UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Fábio Miranda Reina

VERIFICAÇÃO DE INTEGRIDADE DE BANCO DE DADOS DO TIPO GRAFO

Florianópolis

2017

Fábio Miranda Reina

VERIFICAÇÃO DE INTEGRIDADE DE BANCO DE DADOS DO TIPO GRAFO

Trabalho de Conclusão de Curso submetido ao curso de Ciências da Computação da Universidade Federal de Santa Catarina, como requisito para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador Externo: Hylson Vescovi Netto, Prof. Dr. Ciên. Comp.
Profa. Responsável: Luciana de Oliveira Rech, Profa. Dra.

Florianópolis

2017

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Reina, Fábio Miranda
Verificação de Integridede de Banco de Dados do Tipo Grafo / Fábio Miranda Reina ; orientador, Hylson Vescovi Netto, coorientador, Luciana de Oliveira Rech, 2017.
69 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2017.

Inclui referências.

1. Ciências da Computação. 2. Integridade. 3. Banco de Dados tipo Grafo. 4. Verificação de Integridade. I. Netto, Hylson Vescovi. II. Rech, Luciana de Oliveira. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Fábio Miranda Reina

VERIFICAÇÃO DE INTEGRIDADE DE BANCO DE DADOS DO TIPO GRAFO

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo curso de Ciências da Computação.

Florianópolis, 11 de dezembro 2017.

Hylsón Vescovi Netto, Prof. Dr. Ciên. Comp.
Orientador Externo

Luciana de Oliveira Rech, Profa. Dra.
Profa. Responsável

Banca Examinadora:

Frank Augusto Siqueira, Prof. Dr.

Márcio Bastos Castro, Prof. Dr.

AGRADECIMENTOS

Gostaria de agradecer em primeiro lugar a minha família, em especial meu pai, Leonir Reina, e minha mãe, Audenir S. M. Reina, que sempre me deram apoio e são as pessoas que proporcionaram todas as oportunidades que trouxeram até aqui, e minha irmã, feroz defensora, Adriana M. Reina. Sem eles, o caminho que percorri até este ponto certamente teria sido muito mais espinhoso.

Agradeço também a todas as pessoas que me ajudaram de forma direta ou indireta na realização deste trabalho. Estendo meus cumprimentos aos professores, amigos, companheiros de laboratório, assim como aos colaboradores do LaPeSD que me auxiliaram nas atividades realizadas.

Gostaria de lembrar a memória do Prof. Dr. Eng. Lau Cheuk Lung que aceitou orientar o início da minha pesquisa e agradecer aos professores Luciana de Oliveira Rech, Profa. Dr., e Hylson Vescovi Netto, Prof. Dr. Ciên. Comp, por assumirem a orientação e pelas inúmeras contribuições para o aprimoramento deste trabalho.

"A única falta que terá será a desse tempo
que, infelizmente, nunca mais voltará."
Mário Quintana, "O Tempo"

RESUMO

O volume de dados produzidos tem crescido consideravelmente nos últimos anos, tendo sido responsável por incentivar o desenvolvimento de novas estruturas de armazenamento. Juntamente com a elaboração de novas formas de persistir dados, novos desafios de segurança também surgiram. Dentre os modelos criados, banco de dados baseados em grafos tem se tornado uma opção muito atraente quando comparados aos tradicionais bancos relacionais. Apesar de apresentarem vantagens como a facilidade de modelagem e maior velocidade em consultas complexas, estes novos modelos também precisam de cuidados com a integridade da informação armazenada. Este trabalho apresenta o Guardian, uma solução de fácil aplicação para a verificação da integridade de dados armazenados em bancos do tipo grafo. Na literatura foram encontradas propostas com foco na estrutura do grafo e pouca atenção para a informação. Portanto buscou-se técnicas usadas em modelos tradicionais e escolheu-se como base o trabalho de (SILVERIO, 2014) que aborda o problema em bancos relacionais. Os métodos foram propostos de modo a serem independentes do sistema de banco de dados e utilizam funções criptográficas para gerar valores que auxiliam na detecção de modificações maliciosas da informação. Para isso, para cada nó do grafo são gerados dois valores, *hash* e *cHash*. O primeiro é calculado sobre a concatenação dos atributos do nó e garante sua integridade enquanto o segundo usa o *hash* do próprio nó concatenado com o de seus vizinhos e assegura corretude do grafo como um todo. Para validar a proposta, são apresentados os resultados da execução do Guardian em diversos cenários, mostrando que os custos de tempo de aplicação e o tamanho adicional do grafo são aceitáveis.

Palavras-chave: Banco de dados baseados em grafo. Integridade de dados. Segurança de dados.

ABSTRACT

The volume of new data has been raising considerably in the past years in a way to encourage the development of new storage structures. Along with the elaboration of new ways to persist data, new security challenges also appeared. Between the new models, graph databases have been turning very attractive when compared to the traditional relational databases. In addition to the advantages like easy modeling and faster execution of complex queries, these new models also need care to the stored information. These project presents the Guardian, an easy to implement solution to verify data integrity on graph databases. There were found proposals that give focus on the graph structure and a small attention to the information. For that reason, techniques used on traditional databases were searched and the solution presented by (SILVERIO, 2014) for relational databases was chosen as base for this project. The presented methods were proposed to be independend from the database system and cryptography functions were used to generate values that help to detect malicious modifications on the data. For each node of in the graph two values are generated, *hash* and *cHash*. The first one is calculated over the concatenation of all attributes in the node that garanties its integrity while the second uses the node's *hash* and its neighbors *hash* that garanties the hole graph correctness. To validate the proposed, results from the Guardian's execution under diferent cenarios are presented showing that the time cost and the additional size of the graph are acceptable.

Keywords: Graph database. Data integrity. Data security. Completeness, Correctness.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SQL	Structured Query Language
NOSQL	Not Only SQL
BD	Banco de Dados
HMAC	Keyed-hash Message Authentication Code
MAC	Message Authentication Code
SGBD	sistema de gerenciamento de banco de dados
VO	*Verification Object*
SDAG	Search Diagonal Acyclic Graph
MHT	*Merkle Hash Technique*
DAG	*Directed Acyclic Graph*
CMAC	Chained MAC
MB-Tree	Merkle B-Tree
MD5	Message-Digest algorithm 5
SHA1	Secure Hash Algorithm 1
SHA256	Secure Hash Algorithm 256

SUMÁRIO

1 INTRODUÇÃO

A quantidade de informação gerada atualmente tem crescido a uma velocidade muito alta, de forma que para armazenar todo esse volume de dados, tem-se buscado novas estruturas de armazenamento diferentes dos atuais bancos de dados relacionais. Essa busca por novas tecnologias ocorre devido ao fato de que novos sistemas têm exigido maior robustez e capacidade de suportar acesso simultâneo de diversos usuários. Assim sendo, o BD dessas aplicações precisa atender de forma eficaz um grande número de requisições em paralelo, e assim, os modelos relacionais nem sempre tem conseguido atender tais processos da forma desejada (PEREIRA H P BORGES; UNITRI, 2014).

Essa necessidade de estruturas com melhor desempenho impulsionou o desenvolvimento do conceito do Not Only SQL (NOSQL). O termo foi usado inicialmente em 1998 no trabalho de Carlo Strozzi (STROZZI, 1998), mas somente começou a ganhar força em 2004 quando a empresa Google lançou o banco de dados (BD) BigTable (GOOGLE, 2004). Em seguida outros também surgiram, tais como Cassandra (APACHE, 2007), MongoDB (MONGOBD, 2007) e Neo4J (NEO4J, 2002; WEBBER, 2012).

Dentre os modelos NOSQL, uma vertente que apresenta grande potencial é a de bancos baseados em grafos, os quais buscam e armazenam seus dados na estrutura de vértices e arestas, representando entidades e relações, respectivamente. O NOSQL e a modelagem de bancos baseados em grafos não são tecnologias recentes. Porém seu desenvolvimento foi alavancado pela necessidade de se trabalhar com dados tipicamente estruturados em forma de grafos nos sistemas que apresentaram utilização massiva nos últimos anos, como as redes sociais. Esses sistemas precisam armazenar um grande volume de dados, fato que impulsionou o interesse na área de bancos de dados em grafos (ERVEN, 2015).

Apesar da maior eficácia no armazenamento, BDs em grafos herdaram alguns problemas das usuais estruturas relacionais normalmente usadas. Uma das questões mais importantes é a da garantia de segurança da informação. Tal fato se agrava quando considera-se que todo dado produzido precisa ser armazenado em algum banco e muitas vezes não se tem disponíveis os recursos necessários, sejam eles *hardware*, *software* ou humano. Conforme apresentado por (SILVERIO, 2014), é cada vez mais comum a terceirização de recursos de armazenamento, como por exemplo os serviços de armazenamento em nuvem. Tais pro-

vedores fornecem confiabilidade ao dado, no sentido de manter o armazenamento e garantir a recuperação dos dados. Porém, a manipulação da informação e questões de integridade do dado não são fornecidas por provedores em nuvem. Dessa forma se faz necessária a existência de mecanismos que garantam a segurança do manuseio dos dados assim como sua integridade.

A verificação e a garantia de que uma consulta à base de dados retornará a informação completa e totalmente correta ainda é um problema bastante comum a todo o tipo de BD, seja ele relacional ou baseado em NOSQL. Neste caso, se nenhuma verificação da informação é feita pode-se estar assumindo o risco de que os dados tenham sofrido alterações, maliciosamente ou não.

Existem abordagens para tratar esse problema mas, em geral, não se garante que todo tipo de alteração é identificada, a menos que se combine mais de uma proposta. Algumas ideias de tratamento detectam apenas inserções e deleções, enquanto outras identificam apenas atualizações, e mesmo que existam técnicas de conferência dos dados, nada garante que tais medidas são realmente aplicadas pelos provedores de serviços de armazenamento. Dessa forma, as informações ficam bastante vulneráveis a ataques, enquanto os usuários perdem a certeza de que seus dados estão corretos.

O presente trabalho apresenta o Guardian (***Gr**aph **D**atabase **I**ntegrity Verific**a**ti**o**n*), um sistema desenvolvido para garantir a integridade de dados em bancos de dados do tipo grafo. O Guardian foi especificado e implementado sobre o Neo4J, um banco de dados em grafos conhecido e disponível.

1.1 MOTIVAÇÃO

Devido à popularização dos serviços de redes sociais e também dos smartphones, tornou-se muito fácil e cada vez maior a produção de dados, assim como o uso de serviços de armazenamento remoto, na nuvem. Por esse motivo, empresas prestadoras desse serviço enfrentam os desafios de gerenciar, extrair informações úteis (HO et al., 2012) e principalmente armazenar todo este conteúdo de forma segura.

A principal motivação deste trabalho está no problema de como verificar e garantir a integridade da informação retornada ao fazer uma consulta à base de dados. Para isso é possível encontrar diferentes técnicas já desenvolvidas, como as apresentadas por (ARSHAD et al., 2014) e (SILVERIO, 2014), entre outros. No entanto, estas não são

medidas que um usuário comum pode aplicar, já que muitas vezes sua implantação implica em modificações no modo como os BDs ou gerenciadores de BD tratam os dados. Isto é, são implementações que devem ser feitas pelos desenvolvedores dos bancos.

Pode-se citar também como oportunidade de desenvolvimento a ausência de uma solução que apresente as três principais operações de um BD, sendo elas inserção, deleção e atualização. Grande parte das propostas existentes tratam apenas uma ou duas dessas operações, de modo que para cobrir todas elas é preciso combinar mais de uma técnica. Dessa forma, o usuário final acaba tendo poucas opções de ação, o que o obriga a confiar que o banco usado em sua aplicação toma todas a medidas necessárias para a proteção dos dados.

Além da demanda de uma solução completa, outro motivo que impulsiona o desenvolvimento do presente trabalho é que as técnicas atualmente disponíveis são direcionadas a BDs relacionais e pouco se encontra na literatura a respeito de BDs do tipo grafo, como em (ARSHAD et al., 2014), (ERVEN, 2015), (ANGELES; GUTIERREZ, 2008), (HO et al., 2012). Esse tipo de banco possui os mesmos problemas de segurança e integridade já citados mas em muitos dos casos não é possível reaproveitar os processos usados em BDs relacionais, já que se tratam de estruturas completamente diferentes. Dentre as diferenças, pode-se citar a ocorrência de subestruturas desconexas no grafo, isto é, o grafo que estrutura o banco está dividido em dois ou mais subgrafos que não possuem qualquer relação entre si. Um exemplo desse cenário é uma rede social na qual existem pelo menos dois grupos de amigos tal que nenhum integrante de um grupo conhece alguém do outro grupo.

