLEMA

A segurança prevista em bancos de dados é focada principalmente em controles de acesso que buscam certificar que todos os acessos diretos ao sistema ocorram exclusivamente de acordo com modalidades e regras pré-estabelecidas, observadas por políticas de proteção ao dado. No que diz respeito a confidencialidade, a criptografia é a ferramenta mais utilizada para a proteção dos campos.

Em vista disso, pretende-se propor um modelo de implantação que permita um meio de acesso comprovadamente seguro, tanto na atualização quanto na consulta a dados, com perda de eficiência tolerada em suas transações, isto é, com menor interferência em seu desempenho possível.

1.2. HIPÓTESE

Apresentar um sistema que provenha confidencialidade prática e comprovada perante certos ataques, usando uma coleção de modelos de criptografia que atuem individualmente de forma personalizada nos campos e possibilite a consulta eficaz sem o vazamento de dados. Porém, o uso dos algoritmos pode aumentar o tamanho dos dados armazenados, o que por sua vez, exigem o aumento do tamanho dos campos do banco de dados. Além disso, a criptografia dentro de um banco de dados pode retardar pesquisas, devido a computação adicional necessária para decifrar o conteúdo dos campos cifrados cada vez que os dados são recuperados. Assim, é ponto focal desse trabalho avaliar esse impacto.

1.3. OBJETIVOS

Dar garantia de confidencialidade e integridade aos dados hospedados em bancos de dados a partir de modelos criptográficos ajustáveis aos campos das tabelas que permitam operações de consultas com dados cifrados, com um mínimo prejuízo adicional no desempenho da recuperação de dados.

1.4. METODOLOGIA

A metodologia do projeto tem por base uma pesquisa bibliográfica visando o aprofundamento dos conceitos referentes a bancos de dados massivos e as técnicas aplicadas de armazenamento distribuído – tal como o Hadoop e o uso do *framework* MapReduce para escrever e processar de forma escalável aplicações distribuídas em grandes volumes de dados. Adicionalmente, serão estudados os princípios de aplicação de mecanismos de segurança em dados e do uso de esquemas criptográficos no armazenamento e acesso a dados. Em seguida, por pesquisa referencial, identifica-se as soluções teóricas utilizáveis e as possibilidades de aplicação de modelos criptográficos sobre os diferentes campos de dados de acordo com o tipo de consulta a ser feita posteriormente.

O modelo previsto é aplicado em um cenário real: um sistema distribuído hospedado pelo Google (*Encrypted BigQuery*) para reescrever as consultas realizadas de maneira a acessar os dados cifrados sem permitir diretamente que o acesso de usuários ao servidor exponha seu real conteúdo.

1.5. ORGANIZAÇÃO DO TRABALHO

O capítulo 2 objetiva uma revisão dos conceitos principais abordados, incluindo principalmente banco de dados massivos e os modos de execução e tecnologias referentes a Hadoop e MapReduce, que esclarecerão o funcionamento da ferramenta escolhida, bem como os princípios de segurança no armazenamento e na transmissão de dados e informações. No capítulo 3 é apresentada a ferramenta EBQ (*Encrypted BigQuery)*, como um sistema para a realização de consultas similares ao SQL sobre um banco de dados criptografado, juntamente com o conjunto de esquemas criptográficos usados nas consultas ajustáveis.

O capítulo 4 apresenta o modelo proposto, a implementação realizada, os resultados obtidos, as análises efetuadas durante os diversos cenários e as limitações do trabalho realizado. O capítulo final apresenta as conclusões obtidas e aponta trabalhos futuros que podem ser desenvolvidos tendo por base este trabalho

2. CONCEITUAÇÃO E ESTADO DA ARTE

Este capítulo tem como foco a revisão dos principais conceitos relativo a banco de dados; sistemas de armazenamento massivo de dados; sistema de processamento em aplicações distribuídas, como a tecnologia Hadoop/MapReduce; e a segurança aplicada nessas tecnologias. Com o intuito de fornecer um cenário amplo capaz de abranger o tema e ao mesmo tempo, ser possível a identificação e separação dos conceitos e assuntos correlatos, foi realizada uma divisão dos assuntos em tópicos específicos.

2.1. BANCO DE DADOS

Define-se banco de dados como um conjunto de dados integrados que correspondem a um determinado domínio (Elmasri & Navathe, 2005). O conteúdo presente em um banco de dados representa uma visão imediata do estado do mundo real, onde cada mudança em um item do banco de dados reflete uma mudança ocorrida na realidade. Da mesma forma, uma base de dados é projetada e instanciada com dados que seguem uma finalidade e são acessados por um grupo de usuários e/ou aplicações.

Porém, no armazenamento e acesso aos dados nessas bases, diversos problemas estão associados, sendo os principais:

- Garantia de integridade de dados, isto é, que um dado informado seja armazenado e acessado de forma consistente;
- Capacidade de acesso concorrente, isto é, que hajam mecanismos para suportar o acesso simultâneo e a atualização de um mesmo dado;
- A existência de um controle de redundância: para impedir que um mesmo dado seja registrado diversas vezes quando não for desejado e justificável;
- Suporte à padronização de forma a permitir a implementação de regras e restrições sobre os dados que serão armazenados ou recuperados.

Neste sentido, são disponibilizados os Sistemas Gerenciadores de Banco de Dados - SGBDs. O SGBD é um sistema constituído por um conjunto de dados associados e um conjunto de programas de acesso e manipulação desses dados. O principal objetivo de um SGBD é proporcionar um ambiente tanto conveniente quanto

eficiente para a recuperação e armazenamento das informações do banco de dados perante diversos níveis de acesso.

Um Sistema de Banco de Dados é um ambiente de *hardware* e *software,* projetado para gerir grandes volumes de dados, constituído de dados armazenados em um banco de dados, pelo SGBD e as aplicações, conforme Figura 2.1.

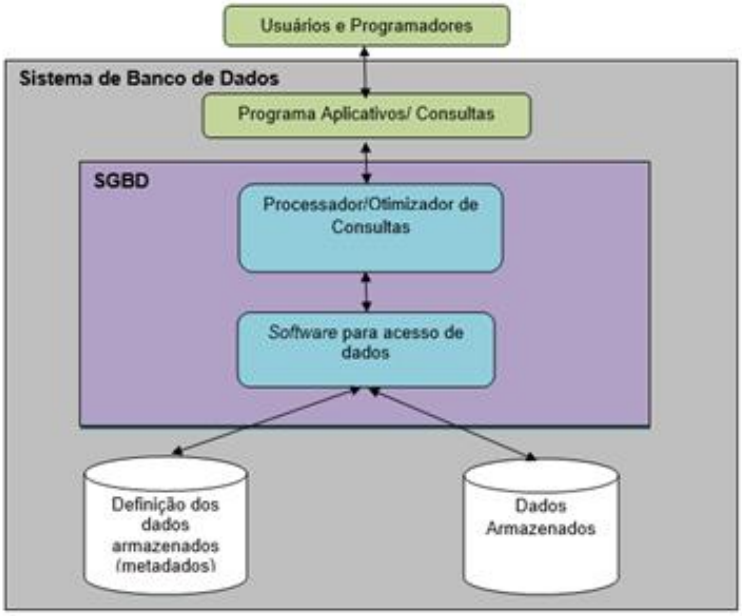

Figura 2.1: Sistema de Banco de Dados.

Os SGBDs são uma coleção de programas que permitem aos usuários criar e manter uma coleção de dados de forma que se dê garantia de integridades, concorrência de acesso, controle de redundância, suporte a padronização, entre outros fenômenos que surgem do uso de banco de dados por múltiplos usuários. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações (Elmasri & Navathe, 2005). Pelo acesso ao SGBD, o usuário não acessa diretamente o recurso de armazenamento de dados mas realiza requisições ao SGBD, que implementa essas requisições junto aos dados armazenados.

O primeiro SGBD comercial surgiu no final de 1960 com base nos primitivos sistemas de arquivos disponíveis na época, os quais não controlavam o acesso concorrente por vários usuários ou processos (Takai *et al.,* 2005). Os SGBDs precursores se utilizam de estruturas baseadas em modelos hierárquicos ou em

modelos em forma de rede para armazenamento dos dados. Cood (1970) apresentou um artigo que estabeleceu os princípios para implementação de um modelo eficiente visando a organização do armazenamento e padronização do acesso a dados, que se tornou a referência na implementação e disponibilização dos SGBDs atuais: o Modelo Relacional

Porém, com o crescente volume de informações trocadas, bem como o avanço da tecnologia, a computação distribuída tornou-se essencial no processamento e armazenamento de informações. O aumento na quantidade e complexidade de serviços oferecidos na Web levou a uma quantidade massiva de dados. Para esse processamento massivo de dados, o desempenho e a disponibilidade são fatores críticos que precisam ser avaliados, pois mecanismos convencionais de gerenciamento de dados não oferecem o suporte adequado (Goldman *et al.*, 2012). Dessa forma, o sistema de banco de dados também teve de se adaptar e buscar ferramentas que suportassem a quantidade crescente de dados e otimizassem o desempenho das aplicações, tendo como principal impacto a revisão do uso dos princípios oriundos do modelo relacional proposto por Codd.

2.1.1. Arquitetura de SGBDs

Takai *et al.* (2005) classificam a apresentação de um SGBD de diversas formas: plataformas centralizadas, sistemas de computador pessoal, banco de dados cliente-servidor e banco de dados distribuídos.

Nas plataformas centralizadas, há uma estrutura de grande porte com ampla capacidade de processamento, o qual hospeda o SGBD e emuladores para as aplicações. Tem como vantagem permitir que muitos usuários manipulem um grande volume de dados, porém exige alto custo devido ao ambiente especial para os *mainframes* e soluções centralizadas.

Na arquitetura de sistemas de computador pessoal, os computadores trabalham em modo *stand-alone*, ou seja, fazem os seus processamentos sozinhos. Com a evolução dos equipamentos, um computador que antes possuía uma capacidade de processamento limitada, hoje pode realizar as operações sem perda de eficiência. Assim, no que diz respeito a SGBDs, funcionam como hospedeiros e terminais. Sua principal vantagem é a simplicidade.

No modo cliente-servidor, o cliente executa as tarefas do aplicativo, ou seja, fornece a interface do usuário - tela, e processamento de entrada e saída. O servidor executa as consultas no SGBD e retorna os resultados ao cliente. A principal vantagem desta arquitetura é a divisão do processamento entre dois sistemas, o que reduz o tráfego de dados na rede.

A arquitetura distribuída funciona com cada servidor atuando como no sistema cliente-servidor, entretanto as consultas dos aplicativos são feitas para qualquer servidor indistintamente. Caso a informação solicitada seja mantida por outro servidor ou servidores, o sistema encarrega-se de obter a informação necessária, de maneira transparente para o aplicativo, que passa a atuar consultando a rede, independente de conhecer seus servidores. É utilizada quando há grande volume de dados e a necessidade de distribuí-los em diversos servidores. Neste sistema há também diversas aplicações que consultam a rede para acessar os dados necessários, porém sem o conhecimento explícito de quais servidores dispõem desses dados.

A Figura 2.2 apresenta as formas de arquitetura largamente utilizadas hoje em dia e descritas anteriormente.

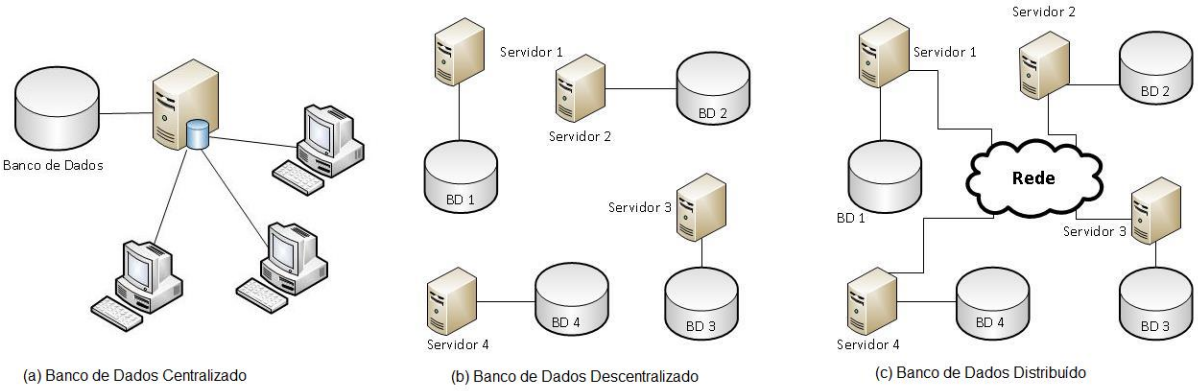

Figura 2.2: Arquiteturas de SGBDs.

2.1.2. Modelo de Base de Dados

Modelos de dados são descrições da estrutura presente em um banco de dados. Com base na estrutura, pode-se avaliar os tipos de dados e quais relacionamentos e restrições os mesmos estão sujeitos. Segundo Takai *et al.* (2005), pode-se classificar os modelos de dados baseando-se nos tipos de conceitos que fornecem para descrever a estrutura da base de dados.

Modelos de Dados Conceituais ou de Alto-Nível fornecem conceitos próximos à percepção dos usuários. Já os Modelos de Dados Físicos ou de Baixo-Nível fornecem conceitos que descrevem os detalhes de como os dados são armazenados no computador (Elmasri & Navathe, 2005). Entre esses dois extremos está uma classe de Modelos de Dados Representacionais (ou de implementação), que oferece os conceitos que podem ser entendidos pelos usuários finais, mas não excessivamente distantes da forma como os dados estão organizados dentro do computador.

Um Modelo de Dados Conceitual utiliza conceitos como entidade, atributos e relacionamentos. Uma entidade é um objeto que é representado na base de dados. Um atributo é uma propriedade que descreve algum aspecto de um objeto, podem ser classificados em simples ou compostos, uni ou multivalorados (Elmasri & Navathe, 2005). Os relacionamentos indicam as associações entre as entidades, bem como quantas entidades participam (grau de relacionamento) e quantas associações são feitas (grau de cardinalidade). O uso de Diagramas de Entidade-Relacionamento (DER) facilita o processo de projeto de um banco de dados.

Por ser um modelo intermediário, o Modelo de Dados Representacionais oculta alguns detalhes de armazenamento de dados, porém, podem ser implementados em um sistema de computador de maneira direta, agrupados no tempo segundo marcos de forte mudança nos seus principais paradigmas. São eles:

2.1.2.1. Modelos de Pré-Relacionais

- Hierárquico (Figura 2.3): modelo pioneiro no projeto de banco de dados. Os dados são organizados em hierarquias ou árvores, sendo cada nó da árvore uma coleção de atributos. O registro da hierarquia que precede a outros é o registro-pai, os outros são chamados de registros-filhos. Uma ligação é uma associação que é feita apenas entre dois registros. O relacionamento entre um registro-pai e vários registros-filhos possui cardinalidade 1:N (Takai *et al*., 2005). Os dados organizados segundo este modelo podem ser acessados segundo uma sequência hierárquica com uma navegação *top-down.*

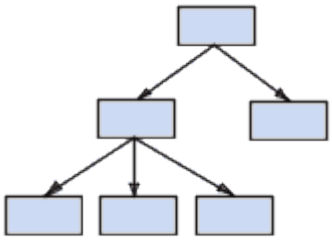

Figura 2.3: Modelo de dados Hierárquico (Os modelos de SGBD).

- Rede (Figura 2.4): surgiu como extensão ao modelo hierárquico, retirando o conceito de hierarquia e permitindo que haja várias associações entre dois registros. Tem a sua estrutura em forma de grafos. Ao contrário do Modelo Hierárquico, em que qualquer acesso aos dados passa pela raiz, o modelo em rede possibilita acesso a qualquer nó da rede sem passar pela raiz. Para esses dois modelos, qualquer acesso à base de dados – inserção, consulta, alteração ou remoção – é feito em um registro de cada vez (Takai *et* al., 2005).

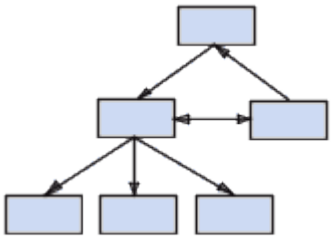

Figura 2.4: Modelo dados em Rede (Os modelos de SGBD)**.**

2.1.2.2. Modelo Relacional

O modelo relacional (Figura 2.5) surgiu devido às necessidades de aumentar a independência de dados nos sistemas gerenciadores de banco de dados, provendo um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados (Takai *et* al., 2005). Tendo por base a teoria dos conjuntos e a álgebra relacional, é organizado em forma de tabelas. Na nomenclatura do modelo relacional formal, uma linha é chamada tupla, um cabeçalho de coluna é um atributo e a tabela é conhecida como relação (Elmasri & Natathe, 2005). Cada tupla representa uma coleção de valores de dados relacionados que correspondem a uma entidade ou relacionamento do mundo real.