1.2 OBJETIVOS

1.2.1 Objetivos Gerais

O presente trabalho tem como propósito desenvolver um sistema que garanta a integridade de dados armazenados em grafos.

1.2.2 Objetivos Específicos

Os objetivos específicos são:

- Especificar um algoritmo para verificar a integridade de grafos conexos em BDs do tipo grafo;

- Disponibilizar uma técnica para modificar grafos desconexos de modo a torná-los compatíveis com a verificação de integridade de grafos conexos;

- Desenvolver e implementar protótipo do Guardian, a fim de verificar sua viabilidade.

- Validar o sistema Guardian por meio de cenários de teste relevantes.

1.3 METODOLOGIA

O presente trabalho iniciou-se pela constatação de que não havia, até o momento, solução para prover o mecanismos de segurança no contexto de BDs em grafos, como a desenvolvida no trabalho de (SILVERIO, 2014).

A partir de então iniciou-se o estudo de artigos relacionados à garantia de integridade da informação armazenada em bancos de dados. Como o foco escolhido foi BDs do tipo grafo, buscou-se textos e relatórios técnicos que usassem este tipo de estrutura, no entanto, o material encontrado trata mais da integridade e da confidencialidade da estrutura ao invés de se preocupar com os dados armazenados.

A fim de encontrar soluções para aplicar na informação, estudou-se técnicas usadas em BDs convencionais, que usam a estrutura relacional, com o objetivo de encontrar métodos que pudessem ser adaptados para o modelo de grafos de forma eficiente e de fácil implantação. Em complemento, escolheu-se a proposta apresentada por (SILVERIO, 2014) para ser trabalhada e desenvolvida para BDs do tipo grafo.

Para a verificação da viabilidade do projeto, foi desenvolvido um protótipo do Guardian além de executados alguns cenários de teste. A partir dos resultados obtidos, foi possível aperfeiçoar a implementação, bem como obter análise utilizada para demonstrar eficiência e também comparar com técnicas correlatas.

Por fim elaborou-se as considerações finais reafirmando o benefício do uso da proposta desenvolvida, bem como a apresentação de trabalhos futuros.

1.4 ORGANIZAÇÃO DO TRABALHO

As próximas seções deste documento estão organizadas da seguinte maneira: no Capítulo 2 são apresentados os conceitos mais relevantes ao trabalho; no Capítulo 3 são discutidos trabalhos relacionados que proporcionam uma visão geral do estado da arte em segurança de banco de dados; no Capítulo 4 é apresentado o sistema Guardian. O Capítulo 5 relata as avaliações realizadas. Finalmente, no Capítulo 6 são apresentadas as considerações finais do trabalho.

2 CONCEITOS FUNDAMENTAIS

Neste capítulo são apresentados definições e conceitos fundamentais usados no desenvolvimento do presente trabalho.

2.1 GRAFOS

Segundo a definição de grafos em (GOLUMBIC, 2004), um grafo $G(V, E)$ consiste de um conjunto não finito de vértices V e uma relação binária irreflexiva em V, que pode ser representada por um conjunto de pares ordenados E ou como uma função de V em seu conjunto potência, $Adj : V \rightarrow P(V)$, tal que $Adj(v)$ é o conjunto adjacência dos vértices v, e o par ordenado (v, w) e E representa uma aresta de v para w. Assim sendo, dado um conjunto $V = v_1, v_2, ..., v_n$, as possíveis relações sobre este conjunto são $E = (v_1, v_2), (v_1, v_3), ..., (v_n - 1, v_n)$, e cada par ordenado é dito arco ou aresta.

Dado um grafo $G(V, E)$, define-se ordem com sendo o número de vértice que compõem o conjunto V, ou seja $ordem = |V|$. Da mesma forma, denota-se também tamanho de um grafo como sendo o número de arestas que fazem parte do conjunto E, isto é, $tamanho = |E|$. Outra definição importante é o grau de um vértice, determinado pelo número de arestas ligadas à este vértice. No caso de um grafo direcionado, esta definição pode ser dividida em grau de entrada, dado pelo número de arestas que chegam ao vértice, e grau de saída, dado pelo número de arestas que partem dele (ERVEN, 2015).

Ao aplicar essa teoria aos BDs do tipo grafo, é comum fazer uso dos termos *nó* e *relação* para se referir respectivamente a um vértice e um aresta do grafo que estrutura o banco.

2.2 INTEGRIDADE DE DADOS

De acordo com (MAO, 2003) a garantia de integridade é um problema antigo da criptografia. Erros podem ser inseridos na mensagem de diversas origens, seja de forma maliciosa como no caso de um ataque ou mesmo por algum problema nas redes de comunicação por onde as informações são transmitidas.

Independentemente da fonte de erro, utilizar dados alterados é perigoso e pode gerar grandes falhas no sistema. O princípio básico

para a garantia de integridade é codificar a mensagem com uma chave conhecida apenas pelo seu emissor e destinatário. Dessa forma a decodificação só é possível para aqueles que conhecem o código usado. Por outro lado, ainda existe a possibilidade de um atacante conseguir gerar uma codificação válida para a mensagem em questão (SILVERIO, 2014).

Os principais aspectos para se garantir integridade são:

- **Corretude:** garante que a informação não foi modificada.
- **Completude** garante que o retorno de uma *query* ao banco não possui informação omitida.
- **Atualidade** garante que o dado retornado está em sua versão mais atual.

Dentre as possíveis formas de se garantir a integridade de uma mensagem transmitida está a implantação de um sistema de canal duplo em que um deles é um meio simples e inseguro, como a internet, e o outro é seguro, livre de interferências. Neste caso a mensagem é enviada duas vezes e o receptor pode comparar as duas para verificar se ambas são idênticas e se houve alteração de dados ou não. Como o custo de implantação de um canal seguro é elevado, o que ocorre na prática é o envio do valor de verificação juntamente com a mensagem usando o mesmo canal simples (SILVERIO, 2014).

A Figura 1 mostra a verificação de integridade com o uso de um canal único para a transmissão da mensagem. No exemplo tem-se que o emissor, antes de realizar a transmissão, codifica a mensagem original m com a função *C()* usando a chave de codificação k_c. No caminho para o receptor, a mensagem codificada pode sofrer interferência, caracterizada pela função *I()*, de um atacante ou mesmo do próprio meio. Por fim, o receptor recebe a mensagem *m'* cuja integridade é verificada pela função *V()* usando a chave de verificação k_v. Assim sendo:

- m é a mensagem original;
- m' é a mensagem recebida;
- $C()$ é a função de codificação da mensagem
- $I()$ é uma função que representa a interferência sobre a mensagem, e sua origem pode ser de um atacante ou do próprio meio;
- $V()$ é a função de verificação da mensagem;
- k_c, k_v são as chaves de codificação e verificação respectivamente.

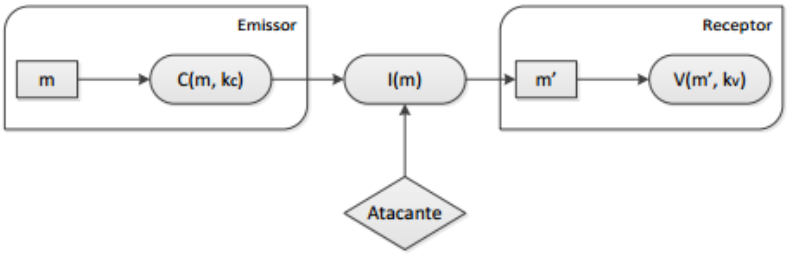


Figura 1 – Modelo de canal simples (SILVERIO, 2014)

2.3 TÉCNICAS DE CRIPTOGRAFIA

As técnicas de criptografia simétricas são os modelos mais tradicionais de codificação de mensagens. Nesse tipo, as chaves k_c e k_v trocadas entre remetente e destinatário são exatamente as mesmas e devem ser mantidas secretas. Dentre as funções existentes pode-se citar Funções *Hash* e *Message Authentication Code*, que serão detalhadas na seção a seguir (AMARO, 2007).

2.3.1 Funções de *Hash*

Uma função *hash* é um algoritmo matemático que mapeia uma *string* de tamanho variável para uma *string* diferente de tamanho fixo (SCHNEIER, 1996). É também dita como uma função de único caminho (*one-way function*), isto é, uma função fácil de computar sobre qualquer entrada mas muito difícil de inverter dado uma imagem qualquer (ARSHAD et al., 2014).

Para uma função *hash* ser considerada segura é preciso satisfazer as seguintes propriedades:

- **Eficiência:** o cálculo deve ocorrer em tempo polinomial, ou seja, deve ser rápido e simples.
- **Resistência à pré-imagem:** dado um valor de *hash* h, deve ser inviável encontrar uma imagem x tal que $f(x) = h$, onde $f()$ é a função de cálculo do *hash*.
- **Resistência à colisão:** duas mensagens diferentes não devem

resultar em um mesmo valor de *hash*, depois de ser aplicada a função $f()$.

2.3.2 *Keyed-hash Message Authentication Code*

As funções de *hash* proporcionam um meio de verificação da integridade de uma mensagem transmitida ou armazenada em um meio não confiável. Porém, há casos em que, além da integridade, é preciso também verificar a autenticidade da informação. Para esses casos existe uma forma de realizar tal verificação, chamada *Message Authentication Code* (MAC), que usa uma chave secreta (KRAWCZYK et al., 1997).

Tipicamente, MACs são usados por usuários que compartilham a chave secreta a fim de validar as mensagens trocadas entre eles. Para isso faz-se uso de um método baseado em funções *hash* chamado de *Keyed-hash Message Authentication Code* (HMAC) que aplicam as funções de *hash*, como apresentado na Seção 2.3.1, e adicionam um novo parâmetro, a chave secreta.

A equação 2.1 esquematiza o cálculo do MAC, de modo que h é a função de *hash*, k é a chave secreta, m é a mensagem e $||$ representa a operação de concatenação (SILVERIO, 2014).

$$MAC = h(k||m) \tag{2.1}$$

A equação 2.2, conforme apresentado por (KRAWCZYK et al., 1997), esquematiza o cálculo da função HMAC.

$$HMAC(k, m) = h((k \oplus opad)||h(k \oplus ipad||m)) \tag{2.2}$$

Os componentes da equação 2.2 estão dispostos a seguir.

- ***h*** é a função de *hash*;
- ***k*** é a chave secreta;
- ***m*** é a mensagem;
- ***opad*** (do inglês, *outer padding*) é a constante *0x5c5c5c. . . 5c5c*, definida pela RFC2104;
- ***ipad*** (do inglês, *inner padding*) é a constante *0x363636. . . 3636*, definida pela RFC2104;
- $\oplus$ é a operação de *ou exclusivo*;
- $||$ é a operação de concatenação.

2.4 RESUMO DO CAPÍTULO

Este capítulo apresenta os conceitos básicos para a compreensão do trabalho, além de descrever os principais aspectos e técnicas para se tratar a verificação de integridade da informação.

Inicialmente é definido o conceito de grafo, já que os BDs alvos do foco deste trabalho são construídos usando esta estrutura de dados. Em seguida, é apresentado o conceito de integridade, como foi usado no projeto. Posteriormente são descritas técnicas de criptografia simétrica e as funções de *hash* e HMAC. Essas técnicas e funções estão entres os modelos mais tradicionais de codificação de mensagens e estão presentes em diversos sistemas usados no dia-a-dia.

3 TRABALHOS RELACIONADOS

3.1 INTRODUÇÃO

Alguns esquemas de segurança em banco de dados, tais como estruturas autenticadas baseadas em *Merkle Hash Trees* e *Skip-Lists*, foram propostos a fim de verificar a integridade da informação . Essas técnicas tentam em geral solucionar um dos seguintes aspectos da informação (SILVERIO et al., 2014):

- *Corretude:* do ponto de vista da integridade, corretude significa que a informação não sofreu qualquer tipo de alteração.
- *Completude:* significa que todas as possíveis tuplas que satisfazem a consulta feita pelo usuário são retornadas pelo servidor.

A maioria dessas técnicas depende da modificação do núcleo dos bancos ou do desenvolvimento de um novo Sistema de Gerenciamento de Banco de Dados (SGBD), o que dificulta bastante a utilização de mecanismos de garantia de integridade em cenários do mundo real. Esse contratempo se torna ainda mais evidente quando se trata de adicionar proteção de integridade a sistemas que já estão em uso.

Outra parte dos trabalhos usa estruturas autenticadas com base em *Merkle Hash Trees* e *Skip-Lists*. A adoção dessas propostas é bem mais viável, já que não requer mudanças no núcleo do BD; no entanto, se limita a bancos estáticos. Assim sendo, estruturas autenticadas são pouco eficientes para bancos dinâmicos, pois exigem que a estrutura seja recalculada após cada atualização.

Além dessas limitações, grande parte dessas técnicas é aplicável apenas em BDs relacionais. No contexto de bancos de dados em grafos, pouco se consegue aproveitar dos métodos existentes devido em especial à grande diferença de estrutura entre os bancos de dados relacionais e os BDs em grafos.

3.2 DETECÇÃO DE VIOLAÇÕES DE INTEGRIDADE DE TABELAS RELACIONAIS COM AUTENTICAÇÃO DE MENSAGENS

O principal trabalho, que deu base ao presente, é a abordagem apresentada por (SILVERIO, 2014) em sua dissertação. Nela o autor implementa uma forma de verificação para banco de dados relacionais.

Sua proposta consiste em adicionar em cada tabela uma nova coluna responsável por armazenar o resultado da função MAC aplicada sobre a concatenação do conteúdo de cada linha segundo a formula $MAC_n = MAC(k, Coluna_1||...||Coluna_i)$, onde k é uma chave conhecida apenas pela aplicação (Figura 2).

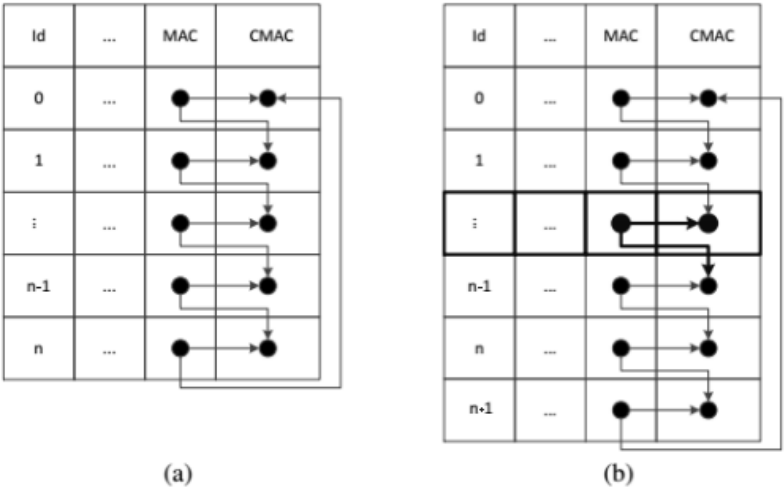


Figura 2 – Representação de tabela com coluna CMAC: (a) corrente circular gerada após a criação das colunas MAC e CMAC e (b) representação com uma nova linha incluída. Fonte: (SILVERIO, 2014).

O uso do MAC assegura a integridade de operações do tipo inserção e atualização, mas ainda deixa a tabela vulnerável a deleções não autorizadas. Como solução é proposto um algoritmo que liga uma linha à sua sucessora, chamado *Chained-MAC* (CMAC), cujo resultado é armazenado em uma nova coluna e calculado para a linha n segundo a fórmula $CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n))$, tal que $\oplus$ corresponde à operação XOR (ou exclusivo).

Dessa maneira, o CMAC conecta as linhas de uma forma que um atacante não consegue apagar uma linha sem ser detectado, já que ele não tem conhecimento da chave secreta para produzir novos valores válidos para as linhas adjacentes.

As operações são bastante eficientes já que são calculados apenas dois MACs e uma operação XOR com esses dois valores. A atualização dos valores, por exemplo, não impacta na eficiência. Outro ponto

importante é que o CMAC não é uma operação em cascata, ou seja, só precisa ser calculado quando o valor MAC de uma determinada linha é alterado. Na geração do valor da coluna MAC o conteúdo concatenado da linha em questão é usado na função, enquanto para a geração do valor da coluna CMAC são usados os MACs da mesma linha e o da linha anterior (Figura 2-a).

Apesar de linhas adjacentes estarem protegidas, a primeira e a última linhas continuam vulneráveis a ataques de deleção. Isso é possível pois a primeira linha não tem uma anterior, assim como a última não tem uma subsequente para se ligar. Dessa forma, é proposto que se faça um CMAC circular, de modo que a última linha se ligue à primeira.

As operações são:

- *Adição de linhas:* ocorre de forma direta. O cliente deve calcular o MAC da linha e em seguida seu valor de CMAC usando os MACs da linha anterior e o da linha que está sendo adicionada (Figura 2-b).

- *Atualização de linhas:* semelhante à inserções, os valores de MAC e CMAC devem ser recalculados e em seguida deve-se atualizar o CMAC da linha subsequente.

- *Deleção de linhas:* Ao deletar uma linha também deve se atualizar o valor de CMAC da linha subsequente usando o valor da linha anterior à excluída.

- *Verificação de integridade:* pode ocorrer em diferentes granularidades, desde que os valores de MAC e CMAC dos grupos desejados estejam preenchidos.

- *Verificação de completude:* é possível devido aos valores de CMAC que ligam uma linha a outra. Portanto se algum dados se perder, a verificação do *hash* falha.

Outros trabalhos voltados para BDs relacionais encontrados na literatura são baseados em estruturas autenticadas: *Merkle Hash Trees* e *Skip-Lists*. O trabalho de (LI et al., 2007) utiliza uma extensão de árvores B^+ com informações resumidas, nomeada como *Merkle B-Tree* (MB-Tree). Esta MB-Tree é usada para fornecer prova de corretude e completude da informação. Apesar de apresentar bons resultados, para que a ideia seja implantada o BD precisa ser adaptado, assim como a árvore B+ precisa ser expandida para suportar uma *Merkle Hash Tree*.

A proposta de (BATTISTA; PALAZZI, 2006) fala de se implementar uma *Skip-List* autenticada em uma tabela relacional. Neste caso cria-se uma nova tabela (*security table*) que armazena uma *Skip-List* autenticada que é usada para assegurar autenticidade e completude para as consultas realizadas. Esta opção foge do problema de se desenvolver um novo SGBD, no entanto seus resultados são superficiais e não há como avaliar a real sobrecarga em termos de operações SQL.

Pode-se citar também o trabalho de (MIKLAU; SUCIU, 2005), que implementa uma árvore *hash* em uma tabela relacional, proporcionando verificações de integridade. Para isso o cliente precisa reconstruir a árvore e comparar o nó raiz calculado com o armazenado anteriormente. Apesar de se usar funções simples de criptografia, a construção da árvore compromete a eficiência do método.

3.3 PROTEÇÃO DE INTEGRIDADE DE GRAFOS VIA CRIPTOGRAFIA DE ÁRVORE DE TRAVESSIA

Assim como em bancos relacionais, ao trabalhar com um novo modelo, frequentemente é necessário checar se uma dada estrutura foi atualizada ou não ou mesmo se dois grafos são idênticos. No entanto, propostas similares à apresentada na Seção 3.2 ainda não foram encontradas na literatura, no contexto de BD do tipo grafo. O que existe está mais voltado para a criptografia dos dados e a garantia de sigilo da informação no caso de compartilhamento de parte do grafo. Para isso funções de *hash* são usadas.

No caso de esquemas de *hash* como SHA1 e SHA2, uma mensagem é compartilhada por inteiro ou simplesmente não é compartilhada. Por outro lado, quando se trata de um grafo, segundo (ARSHAD et al., 2014), pode-se compartilhar apenas partes dessa estrutura, ou seja, apenas um subgrafo. Neste caso, esquemas tradicionais de codificação permitem o vazamento de informações acerca do restante do grafo. Isto ocorre pois junto com o subgrafo, são enviados Objetos de Verificação (VO - *Verification Objects*), usados na validação da integridade, que acabam revelando informações sobre os dados não transmitidos. Com isso, o desafio está em fazer o *hash* das partes não compartilhadas do grafo (ARSHAD et al., 2014).

O trabalho de (ARSHAD et al., 2014) apresenta este tipo de estudo. Nele o autor apresenta uma técnica para criptografar o grafo a partir de sua árvore de travessia gerada. A árvore é construída por meio de uma busca em profundidade no grafo em que são atribuídos

valores de pré-ordem e pós-ordem para cada nó visitado e então gera-se um valor *hash* resistente a colisões para a estrutura resultante.

Um dos desafios para a proposta de (ARSHAD et al., 2014) é que esta não suporta a dinamicidade do grafo. Assim, o *hash* precisa ser recalculado para toda a estrutura no caso de qualquer alteração.

O autor ressalta as três propriedades de uma função *hash*: eficiência, resistência a pre-imagem e resistência à colisões. A Seção 2.3.1 do presente trabalho descreve essas propriedades, além de apresentar algoritmos para a geração de esquemas *hash* de grafos.

3.4 OUTRAS TÉCNICAS

A técnica de *Merkle Hash*, do inglês *Merkle Hash Technique* (MHT), tem sido proposta para árvores e estendidas para grafos direcionados acíclicos. Outro método é o *Search Diagonal Acyclic Graph* (SDAG), também baseado em *Merkle Hash Trees*. Esses métodos assumem que atualizações não transformarão a estrutura de árvore em um grafo não-árvore, de forma que se limitam as operações que podem ser executadas em um banco do tipo grafo.

Sem um método mais genérico e eficiente, o que pode ser feito atualmente é determinar o tipo de grafo a ser feito o *hash* e então determinar o método mais aplicável.

3.4.1 Função de *hash* em Árvores Usando *Merkle Hash Technique* (MTH)

A MTH é uma técnica que pode ser aplicada em estruturas do tipo árvore, grafos acíclicos e direcionados. Dada a árvore, esta é processada das folhas para as raízes. Dado um nó x da árvore $T(V, E)$, se x é uma folha, então $mh(x) = H(c_x)$, caso contrário, $mh(x) = H(mh(y_1), ..., mh(y_n))$ tal que y_n's são filhos de x (MERKLE, 1989).

Considerando a árvore da Figura 3, os valores de $mh(e)$ e $mh(f)$ são calculados por $H(c_e)$ e $H(c_f)$ respectivamente. Esses resultados são então usados para computar $mh(d) = H(mh(e)||mh(f))$. Em seguida $mh(b)$ é obtido por $H(mh(d))$. Por fim obtém-se $mh(c) = H(c_c)$ e $mh(a) = H(mh(b)||mh(c))$.

No caso de compartilhamento de uma subárvore, é também compartilhado um conjunto de Objetos de Verificação (VO) que são usados

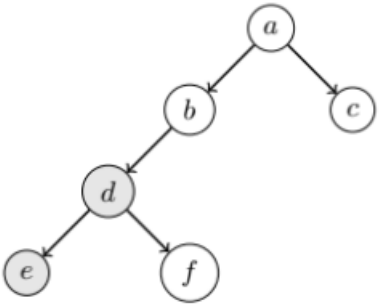


Figura 3 – Exemplo de grafo em forma de árvore. Os nós sombreados destacam um subgrafo da estrutura. Fonte: (ARSHAD et al., 2014).

junto à subárvore para calcular a função $mh()$ de toda a árvore e então comparar com o original (MARTEL et al., 2001). Esse conjunto de VO consiste em uma exposição de informações acerca da parte não compartilhada do grafo.

3.4.2 Função de *hash* em Grafos

O desafio em se fazer o *hashing* de grafos vem dos fatos de que: i) um grafo pode ter ciclos, ii) mudanças nos nós podem afetar muitos outros nós, e iii) um grafo pode ser compartilhado apenas parcialmente em termos de subgrafos. Em outras palavras, há a possibilidade de se partilhar apenas um subgrafo de interesse ao invés de se revelar todos os nós do grafo, conforme discutido no trabalho de (ARSHAD et al., 2014).

Qualquer atualização de um nó implica na alteração do *hash* de todos os nó relacionados ao primeiro. Em geral em um SDAG $G(V, E)$, uma modificação em um nó afeta $V + E$ nós (MARTEL et al., 2001). A Figura 4 mostra um grafo acíclico direto (do inglês *Directed Acyclic Graph* - DAG) que ilustra essa situação. No grafo apresentado, o nó f tem três arestas de entrada. Ao se aplicar a técnica SDAG, o *hash* de f influencia os *hashes* dos três outros nós ligados a ele, b, c e d. Nesse caso qualquer alteração em f afeta todos os nós exceto e.