O objetivo do modelo é representar os dados de forma mais simples, através de um de conjuntos de tabelas inter-relacionadas. Este modelo abandona os conceitos anteriores, tornando os bancos de dados mais flexíveis, tanto na forma de representar as relações entre os dados, como na tarefa de modificação de sua estrutura, sem ter que reconstruir todo o banco de dados. Tem como vantagem sobre seus antecessores a representação simples dos dados e a facilidade com que consultas complexas podem ser expressas.

Os primeiros produtos relacionais começaram a aparecer no final da década de 1970. Hoje, a maioria dos sistemas de banco de dados é relacional: IBM: DB2; Microsoft: SQL Server; Oracle: 9i, 10g, 11g MySqlAB: MySql; Open Source: PostgreSQL; Sybase Server (Sybase). A principal linguagem de criação, consultas e manipulação de dados em sistemas de bancos de dados relacionais é o SQL (*Structured Query Language*).

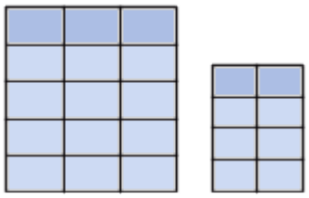

Figura 2.5: Modelo de dados Relacionais (Os modelos de SGBD).

2.1.2.3. Modelos Pós-Relacional

- Modelo Orientado a Objetos (Figura 2.6): Surgiu em meados de 1980 para armazenamento de dados complexos, não adequados aos sistemas relacionais. Foram inspirados nas práticas de programação orientada a objetos e incorporaram algumas de suas funcionalidades como encapsulamento de operações, herança de objetos e registros de dados abstratos, já difundidos em linguagens de programação como o SmallTalk e o C++. Assim, os dados são armazenados sob a forma de objetos e só podem ser manipulados por métodos definidos nas classes de que pertencem esses objetos (Takai *et* al., 2005). Os objetos podem conter referências para outros objetos, e as aplicações podem acessar os dados requeridos usando um estilo de navegação que contenha um conjunto de linguagem de programação orientada a objetos. A filosofia do modelo

de dados orientado a objetos consiste em agrupar os dados e o código que manipula estes dados em um único objeto, estruturando-os de forma que possam ser agrupados em classes. Isso significa que os objetos de banco de dados agrupados podem usar o mesmo mecanismo de herança para definir superclasses e subclasses de objetos, criando assim hierarquias.

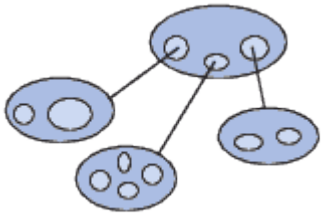

Figura 2.6: Modelo de dados Orientado a Objetos (Os modelos de SGBD).

- Sistemas Objetos-Relacionais: Fornecedores de bancos de dados relacionais adicionaram a seus produtos capacidade de incorporar objetos mais complexos (imagem, som e vídeo) além de recursos de orientação a objetos. No entanto, isso não os torna sistemas puramente orientados a objetos, apesar de sua denominação ORDMS – *Object-Relacional Database Management System* (Sistema de Gerenciamento de Banco de Dados Objeto-Relacional). Esses sistemas na realidade implementam uma camada de abstração de dados em cima de métodos relacionais, o que torna possível a manipulação de dados mais complexos. Seguem, portanto, as especificações da SQL3 que fornecem capacidades estendidas e de objetos adicionadas ao padrão SQL. Todas as características relacionais permanecem, ou seja, as tabelas continuam a existir, porém elas possuem alguns recursos adicionais.

2.1.2.4. Modelos de Armazenamento NoSQL

A Web tem revolucionado a forma como criamos conteúdo e trocamos informações, além de habilitar o desenvolvimento de uma série de novas aplicações. O grande volume de dados gerado por aplicações Web, juntamente com os requisitos diferenciados destas aplicações, como a escalabilidade sob demanda e o elevado grau de disponibilidade, têm contribuído para o surgimento de novos paradigmas e tecnologias.

Neste contexto uma nova categoria de Banco de Dados, chamada NoSQL *(Not Only SQL)*, foi proposta com o objetivo de atender aos requisitos de gerenciamento de grandes volumes de dados, semiestruturados ou não estruturados. São baseados em conceitos de Banco de Dados Paralelos e/ou Distribuídos e aplicam em sua essência uma quebra do paradigma ACID (Cattel, 2011). Tiveram como base de seu desenvolvimento as necessidades e especificações vindas de trabalhos da Amazon e Google. Hoje, os modelos de armazenadores NoSQL apresentam uma sistemática que os agrupam em quatro tipos básicos, a saber:

- Chave-Valor (*Key-Value Stores*): Modelo simples que permite a visualização do banco de dados como uma grande tabela hash. Exs.: Dynamo DB (Amazon), Voldemor, Berkeley DB, Hadoop, Scalaris e Redis.
- Orientado a Colunas (*BigTable-style Databases*): "Banco de dados" que teve como referência o modelo BigTable do Google, e que dentre suas características podemos destacar o particionamento dos dados com forte consistência dos mesmos. Exs.: Cassandra, Google BigTable e Hyperbase.
- Orientado a Documentos (*Document Databases*): Armazena coleções de documentos, ou seja, objetos com identificadores únicos e um conjunto de campos *(strings*, lista ou documentos aninhados), assemelhando ao modelo chave-valor, porém cada documento tem um conjunto de campos-chaves e os valores destes campos. Exs.: CouchDB e o MongoDB.
- Orientado a Grafos (*Graph Databases*): Possui três componentes básicos (nós, relacionamentos, propriedades) que permitem que o SGBD possa ser visto como um multigrafo rotulado e direcionado, onde cada par de nós pode ser conectado por mais de uma aresta. Exs.: Neo4j, o InfoGrid, o Sones e o HyperGraphDB.

Com a crescente necessidade de inserir, consultar ou atualizar os dados de maneiras diferentes, muitos SGBDs utilizam modelos híbridos que herdam os aspectos mais vantajosos de diferentes modelos de dados, como a fácil estrutura de consulta baseada em SQL e a indexação de dados complexos do modelo orientado a objetos.

2.1.3. ACID

As propriedades ACID são importantes conceitos quanto a transações em banco de dados. Tais propriedades definem se um sistema de banco de dados é confiável ou não. Correspondem a atomicidade, consistência, isolamento e durabilidade (Elmasri & Navathe, 2005).

- **Atomicidade**: cada transação é tratada como atômica, ou seja, deve ser executada por completo. Se uma parte da transação falhar, toda transação falhará. É responsabilidade do subsistema de restauração de transações do SGBD garantir a atomicidade. Qualquer ação que constitui falha na unidade de trabalho, a transação deve ser desfeita (*rollback*). Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada (*commit*).
- **Consistência**: uma transação preservará a consistência se a sua execução completa fizer o banco de dados passar de um estado consistente para o outro, isto é, garante que todos os dados serão escritos no banco de dados. Define-se por Elmasri & Navathe (2005) que um estado do banco de dados é a coleção de todos os itens de dados armazenados (valores) no banco de dados em um dado momento. Um estado consistente do banco de dados satisfaz as restrições especificadas no esquema, bem como quaisquer outras restrições que devam controlar o banco de dados.
- **Isolamento**: permite que duas ou mais pessoas acessem uma mesma base de dados ao mesmo tempo e o sistema executa o controle para que um acesso não interfira no outro. Uma transação deve ser executada como se estivesse isolada das demais. Significa que, a execução de uma transação não deve sofrer interferência de quaisquer outras transações concorrentes.
- **Durabilidade**: as mudanças aplicadas ao banco de dados por uma transição efetivada (*commit*) devem permanecer no banco de dados. Essas mudanças não devem ser perdidas em razão de uma falha. A durabilidade é garantida através de *backups* e logs de transações que facilitam a restauração das transações cometidas, frente aos problemas de *hardware* ou *software* que possam ocorrer.

2.1.4. Modelo NoSQL

O modelo NoSQL, termo utilizado por primeira vez em julho de 2009, em uma conferência de novos modelos de banco de dados, não possui uma definição clara segundo os próprios organizadores da conferência. Entre as características apresentadas pelos bancos de dados NoSQL, estão:

- **Não-uso do SQL**: alguns dos bancos de dados NoSQL possuem suas próprias linguagens, que normalmente são bem semelhantes ao SQL, com o objetivo de facilitar o aprendizado.
- ***Open-source***: geralmente os bancos NoSQL são de código aberto.
- **Implementação em *clusters***: a grande maioria (mas não todos) dos bancos NoSQL são desenhados para rodar em *clusters*, o que afeta diretamente a propriedade de consistência dos dados, já que o modelo de dados adotado pelos bancos NoSQL não incluem as propriedades ACID.
- **Sem esquema**: normalmente operam sem um esquema, permitindo que o administrador adicione campos ao banco de dados sem se preocupar com mudanças na estrutura (esquema).

Uma das características marcantes desses modelos é implementar um modelo diferenciado de consistência, em geral, a consistência eventual (Cattel, 2011). A consistência eventual é uma característica de bancos NoSQL relacionada ao fato de que consistência nem sempre pode ser mantida entre os diversos pontos de distribuição de dados. Esta característica tem como princípio o teorema CAP (Brewer, 2012), que diz que, em um dado momento, só é possível garantir duas de três propriedades entre consistência, disponibilidade e tolerância à partição. Como no contexto de seus usos, os armazenadores NoSQL geralmente privilegiam a disponibilidade e a tolerância à partição, as propriedades ACID de consistência não podem ser obedecidas simultaneamente. Para esse novo modelo, tem-se outro conjunto de projetos denominado BASE (Basicamente disponível, Estado leve e Consistente em momento indeterminado) (Cattel, 2011). Neste sentido, sistemas que utilizam modelo de armazenamento NoSQL toleram inconsistências temporárias a fim de poder priorizar disponibilidade.

2.2. ARMAZENAMENTO MASSIVO DE DADOS PARA SISTEMAS DISTRIBUÍDOS

Diariamente sistemas corporativos, serviços e sistemas Web, mídias sociais, entre outros, produzem juntos um volume impressionante de dados. O armazenamento desses dados é feito, em sua maioria, de maneira não estruturada, utilizando sistemas, linguagens e formatos distintos, que muitas vezes não são compatíveis entre si.

Caracterizando essa tendência de geração de grandes volumes de dados pelas aplicações surgiu o conceito denominado “Big Data”, que trata não só do volume de dados, mas também quanto a variedade de estruturas e formatos e velocidade de processamento (*Big Data Definition*). Ferramentas que tratam Big Data utilizam a computação para criar soluções capazes de analisar grandes bases de dados, processar seus pesados cálculos, identificar comportamentos e disponibilizar serviços especializados em seus domínios (Goldman *et al.,* 2012). Como o processamento de problemas complexos em grande volume de dados durariam muito tempo (em torno de horas ou dias) a computação paralela e distribuída veio como alternativas para atenuar este gasto de tempo.

A computação distribuída é constituída de *clusters* e grades computacionais funcionando em um conjunto de computadores comuns, os quais juntos, agregam alto poder de processamento a um custo relativamente baixo. Trabalham de forma a dividir uma ação ou tarefa em subtarefas, executando-as paralelamente em diversas máquinas.

Baseados nessas arquiteturas e suprindo as necessidades do contexto apresentado, foram desenvolvidas diversas abordagem para o armazenamento massivo de dados, entre elas, a plataforma Hadoop, que encontra-se consolidada tanto no ambiente empresarial quanto acadêmico.

2.2.1. Sistemas de Arquivos Distribuídos Hadoop

A plataforma de desenvolvimento Hadoop foi desenhada para resolver problemas onde há uma grande quantidade de dados – talvez uma mistura de dados complexos e estruturados. É em situações nas quais se deseja fazer uma análise profunda e computacionalmente extensa, como *clustering* e *targeting*.

Hadoop foi desenvolvido pela Apache Software Foundation para ser executado em um grande número de máquinas que não compartem memória nem discos. O

HDFS - *Hadoop Distributed File System* é o sistema de arquivos distribuído do Hadoop. Quando os dados são carregados usando o *framework*, são divididos em blocos que depois se propagam através de diferentes servidores. Não há um lugar específico onde acessar todos os dados, Hadoop realiza um segmento para saber onde residem os dados. E devido ao grande número de repositórios de cópia, os dados armazenados em um servidor, que porventura, se desconecta ou morre, podem ser reproduzidos de forma automática a partir de uma cópia conhecida.

Em um sistema de banco de dados centralizado, um disco grande é conectado a vários processadores (de 4 a 16 processadores, por exemplo), mas esta é toda a potência que pode chegar. Em um *cluster* Hadoop, cada um destes servidores possui dois, quatro ou oito CPUs. É possível executar o trabalho de indexação mediante o envio de seu código a cada uma das dezenas de servidores no *cluster* e cada servidor funciona na sua própria porção de dados. Os resultados se entregam de novo a um todo unificado. Isto é MapReduce: atribuir a operação a todos os servidores e depois reduzir os resultados de novo em um único conjunto de resultados (White, 2012).

Arquitetonicamente, a razão pela qual é capaz de fazer frente a uma grande quantidade de dados é que Hadoop se expande. E a razão pela qual é capaz de fazer perguntas computacionalmente complicadas é porque tem todos estes processadores, que trabalham em paralelo, aproveitados em conjunto. O mecanismo responsável por essa computação em paralelo é o MapReduce, visto a seguir.

2.2.2. Framework para processamento de dados massivos: Map/Reduce

O Map/Reduce é um *framework* computacional para processamento paralelo criado pelo Google. Ele abstrai as dificuldades do trabalho com dados distribuídos, de modo que cada nó é independente e autossuficiente, eliminando quaisquer problemas que o compartilhamento de informações pode trazer (Apache Hadoop).
Consiste das seguintes funções: (Goldman *et al.*, 2012):

- *Map*: Recebe uma lista como entrada, aplica uma função e gera uma nova lista como saída. Particularmente no uso em conjunto do Hadoop com técnicas de Map/Reduce, as funções *Map* utilizam os blocos do arquivo armazenado como entrada. Podem ser, por exemplo, uma linha em um arquivo de log ou de uma tabela. Os blocos podem ser processados em paralelo em diversas máquinas do

aglomerado. Como saída, as funções *Map* produzem, normalmente, outros pares chave/valor.

- *Shuffle*: A etapa de *shuffle* é responsável por organizar o retorno da função *Map*, atribuindo para a entrada de cada *Reduce* todos os valores associados a uma mesma chave. Esta etapa é realizada pela biblioteca do MapReduce.
- *Reduce*: Recebe o resultado da função *Map* como entrada, aplica uma função para que a entrada seja reduzida a um único valor na saída. No Hadoop as funções *Reduce* são responsáveis por fornecer o resultado final da execução de uma aplicação, juntando os resultados produzidos por funções *Map*, ou seja, os conjuntos de valores associados a uma chave são agrupados em lista e consumidos, retornando uma nova lista de pares chaves/valor.

Um trabalho (*job*) pode ser considerado como uma execução completa de um programa MapReduce incluindo todas as suas fases: *Map, Shuffle e Reduce*. Uma tarefa (*task*) é definida como a execução de uma função (*Map* ou *Reduce*) dentro do *framework.*

O processamento tem três fases: uma fase inicial de mapeamento, onde são executadas diversas tarefas *Map*; uma fase intermediária onde os dados são recolhidos das funções *Map*, agrupados e disponibilizados para as tarefas de *Reduce*; e uma fase de redução onde são executadas diversas tarefas *Reduce*, para agrupar os valores comuns e gerar a saída da aplicação.

2.2.2.1. Componentes

Possui uma arquitetura *master-slave:*

- JobTracker: possui uma função de gerenciamento da execução das tarefas a serem processadas pelo MapReduce. Escolhe diferente nós para processar as tarefas de uma aplicação e monitorá-las enquanto estiverem em execução. Um dos objetivos do monitoramento é, em caso de falha, identificar e reiniciar uma tarefa no mesmo nó ou, em caso de necessidade, em um nó diferente.
- TaskTracker: executa as tarefas MapReduce. Cada instância de TaskTracker está localizada em um nó-escravo. Executa uma tarefa *Map* ou *Reduce* designada pelo JobTracker.