Fazer o *hashing* de grafos com ciclos é ainda mais complexo, visto que grafos não podem ser ordenados topologicamente. Isso ocorre pois a ordenação topológica é uma ordem linear em que cada nó aparece

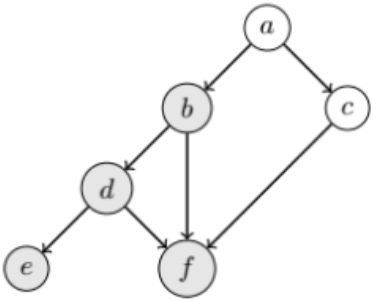


Figura 4 – Representação de um DAG. Os nós sombreados destacam um subgrafo da estrutura. Fonte: (ARSHAD et al., 2014).

antes de todos os nós para os quais este tenha uma aresta de saída. No caso dos grafos da Figura 5, o ciclo envolve os nós *a*, *b*, *c*, *d* e *f*. A dificuldade então surge em como fazer o *hash* desses grafos de modo a preservar o ciclo da estrutura.

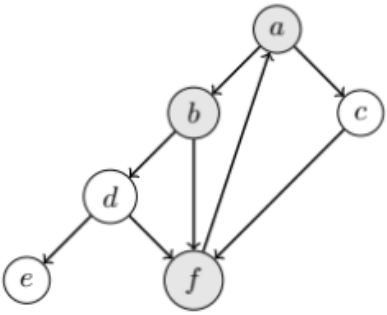


Figura 5 – Representação de um grafo cíclico. O ciclo envolve os nós *a*, *b*, *c*, *d* e *f*. Os nós sombreados destacam um subgrafo da estrutura. Fonte: (ARSHAD et al., 2014).

3.5 RESUMO DO CAPÍTULO

Neste capítulo foi apresentada e discutida a forma como o problema da verificação da integridade de dados é tratado em trabalhos

correlatos. Verificou-se que a maioria das técnicas depende de modificações nos BDs que vão além do que usuários podem realizar. Apresentou-se também um outro grupo de trabalhos, mais flexíveis quanto à aplicação, no entanto pouco eficientes quando implantados em grafos dinâmicos. Por fim, destacou-se o pouco aproveitamento das técnicas existentes, já que estas são, em sua maioria, direcionadas a BDs relacionais e não a bancos do tipo grafo.

4 MÉTODO PROPOSTO

O objetivo deste trabalho é garantir a corretude de dados em BDs do tipo grafo, identificando a ocorrência de modificações não autorizadas nas informações ou na estrutura do BD. É necessário também garantir a integridade da estrutura do grafo, além de lidar com as diferentes características dos grafos. Tais recursos são disponibilizados no sistema denominado Guardian (***Gr**aph **D**atabase **I**ntegrity Verific**a**ti**o**n*), especificado e descrito a seguir. Conforme definido na Seção 2.1, um grafo é composto por vértices e arestas que também podem ser chamados de nós e relações respectivamente.

O Guardian calcula um valor de *hash* (H_{node}) a partir da concatenação dos atributos de um nó, e armazena o resultado como um novo atributo. Dessa forma pode-se identificar a ocorrência de atualizações não autorizadas. Apesar de já verificar a completude dos dados em cada nó do grafo, a estrutura ainda fica vulnerável ao ataque de inserção e/ou deleção de vértices e arestas. Em outras palavras, um atacante poderia remover uma entidade do banco de forma que o sistema não detectasse a diferença.

Ressalta-se que a função de criptografia escolhida para a geração dos valores de *hash* e *cHash* deve ser uma *Keyed-hash Function*, como a apresentada na Seção 2.3.2. Em outras palavras, a função de criptografia deve utilizar uma chave privada na codificação dos valores, provendo segurança, já que apenas pessoas autorizadas terão conhecimento dessa chave.

A fim de evitar o ataque de inserção e/ou deleção, calcula-se um novo valor de *hash* (o *cHash*) a partir da concatenação do H_{node} do nó atual com os H_{node} de cada um dos nós vizinhos (Figura 6). Dessa forma cria-se uma corrente ligando todos os nós de modo que se um nó for deletado ou inserido, ou uma relação feita ou apagada, o *cHash* das entidades vizinhas precisa ser atualizado também para que a autenticação continue correta. Caso contrário a verificação da integridade irá falhar na comparação com o *hash* verdadeiro, já que o atacante não terá a chave privada para recalcular e atualizar os valores necessários. Já na ocorrência de uma mudança autorizada apenas os valores de *cHash* dos nós vizinhos precisam ser atualizados evitando assim o reprocessamento dos valores de *hash* no restante do banco.

É importante mencionar que a ordem de leitura dos nós vizinhos é fundamental. Isso significa que cada sequência de nós gera um *hash* diferente. Assim sendo é essencial que os nós sejam ordenados por

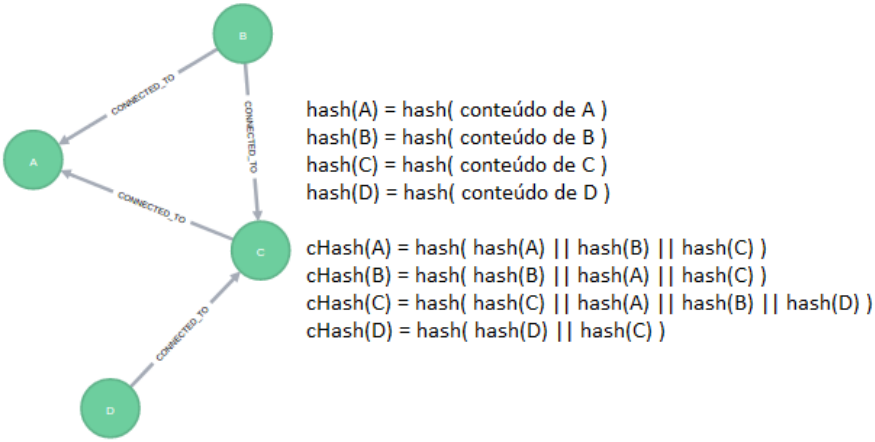


Figura 6 – *Hash* e *cHash* são calculados sobre a concatenação do conteúdo de cada nó. A relação entre os nós é *connected_to*. Fonte: o autor.

algum atributo antes da aplicação dos algoritmos. No caso de o banco ter a estrutura baseada em um grafo direcionado, também é necessário considerar a informação de sentido da aresta, do contrário uma inversão de direção não seria detectada. Dessa forma, o cálculo do *cHash* segue a equação 4.1 tal que dado o vértice a, os nós de b a n correspondem aos vizinhos de a; $a \rightarrow b$ corresponde à aresta entre a e b e $a \rightarrow n$ corresponde à relação entre a e n.

$$cHash_a = hash(hash_a||hash_b||hash_{a\rightarrow b}||...||hash_n||hash_{a\rightarrow n}) \quad (4.1)$$

As operações apresentadas na Figura 6 garantem que qualquer modificação não autorizada da estrutura do grafo ou do conteúdo de cada nó será detectada no momento da verificação dos valores de *hash* e *cHash*. No entanto, a proposta é de trabalhar com grafos que podem apresentar subestruturas desconexas como ilustrado na Figura 7. Neste caso só é possível garantir a integridade de cada subgrafo individualmente. Isto significa que se um atacante tentar, por exemplo, apagar completamente uma dessas subestruturas, o Guardian não seria capaz de detectar a ocorrência do ataque.

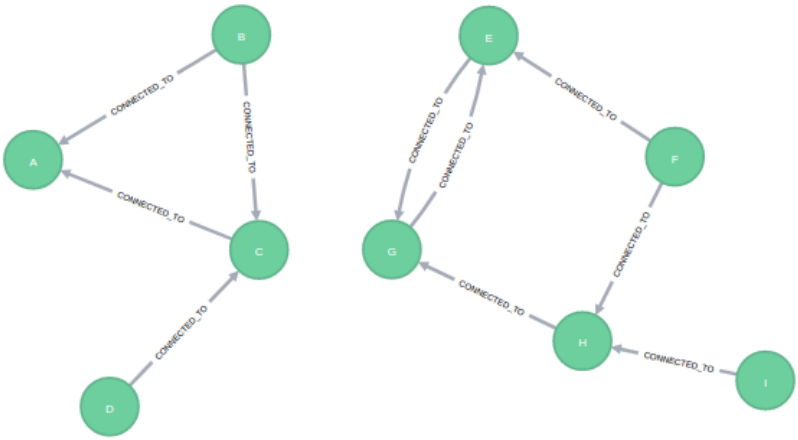


Figura 7 – Grafo com subestruturas desconexas. As relações entre os nós são *connected_ to*. Fonte: o autor.

A maneira proposta para tratar essa questão é criar um novo nó, o qual será a "*raiz*" (R) ou nó central de toda a estrutura. Esse novo vértice deve ser conectado a todos os nós ou subgrafos existentes, como apresentado na Figura 8, por meio de uma relação de posse e pertencimento. Assim, pode-se dizer que uma subestrutura pertence ao grafo cujo nó central é o nó *raiz* criado.

A relação do vértice *raiz* (*root*) com os demais pode ser feita segundo as estratégias $LigaTodos$ ou $LigaMenor$. A diferença está no modo como o nó *raiz* é ligado aos outros nós do grafo. No primeiro caso, o *root* se conecta a todos os nós do grafo enquanto a segunda opção se liga apenas ao nó de menor *id* (menor índice) de cada subestrutura que compõe o grafo. Dessa forma, pode-se dizer que com a estratégia $LigaTodos$ o número de novas relações é o mesmo que a quantidade nós no grafo. Por outro lado, com o $LigaMenor$, o número de novas ligações é igual a quantidade de subgrafos do banco.

Em seguida, deve-se repetir os passos para calcular os valores de *cHash* considerando esse novo nó. A equação 4.2 ilustra a operação para obter o *cHash* do nó *raiz*. Nela tem-se que os nós de a até n representam todos os vizinhos do nó *raiz*. Dessa forma é possível garantir uma corrente ligando todas as partes do grafo e então assegurar a integridade da informação armazenada no banco.

$$cHash_{raiz} = hash_a||...||hash_n \tag{4.2}$$

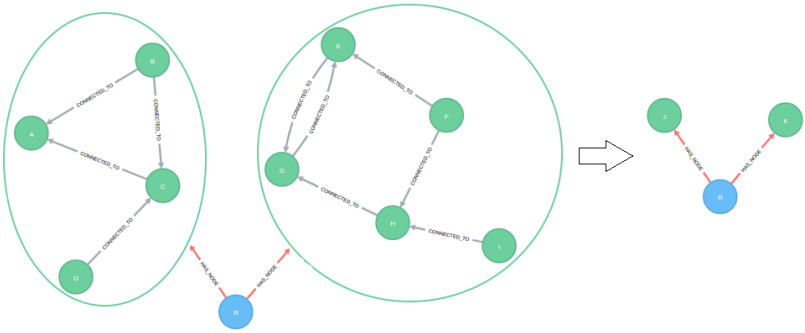


Figura 8 – Criação do nó raiz considerando cada subestrutura como parte de um mesmo grafo. As relações entre os nós de cada subgrafo são *connected_to*, enquanto as relações do nó raiz com os subgrafos são *has_node*. Fonte: o autor.

4.1 OPERAÇÕES

Será descrito a seguir como o Guardian realiza as operações de adição, atualização e remoção de nós em um grafo. Todas essas operações envolvem a geração de um *hash* e de um *cHash*, cujo procedimento também está descrito a seguir (Seções 4.1.4 e 4.1.5).

4.1.1 Adição de nós

A adição de novos nós ocorre de forma direta. O cliente deve gerar o *hash* a partir dos atributos do nó, conectá-lo a seus vizinhos e a seguir atualizar seu valor de *cHash* e de seus vizinhos.

Inicialmente (Algoritmo 1), são concatenados os atributos do objeto recebido como parâmetro (linha 2), em seguida, obtém-se o *hash* da concatenação resultante (linha 3) e então é calculado o *cHash* do nó usando o valor do *hash* gerado no passo anterior (linha 4). Depois de obter os valores de *hash*, executa-se o comando de criação de um novo nó (linha 5). Após a criação do nó, executa-se o algoritmo de criação

das ligações para conectar o novo nó a seus vizinhos (linha 6), e por fim, é chamada a função de atualização do *cHash*, com o parâmetro: identificador do nó (linha 7).

Algoritmo 1: Criar Nó

```
1 Entrada: objeto ou atributos do novo nó;
2 mensagem:= concatenação de todos os atributos do objeto;
3 h1:= cálculo do hash da mensagem;
4 cHash:= cálculo do hash de h1;
5 executa o comando de criação de novo nó passando como parâmetro os
    atributos do objeto, h1 e cHash;
6 criarLigacoes(); // cria as relações do novo nó
7 atualizarcHash(ID do nó); // Algoritmo 7
```

4.1.2 Remoção de nós

Ao remover um nó é necessário atualizar o *cHash* dos demais vértices a que este possuía ligação. O mesmo acontece quando se deseja desfazer apenas uma relação. No primeiro caso, antes de se apagar um vértice, deve-se identificar todos os seus vizinhos e só então removê-lo. No caso de uma relação, é preciso identificar os dois nós envolvidos e então desfazer a ligação. Nesses dois casos (remoção de um vértice ou aresta) é criada uma lista de nós os quais devem ter seus *cHashes* atualizados.

O Algoritmo 2 apresenta os passos para a remoção de um vértice. Tem-se como entrada o identificador do nó a ser removido. Primeiramente identifica-se os vértices vizinhos e os agrupa em uma lista (linha 2). Em seguida executa-se o comando para apagar o nó desejado (linha 3). Por fim, para cada nó existente na lista criada anteriormente, atualiza-se o *cHash* (linha 6).

O Algoritmo 3 mostra os passos para remover uma aresta. Inicialmente passa-se como entrada o identificador da relação (linha 1). Em seguida, identificam-se os dois vértices que compõem tal ligação e ambos são adicionados em uma lista, *envolvidos* (linha 2). O próximo passo é apagar a aresta (linha 3) e por fim, para cada nó na lista de *envolvidos*, atualizar o *cHash* (linha 6).

Algoritmo 2: Remover Nó

```
1 Entrada: nó ID;
2 vizinhos:= lista de vizinho do nó atual, de id = ID;
3 executa comando de remoção do nó de id = ID
4 se vizinhos é diferente de vazio então
5 |   para ∀ vizinho ∈ vizinhos faça
6 |   |   atualizarcHash(ID do vizinho); // Algoritmo 7
7 |   fim
8 fim
```

Algoritmo 3: Remover Relação

```
1 Entrada: relação ID;
2 envolvidos:= lista nós que compõem a relação, de id = ID;
3 executa comando de remoção da relação de id = ID
4 se envolvidos é diferente de vazio então
5 |   para ∀ nó ∈ envolvidos faça
6 |   |   atualizarcHash(ID do nó); // Algoritmo 7
7 |   fim
8 fim
```

4.1.3 Atualização de nós

Mediante atualização de um nó, ou seja, mudanças no valor de algum atributo, deve-se atualizar os valores de *hash* e *cHash*. O Algoritmo 4 detalha a atualização de um nó. Tem-se como parâmetro de entrada o identificador do nó modificado (linha 1). Inicialmente atualiza-se o valor de *hash* do vértice (linha 2). Em seguida corrige-se o valor do *cHash* (linha 3). Após isso, deve-se verificar se o nó atual possui vizinhos para que estes tenham seus *cHashes* corrigidos (linha 4). Se houver vizinhos, deve-se atualizar seus *cHashes* (linha 7).

4.1.4 Atualização do *Hash*

A atualização do valor de *hash* ocorre sempre que algum atributo do nó tem seu valor alterado. O Algoritmo 5 demonstra os passos para a atualização do valor. Neste caso toma-se como parâmetro de entrada o identificador do nó modificado (linha 1) e então agrupa-se todos seus atributos em uma lista (linha 2) e os concatena em uma

Algoritmo 4: Pós Atualização de Nó

```
1 Entrada: nó ID;
2 atualizaHash(ID) // Algoritmo 5
3 atualizacHash(ID) // Algoritmo 7
4 vizinhos = lista de nós ligados ao nó atual, de id = ID;
5 se nó atual possui vizinhos então
6 |   para ∀ nó ∈ vizinhos faça
7 |   |   atualizacHash(ID do nó); // Algoritmo 7
8 |   fim
9 fim
```

string temporária (linha 3). Em seguida, esta *string* é passada à função de criptografia que devolve uma mensagem codificada (linha 4) a qual é armazenada no atributo *hash* do nó (linha 5).

Algoritmo 5: Atualizar *Hash*

```
1 Entrada: nó ID;
2 atributos:= lista atributos do nó atual;
3 temp:= concatenação dos valores de cada atributo ∈ atributos;
4 hash:= hash de (temp); // valor retornado a função de criptografia
5 executa Cypher de atualização do nó setando o nova valor de Hash
  calculado na linha 4
```

De modo semelhante, o Algoritmo 6 mostra como ocorre a atualização de todos os *hashes* do grafo. Antes de se seguir os passos do para atualizar o *hash* de um único nó, identifica-se todos os vértices do grafo e adiciona seus *ids* em um lista (linha 2). Por fim, para cada nó desta lista corrige-se o valor de *hash* (linha 5).

Algoritmo 6: Atualizar todos os *hashes*

```
1 Entrada: - ;
2 lista:= lista de nós do grafo;
3 se lista é diferente de vazio então
4 |   para ∀ nó ∈ lista faça
5 |   |   atualizarHash(ID do nó); // Algoritmo 5
6 |   fim
7 fim
```

4.1.5 Atualização do *cHash*

A atualização do *cHash* deve ocorrer sempre que alguma modificação da estrutura ou da informação acontecer. O Algoritmo 7 apresenta os passos para atualização do valor de *cHash*. Tem-se como como parâmetro de entrada o identificador do nó alvo. Inicialmente, obtêm-se o *hash* do nó atual (linha 2), em seguida é identificada a lista de vizinhos do nó atual (linha 3). O próximo passo é armazenar em uma variável temporária a concatenação do *hash* de cada um dos vizinhos do nó atual (linha 4), e então concatena-se o *hash* atual com o valor temporário gerado no passo anterior (linha 5). Por fim, executa-se o comando de atualização do nó, usando a sintaxe Cypher (linha 6).

Algoritmo 7: Atualizar *cHash*

```
1 Entrada: nó ID;
2 hash_a:= hash do nó atual, de id igual a ID;
3 vizinhos:= lista de vizinhos do nó atual;
4 temp:= concatenação dos hashes de cada vizinho ∈ vizinhos;
5 cHash:= hash de (hash_a + temp);
6 executa Cypher de atualização do nó setando o nova valor de cHash
    calculado na linha 5
```

O Algoritmo 8 apresenta os passos para a atualização de todos os *cHashes*. A operação ocorre de forma semelhante à atualizacao de um único valor. Antes de aplicar o algoritmo de atualização do *cHash* identifica-se todos os vértices do grafo e adiciona seus *ids* em um lista (linha 2). Em seguida, para cada nó desta lista, executa-se a função de atualização do *cHash* (linha 5).

Algoritmo 8: Atualizer Todos os *cHashes*

```
1 Entrada: - ;
2 lista:= lista de nós do grafo;
3 se lista é diferente de vazio então
4 |   para ∀ nó ∈ lista faça
5 |   |   atualizarcHash(ID do nó); // Algoritmo 7
6 |   fim
7 fim
```

4.2 RESUMO DO CAPÍTULO

Neste capítulo são apresentadas as técnicas usadas no sistema Guardian para realizar a verificação de integridade de dados armazenados em BDs do tipo grafo. Dentre elas são descritos os cálculos para a geração dos valores de *hash* e *cHash*. O primeiro valor é responsável por garantir a integridade da informação de cada vértice individualmente, enquanto o segundo assegura a integridade do grafo como um todo de forma a detectar a remoção de algum nó ou subgrafo da estrutura. Em seguida é elaborada uma estratégia para organização de grafos desconexos a fim de torná-los compatíveis com o Guardian.

Finalmente, a Seção 4.1 traz a descrição e os algoritmos das operações realizadas pelo Guardian. Dentre elas estão as principais ações de um BD, tais como inserção, atualização e remoção, bem como os passos para a atualização dos valores de *hash* e *cHash*.

5 AVALIAÇÃO

O sistema Guardian, proposto neste trabalho, foi submetido inicialmente a um teste de viabilidade, usado para confirmar que a técnica poderia ser implementada (Seção 5.1). Em seguida, foi desenvolvida uma nova versão do Guardian para a realização dos testes de utilização dos algoritmos propostos com os quais pode-se coletar informações sobre o tempo de execução e o tamanho adicional do grafo gerados (Seção 5.2).

Todas as execuções foram realizadas em uma máquina com processador Intel Core i7-4510U, 64 bits, com CPU a 2Ghz e memória RAM de 16GB, enquanto o banco de dados usado foi o Neo4J na versão 3.1.4. Os testes foram realizados de modo que servidor do banco Neo4j e Guardian eram as únicas aplicações em execução além dos processos normais da máquina.

As medições deste trabalho foram feitas todas no lado do cliente. O cenário testado foi o de um usuário que consulta o banco para obter os dados necessários, processa os valores de *hash* e *cHash* localmente rodando a aplicação em seu computador e então devolve o resultado ao SGBD para a atualização dos valores no banco.

5.1 TESTE DE VIABILIDADE

Para fazer a validação do Guardian e verificar sua viabilidade, foi desenvolvido um protótipo de código aberto com o qual se realizou o primeiro teste (GITHUB, 2017b). Esta verificação simulou uma pequena rede social na qual existem dois grupos de amizade divididos de modo que membros de um grupo não possuem qualquer ligação com os membros do segundo grupo, como mostra a Figura 9. A aplicação de teste constrói o grafo pré definido da rede social apresentada e também insere o nó *raiz* do grafo.

No momento da inserção de um novo nó, faz-se a coleta de todos os seus atributos, que são concatenados em uma *string*. Essa *string* é criptografada de acordo com o algoritmo definido. Inicialmente o resultado desse processo é atribuído aos campos de *hash* e *cHash* do vértice. Esse passo é feito para que se possa garantir a integridade das informações de cada nó. Dessa forma, qualquer alteração no conteúdo dos dados resultará em um *hash* diferente do calculado inicialmente, caracterizando um possível ataque caso tenha sido realizada por um

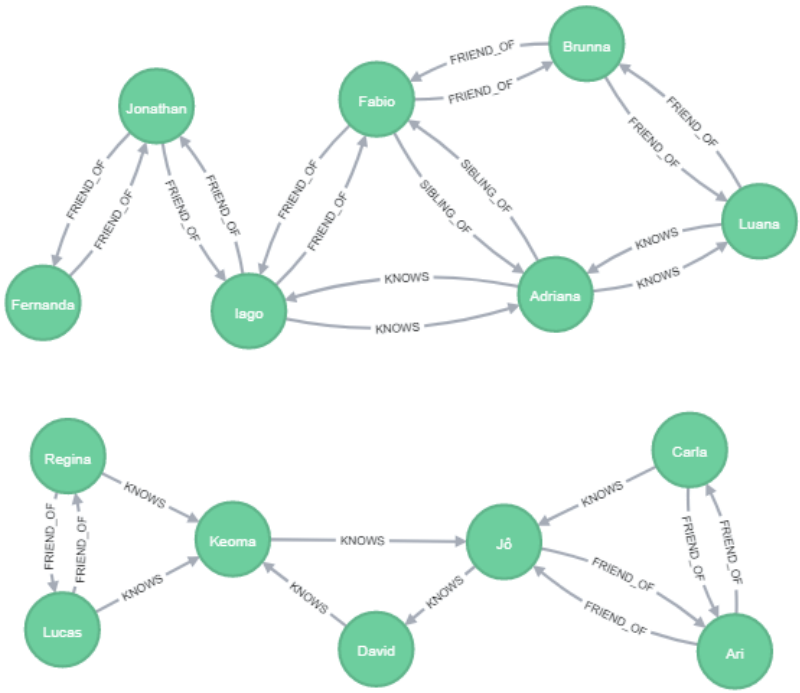


Figura 9 – Grafo da rede social simulada composto por dois subgrafos desconexos. Cada nó representa um usuário, que se conecta a outros pelas relações "conhece" (*knows*), "amigo de" ($friend\_of$) ou "irmão de" ($sibling\_of$). Fonte: o autor.

usuário não autorizado. Para fins de teste o algoritmo de criptografia usado foi o BCrypt (PROVOS; MAZIERES, 1999).