É possível observar na Figura 2.7 a integração do mecanismo MapRaduce e seus componentes juntamente com o HDFS, composto por NameNode e DataNodes, que operam como o nó principal de gerenciamento e os nós que tratam do armazenamento de blocos, respectivamente.

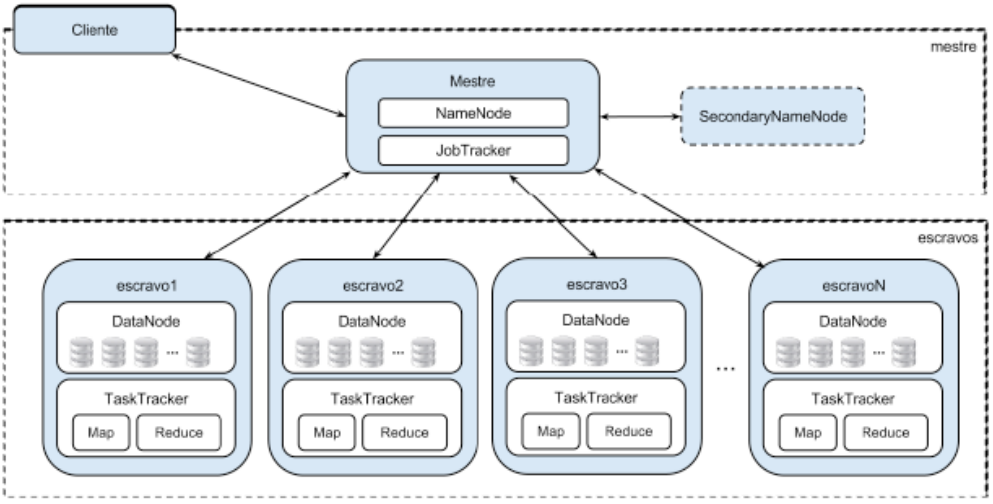

Figura 2.7: Processos Hadoop (Goldman *et al.,* 2012).

2.3. SISTEMAS DE ARMAZENAMENTO E RECUPERAÇÃO MASSIVA DE DADOS

Pode-se classificar os sistemas de armazenamento de dados pela forma como os dados serão guardados no disco. Podem ser orientados por linhas (*row-store)* ou por colunas (*column-store*). Os sistemas de armazenamento orientado por linhas armazenam e recuperam linhas inteiras do disco, enquanto que, nos sistemas de armazenamento colunar, cada coluna é armazenada de forma contigua e separada no disco. Para este trabalho, serão analisados os modelos de armazenamento orientados por colunas.

Apesar da ideia de particionar verticalmente as tabelas de uma base de dados para melhorar a performance ter surgido na década de 70, somente nos anos 2000 foram iniciadas as pesquisas acerca do tema. MonetDB (Boncz *et al.,* 2005) e C-Store foram sistemas pioneiros projetados com base de dados orientada por colunas e execução de consultas vetorizadas. Basicamente, em um banco de dados *column-store*, os valores armazenados em colunas são compactados e comprimidos para melhorar a eficiência na leitura (Abadi *et al.*, 2008). A operação de compressão é baseada no uso de algoritmos que identificam atributos de valores similares e os

agrupam, diminuindo assim, o espaço em disco gasto por uma base de dados. Dessa forma, suas operações são mais rápidas que banco de dados tradicionais por apresentarem maior eficiência E/S para consultas de apenas que solicitam leitura.

Um dos exemplos mais importantes de base de dados orientado por colunas é o Google BigTable, um sistema de armazenamento distribuído criado pelo Google com o objetivo de lidar com grandes volumes de dados estruturados gerados por projetos e serviços da organização. O BigTable pode ser definido como um mapa distribuído, multidimensional, ordenado e persistente (Chang *et al.*, 2008). Cada valor do mapa é indexado usando uma chave de linha, uma chave de coluna e uma marcação de tempo.

A chave de linha é usada para manter a ordem dos dados, com um intervalo de linhas, denominado *tablet*, dividido dinamicamente. Assim, a leitura de pequenos intervalos de linhas é eficiente, pois requer comunicação com um número pequeno de máquinas. As chaves de colunas são agrupadas em conjuntos chamados de famílias de colunas, que formam a unidade básica de controle de acesso. Todos os dados armazenados em famílias de colunas são geralmente do mesmo tipo e passam por algoritmos de compressão para atingir alta capacidade. O uso de marcações de tempo garante a cada célula do BigTable a possibilidade de conter múltiplas versões.

Depois de sua implementação, o BigTable inspirou diversas organizações a criarem modelos derivados. Inclui-se o Apache HBase, feito para operar conjuntamente com o HDFS; o Cassandra, utilizado pelo Facebook e o Hypertable, tecnologia livre que serve de alternativa ao HBase. Ainda, aproveitando a ferramenta poderosa que foi desenvolvida e a necessidade de consultar rapidamente dados armazenados nela, o Google projetou Dremel, que é detalhado abaixo.

2.3.1. BigQuery/Dremel

BigQuery é a implementação externa da ferramenta Dremel, usada internamente pelo Google para serviço de consulta em grandes conjuntos de dados. Surgiu da necessidade de uma ferramenta que pudesse lidar com consultas e retornasse resultados rápidos devido ao montante de dados que o Google tem que lidar diariamente.

A vantagem de desempenho do BigQuery vem da sua arquitetura de processamento paralelo. A consulta, de estrutura semelhante ao SQL, é processada

por milhares de servidores em uma estrutura de árvore de execução de vários níveis, com os resultados finais agregados na raiz. O BigQuery armazena os dados em um formato de colunas, de modo que apenas os dados das colunas que estão sendo consultadas são lidos. Essa arquitetura tem ganhos significativos de desempenho de E/S sobre o modelo tradicional de banco de dados relacional, pois não é lido todo o registro para cada consulta.

2.3.1.1. Arquitetura

A combinação de um armazenamento em colunas com uma estrutura de árvore dá ao Google BigQuery o seu ganho de performance. BigQuery armazena os dados em formato colunar, que funciona separando um registro em valores colunares e guardando cada valor em locais diferentes, enquanto um banco de dados normalmente armazena o registro inteiro em um local. A Figura 2.8 apresenta a ideia principal: todos os valores de um campo aninhado como A.B.C, que são armazenados de forma contigua. Dessa forma, A.B.C pode ser recuperado sem acessar A.E ou A.B.D.

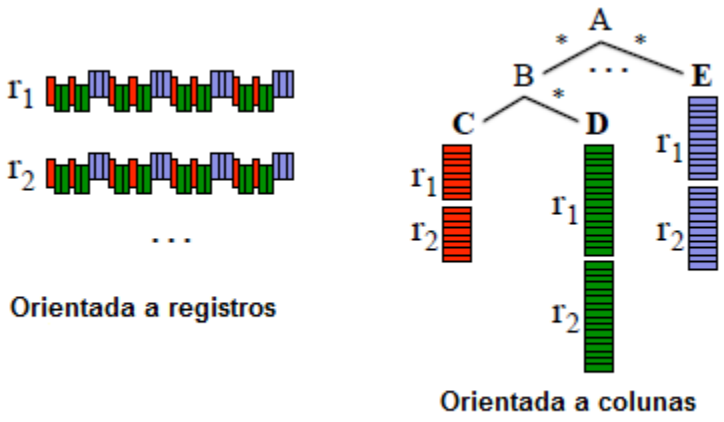

Figura 2.8: Armazenagem colunar BigQuery/Dremel (Adaptação de Melnik et. al, 2010).

Usar um armazenamento colunar representa uma diminuição no tráfego, uma vez que, ao fazer uma pesquisa, apenas a coluna apropriada será lida, sem o processamento desnecessário das outras colunas. Além disso, garante uma alta taxa de compressão, pois cada coluna pode ter valores semelhantes, especialmente se a cardinalidade da coluna (variação de possíveis valores da coluna) é baixa, é mais fácil ganhar taxas de compressão mais elevadas do que o armazenamento baseado em

linhas (Sato, 2012). Apresenta como desvantagem a deficiência na atualização dos campos, por isso BigQuery não admite a atualização de registros.

Um dos desafios enfrentados pelo Google na concepção do BigQuery/Dremel foi a forma de enviar consultas e recolher resultados em dezenas de milhares de máquinas em questão de segundos. O desafio foi resolvido usando a arquitetura em árvore (Sato, 2012). Utiliza-se uma árvore de serviço de vários níveis que recebe as consultas, lê os metadados das tabelas e encaminha a consulta para o próximo nível da árvore. Os servidores no final da árvore comunicam-se com a camada de armazenamento e acessam os dados, agregando os resultados ao final do processo, conforme Figura 2.9.

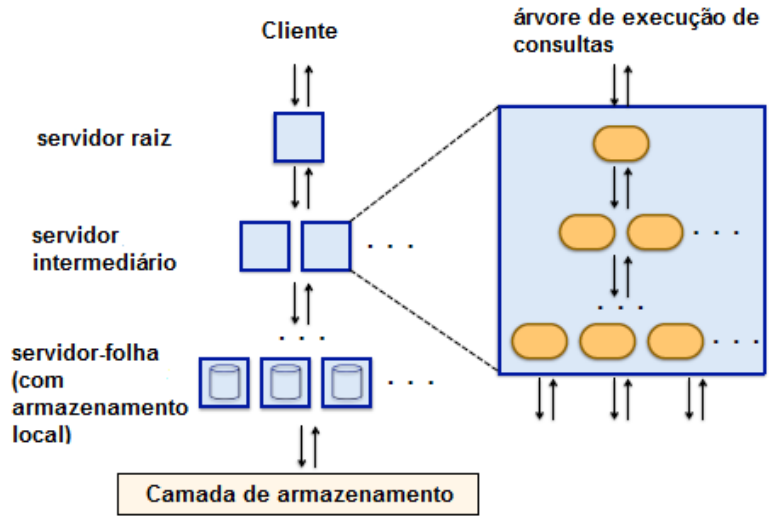

Figura 2.9: Arquitetura de consulta em árvore (Adaptação de Melnik et. al, 2010).

2.3.1.2. Comparativo Google BigQuery e MapReduce

Assim como a tecnologia MapReduce, o BigQuery também trabalha no processamento de Big Data, porém existem algumas diferenças entre essas duas tecnologias: BigQuery foi projetado como uma ferramenta de análise de dados interativos para grandes conjuntos de dados, enquanto o MapReduce foi concebido como um *framework* para processamento em lotes (*Batch processing)* em grandes conjuntos de dados.

Apesar de MapReduce e BigQuery serem ambas infraestruturas de computação em paralelo, segundo Sato (2012), BigQuery executa uma consulta em Big Data em alguns segundos, enquanto MapReduce pode levar alguns minutos ou horas para

terminar a consulta. Essa diferença se dá quando as tecnologias não são aplicadas para o seu propósito de origem, não desempenhando o máximo que podem oferecer. Dessa forma, BigQuery apresenta melhores resultados quando usado para consultas (*queries*) em *ad hoc* e análises de dados de tentativa e erro em grandes conjuntos de dados. Já MapReduce apresenta a sua melhor performance em processamento *batch* em grande volume de dados para reunião ou conversão mais demorada de dados; é a melhor opção quando é preciso processar dados não estruturados de forma programática, porém podem ser usados de forma complementar.

2.4. SEGURANÇA NO ARMAZENAMENTO E TRANSMISSÃO DE DADOS

Para que um sistema seja seguro seguindo as definições de Stallings (2010), deve-se analisar o cumprimento dos princípios de disponibilidade, integridade, confidencialidade, autenticidade e irretratabilidade.

A disponibilidade garante que o sistema computacional (*hardware* e *software*) se mantenha operacional de forma eficiente e possua a capacidade de se recuperar rápida e completamente em caso de falhas. Uma interrupção põe em risco a disponibilidade dos recursos (Kahate, 2013).

A integridade assegura que o sistema não altere os dados ou informações armazenados ou transmitidos, bem como não permitir que alterações involuntárias ou intencionais ocorram. O ataque contra a integridade de uma mensagem é o de modificação (Kahate, 2013).

A confidencialidade tem o papel de permitir somente que pessoas, entidades e processos autorizados tenham acesso aos dados e informações armazenados ou transmitidos, no momento e na forma autorizada. A interceptação causa perda de confidencialidade da mensagem (Stallings, 2010).

A autenticidade atesta com exatidão o originador do dado ou informação, bem como o conteúdo da mensagem. O ataque de personificação prejudica a propriedade de autenticidade em um sistema. E, por fim, a irretratabilidade atesta a impossibilidade de negar a participação em uma transação eletrônica (Kahate, 2013).

Os tipos de ataques sofridos podem ser divididos em dois: ativo ou passivo. Um ataque passivo pretende aprender ou usar a informação encontrada no sistema, entretanto, não afeta os recursos do mesmo. São caracterizados por um procedimento de espionagem ou monitoramento da rede. Um ataque ativo tenta alterar a informação

ou a operação realizada no sistema, modificando-a ou criando uma versão falsa (Stallings, 2010).

O serviço básico provido pela criptografia é a habilidade de mandar informação entre dois participantes de uma maneira que previna a leitura por outros. Por meio de uma chave, um texto em claro é codificado por um algoritmo de criptografia resultando em um texto cifrado. É importante que um algoritmo de criptografia seja razoavelmente eficiente para que possa ser calculado. Esquemas criptográficos não são impossíveis de serem quebrados sem a chave, dessa forma, o nível de segurança de um esquema depende de quanto esforço ou recursos são alocados pelo atacante para quebrá-lo (Kaufman *et al.*, 2002).

Define-se formalmente em termos criptográficos os conceitos de criptografia simétrica, mesma chave usada para codificação e decodificação, e algoritmos criptográficos:

Seja $\{0,1\}^n$ o conjunto de *strings* binários de tamanho *n* e $k \leftarrow \mathcal{K}$ uma chave *k* escolhida aleatória e uniformemente do conjunto $\mathcal{K}$. Conforme definido em Goldreich (2009), um esquema criptográfico simétrico é um trio *(K, C, D)*, sendo $\mathcal{K} = \{0,1\}^n$ o conjunto de chaves, $\mathcal{X} = \{0,1\}^m$ o conjunto de textos em claro e $\mathcal{Y} = \{0,1\}^l$o conjunto de textos cifrados, onde $C : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ é um algoritmo de codificação que mapeia a chave $k \in \mathcal{K}$ e o texto em claro $x \in \mathcal{X}$ em um texto cifrado correspondente $c \in \mathcal{Y}$. $D : \mathcal{K} \times \mathcal{Y} \rightarrow \mathcal{X}$ é um algoritmo de decodificação que mapeia a chave *k* e a cifra *c* no texto em claro *x* correspondente. Nota-se que $D_k\big(C_k(x)\big) = x$ e o tamanho *n* da chave é chamado parâmetro de segurança do esquema.

2.4.1. Cifras de Bloco

Uma cifra de bloco é um esquema de Cifragem/Decifragem em que um bloco de texto em claro é tratado como um todo e usado para produzir um bloco de texto cifrado de igual comprimento (Stallings, 2010). Muitas cifras de bloco tem uma estrutura de Feistel, que é constituída por um número de rodadas idênticas de processamento. Em cada rodada, uma substituição é realizada em metade dos dados a serem processados, seguida por uma permutação que troca as duas metades. A chave original é expandida para que uma chave diferente seja usada para cada rodada.

O *Data Encryption Standard* (DES) tem sido o mais utilizado algoritmo de criptografia até recentemente, principalmente na sua versão 3DES. Exibe a estrutura

de Feistel clássica. DES usa um bloco de 64 bits e uma chave de 56 bits (Stallings, 2010).

Com a evolução de novas aplicações, o DES tornou-se não mais adequado por sua chave muito pequena, e a sua implementação tripla ser muito lenta. Surge então, o *Advanced Encryption Standard* (AES), uma cifra de bloco que substitui o uso do DES. Usa um bloco de 128 bits e chaves de 128, 192 ou 256 bits (Stallings, 2010). AES não usa uma estrutura Feistel, como alternativa, usa o algoritmo de Rijndael: cada rodada consiste em quatro funções separadas: substituição de bytes, permutação, operações aritméticas sobre um campo finito e XOR com a chave. Ao contrário da estrutura de Feistel, no AES todo o bloco é processado, de maneira que é tratado como uma matriz que realiza as quatro funções mostradas na Figura 2.10 em diversas rodadas.