Após a inserção do novo vértice, criam-se as relações pré definidas com seus respectivos vizinhos, assim como a ligação com o nó *raiz* do grafo (Figura 10). Por fim, atualiza-se o valor de *cHash* de todos os nós. Essa atualização faz uso dos *hashes* do nó atual e dos seus vizinhos diretos para gerar um novo valor de *hash* que é armazenado no atributo *cHash*. Com essa atualização, é possível garantir a integridade do subgrafo evitando que um vértice ou aresta possam ser apagados sem que a modificação seja detectada.

Figura 10 – Grafo resultante após aplicação do método proposto. As relações entre os nós são do tipo relações "conhece" (*knows*), "amigo de" (*friend_of*) ou "irmão de" (*sibling_of*) e as do nó *raiz* com os demais são do tipo "possui_nó" (*has_node*). Fonte: o autor.

5.2 TESTES DE AVALIAÇÃO

Após confirmar a viabilidade do Guardian (GITHUB, 2017b), realizou-se a refatoração do código a fim de melhorar sua eficiência. A partir das características do problema a ser resolvido e de execuções preliminares do Guardian, foram criadas hipóteses para orientar o desenvolvimento dos experimentos. As hipóteses estão descritas a seguir.

H1: **O tempo de aplicação do *hash*/*cHash* é linearmente proporcional ao número de nós/vértices**. Essa hipótese surgiu da intuição de que quanto maior o número de nós e vértices a serem processados, maior seria o tempo de aplicação do Guardian.

Partiu-se da suposição de linearidade entre o tempo e o número de nós/vértices por essa relação aparentar ser a mais natural na situação considerada.

H2: **O tamanho adicional do grafo resultante da aplicação do *hash*/*cHash* é menor do que 5% do tamanho do BD**. Essa hipótese surgiu da suposição de que o acréscimo dos valores de *hash* e *cHash* não aumentaria em grande quantidade o espaço ocupado pelo grafo e decidiu-se que 5% de aumento do tamanho seria bastante aceitável.

H3: **O método de criptografia utilizado influencia significativamente no tempo necessário para aplicar a solução dada pelo Guardian**. A origem desta hipótese se baseou no pensamento de que os algoritmos de criptografia usados poderiam ter tempos de execução bastantes discrepantes devido a suas diferentes complexidades.

H4: **O método de *hash* utilizado influencia no tamanho adicional do banco gerado ao aplicar o Guardian aos grafos**. A hipótese surgiu a partir do fato de que cada um dos tipos de criptografia usados geram *hashes* de tamanhos diferentes. Isto é, a *string* codificada resultante da aplicação do MD5 é menor que a gerada pelo SHA1, assim como esta é menor que a devolvida pelo SHA256.

H5: **O método *ligaTodos* é mais rápido do que o *ligaMenor***. Essa hipótese surgiu do conhecimento de que o método *ligaMenor* precisa de mais operações para decidir qual nó deve ser conectado ao nó *raiz*, o que não ocorre com o *ligaTodos*.

H6: **O método *ligaTodos* ocupa mais espaço que o *ligaMenor*, e gera maior tamanho adicional do grafo** . Esta hipótese partiu do pressuposto de que, com o método *ligaTodos*, mais arestas seriam criadas do que com o *ligaMenor*. Pode-se dizer que essa afirmativa está correta em partes, no entanto não há garantias de que ela sempre se confirmará.

H7: **A solução de *hash*/*cHash* com uso da estratégia *ligaTodos* pode ser aplicada a todo grafo desconexo**. Essa hipótese surgiu do questionamento da necessidade de se identificar os subgrafos do banco.

H8: **Maior tamanho de nó vai impactar no tempo/esforço da geração do *hash*/*cHash***. Essa hipótese surgiu da proposta de garantir a integridade de todos os atributos do nó. Para isto, a técnica proposta concatena o valor de todos os atributos do vértice em uma *string* antes de aplicar a função de criptografia.

H9: **Quanto maior o número de relações entre os vértices, maior é o tempo de execução**. Essa hipótese surgiu a partir da discussão gerada pela Hipótese 8. Observou-se que se o tempo de execução é afetado pelo número de propriedades de um nó. O mesmo poderia acontecer em relação ao grau do vértice, ou seja, uma maior quantidade de relações poderia deteriorar o desempenho.

Para realizar a verificação dessas hipóteses, foram necessárias diversas variações de parâmetros na execução do Guardian, que geraram diferentes cenários de teste (Tabela 1). Variou-se o tamanho do grafo a ser processado, de maneira que foram gerados novos grafos com dez, cem, um mil, dez mil e cem mil nós. Para isso, foi adicionado ao Guardian um método de geração de grafo que cria a base com o tamanho indicado.

Outra variação realizada foi o tipo de algoritmo de criptografia. Foram utilizados os algoritmos MD5(RIVEST, 1992), SHA1(JONES, 2001) e SHA256(WOLRICH et al., 2014), (FRANCIA; FRANCIA, 2007). Cada algoritmo de criptografia gerou cinco cenários de teste diferentes e cada um deles foi executado pelo menos quinze vezes. Nas cinco primeiras, grupo 1, apenas o valor de *hash* foi gerado em uma mesma execução, e armazenado como atributo do nó. Nas cinco seguintes, grupo 2, considerou-se o grafo no estado resultante da geração do valor do *hash*, com este valor já armazenado no vértice, e então executou-se a aplicação para gerar apenas o valor de *cHash* em uma mesma execução e armazená-lo. Já nas cinco últimas, grupo 3, considerou-se o grafo original, o qual não possui os valores de *hash* e *cHash*, e então executou-se a aplicação para que fossem gerados os dois valores em uma mesma execução.

Os cenários de teste são listados na Tabela 1 e os resultados, apresentados na Seção 5.2.1, representam a média dos valores obtidos em cada grupo de execuções, conforme descrito acima.

O experimento (código da aplicação e grafos gerados) está integralmente disponível em forma de código aberto (GITHUB, 2017a).

Tabela 1 – Tabela de Cenários.

Cenário	Algoritmo de Criptografia	Número de Nós
1	MD5	10
2	MD5	100
3	MD5	1000
4	MD5	10000
5	MD5	100000
6	SHA1	10
7	SHA1	100
8	SHA1	1000
9	SHA1	10000
10	SHA1	100000
11	SHA256	10
12	SHA256	100
13	SHA256	1000
14	SHA256	10000
15	SHA256	100000

Fonte: o autor.

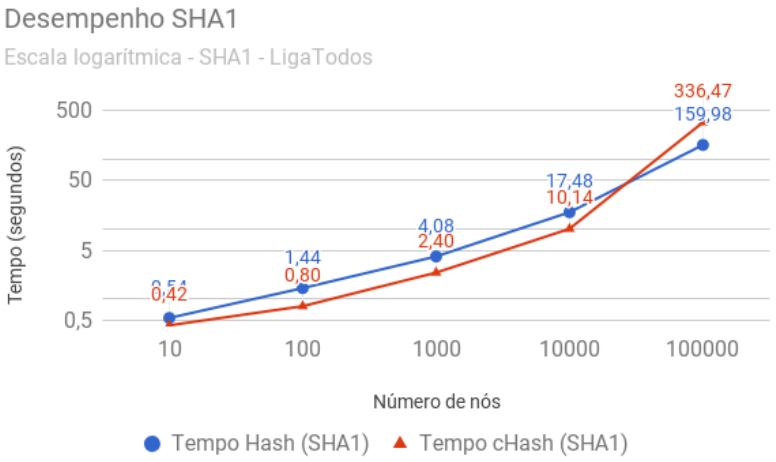
5.2.1 Verificação das Hipóteses

Hipótese 1: O tempo de aplicação do *hash*/*cHash* é linearmente proporcional ao número de nós/vértices.

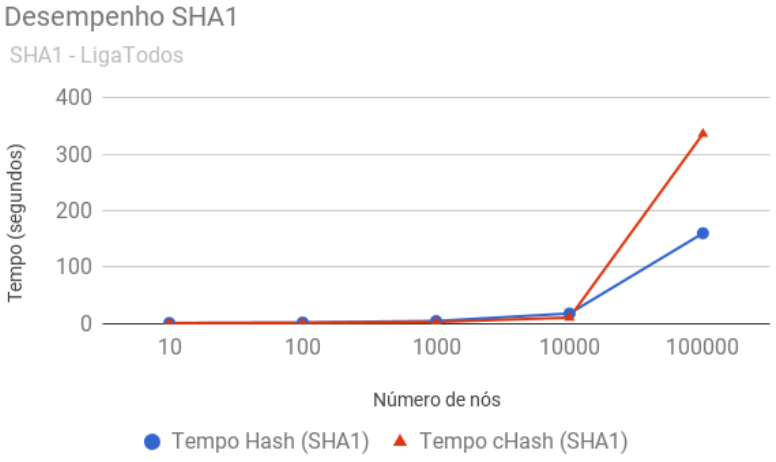
A Figura 11 apresenta os gráficos do tempo de aplicação do Guardian nos diferentes tamanhos de grafo usando o algoritmo de criptografia SHA256. Os demais algoritmos apresentaram valores diferentes, mas o comportamento da curva foi o mesmo. Nesta, o primeiro gráfico é apresentado em escala logarítmica enquanto o segundo encontra-se em escala decimal. Como pode ser verificado, o tempo cresce de forma exponencial conforme o número de nós do grafo aumenta.

De acordo com os resultados, a solução mostrou-se ideal para problemas com grafos de tamanhos medianos. No entanto, a soma dos tempos de geração dos valores de *hash* e *cHash* ainda são aceitáveis considerando-se o tamanho do grafo e que a atualização dos dois valores ocorreria em todo grafo apenas na primeira execução, já que as demais modificações afetariam somente uma parte dos nós do grafo.

Portanto conclui-se que a Hipótese 1 é falsa, já que o tempo de aplicação do Guardian não é linearmente proporcional ao número de



(a)



(b)

Figura 11 – Gráficos do tempo de geração dos valores de *hash* e *cHash*: escala logarítmica (a) e escala normal (b). Fonte: o autor.

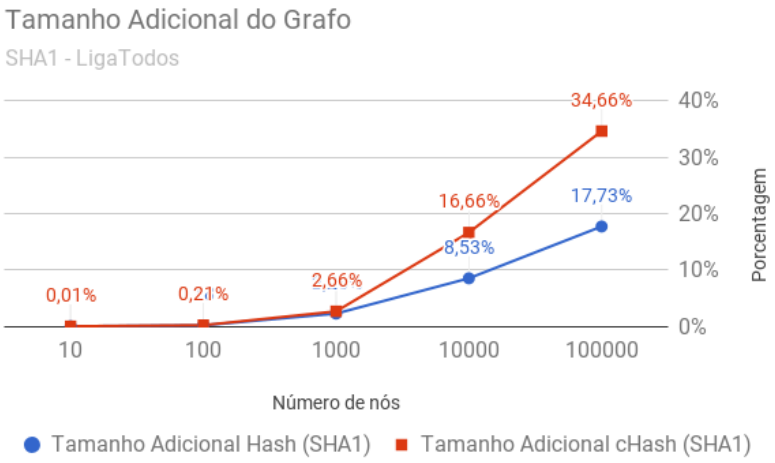


Figura 12 – Gráfico do tamanho adicional do BD gerado pelo *hash* e *cHash*. Fonte: o autor.

nós.

Hipótese 2: O tamanho adicional do grafo resultante da aplicação do *hash*/*cHash* é menor do que 5% do tamanho do BD

Observando-se Figura 12 verifica-se que o tamanho adicional do grafo foi maior que 5% para grafos com mais de 10.000 nós. Nela é apresentado o gráfico do tamanho adicional criado após a geração dos valores de *hash* e *cHash* nos grafos dos diferentes tamanhos.

Dessa forma conclui-se que a Hipótese 2 é falsa já que o tamanho adicional do banco gerado pela aplicação do *hash*/*cHash* cresce conforme o número de o número de nós no grafo aumenta, e nem sempre é menor que 5%.

Hipótese 3: O método de criptografia utilizado influencia significativamente no tempo necessário para aplicar a solução dada pelo Guardian.

A Figura 13 mostra a comparação entre os tempos gastos na aplicação do Guardian com o uso de cada algoritmo de criptografia.

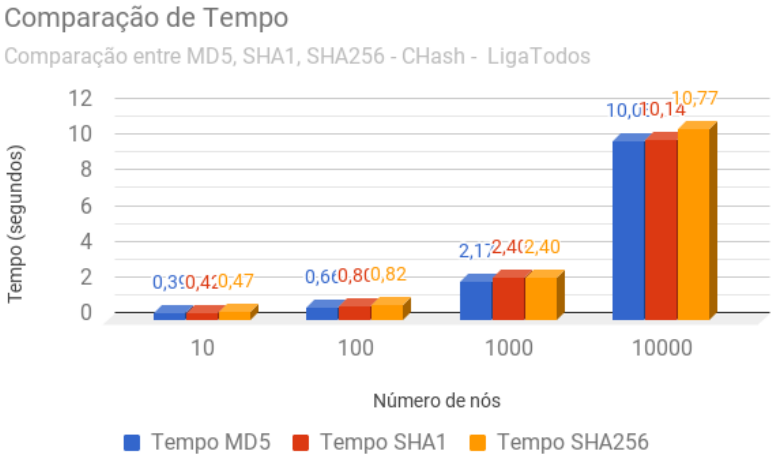


Figura 13 – Gráfico de comparação dos tempos gastos pelos diferentes algoritmos de criptografia. Fonte: o autor.

Dentre os algoritmos usados, MD5, SHA1 e SHA256, observou-se que o método criptográfico não influencia significativamente no tempo de aplicação da solução, ou seja, a codificação dos valores de *hash* e *cHash* pouco interfere no tempo necessário para a implantação do método proposto.

Ao analisar os resultados coletados, nota-se que ocorre uma pequena variação de tempo entre os diferentes algoritmos em grafos com o mesmo número de nós, no entanto a diferença de valores é pouco significativa.

A Figura 14 ilustra a similaridade das curvas dos gráficos de tempo dos três tipos criptográficos usados. Apesar de os valores numéricos variarem de acordo com o algoritmo de criptografia pode-se verificar que a forma do gráfico se mantém semelhante.

Desse modo, para este caso, pode-se concluir que a Hipótese 3 é falsa já que a alteração no método criptográfico não afetou significativamente o desempenho da solução.

Hipótese 4: O método de criptografia utilizado influencia no tamanho adicional do BD gerado ao aplicar o Guardian aos grafos.

A Figura 15 traz uma comparação dos tamanhos adicionais ge-

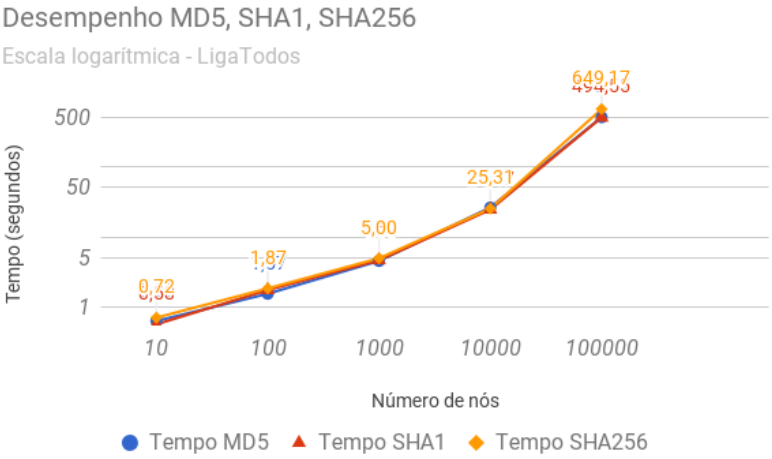


Figura 14 – Gráfico de comparação do desempenho de tempo dos três algoritmos de criptografia. Fonte: o autor.

rados por cada um dos algoritmos usados. Como esperado, observa-se uma variação dos valores dependendo do algoritmo de criptografia usado. Assim como ocorre na Hipótese 2, a diferença do tamanho adicional entre os algoritmos de criptografia se torna mais significativa em grafos com número de vértices acima de 10.000 nós. Portanto conclui-se que a Hipótese 4 é verdadeira.

Hipótese 5: O método *ligaTodos* é mais rápido do que o método *ligaMenor*.

Os testes foram executados usando dois métodos nomeados de *ligaTodos* e *ligaMenor*. No primeiro método, o nó *raiz* é conectado a todos os nós do grafo, e no segundo, o nó *raiz* é conectado apenas ao nó de menor índice de cada subgrafo.

A Figura 16 apresenta o gráfico da comparação entre os tempos gastos para geração do *hash* e do *cHash* em ambos os métodos, *ligaTodos* e *ligaMenor*. A curva apresentou o mesmo formato nos três métodos de criptografia, por isso foi representado apenas o gráfico utilizando o método MD5.

A escolha do nó a ser conectado ao *raiz* pela estratégia *ligaMenor* exige maior número de operações, o que não ocorre com a *ligaTodos*. Isso significa que este último exige menor processamento contraposto

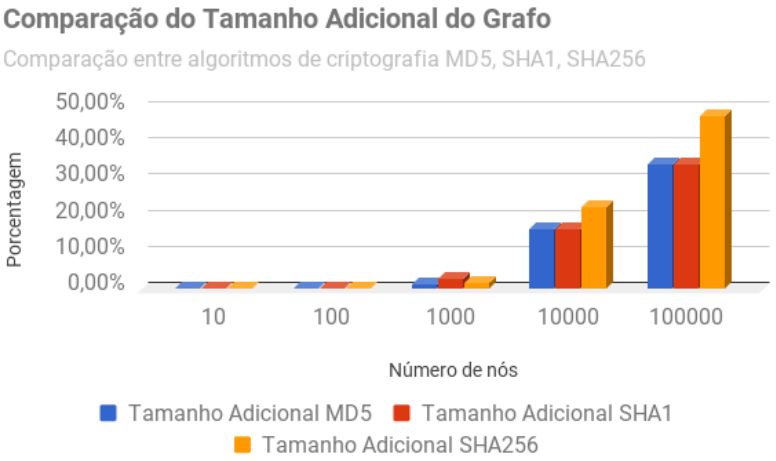


Figura 15 – Gráficos de comparação dos tamanhos adicionais gerados pelos diferentes algoritmos de criptografia. Fonte: o autor.

ao primeiro, o qual precisa encontrar o menor nó de cada subestrutura do grafo. Ainda que este fosse totalmente conexo, o método *ligaMenor* precisaria comparar todos os nós e por fim ligar o menor ao nó *raiz*. Portanto, esse trabalho extra resulta em maior tempo de execução e deste modo, pode-se concluir que a Hipótese 5 é verdadeira.

Hipótese 6: O método *ligaTodos* ocupa mais espaço que o *ligaMenor*, e gera maior tamanho adicional do grafo

A Figura 17 traz a comparação entre as curvas de ambos os métodos, obtidas nos testes executados. O mesmo padrão pode ser observado com demais tipos de criptografia, motivo pelo qual optou-se por apresentar apenas resultados com uso do método MD5.

Conforme já mencionado, o método *ligaMenor* conecta apenas um nó de cada subestrutura ao nó *raiz*. Dessa forma, caso o grafo não possua subgrafos, ou seja, trata-se de um grafo totalmente conexo, apenas uma aresta seria criada para ligá-lo ao nó raiz, enquanto o outro método precisaria criar um número de arestas igual ao número de vértices.

Assim sendo, como não há garantia de que o grafo sempre atenderá a condição de ser totalmente conexo, também não se pode afirmar que o tamanho adicional do *ligaTodos* será sempre maior. Portanto

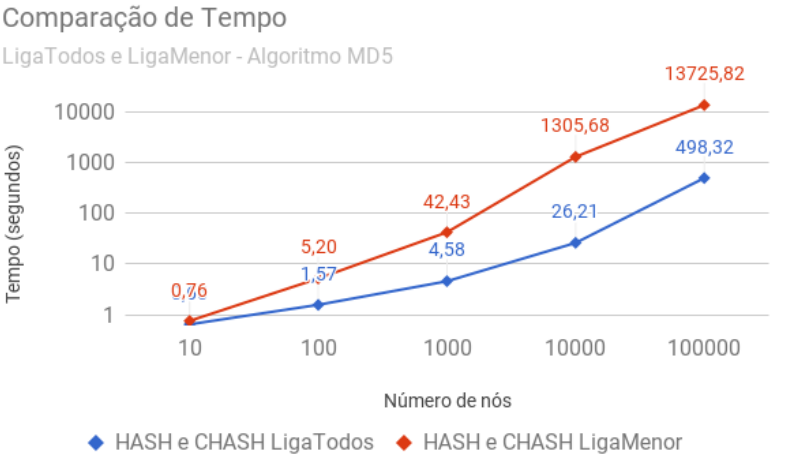


Figura 16 – Gráfico de comparação dos tempos gerados pelos métodos *ligaTodos* e *ligaMenor*. Fonte: o autor.

pode-se concluir que a Hipótese 6 é falsa.

Hipótese 7: A solução de *hash/cHash* com uso da técnica apresentada *ligaTodos* pode ser aplicada a todo grafo desconexo.

Conforme mencionado, existiu o questionamento sobre a necessidade de o Guardian precisar identificar as subestruturas do grafo. Tal verificação não se fez necessária pois o próprio BD já realiza esse procedimento ao se fazer uma consulta pela lista de todos os nós presentes no grafo.

Assim, o Guardian pode ser aplicado a todo tipo de grafo desconexo, assim como aos conexos, não importando o tipo da técnica usada, *ligaTodos* ou *ligaMenor*.

O *ligaTodos* faz uma leitura completa no grafo e liga o nó *raiz* a todos os outros encontrados. De forma semelhante, a técnica *ligaMenor* faz uma varredura completa identificando os subgrafos que não possuem ligação com o nó *raiz*. Em seguida, compara os nós da subestrutura e identifica o de menor índice, o qual será conectado ao *raiz*. Desta forma conclui-se que a Hipótese 7 é verdadeira.

Hipótese 8: Maior tamanho de nó vai impactar no tempo/esforço da geração do *hash/cHash*.

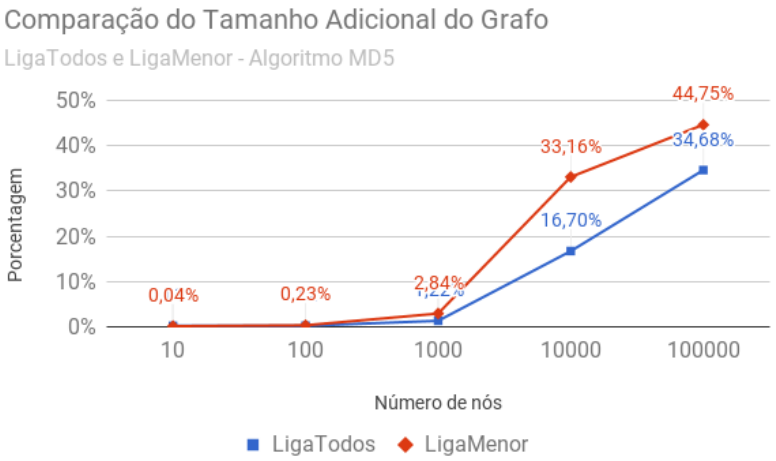


Figura 17 – Gráfico de comparação do tamanho adicional gerado pelos métodos *ligaTodos* e *ligaMenor*. Fonte: o autor.

O gráfico apresentado na Figura 18 mostra a comparação do tempo gasto na aplicação do Guardian em grafo de mesmo tamanho, sendo que o grafo (B) possui nós com tamanho dez vezes maior que os nós do grafo (A).

A fim de verificar esta hipótese comparou-se dados gerados pelo grafo de mesmo número de nós, no entanto com dez vezes mais informação no vértice. Assim sendo diz-se que o novo nó é dez vezes maior que o nó anterior.

Após execução dos testes, verificou-se que o tamanho do nó impacta no tempo de aplicação da solução, ou seja, quanto maior a quantidade de informação em cada nó, maior o seu tamanho, e maior o tempo necessário para geração do *hash* e *cHash*. Este fato ocorre pois para a geração dos valores é preciso coletar e concatenar todos os atributos do nó. Dessa forma, se um nó possui muitos atributos, gasta-se mais tempo para agrupar toda a informação necessária. Assim, pode-se concluir que a Hipótese 8 é verdadeira.