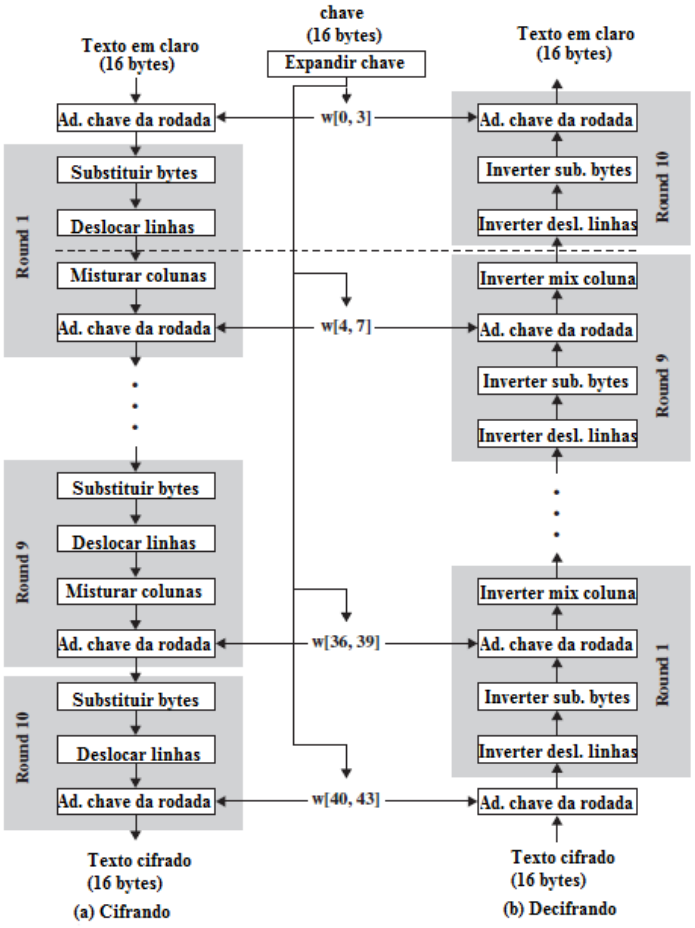

Figura 2.10: AES Cifragem e Decifragem (Adaptação de Stallings, 2010).

2.4.2. Modos de Operação

No caso em que a mensagem a ser cifrada não tem o tamanho exato que os blocos acima citados, geralmente maiores, é necessário separá-las em blocos e usar os modos de operação.

- *Eletronic Code Book* (ECB): A Figura 2.11 apresenta cada bloco codificado de forma independente usando a mesma chave. Blocos de texto em claro iguais terão textos cifrados iguais. Dessa forma deve ser usado em mensagens pequenas onde há menos incidências de blocos iguais (Kahate, 2013).

Figura 2.11: Diagrama EBC (Adaptação de Kantarcioglu).

- *Cipher Blocking Chaining* (CBC): Na Figura 2.12 nota-se que é feita um XOR do bloco de texto em claro com o resultado cifrado do bloco anterior, e então é cifrado o bloco. A primeira operação XOR deve ser feita com um vetor de inicialização (VI) gerado aleatoriamente que deve ser enviado com o texto. Blocos de texto em claro idênticos são cifrados diferentemente, além disso, o último bloco de texto cifrado depende de todo o texto em claro.

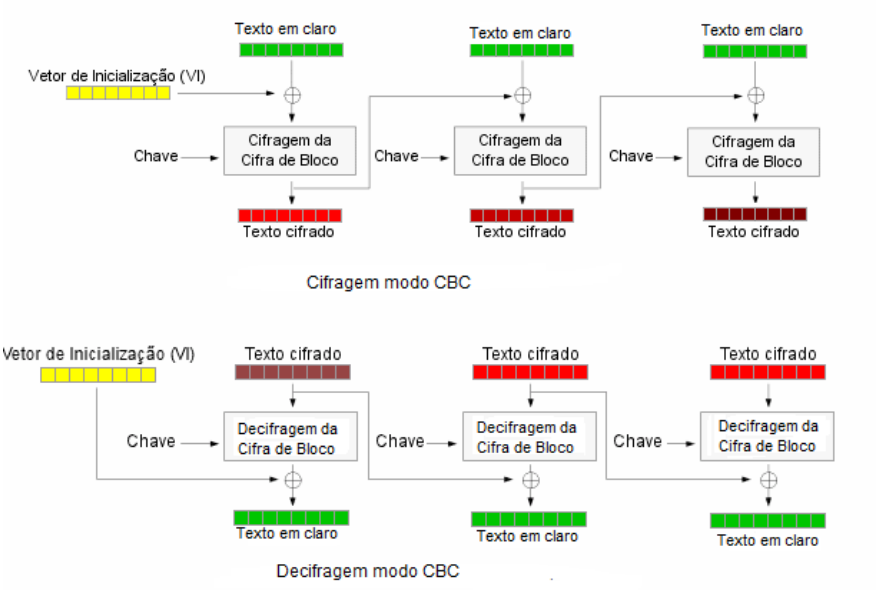

Figura 2.12: Diagrama CBC (Adaptação de Kantarcioglu).

- *Cipher Feedback Mode* (CFB): A Figura 2.13 mostra que o bloco de texto anteriormente cifrado é usado como entrada para o algoritmo criptográfico para produzir uma saída pseudoaleatória, em seguida é feito XOR com o texto em claro para produzir a próxima unidade de texto cifrado (Stallings, 2010). Assim como no modo CBC, um VI é inserido como entrada da primeira operação XOR.

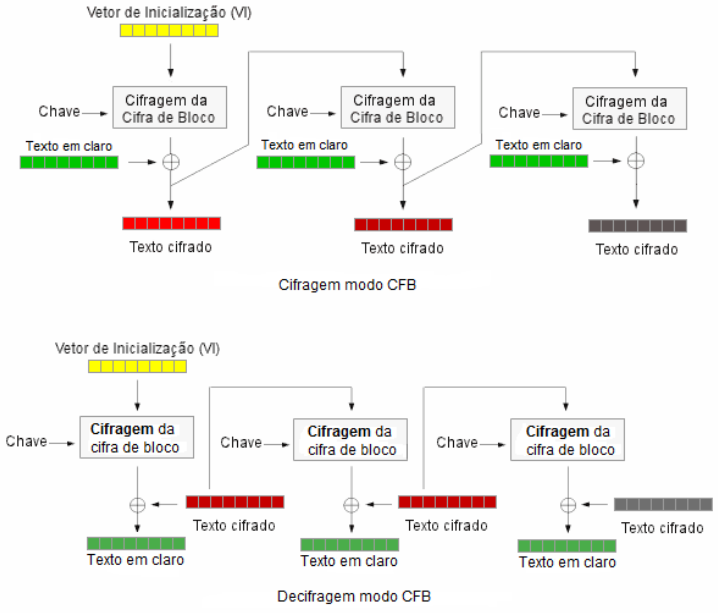

Figura 2.13: Diagrama CFB (Adaptação de Kantarcioglu).

- *Output Feedback Mode*(OFB): semelhante ao CFB, exceto que, a entrada para o algoritmo criptográfico é a saída do algoritmo criptográfico anterior (Figura 2.14). São usados blocos completos (Stallings, 2010).

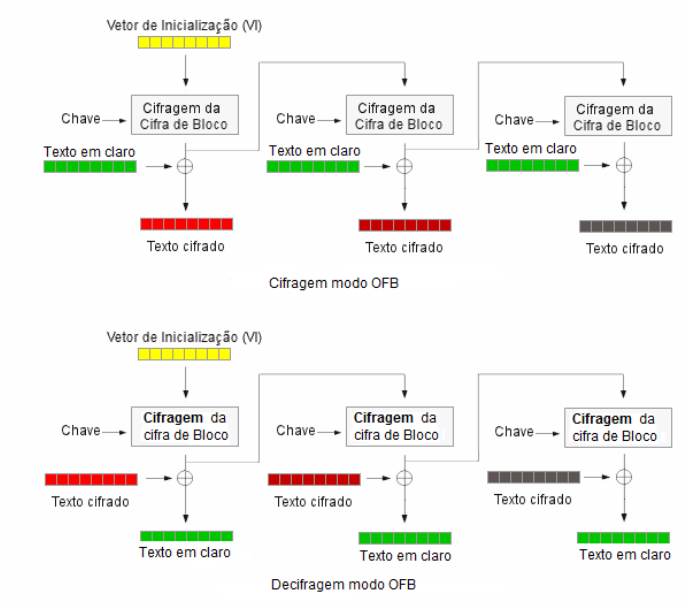

Figura 2.14: Diagrama OFB (Adaptação de Kantarcioglu).

2.4.3. Tipos de ataque a cifra

Uma cifra é considerada segura se um atacante tem dificuldade comprovadas para obter a chave ou texto em claro desejado tendo apenas o texto cifrado como base. A Tabela 2.1 evidencia os tipos de ataque a uma cifra.

Tabela 2.1: Tipos de ataques a cifra (Kahate, 2013).

Ataque	**Elementos conhecidos pelo atacante**	**Elementos que o atacante quer descobrir**
Apenas texto cifrado *(Cipher text only)*	▪ Texto cifrado de várias mensagens, as quais todas são cifradas com a mesma chave ▪ Algoritmo usado	▪ Mensagens de texto em claro correspondentes a esses textos cifrados ▪ Chave usada na criptografia
Texto cifrado conhecido *(Known cipher text)*	▪ Texto cifrado de várias mensagens, as quais todas são cifradas com a mesma chave	▪ Chave usada na criptografia ▪ Algoritmo para decifrar as mensagens de texto em claro com a mesma chave

	▪ Mensagens de texto em claro correspondentes às cifras acima.	
Texto em claro escolhido *(Chosen plain text)*	▪ Texto cifrado e as mensagens de texto em claro associadas a ele ▪ Escolhe o texto em claro a ser cifrado	▪ Chave usada para a criptografia ▪ Algoritmo para decifrar o texto cifrado com a mesma chave
Texto cifrado escolhido *(Chosen chipher text)*	▪ Texto cifrado de várias mensagens a serem decifradas ▪ Mensagens de texto em claro correspondentes	▪ Chave usada para a criptografia

2.4.4. Indistinção a ataques

Uma propriedade que garante a segurança de uma cifra é a indistinção. Intuitivamente, se um esquema possui essa propriedade, o atacante não consegue distinguir pares de textos cifrados baseado na mensagem que foi criptografada. Assim, um criptossistema é considerado seguro em termos de indistinção se nenhum atacante, dada a cifra de uma mensagem escolhida aleatoriamente de um conjunto de dois elementos determinados pelo atacante, pode identificar qual mensagem foi escolhida com probabilidade significantemente maior que uma escolha aleatória (1/2) (Goldreich, 2009). Se um adversário consegue diferenciar o texto cifrado escolhido com probabilidade maior que 1/2, então é dito que esse adversário possui uma vantagem em indistinção e o esquema não é considerado seguro em termos de indistinção. Definido formalmente por Golderich (2009), um esquema criptográfico (*K, C, D)* é indistintamente seguro se para todo $x, y \in \mathcal{X}$, todo algoritmo *A*, todo polinômio *p* e todo *n* suficientemente grande, com a vantagem $AdvA_{xy} < 1/p(n)$. A vantagem é definida como:

$$AdvA_{xy} = |\Pr[A(C_k(x)) = 1] - \Pr[A(C_k(y)) = 1]| \qquad (1)$$

Partindo desse princípio, algumas definições a respeito de indistinção foram desenvolvidas baseadas em Goldreich (2009).

2.4.4.1. Indistinção sobre ataque de texto em claro escolhido

Um esquema de criptografia *(K, C, D)* é indistintamente seguro sobre um ataque de texto em claro escolhido (*Indistinguishability under chosen plain text attack* – IND-CPA) se para todo $x, y \in \mathcal{X}$, todo algoritmo A^{C_k} com acesso a um oráculo de criptografia C_k ,todo polinômio positivo *p* e para todo *n* suficientemente grande, a vantagem $AdvA_{xy}^{C_k} < 1/p(n)$. A vantagem é definida como:

$$AdvA_{xy}^{C_k} = |\Pr\left[A^{C_k}\left(C_k(x)\right) = 1\right] - \Pr\left[A^{C_k}\left(C_k(y)\right) = 1\right]| \quad (2)$$

2.4.4.2. Indistinção sobre ataque de texto cifrado escolhido previamente/ Indistinção sobre ataque de texto cifrado escolhido posteriormente

IND-CCA e IND-CCA2 (*Indistinguishability a-priori chosen-cipher text attack* – IND-CCA/ *Indistinguishability posteriori chosen-cipher text attack* – IND-CCA2) usam uma definição semelhante a IND-CPA. Contudo, além do uso de um oráculo de criptografia é dado acesso a um oráculo de decodificação que retorna o texto em claro de cifras enviadas pelo atacante. Na definição *a-priori*, é permitido ao usuário consultar esse novo oráculo somente até receber a cifra como desafio. Na definição *posteriori*, também chamada de adaptativa, o atacante pode continuar consultando o oráculo de decodificação mesmo depois de ter recebido a cifra como desafio. Um ataque de texto cifrado escolhido pode ser descrito:

i. O desafiante gera uma chave $k \leftarrow \mathcal{K}$.
ii. O adversário solicita ao oráculo de decodificação os textos em claro correspondentes às cifras de sua escolha.
iii. O desafiante gera duas *strings* de textos em claro e envia ao adversário a cifra de uma delas.
iv. O adversário tenta adivinhar qual das duas *strings* lhe foi dada.

Na definição *posteriori* o adversário pode pedir a decodificação de mais cifras, exceto pela decodificação do desafio recebido. Definido formalmente por Golderich (2009), um esquema criptográfico *(K, C, D)* é indistintamente seguro a respeito de um

ataque de texto cifrado escolhido posteriormente se para todo $x, y \in \mathcal{X}$, todo algoritmo A^{D_k} com acesso a um oráculo de decodificação D_k, todo polinômio positivo *p* e para todo *n* suficientemente grande, a vantagem $AdvA_{xy}^{D_k} < 1/p(n)$. A vantagem é definida como:

$$AdvA_{xy}^{D_k} = |\Pr\left[A^{D_k}\left(C_k(x)\right) = 1\right] - \Pr\left[A^{D_k}\left(C_k(y)\right) = 1\right]| \quad (3)$$

As definições de indistinção sobre ataques a cifras são comumente utilizadas para caracterizar criptossistemas quanto a sua segurança, sendo IND-CCA2 a definição mais forte das três apresentadas.

2.4.5. Funções Hash

Define-se hash em Stallings (2010) como uma função que mapeia uma mensagem de tamanho variável em um valor hash de tamanho fixo ou uma versão reduzida da mensagem. Seu funcionamento é baseado no uso repetitivo de funções de compressão. Funções de hash tem como seu principal objetivo a integridade de dados, por isso são úteis para autenticação de mensagens e verificação da integridade, pois a mudança, por menor que seja, em uma mensagem *m* resultará numa mudança na hash *H(m)* (Kaufman *et al.*, 2002). Para que seja útil, uma função de hash deve obedecer certas propriedades.

A propriedade de não-inversão está relacionada a incapacidade de descobrir *m* dado um *H(m)* (Stallings, 2010). Ou seja, é fácil gerar um código a partir de uma mensagem, mas praticamente impossível gerar uma mensagem a partir de um código. Assim, a melhor maneira para descobrir a mensagem por um código é um ataque força bruta, tenta-se todo *m* possível tal que a sua função hash *h* seja igual a *H(m)*.

A propriedade de resistência fraca garante a impossibilidade de gerar uma mensagem alternativa com o mesmo valor de hash da mensagem original. Um ataque para quebrar a resistência a colisão seria achar qualquer outra mensagem que tenha o mesmo valor de hash.

A resistência a colisão forte implica que deve-se achar um par de mensagens *x* e *y* que tenham um mesmo valor de hash tal que *H(x)= H(y)* (Stallings, 2010). Um ataque a esse tipo de resistência é chamado ataque de Aniversário (Kaufman *et. al*, 2002). Suponha que um código de hash de 64 bits seja seguro, se um código hash

criptografado *H(m)* for transmitido concatenado com a mensagem não criptografada correspondente *m*, então um atacante precisaria encontrar *m'* tal que *H(m') = H(m)*, para substituir *m* por *m'* e enganar o receptor.

A Tabela 2.2 resume as propriedades necessárias para uma função de hash. Uma função de hash é dita fraca se respeita todas as cinco primeiras propriedades, uma vez que a propriedade de resistência a colisão não implica em não-inversão e vice-versa. Uma função de hash é dita forte quando atende às seis propriedades mostradas (Stallings, 2010).

Tabela 2.2: Resumo de propriedade necessária para uma função hash (Stallings, 2010).

Propriedades	**Descrição**
Entrada com tamanho variável	A função hash *H( )* pode ser aplicada a um bloco de dados de qualquer tamanho
Saída com tamanho fixo	A função hash *H( )* produz uma saída de comprimento fixado
Eficiência	*H(x)* é relativamente fácil de ser computada para cada *x* dado, fazendo tanto as implementações de *hardware* e *software* práticas
Unidirecional ou não-inversão (*Preimage resistant)*	Para cada valor de hash *h* dado, é computacionalmente inviável achar *y* de tal modo que *H(y)=h*
Resistência fraca a colisão (*Second preimage resistant)*	Para cada bloco *x* qualquer, é computacionalmente inviável achar $y \neq x$ com *H(y) = H(x)*
Resistência forte a colisão	É computacionalmente inviável achar qualquer par *(x, y)* de tal modo que *H(y) = H(x)*

Como exemplos de hashes pode-se citar:

- MD4 *Message Digest Algorithm* (RFC 1320): Foi projetado por Ron Rivest em 1990 e introduzido com o objetivo de que fosse uma função rápida de 128 bits, porém foi demonstrado em 2004 que o algoritmo não é seguro. Demonstrou-se

que era possível achar colisões para MD4 em menos de um minuto utilizando um computador simples.

- MD5, *Message Digest Algorithm* RDA-MD5 (RFC 1321). É uma versão melhorada de MD4 com saída de 128 bits que adicionou mais elementos às etapas do algoritmo, tornando-o mais lento porém mais seguro. A função tornou-se um padrão mundial, porém foi demonstrado também uma drástica redução no tempo para encontrar colisões.
- SHA-1, *Secure Hash Algorithm* (NIST-FIPS 180-1). Foi aprovado pelo governo dos EUA em 1995. É muito similar, no seu modo de operação, com o MD5. Este algoritmo é ligeiramente mais lento do que MD5, mas a saída de 160 bits o faz mais seguro frente à procura de colisões usando a força bruta. Até agora mostrou-se seguro contra quebra de resistência a colisão e inversão.

2.5. TRABALHOS RELACIONADOS

Aplicações online estão sujeitas ao roubo de informação sigilosa, pois atacantes podem tirar proveito de servidores inseguros e obter acesso a informação. As informações relevantes podem ser cifradas antes de serem enviadas para os servidores, entretanto, processar *queries* sobre os dados cifrados é difícil. Uma abordagem é projetar um esquema que possa executar funções arbitrárias nos dados cifrados como se estivesse executando sobre o texto em claro. A criptografia totalmente homomórfica (Gentry, 2009) é um conceito teórico que atende a esses objetivos na garantia de confidencialidade, porém é considerada impraticável no momento.

2.5.1. Busca de Palavras

Song *et al.*(2000) descrevem esquemas criptográficos para efetuar uma busca por palavras-chave em dados cifrados usando um servidor inseguro. Estes esquemas são usados na composição deste trabalho. O esquema é simples e rápido, para um documento de tamanho *n*, os algoritmos de criptografia e busca precisa de apenas *O(n)* operações de cifra de fluxo e de bloco. Não inserem quase nenhum espaço ou *overhead.*

Um método para buscas exatas que não requer a leitura de toda a base de dados é apresentado em Amanatidis *et al.* (2007). Bao *et al.* (2008) aprimora as

técnicas de busca em dados criptografados para o caso de multiusuários. Boneh & Waters (2007) mostram esquemas de chaves públicas para comparação, checagem de subconjuntos e consultas de conjunção em dados cifrados, porém esses esquemas têm cifras de tamanho exponencial em relação ao texto em claro, limitando a sua aplicação prática.

Boldyreva *et al.* (2009) apresenta um modelo de criptografia simétrica com preservação de ordem. Tal modelo é equivalente a um mapeamento aleatório que preserva a ordem. Entretanto, devido a essa propriedade, torna-se mais fraca que um modelo determinístico pois o adversário pode obter o conhecimento da ordem. O modelo provido permite relações de ordem entre dois dados baseadas nos seus valores cifrados, sem revelar os dados reais. Se uma coluna é criptografada com a preservação de ordem, o servidor pode realizar consultas de atributos inseridos em um determinado intervalo.

2.5.2. Coleção de esquemas de consultas em dados cifrados

CryptDB (Popa *et al.*, 2011) é um sistema que provê confidencialidade prática e comprovada para aplicações baseadas em bancos de dados SQL. Evita a criptografia totalmente homomórfica, no entanto, processa as consultas usando uma coleção de esquemas criptográficos eficientes admitidos pela configuração SQL (Popa *et al.*, 2011). CryptDB direciona as ameaças passivas para o SGBD e as ameaças ativas para o Proxy, que atua como um agente mediador entre o cliente e o SGBD. As consultas vindas do cliente são interceptadas pelo Proxy que as reescreve de forma que possam ser executadas em dados cifrados. O servidor efetua as consultas e retorna os textos cifrados correspondente para o Proxy. O proxy decifra as cifras recebidas e envia os resultados para o cliente (Figura 2.15).

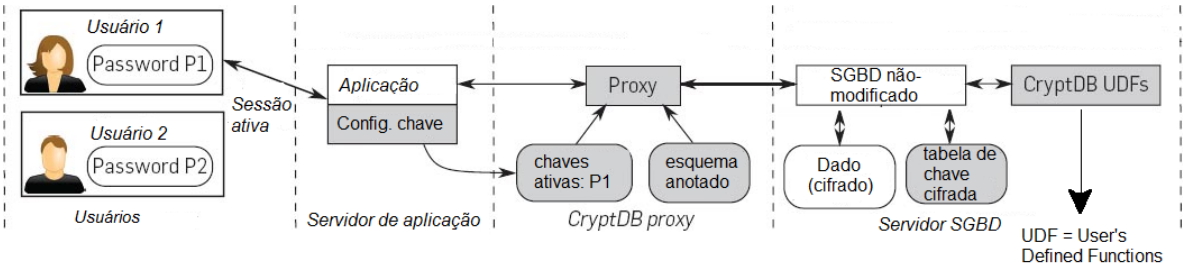

Figura 2.15: Arquitetura do CryptDB (Adaptação de Popa *et al.*,2011).

O diferencial desta ferramenta é o uso de criptografia em camadas, ou cebolas, que são definidas com base nos requisitos de acesso, nos tipos de consulta e previsões anteriores para revestir os dados com as camadas apropriadas. O proxy, ao receber a consulta, dinamicamente retira as camadas para chegar a camada correspondente à classe de execução dada. O servidor processa os dados sobre as camadas adequadas e os envia de volta ao proxy. Como as colunas são criptografas com as mesmas chaves contabilizadas pelo proxy, informações de igualdade serão vazadas apenas durante as consultas de igualdade, assim, a informação é divulgada apenas quando solicitada. Suas vantagens são: uma grande variedade de consultas suportadas, o SGBD não tem nenhum acesso aos dados em claro, requer pouca alteração a um sistema de banco de dados para que fique compatível e o método complexo de criptografia em camadas aprimora a segurança dos dados, bem como fornece apenas o nível necessário de proteção para diferentes subconjuntos dos mesmos dados. Apesar de apresentar uma ferramenta de consulta a dados criptografados comprovadamente rápida (Popa *et al.*, 2011), o processamento de várias camadas da cebola torna-se computacionalmente intenso, podendo causar problemas de consistência de dados devido ao crescimento dos mesmos no banco de dados. Transações podem ser atrasadas ou interceptadas. Em algumas etapas, CryptDB vaza dados e, ao longo do tempo, pode permitir aos atacantes estudar o *layout*, o que vai finalmente permitir-lhes espionar dados de usuários. Pode não ser adequado para o processamento de dados em tempo real, o qual um processamento rápido e confiável é necessário para os dados, sem comprometer o desempenho como CryptDB faz.

Tu *et al.* (2013) é apresentado o Monomi, um sistema de execução de consultas analíticas em dados confidenciais usando um servidor não confiável. Monomi usa uma execução de consulta dividida em cliente/servidor (*split client/server query execution*) para realizar consultas complexas que são primeiramente processadas no servidor e tem o resto do processamento terminado no cliente. Introduzem técnicas que contribuem na melhora de performance de algumas consultas analíticas em dados cifrados: pré-processamento por linha, criptografia eficiente em termos de espaço, além de adição homomórfica agrupada e pré-filtragem. Finalmente, Monomi mostra que o uso de um planejamento melhora o desempenho

sobre algoritmos “gananciosos”, escolhendo projetos físicos eficientes e planos de execução de consultas.

Mylar (Popa *et al.,* 2014) é um *framework* de aplicação Web que permite aos desenvolvedores protegerem dados confidenciais diante de servidores com propensão a insegurança. Aproveita-se a troca de dados entre *browser* e servidor, em vez do HTML, para cifrar todos os dados armazenados no servidor e decifrá-los apenas no *browser* do cliente. Esse *framework* provê uma abstração para compartilhar dados com segurança entre os usuários, e utiliza uma extensão do *browser* para verificar o código baixado do servidor que roda no navegador. Para buscas de palavras-chave, que não são práticas de serem executadas no navegador, Mylar insere um esquema criptográfico para realizar as buscas no servidor em dados cifrados com chaves diferentes. Resultados experimentais mostram que o uso de Mylar requer poucas modificações na aplicação e que apresenta um *overhead* modesto de desempenho (Popa *et al.*, 2014).

Arasu *et al.* (2013) apresentam o Cipherbase, um sistema de banco de dados SQL completo que permite as organizações utilizarem as vantagens da computação em nuvem e ao mesmo tempo manterem a confidencialidade dos dados sensíveis. Cipherbase é baseado em uma arquitetura de coprocessadores com o objetivo de decompor o processamento entre o *hardware* tradicional (inseguro) e o confiável. Utiliza a combinação de criptografia de dado estático (*encryption at rest),* servidores seguros e o uso de criptografia parcialmente homomórfica para atingir segurança ortogonal, isto é, permitir que as organizações desenvolvam suas aplicações e configurem suas políticas de nível de segurança independentemente de qualquer desempenho, escalabilidade ou custo. Possui uma forte ligação entre os projetos de *hardware/software*, que integra FPGAs (*Field-Programmable Gate Array*) como *hardware* confiável no sistema de banco de dados SQL Server da Microsoft. Cipherbase ainda está no processo de pesquisa e desenvolvimento pelo Microsoft Research.

3. MODELO DE BANCOS DE DADOS SEGURO HOSPEDADO EM NUVEM

Este capítulo apresenta o conjunto de esquemas criptográficos adequados para cada tipo de consulta utilizadas em dados criptografados, de forma que um servidor inseguro não obtenha nenhum tipo de informação a respeito do conteúdo armazenado. Em seguida é apresentada a ferramenta *Encrypted BigQuery*, que utiliza esses blocos de criptografia e os implementa em seu Sistema de Banco de Dados. Como EBQ utiliza de forma semelhante os mesmos esquemas criptográficos aplicados no CryptDB, apresenta-se uma comparação entre as duas ferramentas, expondo as suas diferenças e suas aplicações correspondentes.

3.1. CONSULTAS EM DADOS CRIPTOGRAFADOS

Para a consulta de dados criptografados é necessário um conjunto de criptossistemas que os trate adequadamente sem a perda de confidencialidade. Dessa forma, os dados brutos são cifrados de maneiras diferentes antes de serem inseridos em um banco de dados. Dependendo do tipo de consulta a ser realizada num determinado campo, uma técnica distinta será usada. São elas:

3.1.1. Algoritmo Probabilístico

A cifra é dita probabilística quando, para valores diferentes existirão cifras diferentes com grande probabilidade (Kaufman *et al.*, 2002). Para uma construção eficiente é usada uma cifra de bloco, como AES no modo CBC juntamente com um vetor de inicialização (VI) gerado aleatoriamente.

Provê máxima segurança devido a propriedade de indistinção sobre um Ataque de Texto em Claro Escolhido (IND-CPA), um adversário será incapaz de distinguir pares de textos cifrados baseados em mensagens cifradas por ele. Apesar de garantir confidencialidade e integridade dos dados em um banco de dados, não é possível realizar manipulações com os dados cifrados, exceto o comando SELECT.

3.1.2. Algoritmo Determinístico

O modelo de criptografia é dito determinístico quando é gerada a mesma cifra para a mesma mensagem em claro. O esquema é feito com a cifra de bloco AES no modo CBC com um VI de zeros. Assim, o esquema determinístico realiza uma permutação pseudoaleatória (Goldreich, 2009) por meio do uso de cifras de bloco, o

que representa uma diminuição no nível de segurança: adversários podem ter o conhecimento de quais valores cifrados correspondem ao mesmo valor em claro.

A escolha desse esquema permite a realização de consultas de igualdade, isto é, pode realizar o comando SELECT com predicados de igualdade, junções de igualdade, COUNT. (Popa *et al.,* 2011).

3.1.3. Busca por Palavras

Este esquema não é necessariamente um esquema de criptografia. É calculada a função hash, normalmente SHA-1, de todas as sequências possíveis de palavras. Em seguida, os hashes são mantidos em um campo e separados por espaços. Caso haja algum caractere especial inserido no texto, este deve ser informado no arquivo esquema.

Representa o nível mais baixo de segurança implementado, uma vez que hash é uma função pública e facilmente reproduzida. Um ataque de dicionário conseguiria relacionar a mensagem em claro com o seu hash correspondente. Portanto, é utilizado na criptografia de campos que necessitem da funcionalidade de busca por palavras, entretanto não é necessário um nível de segurança alto. Podem ser usadas como atributo para cláusula WHERE com checagem de conteúdo (usando CONTAINS) com palavras-chaves inteiras, mas não aparecem em consultas SELECT simples.

3.1.4. Busca Probabilística por Palavras

A busca probabilística consiste em buscar uma palavra *W* e retornar todas as posições onde *W* aparece no texto em claro (Song *et. al,* 2000). Erros quanto o aparecimento de posições erradas podem ocorrer, porém podem ser controlados ajustando o tamanho do texto cifrado gerado. Sendo $\mathcal{Y} = \{0,1\}^l$ o conjunto de textos cifrados, cada posição errada será retornada com probabilidade $1/2^l$ (Song *et. al*, 2000).

Primeiramente o texto é dividido em palavras-chave usando delimitadores padrão de uma palavra em português. As repetições são eliminadas, as posições aleatoriamente permutadas para garantir mais segurança e é feito o preenchimento (*padding)* para que cada palavra tenha o mesmo tamanho.

A ideia principal do algoritmo proposto por Song *et. al* (2000) é cifrar um texto composto por $W_1, ..., W_z$ realizando a operação XOR bit a bit entre o texto em claro e

uma sequência pseudoaleatória de bits com uma estrutura específica. Essa configuração permite a busca de dados sem revelar nada sobre o conteúdo do texto em claro.

Inicialmente, cada palavra *W* do texto é cifrado previamente usando um algoritmo determinístico $C_{k''}$(como EBC, ou caso a palavra seja muito grande, o modo CBC com um VI constante). Esse processo permite que $C_{k''}(x)$ dependa apenas de *x* e não da posição *i* onde *x* se encontra. Define-se cada palavra criptografada como sendo $X_i = C_{k''}(W_i)$e a sequência de palavras pré-criptografadas como $X_1, ..., X_z$.

A chave k_i é escolhida por meio de uma função pseudoaleatória $f : \mathcal{K}_F \times \{0,1\}^* \rightarrow \mathcal{K}_F$ de modo que $k_i = f_{k'}(W_i)$. Escolher uma chave em função da palavra W_i garante uma busca controlada, a qual permite que o servidor identifique todos os lugares onde *W* pode aparecer, mas não revela nada sobre as posições *i* onde $W_i \neq W$.

Usa-se um gerador pseudoaleatório para a geração da sequência de valores $S_1,..., S_z$. Tal sequência é cifrada usando uma função pseudoaleatória $F : \mathcal{K}_F \times \mathcal{X} \rightarrow \mathcal{Y}$, de modo que cada S_i é concatenada com a sua cifra, resultando em $T_i = \langle S_i, F_{ki}(X_i)\rangle$. Em seguida, a cifra C_i é construída para obter $C_i = X_i \oplus T_i$, onde $X_i = C_{k''}(W_i)$ e $T_i = \langle S_i, F_{ki}(X_i)\rangle$ e então enviada ao servidor.

Para buscar por uma palavra *W*, o usuário calcula $X = C_{k''}(W)$ e $k = f_{k'}(W)$ e envia os resultados concatenados $\langle X, k\rangle$. O servidor faz a busca por *X* checando no texto cifrado se $C_i \oplus X_i$ está na forma $\langle S_i, F_{ki}(X_i)\rangle$ e retornando a posição correta.

Oferece um nível maior de segurança que o modelo anterior, uma vez que provê consultas isoladas para as buscas, ou seja, o servidor inseguro não pode aprender nada do texto em claro além do resultado da busca. Fornece uma busca controlada para que o servidor não possa buscar uma palavra qualquer sem a autorização do usuário e, além disso provê a consulta oculta de maneira que o usuário solicite uma busca de uma palavra secreta ao servidor sem revelar a palavra ao mesmo. Permite consultas SELECT para checagem de conteúdo, com o uso da cláusula WHERE e atributo CONTAINS ou LIKE.

3.1.5. Criptografia Homomórfica

Criptografia homomórfica é uma forma de criptografia que permite lidar com tipos específicos de cálculos em cifras e gerar um resultado também cifrado que, quando decifrado, corresponde ao resultado de operações realizadas em textos em claro. Ou seja, uma cifra *C( )* é dita homomórfica quando, dado *C(x) e C(y*), é possível obter *C(x # y)* sem decifrar x e y, para alguma operação # (Gentry, 2009). A utilização de uma cifra puramente homomórfica é comprovadamente lenta (Cooney, 2009), então a alternativa eficiente a ser escolhida é uma cifra parcialmente homomórfica, como Paillier (1999), que é um esquema seguro IND-CPA.

Para compreender melhor como o criptossistema de Pailler funciona deve-se definir alguns conceitos matemáticos (Oliveira Santos, 1998):

- A Função de Euler, *Φ(n),* retorna o número de inteiros positivos menores ou iguais a *n* que são primos entre si com *n. Se n* pode ser fatorado em dois primos distintos *p* e *q (n = pq)*, então: $\phi(n) = (p-1)(q-1)$
- A Função de Carmichael, *λ(n),* retorna o m.m.c. (Mínimo Múltiplo Comum) de todos os fatores da função *Φ(n). Se n* pode ser fatorado em dois primos distintos *p* e *q*, então: $\lambda(n) = mmc(p-1, q-1)$
- Inverso Modular de um inteiro *a* modulo *n* é aquele que, a multiplicação de *a* pelo seu inverso resulta em resto 1. Isto é, seja *x* o inverso modular de *a* tal que:
  $a^{-1} \equiv x \; mod \; n$ ou $ax \equiv 1 \; mod \; n$
- $\mathbb{Z}_n$: Conjunto de números inteiros *n.*
- $\mathbb{Z}_n^*$: Conjunto de números primos entre si com *n.* $|\mathbb{Z}_n^*| = \phi(n)$.
- $\mathbb{Z}_{n^2}^*$: Conjunto de números primos entre si com *n².* $\left|\mathbb{Z}_{n^2}^*\right| = n\phi(n) = \phi(n^2)$.

Diferentemente dos esquemas usados até agora, o criptossistema de Paillier utiliza algoritmos de criptografia assimétrica, os quais a chave usada para cifrar a mensagem não é a mesma para decifrá-la. As mensagens são cifradas com uma chave pública e não podem ser decifradas sem a chave privada do destinatário.

Para a geração da chave são escolhidos um gerador *g*, onde $g \in \mathbb{Z}_{n^2}^*$ e dois números primos aleatórios grandes *p* e *q*, tal que:

$$n = pq \quad \text{e} \quad mdc\big(pq, (p-1)(q-1)\big) = 1$$

Em seguida, calcula-se a função de Carmichael *λ(n)* e o inverso modular

$$\mu = \left(L(g^{\lambda} \bmod n^2)\right)^{-1} \bmod n$$

Onde a função *L( )* é definida como $L(u) = {}^{u-1}/_{n}$. Dessa forma, o inverso modular só irá existir se foi escolhido um gerador *g* válido. Finalmente, como resultado, são obtidos os pares *(n,g)* para chave pública e *(λ,μ)* para chave privada. A Tabela 3.1 exibe os processos referentes às operações de codificação e decodificação.

Tabela 3.1: Função criptográfica de Paillier – Codificação e Decodificação.

Cifrar	**Decifrar**
▪ Seja *m* a mensagem de texto em claro a ser cifrada, onde $\boldsymbol{m} \in \mathbb{Z}_{\boldsymbol{n}}$ tal que $\boldsymbol{m} < \boldsymbol{n}$. ▪ É escolhido aleatoriamente *r*, onde $\boldsymbol{r} \in \mathbb{Z}_{\boldsymbol{n}}^{*}$ tal que $\boldsymbol{r} < \boldsymbol{n}$. ▪ O texto cifrado será: $\boldsymbol{c = g^{m} r^{n} \bmod n^2}$	▪ Seja *c* o texto cifrado, onde $c \in \mathbb{Z}_{n^2}^{*}$, tal que $c < n^2$. ▪ A mensagem decifrada será: $m = L\left(c^{\lambda} \bmod n^2\right).\mu \bmod n$

O aspecto mais notável deste esquema são as propriedades de adição homomórfica que o mesmo adquire, levando às seguintes identidades (Paillier, 1999): $\forall\, m_1, m_2 \in \mathbb{Z}_n \; e \; k \in \mathbb{N}$

- $D(C(m_1)C(m_2) \bmod n^2) = m_1 + m_2 \bmod n$
- $D(C(m)^k \bmod n^2) = km \bmod n$
- $D(C(m_1)g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n$
- $D(C(m_1)^{m_2} \bmod n^2) = m_1 m_2 \bmod n$
- $D(C(m_2)^{m_1} \bmod n^2) = m_1 m_2 \bmod n$

Sendo *D()* a função para decifrar, *C()* a função para cifrar, m_1 e m_2 duas mensagens distintas. Logo, devido às propriedades de uma cifra parcialmente homomórfica, são admitidas as consultas que buscam as manipulações numéricas de soma e média entre colunas.

É apresentado na Tabela 3.2 um resumo dos esquemas criptográficos implementados, bem como a suas aplicações e sintaxe SQL usada nas consultas. As

quatro operações matemáticas básicas, onde um dos termos é um número inserido pelo usuário, podem ser feitas com todos os campos numéricos, uma vez que a resposta da consulta é decifrada primeiro e, em seguida calculada a expressão antes de exibir os resultados.

Tabela 3.2: Resumo esquemas criptográficos.

Esquema	Construção	Funções	Sintaxe SQL
Probabilístico	AES em CBC	Dado estático	SELECT, UPDATE, DELETE
Homomórfico	Pailier (1999)	Adição	SUM, +
Busca por palavras	Hash SHA-1 das palavras	Busca de palavras	SELECT – WHERE, CONTAINS
Busca Prob. por palavras	Song *et. al* (2000)	Busca de palavra	SELECT – WHERE, CONTAINS, ILIKE
Determinístico	AES em CBC com VI nulo	Igualdade	=,!=, IN, COUNT

Com base nos algoritmos apresentados, nota-se que o criptossistema que possui maior nível de segurança é o esquema Probabilístico, seguido da criptografia Homomórfica, Busca Probabilística por Palavras, Busca por Palavras e Determinístico.

3.2. APRESENTAÇÃO *ENCRYPTED BIGQUERY*

Apesar de toda a facilidade encontrada em termos de escalabilidade, disponibilidade, flexibilidade e economia para armazenar dados de maneira distribuída, como provê Google BigQuery, o aspecto da segurança ainda é uma preocupação, uma vez que dados sigilosos podem estar expostos. A solução é a criptografia dos dados por parte do cliente antes de hospedá-los em algum serviço de armazenagem, de forma que, apenas o cliente tenha conhecimento do conteúdo hospedado e a realização das consultas também não vaze informações.

Encrypted BigQuery (EBQ) é uma ferramenta que permite a criptografia dos dados por parte do cliente seguido do carregamento (*upload*) para o BigQuery (Tigani & Naidu, 2014). A fonte não cifrada e a chave usada na criptografia não são enviadas

pela rede, assim, não há a possibilidade de qualquer administrador do BigQuery obter acesso aos dados.

O funcionamento do EBQ se inicia com a construção do arquivo esquema, que indica qual será a estrutura dos campos presentes da tabela a ser carregada. EBQ inclui um campo extra (*encrypt*), o qual determina os tipos de esquemas criptográficos, expostos na Tabela 3.2, que serão usados. Feito isso, os dados são carregados de maneira semelhante ao BigQuery, especificando o arquivo da chave usada e seguindo os esquemas criptográficos apontados no campo *encrypt*. O dado é então armazenado na nuvem do Google, mais precisamente no segmento Google Cloud Storage.

As consultas a serem feitas são reescritas antes de serem enviadas para que possam fazer sentido a base de dados do BigQuery. Deste modo, as consultas feitas usando EBQ passam por uma espécie de interpretador, que adapta com criptografia os parâmetros a serem buscados de forma que, a base de dados do BigQuery não obtém conhecimento acerca dos resultados retornados. Semelhantemente, os resultados obtidos são recebidos cifrados, são decodificados com a chave usada pelo EBQ armazenada no cliente e apresentados a aplicação, de acordo com Figura 3.1. Ainda, apresenta a integração com o *framework* Hadoop no processo de carregamento dos dados na plataforma de armazenamento.

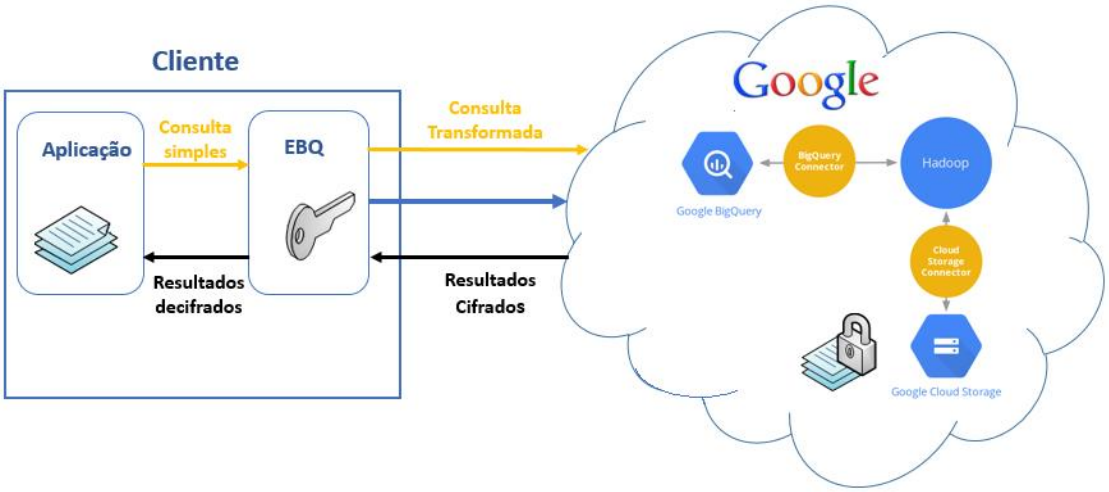

Figura 3.1: Arquitetura EBQ.

3.3. COMPARATIVO EBQ E CRYPTDB

Apesar das ferramentas EBQ e CryptDB seguirem a mesma ideia de utilizar coleções de esquemas criptográficos para a manipulação e consulta de dados armazenados em banco de dados, e a maioria dos blocos de criptografia implementados pelo EBQ foram implementados também no CryptDB, existem algumas funcionalidades e aplicações diferentes para as duas ferramentas usadas.

Primeiramente, o ambiente em que estão sendo aplicadas é diferente. EBQ funciona em um ambiente distribuído, não controlado enquanto que o CryptDB funciona em um ambiente fechado regulado por um proxy. O modelo de dados em que são aplicados os esquemas criptográficos também é diferente: CryptDB é aplicado em um modelo relacional, EBQ é executado em um modelo NoSQL que utiliza a linguagem SQL para consultas. No uso de chaves, CryptDB realiza um encadeamento de chaves com a senha do usuário para que o dado só possa ser decifrado usando a senha de um dos usuários com acesso ao dado, o EBQ utiliza uma mesma chave armazenada no cliente que efetuou o carregamento de dados, ainda não admite multiusuários. Ambos, apresentam soluções práticas e seguramente comprovadas para consultas em dados criptografados, porém a ferramenta EBQ se mostra versátil pois trata dois aspectos atuais: o processamento de grande volume de dados, auxiliado por um ambiente distribuído e a consulta a dados cifrados. A Tabela 3.3 apresenta um quadro comparativo das duas ferramentas.

Tabela 3.3: Comparativo EBQ e CryptDB

Aspectos	CryptDB	EBQ
Modelo de dados	Relacional – MySQL	NoSQL com consultas em SQL
Ambiente	Centralizado, uso de Proxy	Distribuído
Tratamento de chaves	Encadeamento de chaves	Única chave
Oferece processamento de Big Data	Não	Sim
Diferenciais	Uso de camadas de algoritmos criptográficos.	Agrega consulta de dados cifrados com processamento de Big Data.

4. RESULTADOS E ANÁLISES

Este capítulo tem como o objetivo aplicar os modelos criptográficos expostos anteriormente, avaliando o desempenho e custo operacional adicionados pelo criptossistema quando aplicado em um sistema distribuído de armazenamento massivo de dados, como o EBQ. Os cenários utilizados visam abranger uma forma mais complexa de armazenamento de forma a testar a sua eficiência. Os resultados obtidos foram baseados em medições de tempo de execução de determinadas consultas e são apresentados em forma de gráficos e tabelas juntamente com a sua descrição e análise.

4.1. AMBIENTE DE IMPLEMENTAÇÃO

As análises de desempenho e *overhead* tem como ambiente de execução um computador com processador core I5 1,8GHz com 6GB de memória RAM e sistema operacional Linux Ubuntu 14.04, executando a ferramenta EBQ que está conectada ao Google BigQuery via internet. As configurações do ambiente hospedado no Google foram feitas seguindo o padrão.

EBQ é executado por meio de uma ferramenta baseada em python que acessa BigQuery usando a linha de comando. Todas as operações realizadas são feitas por esse meio de acesso, que adapta as requisições feitas e invoca as funções de inserção, consulta e remoção criadas pelo BigQuery.

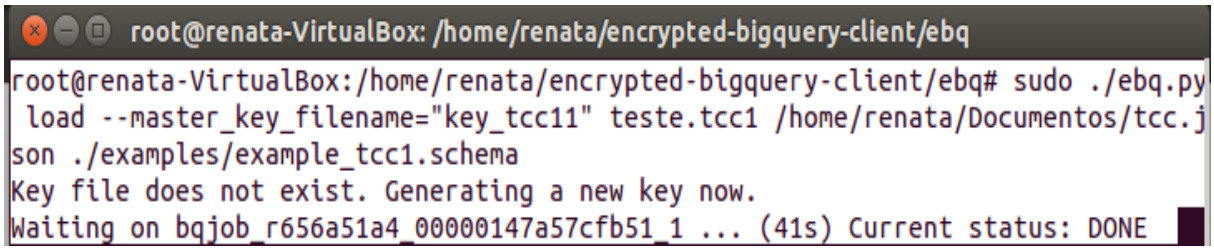

Figura 4.1: Carregamento de dados usando linha de comando.

Na Figura 4.1 é ilustrada a operação de carregamento na base dados. Como visto nas duas primeiras linhas, o usuário invoca a ferramenta EBQ e preenche os campos indicando qual a chave será usada para a criptografia, a tabela de destino, o arquivo a ser carregado e qual esquema será seguido. No caso de não haver nenhuma chave,

uma nova chave é criada e armazenada junto ao cliente. Imediatamente após isso, a requisição se transforma em um *job* a ser cumprido pelo BigQuery.

4.1.1. Cenário de implementação

Para a aplicação e demonstração dos modelos criptográficos foi utilizado um esquema de forma a apresentar a aplicação do uso de famílias de colunas. O exemplo criado tem como tema um cadastro pessoal, o qual possui um número de identificação e famílias de colunas divididas em cadastro, biometria, público, privado e secreto, conforme Figura 4.2.

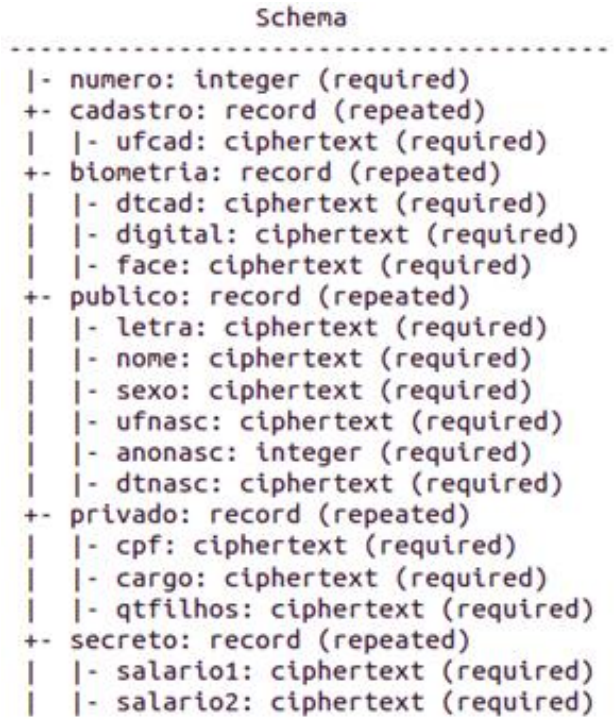

Figura 4.2: Arquivo Esquema.

A Tabela 4.1 apresenta qual tipo de criptografia foi escolhido para cada coluna da tabela. A definição da cifra adequada para cada coluna influencia fortemente o desempenho e o conteúdo da consulta realizada, uma vez que determinadas consultas tem restrições dependendo do nível de segurança escolhido para aquela coluna.

Tabela 4.1: Cifras aplicadas nas famílias de colunas.

Cadastro	
Colunas	**Tipos de Cifras**
UFcad (UF do cadastro)	Determinístico
Biometria	
Colunas	**Tipos de Cifras**
Dtcad (Data do cadastro)	Determinístico
Digital	Determinístico
Face	Determinístico
Público	
Colunas	**Tipos de Cifras**
Letra	Determinístico
Nome	Busca Prob. Palavras
Sexo	Busca por Palavras
UFnasc (UF de nascimento)	Determinístico
Anonasc (Ano de nascimento)	Nenhuma
Dtnasc (Data de nascimento)	Determinística
Privado	
Colunas	**Tipos de Cifras**
CPF	Busca por Palavras
Cargo	Busca Prob. Por Palavras
Qtfilhos (Quantidade de filhos)	Probabilístico
Secreto	
Colunas	**Tipos de Cifras**
Salario1	Homomófica
Salario2	Homomórfica

De posse do arquivo esquema contendo a estrutura, foi realizada a inserção de dados com volumes de entrada de 5000, 50000 e 100000 registros para a análise do impacto que o processamento de consultas sobre grandes volumes de dados criptografados pode gerar no tempo de resposta. Para que essa operação pudesse ser efetuada, ajustes no código tiveram que ser feitos devido a erros apresentados pela ferramenta EBQ que não permitia a inserção de dados no formato JSON, tal formato, inspirado em JavaScript, permite o intercâmbio de dados seguindo uma estrutura aninhada usando coleções de pares nome/valor, como um objeto. Esse formato é aplicado no exemplo do cadastro pessoal.

Após a inserção dos dados, verifica-se pela interface Web do BigQuery, oferecida pelo Google, que os dados armazenados realmente foram cifrados e não apresentam nenhuma informação a respeito do conteúdo inserido, de acordo com a Figura 4.3. Na interface oferecida é apresentada uma descrição da tabela contendo o hash da chave

e o próprio esquema cifrado. Em seguida, é descrito um código de identificação da tabela, o tamanho da tabela e a quantidade de linhas. Logo abaixo, é apresentada uma prévia dos campos da tabela, todos criptografados. Vale ressaltar que todo o tipo de tabela, de estrutura plana ou aninhada, é apresentado de forma única, uma visualização plana. A interface Web do BigQuery também oferece a possibilidade de realizar consultas, porém, usando a extensão EBQ, a composição da consulta deve se basear nos valores cifrados dos campos como atributos.

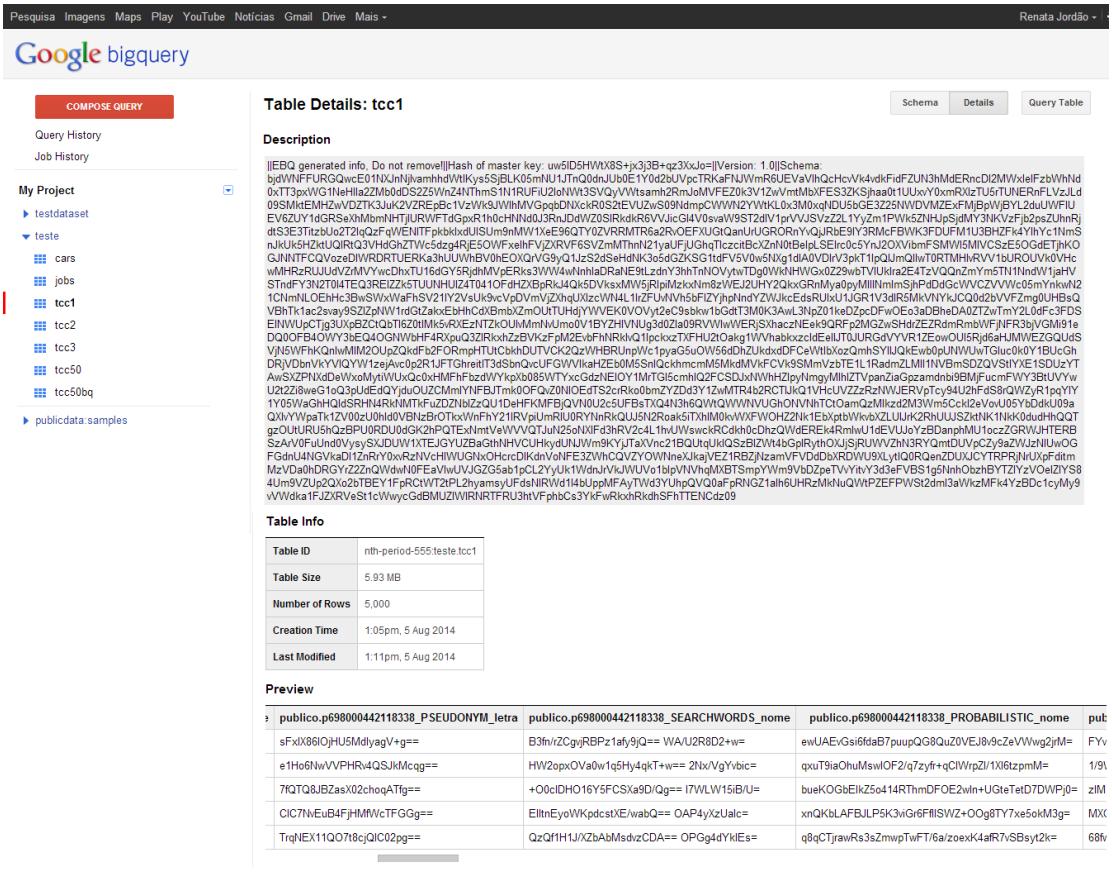

Figura 4.3: Visualização da interface Web do BigQuery.

4.1.2. Realização das consultas

Uma vez carregados os dados, é feita uma bateria de consultas, sob a linguagem SQL (Figura 4.4), com atributos e resultados diversos a fim de avaliar a viabilidade da aplicação dos criptossistemas para grandes volumes de dados.

A primeira consulta realizada consiste de uma operação SELECT simples que requisitava um atributo de cada família de colunas, a saber: o número, a data do cadastro, o nome do cadastrado, sua UF de nascimento, seu cargo e o seu salário.

A segunda rodada de consulta é composta pela clausula WHERE com atributo de igualdade. Essa consulta visa testar a cifra Determinística que admite este tipo e consulta e retorna como resultado o nome e data de nascimento do cadastrado que a UF de nascimento como RJ.

O terceiro teste foi realizado com uma consulta usando a cláusula WHERE com a condição CONTAINS, visando avaliar a eficiência e o resultado da cifra de Busca Probabilística por Palavras, o qual deve retornar a quantidade de filhos do cadastrado quando o mesmo possui um cargo que contém a palavra “advogado”. Como visto anteriormente, a Busca Probabilística por Palavras não prevê a busca por fragmentos, apenas palavras inteiras.

A quarta consulta é feita com a clausula WHERE utilizando um parâmetro sem criptografia que deve retornar os campos de nome, UF de nascimento e data de nascimento de cadastrados com ano de nascimento maior que 1980. Por fim, a última consulta pretende testar o funcionamento da cifra Homomórfica, para isso realiza a consulta solicitando a soma de duas colunas de salários que foram criptografadas com esse criptossistema.

```
root@renata-VirtualBox:/home/renata/encrypted-bigquery-client/ebq# sudo ./ebq.py
 query --master_key_filename=key_tcc11 "SELECT numero, publico.nome,publico.ufna
sc,publico.dtnasc FROM teste.tcc1 where publico.anonasc > 1980"
Waiting on bqjob_r31236394_00000147cbbb6b24_1 ... (0s) Current status: DONE
+--------+--------------+----------------+----------------+
| numero | publico.nome | publico.ufnasc | publico.dtnasc |
+--------+--------------+----------------+----------------+
|      4 |      RICARDO |             ES |       19830319 |
|      6 |    ANASTACIA |             PA |       19850327 |
|      7 |     CUSTODIA |             DF |       19980327 |
|     10 |      GETULIO |             MA |       19980729 |
|     11 |      SILVANO |             AC |       19910510 |
|     15 |       ISRAEL |             MT |       19860419 |
```

Figura 4.4: Exemplo de consulta.

4.2. AVALIAÇÃO DOS RESULTADOS

Baseadas nas tabelas e consultas realizadas, foi avaliado o tempo de resposta que cada consulta EBQ levou, usando como comparação uma tabela sem criptografia com os mesmos registros. Cada processo de consulta foi repetido 50 vezes de forma a obter um tempo médio, uma vez que a maior parte do processamento é realizada

nos servidores do Google em um ambiente não-controlado onde não se pode garantir a expectativa de execução correta e sempre da mesma maneira.

4.2.1. Teste com 5.000 registros

A primeira rodada de testes foi feita executando as consultas em um banco de dados de 5000 registros, o resultado é mostrado na Tabela 4.2.

Tabela 4.2: Tempo médio (em segundos) de resposta a consulta - 5000 registros.

	Consulta simples	**Cláusula WHERE c/ Igualdade**	**Cláusula WHERE - CONTAINS**	**Cláusula WHERE simples**	**Soma Homomórfica**	**Média das consultas**
Sem EBQ	3,96	3,48	3,52	4,24	4,62	3,96
Com EBQ	9,46	5,64	7,3	8,84	12,4	8,73
Variação	238,8%	162%	207,3%	208,5%	268,4%	217%

Conforme a Figura 4.5, as consultas EBQ acrescentam um atraso ao tempo de resposta devido ao volume de dados que os esquemas criptográficos inserem. A consulta simples apresenta valores altos visto que requisitou uma coluna de cada família de colunas. Observa-se também que a consulta de soma homomórfica gera um tempo de resposta grande em relação a consulta sem criptografia, devido ao tempo de processamento da soma das cifras agregado ao tempo de decodificação do resultado na máquina do cliente. Ainda, verifica-se que o uso de criptografia em base de dados aumenta seu tamanho em quase 10 vezes mais que a base de dados comum, pois os valores cifrados são maiores que os valores sem criptografia para alguns esquemas criptográficos, como a cifra homomórfica. Porém, feita em um ambiente propício para tratamento de Big Data, esse aumento não prejudica a eficiência nem a velocidade de consultas dinâmicas.

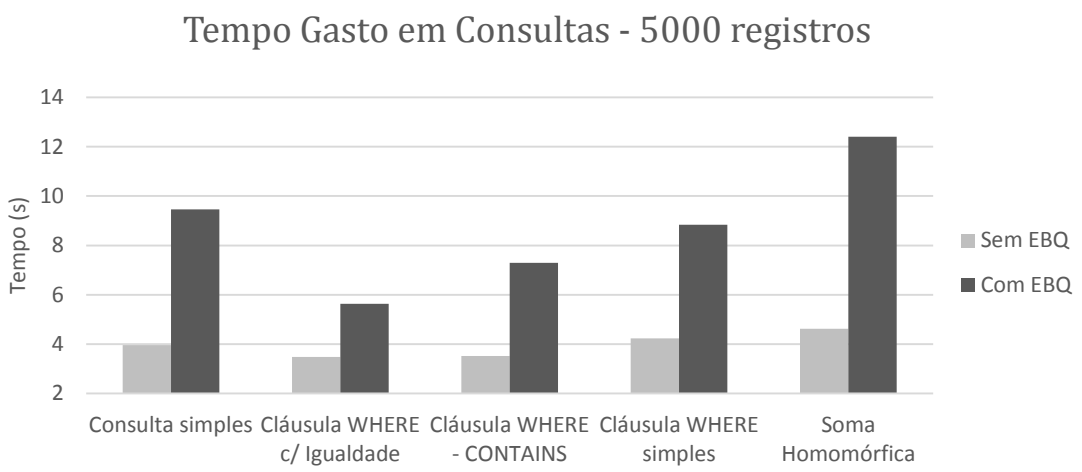

Figura 4.5: Tempo médio (em segundos) gasto em consultas - 5000 registros.

Este argumento pode ser justificado quando a segunda rodada de consultas é feita usando 50000 registros, onde é visto que o aumento no volume de dados não influencia tanto no desempenho de consultas em dados criptografados.

4.2.2. Teste com 50.000 registros

Para o mesmo conjunto de consultas especificadas na seção 4.1.2 foi gerado um banco de dados com um volume de 50000 registros com o objetivo de verificar o grau de impacto que um atraso no tempo de resposta a uma consulta iria causar em um volume maior de dados. Pela Tabela 4.3, nota-se que, mesmo aumentando a quantidade de dados, a margem de atraso permanece aparentemente controlada. A Figura 4.6 confirma essa afirmação apresentando a performance obtida com o número de registros aumentado em 10 vezes o seu carregamento original. Os atrasos encontrados são pequenos se comparados com o tamanho da tabela gerada. Novamente, observa-se que as consultas mais demoradas são aquelas envolvendo a leitura de diferentes famílias de colunas e aquelas que realizam operações em cifras homomórficas, que requerem processamento extra devido a extensão de sua cifra.

Tabela 4.3: Tempo médio (em segundos) de resposta a consulta – 50000 registros.

	Consulta simples	**Cláusula WHERE c/ Igualdade**	**Cláusula WHERE - CONTAINS**	**Cláusula WHERE simples**	**Soma Homomórfica**	**Média das consultas**
Sem EBQ	6,1	5,58	5,04	5,32	5,22	5,45
Com EBQ	9,94	8,68	9,12	8,62	14	10,07
Variação	162,9%	155,5%	180,9%	162%	268,2%	186%

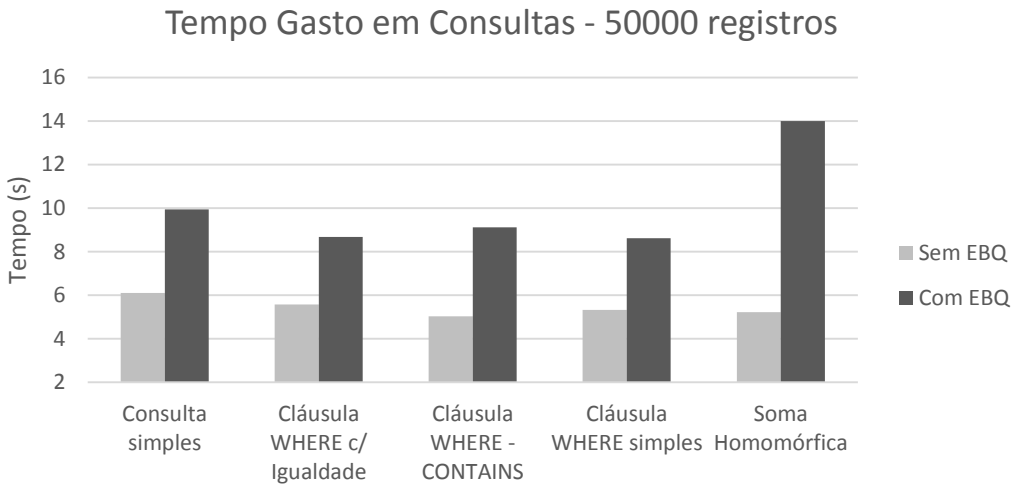

Figura 4.6: Tempo médio (em segundos) gasto em consultas - 50000 registros.

Verifica-se portanto, que a aplicação de consultas diretas sobre grandes volumes de dados criptografados armazenados em ambiente distribuídos, como o caso do EBQ, adiciona um atraso ao intervalo de resposta, porém, a medida que o volume de dados cresce, essa variação se estabiliza e pode ser considerada viável quando esquemas de criptografia eficientes que adicionam uma latência modesta ao todo são adotados.

Baseando-se nos valores de tempo de resposta obtidos, é possível modelar a estrutura de uma base de dados, bem como quais esquemas criptográficos serão usados em cada coluna, de forma eficiente, a fim de otimizar análises de dados de tentativa e erro em grandes conjuntos de dados.

4.2.3. Teste com 100.000 registros e projeções

Realizando uma nova rodada de carregamento de dados, pode-se confirmar a ideia de que o aumento no volume de dados carregados não gera tanto impacto aos atrasos obtidos com o uso de criptografia, como visto na Tabela 4.4 e Figura 4.7.

Tabela 4.4: Tempo médio (em segundos) de resposta a consulta - 100000 registros.

	Consulta simples	**Cláusula WHERE c/ Igualdade**	**Cláusula WHERE - CONTAINS**	**Cláusula WHERE simples**	**Soma Homomórfica**	**Média das consultas**
Sem EBQ	8,96	9,02	7,52	9,66	14,48	9,93
Com EBQ	19,34	19,24	10	10,38	18,88	15,57
Variação	215,8%	213,3%	133%	107,4%	130,4%	160%

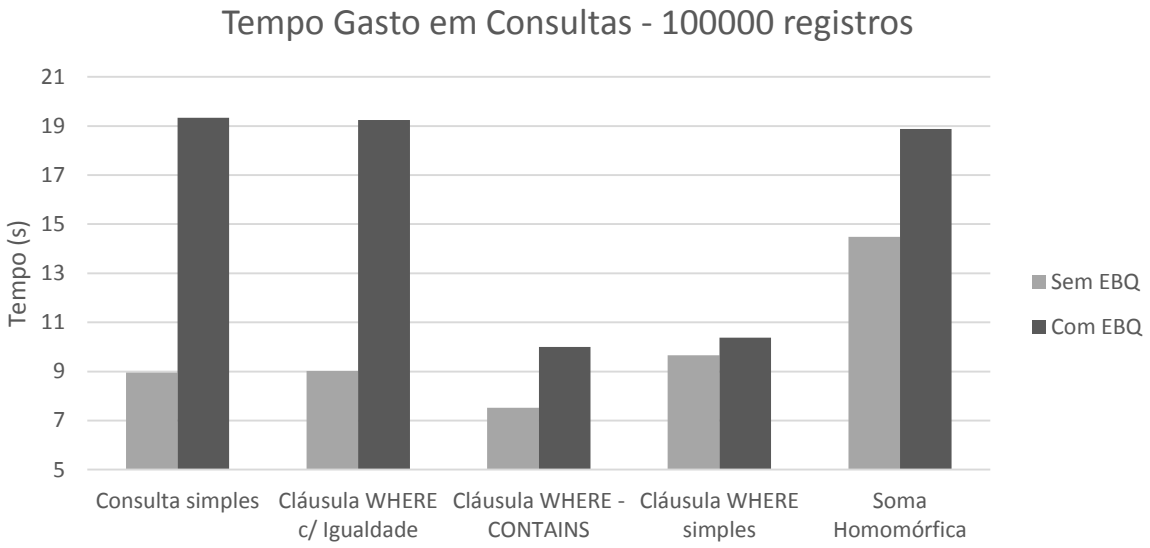

Figura 4.7: Tempo médio (em segundos) gasto em consultas - 100000 registros.

Ainda, fazendo uma projeção utilizando o modelo de consultas diretas em dados armazenados de maneira distribuída, nota-se que o tempo de resposta às consultas tem um comportamento controlado devido ao seu crescimento modesto e quase linear (Figura 4.8). Tal comportamento se deve ao fato de que a criptografia está aliada ao processamento otimizado de grandes volumes de dados, dessa forma, o acréscimo de dados inseridos pela criptografia não influi de maneira tão forte pois a ferramenta

já prevê o processamento de grandes volumes de dados. Contudo, pode-se inferir que essa eficiência possui um ponto de saturação, o qual atinge o volume máximo da capacidade de armazenamento da ferramenta e começa a apresentar valores de atraso que prejudicam o funcionamento da aplicação, o mesmo não está evidenciado nesse trabalho.

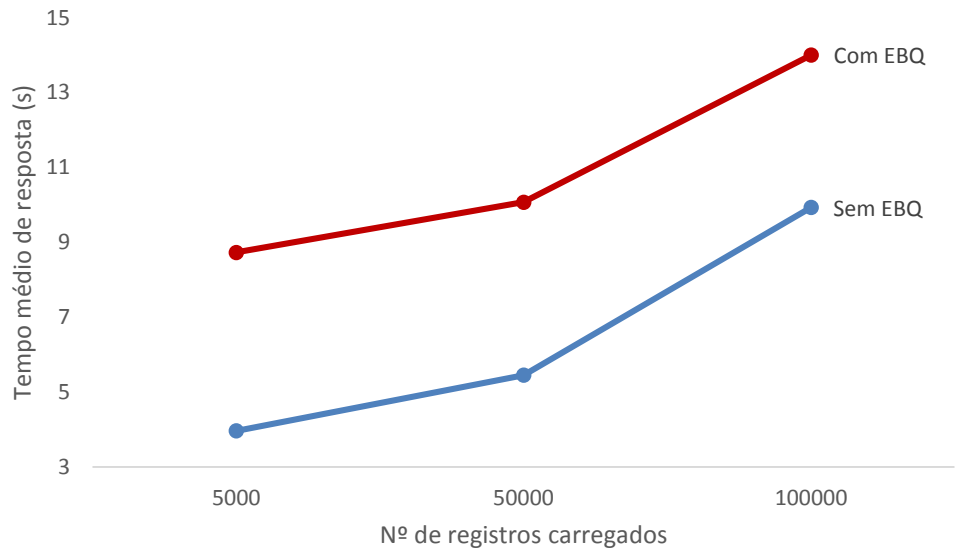

Figura 4.8: Comportamento do tempo de resposta médio com carregamento de vários tamanhos de registro.

4.2.4. Limitações

Mesmo com uma eficiente avaliada, frente a uma latência considerada modesta, a ferramenta utilizada para aplicar os criptossistemas apresentou algumas discrepâncias e limitações em relação ao que está declarado na documentação. Visto que o EBQ roda usando a arquitetura e configurações presentes no BigQuery, a maioria de suas funcionalidades deveriam ser contempladas com a possibilidade do uso de criptografia, porém, o uso de algoritmos criptográficos em consultas diretas limita-se ao resumo da Tabela 4. Muitos predicados de consultas como GROUP BY, DISTINCT, AVG, não são admitidos pela ferramenta, que preza pela simplicidade em suas consultas.

Assim como o BigQuery, a extensão *Encrypted BigQuery* tem dificuldades em permitir a inserção e consulta de tabelas que utilizam o esquema de famílias de colunas. Para processar consultas que utilizam parâmetros de diferentes famílias de colunas, o BigQuery utiliza a função FLATTEN que torna uma das colunas consultadas

plana a fim de que possa ser feita a comparação e a entrega do resultado. Entretanto, essa função ainda não é admitida pela ferramenta EBQ.

Como no processo de carregamento dos dados na base de armazenamento massivo uma única chave é usada para cifrar todos os campos e a mesma armazenada no cliente, uma vulnerabilidade é inserida nesse contexto. Da mesma forma, como o armazenamento da chave é feito somente no cliente que carregou os dados, o acesso de múltiplos usuários capazes de realizar consultas válidas ainda não é previsto pelo EBQ.

Após a exposição das principais limitações encontradas na execução da ferramenta, destaca-se a justificativa de que o *Encrypted BigQuery* trata-se de uma extensão, ainda não totalmente operante, do mecanismo BigQuery de consultas rápidas em grandes volumes de dados. Além disso, aproveita o uso da linguagem SQL para utilizar os algoritmos criptográficos implementados pelo CryptDB, contudo, como a ferramenta não é puramente relacional, algumas funcionalidades se perdem enquanto ganha-se em eficiência e rapidez devido a sua arquitetura NoSQL.

5. CONCLUSÃO E TRABALHOS FUTUROS

Com o crescente número de ameaças e ataques sofridos por banco de dados, a proteção de dados sigilosos e sensíveis tornou-se prioridade para muitas aplicações. Porém, apenas controle de acesso não é o bastante para garantir a confidencialidade e a integridade dos dados, é necessário o uso de criptografia sobre os campos das tabelas. Por esse motivo, foi apresentada neste trabalho a implementação de um modelo de armazenamento de dados criptografados em ambiente distribuído que garantem um meio de acesso comprovadamente seguro, tanto na atualização quanto na consulta a dados, com um mínimo prejuízo adicional no desempenho da recuperação dos dados

A implementação dos cenários e resultados apresentados permitem identificar que a utilização de criptografia em grandes volumes de dados agrega uma carga de dados a mais e um atraso ao tempo de resposta às pesquisas. Entretanto, com o uso de *frameworks* projetados para o tratamento massivo de dados, este gasto extra de tempo e memória não se torna tão relevante, levando em conta a velocidade com que as operações são efetuadas.

Como proposta de trabalhos futuros é sugerida a inserção de uma gama maior de atributos de consultas, muito utilizados por bando de dados tradicionais, como GROUP BY e DISTINCT. Ressalta-se também a importância de uma funcionalidade que possibilite realizar consultas com parâmetros de diferentes famílias de colunas pois trará uma utilidade maior ao serviço de consultas, além de uma eficiência e visão da tabela como um todo.

Além disso, seria interessante a implementação de um servidor seguro de gerenciamento de chaves permitindo que a configuração de multiusuários acesse os dados criptografados. Uma alternativa, seria a modificação do modelo para implementar um proxy que armazenaria as chaves usando o conceito de Criptografia Baseada em Atributo (*Attributed based Encryption* - ABE). Nesse sistema, as chaves de um usuário e mensagens cifradas são classificadas com conjuntos de atributos descritivos e uma chave particular pode decifrar um texto cifrado especial somente se houver uma correspondência entre os atributos do texto cifrado e a chave do usuário.

6. REFERÊNCIAS BIBLIOGÁFICAS

ABADI, D. J., MADDEN, S. R., & HACHEM, N. (2008). Column-stores vs. row-stores: how different are they really? In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 967-980). ACM.

AMANATIDIS, G., BOLDYREVA, A., & O'NEILL, A. (2007). Provably-secure schemes for basic query support in outsourced databases. In Data and Applications Security XXI (pp. 14-30). Springer Berlin Heidelberg.

Apache Hadoop. (s.d.). Acesso em 15 de dezembro de 2013. Disponível em: <http://hadoop.apache.org>.

ARASU, A., BLANAS, S., EGURO, K., KAUSHIK, R., KOSSMANN, D., RAMAMURTHY, R., & VENKATESAN, R. (2013). Orthogonal Security with Cipherbase. In CIDR.

BAO, F., DENG, R. H., DING, X., & YANG, Y. (2008). Private query on encrypted data in multi-user settings. In Information Security Practiceand Experience (pp. 71-85). Springer Berlin Heidelberg.

Big Data Definition - MIKE2.0, the open source methodology for Information Development. Acesso em: 6 de agosto de 2014. Disponível em:< mike2.openmethodology.org>.

BOLDYREVA, A., CHENETTE, N., LEE, Y., & O'NEILL, A. (2009). Order preserving symmetric encryption. In Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). Cologne, Germany.

BONCZ, P. A., ZUKOWSKI, M., & NES, N. (2005). MonetDB/X100: Hyper-Pipelining Query Execution. In CIDR (Vol. 5, pp. 225-237).

BONEH, D., & WATERS, B. (2007). Conjunctive, subset, and range queries on encrypted data. In Theory of cryptography (pp. 535-554). Springer Berlin Heidelberg.

BREWER, E. (2012). CAP twelve years later: How the" rules" have changed. Computer, 45(2), 23-29.

CATTELL, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.

CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., ... & GRUBER, R. E. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 4.

CODD, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377-387.

COONEY, M. (2009). IBM touts encryption innovation; new technology performs calculations on encrypted data without decrypting it. Computer World.

DANG, Q. H. (2012). “Secure Hash Standard (SHS)”, NIST FIPS - 180-1. Acesso em 22 de julho de 2014. Disponível em:<http://www.nist.gov/itl/fipscurrent.cfm>.

ELMASRI, R; & NAVATHE, S.B. (2005). Sistemas de Banco de Dados. Editora: Pearson/Addison Wesley. Edição:4ª.ISBN: 8588639173.

GENTRY, C. (2009). A fully homomorphic encryption scheme. Tese de Doutorado. Stanford University.

GOLDMAN, A., KON, F; JUNIOR, F. P; POLATO, I; PEREIRA, R. DE F.(2012). Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. In: XXXII Congresso da Sociedade Brasileira de Computação, XXXI Jornadas de Atualização em Informática (JAI). Edição: 1ª. Curitiba- PR.

GOLDREICH, O. (2009). Foundations of Cryptography: Volume 2, Basic Applications (Vol. 2). Editora: Cambridge University Press.

HEUSER, CARLOS A. (2001). Projeto de Banco de Dados. Editora: Saga Luzatto. Edição: 4ª.Porto Alegre - RS.

KAHATE, A. (2013). Cryptography and Network Security. Editora: Tata McGraw-Hill, Edição: 3ª. ISBN: 1259029883.

KANTARCIOGLU, M. Modes of Operation. Acesso em 20 de julho de 2014. Disponível em: < http://www.utdallas.edu/~muratk/courses/crypto09s_files/modes.pdf>.

KAUFMAN, C., PERLMAN, R; SPECINER, M. (2002). Network Security: Private Communication in a Public World. Editora: Prentice Hall, Edição: 2ª. ISBN: 9780130460196.

Le COADIC, Y. F. (2004). A Ciência da Informação. Editora: Briquet de Lemos. Edição: 2ª. Brasília-DF.

MELNIK, S., GUBAREV, A., LONG, J. J., ROMER, G., SHIVAKUMAR, S., TOLTON, M., & VASSILAKIS, T. (2010). Dremel: interactive analysis of web-scale datasets. Proceeding of the VLDB Endowment, 3(1-2), 330-339.

OLIVEIRA SANTOS, J. P. DE. (1998). Introdução à teoria dos números. Editora: Instituto de Matemática Pura e Aplicada. Edição: 3ª. ISBN: 8524401427.

Os modelos de SGBD. Acesso em 19 de julho de 2014. Disponível em:<http://pt.kioskea.net/>.

PAILLIER, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Advances in cryptology—EUROCRYPT’99 (pp. 223-238). Springer Berlin Heidelberg.

POPA, R. A., REDFIELD, C., ZELDOVICH, N., & BALAKRISHNAN, H. (2011). Cryptdb: protecting confidentiality with encrypted query processing. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (pp. 85-100). ACM.

POPA, R. A., STARK, E., HELFER, J., VALDEZ, S., ZELDOVICH, N., KAASHOEK, M. F., & BALAKRISHNAN, H. (2014). Building web applications on top of encrypted data using Mylar. In USENIX Symposium of Networked Systems Design and Implementation.

RIVEST, R. "The MD5 Message Digest Algorithm", RFC 1321, abril 1992. Acesso em 22 de julho de 2014. Disponível em: <http://www.ietf.org/rfc/rfc1321.txt >.

RIVEST, R. "The MD5 Message Digest Algorithm", RFC 1320, abril 1992. Acesso em 22 de julho de 2014. Disponível em: <http://www.ietf.org/rfc/rfc1320.txt >.

SATO, K. (2012). “An inside look at google bigquery”. White paper. Acesso em 3 de julho de 2014. Disponível em:<https://cloud.google.com/files/BigQueryTechnicalWP.pdf>.

SONG, D. X., WAGNER, D., & PERRIG, A. (2000). Practical techniques for searches on encrypted data. In Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on (pp. 44-55). IEEE.

STALLINGS, W.(2010). Criptografia e Segurança de Redes: Princípios e Práticas. Editora: Prentice Hall, Edição: 5ª. ISBN: 8576051192. São Paulo - SP.

TAKAI, O.K.; ITALIANO, I.C; FERREIRA, J.E. (2005). Introdução a Banco de Dados. DCC/IME/USP.

TIGANI, J., & NAIDU, S. (2014). Google BigQuery Analytics. Editora: John Wiley & Sons. Edição: 1ª. ISBN: 978-1-118-82482-5.

TU, S., KAASHOEK, M. F., MADDEN, S., & ZELDOVICH, N. (2013). Processing analytical queries over encrypted data. In Proceedings of the VLDB Endowment (Vol. 6, No. 5, pp. 289-300). VLDB Endowment.

Unisys. (2014). Critical Infrastructure: Security Preparedness and Maturity. Independently conducted by Ponemon Institute LLC.

WHITE, T. (2012). Hadoop: The Definitive Guide. Editora: O'Reilly Media/Yahoo Press. Edição: 3ª. ISBN: 9781449311520.