Hipótese 9: Quanto maior o número de relações entre os vértices, maior é o tempo de execução.

A Figura 19 apresenta a proporção do tempo gasto nas gerações dos valores de *hash* e *cHash* pelo Guardian. A partir da análise da

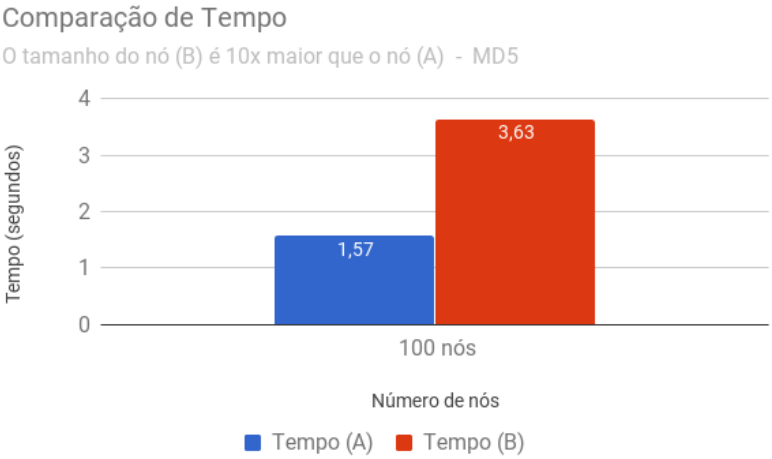


Figura 18 – Gráfico de comparação do tempo com nós dez vezes maior. Fonte: o autor.

Hipótese 8 pode-se dizer que quanto mais atributos o nó contiver, mais tempo é necessário para a aplicação da solução. Como apresentado, isso ocorre devido ao tempo gasto para concatenar todos os valores.

Relembrando o proposto no Capítulo 4, o processo para a geração do valor de *cHash* utiliza o *hash* do nó atual concatenado com o *hash* de cada vizinho. Assim sendo, pode-se dizer que quanto mais relações um vértice possuir, mais tempo será necessário para a geração do valor de *cHash*. Isso ocorre pois a operação precisa consultar outros nós para obter as informações faltantes, o que acaba resultando no aumento do tempo gasto.

Pode-se observar na Figura 19 como o aumento do número de nós e, consequentemente, o aumento do número de ligações dos grafos analisados, faz com que a proporção do tempo gasto para a geração do *cHash* também se eleve. Assim, conclui-se que a Hipótese 9 é verdadeira.

Uma melhor análise da Figura 19 também pode auxiliar na escolha do melhor tipo de aplicação na qual se aplicará o Guardian. A partir dela pode-se inferir que aplicações com muitos vértices exigirão mais tempo para o processamento do *hash*. Por outro lado, o maior grau dos nós (maior número de ligações) do grafo resultará em um pro-

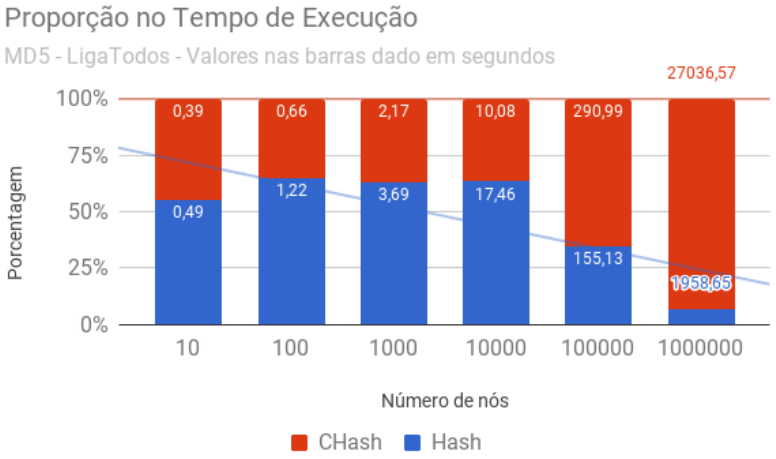


Figura 19 – Gráfico de proporção de tempo usada para a geração dos valores de *hash* e *cHash*. Fonte: o autor.

cessamento mais lento do valor do *cHash*. Dessa forma, cabe ao usuário analisar sua aplicação e equilibrar o uso do Guardian de acordo com suas necessidades e capacidade de processamento.

5.3 RESUMO DO CAPÍTULO

Este capítulo faz a avaliação do Guardian. Inicialmente é descrito o ambiente de execução e a forma como os experimentos foram realizados. Em seguida é relatado o processo de validação do Guardian e a forma como se verifica a viabilidade da proposta. Para isso, foi desenvolvido um protótipo e foram executados cenários de teste que comprovaram sua viabilidade. Finalmente são apresentadas hipóteses sobre o funcionamento e impacto gerado pela solução. Todos os resultados usados na discussão das hipóteses foram obtidos pela execução de diversos cenários de teste executados na versão aprimorada do Guardian.

6 CONCLUSÕES

Neste capítulo são revisitados as motivações e objetivos do trabalho. Em seguida, apresenta-se na visão geral uma síntese das atividades, sob a luz das contribuições e resultados obtidos. Na sequência, as contribuições são revistas e, finalmente, são identificados possíveis trabalhos futuros.

6.1 REVISÃO DAS MOTIVAÇÕES E OBJETIVOS

A multiplicação do volume de informação gerada diariamente é uma verdade cada vez mais comum. Com ela surge o desafio de armazenar toda esse novo conteúdo de forma segura e integra de modo que possa-se garantir a corretude dos dados retornados de uma consulta à base de dados.

Existem na literatura soluções para o problema, no entanto são medidas que, em geral, não podem ser aplicadas pelo proprietário da informação. A maioria das propostas exige alterações no tratamento dos dados feito pelos BDs ou gerenciadores de BD, o que implica em modificações nas bases do banco, que apenas os desenvolvedores poderiam aplicar. Vale lembrar também que grande parte desses trabalhos apresenta métodos de proteção que tratam apenas uma ou duas das principais operações realizadas em um BD, sendo elas, inserção, atualização e remoção.

Outro fator importante que motiva este trabalho é que pouco se encontra sobre técnicas aplicáveis a BDs do tipo grafo, já que essas propostas são direcionadas a BDs relacionais ou focam seus esforços em proteger a estrutura do grafo e pouco ou nada se fala a respeito da integridade da informação.

Dessa forma, pode-se afirmar que BDs do tipo grafo possuem os mesmos problemas de segurança de bancos relacionais, no entanto pouco se pode reaproveitar dos processos já existentes por se tratarem de estruturas completamente diferentes.

Partindo do cenário atual, o presente trabalho apresenta o Guardian como um sistema para verificar a integridade de grafos conexos em BDs do tipo grafo, incluindo estratégias de organização aplicáveis a grafos desconexos de modo a torná-los compatíveis com as verificações propostas. Por fim, também valida a solução proposta e apresenta os resultados obtidos mostrando a aplicabilidade do método.

6.2 VISÃO GERAL DO TRABALHO

Este trabalho destaca a falta de soluções para a verificação de integridade em BDs do tipo grafo e apresenta o Guardian, uma nova proposta para prover mais segurança para estes modelos de banco.

Inicialmente é apresentado o Guardian com a proposta de adicionar dois novos atributos a cada vértice do grafo, *hash* e *cHash*. O primeiro corresponde ao valor criptografado da concatenação dos demais atributos do nó, e é responsável por garantir a integridade do vértice separadamente. O segundo corresponde ao valor codificado da concatenação do *hash* do nó atual com os *hashes* de seus vizinhos. Dessa forma pode-se garantir a integridade do grafo como um todo.

Em seguida são apresentados algoritmos para a realização das principais operações realizadas em um BD, adição de nós, remoção de nós, atualização de nós, assim como os passos para a atualização dos valores de *hash* e *cHash*. De forma geral, as operações são executadas normalmente com o acréscimo das chamadas de atualização do *hash* e *cHash*, conforme necessário, ao final de cada ação.

Ao final, é apresentada a avaliação do Guardian. Nela é verificada a viabilidade da implantação do método e são realizados diversos testes para se mensurar seu desempenho. Os testes abrangem três métodos de criptografia: MD5, SHA1, SHA2, bem como as duas abordagens de ligação do grafo ao nó $raiz$, $LigaTodos$ e $LigaMenor$.

6.3 CONTRIBUIÇÕES DESTE TRABALHO

As contribuições deste trabalho estão listadas a seguir:

- Uma solução para a verificação de integridade de dados de estruturas conexas em BDs do tipo grafo;
- Uma técnica de modificação de grafos desconexos de modo a torná-los compatíveis com a verificação de grafos conexos;
- Um protótipo do sistema Guardian, desenvolvido para verificação da viabilidade da solução (GITHUB, 2017b);
- Uma implementação aprimorada do sistema Guardian, contendo testes em diversos cenários (GITHUB, 2017a).

O Guardian é apresentado com detalhes no Capítulo 4. O Guardian permite verificar a integridade de BDs do tipo grafo com estruturas conexas, de modo que o banco não esteja dividido em subgrafos. A

proposta traz as ações necessárias a serem tomadas para que se possa atestar a integridade da informação retornada.

Completando a contribuição anterior, o trabalho também apresenta duas estratégias de organização de grafos desconexos. Ambas estratégias tornam os BDs compostos por subgrafos compatíveis à aplicação do método proposto. Assim sendo, pode-se afirmar que o Guardian é capaz de certificar a integridade de todo o BD do tipo grafo, seja este conexo ou desconexo.

O Capítulo 6 traz a validação da proposta. Os experimentos apresentados mostram o Guardian aplicado em bases de diferentes tamanhos, combinando os tipos de criptografia e as estratégias de organização. A partir desses resultados pode-se obter o entendimento de seu desempenho, possibilitando assim uma melhor análise para a escolha do grupo de dados mais apropriado ao qual o Guardian pode ser aplicado.

Por fim, também é disponibilizado o código desenvolvido ao longo da realização do trabalho (GITHUB, 2017a, 2017b). Ambas aplicações implementam os principais algoritmos apresentados no Capítulo 4 e foram usadas para a execução de todos os testes praticados. Deste modo o trabalho disponibiliza exemplos de implementação e execução da solução que podem ser usados pela comunidade como base para a adoção do Guardian e o desenvolvimento de novas aplicações.

6.4 TRABALHOS FUTUROS

O Guardian detecta alterações indevidas no grafo, porém não realiza nenhuma ação que permita a recuperação do dado original. Uma sugestão de melhoria do sistema atual consiste em permitir que alterações realizadas de forma incorreta possam ser restauradas para o valor íntegro. Uma possibilidade para efetivar tal recurso é manter uma réplica do banco de dados em outro local (uma cópia secundária). Assim, é possível recorrer a uma cópia íntegra no caso de corrupção do banco original.

A verificação realizada pelo Guardian tem foco na checagem da corretude da informação, isto é, se houve alteração dos dados ou não. Outras conferências que se fazem importantes para serem melhor trabalhadas são é a verificação da completude de uma consulta assim como a confirmação da atualidade do dado. Assim, como melhoria do sistema, sugere-se que seja confirmado se o dado retornado pelo banco não omite informação, bem como se o dado retornado está em sua

versão mais atual.

A realização de novas avaliações também é uma possibilidade de trabalho futuro, a fim de analisar o desempenho e tamanho adicional do grafo gerados pelas operações de adição de um novo nó e de uma nova relação, atualização de vértices e remoção de nó e de aresta. Os testes realizados no Guardian não coletaram essas informações, no entanto prevê-se que a execução dessas ações apresentariam resultados ainda melhores do que os mostrados no Capítulo 6. Pode-se fazer essa afirmação pois tais operações afetariam apenas uma fração dos nós do grafo, diferente do ocorrido nos testes, que agiram em todo o conjunto de vértices.

Uma avaliação que também se mostra interessante é sobre a flexibilidade de implementação do Guardian. As medições deste trabalho foram feitas sempre no lado do cliente. O cenário testado foi o de um usuário que consulta o banco para obter os dados necessários, processa os valores de *hash* e *cHash* localmente rodando a aplicação em seu computador e então devolve o resultado ao SGBD para a atualização dos valores no banco. A sugestão é que essas operações também sejam avaliadas no SGBD e no próprio servidor, assim como em cenários de sistemas distribuídos. Desta forma, todo esse novo estudo proporcionará opções de configuração e melhor adequação aos mais diversos contextos.

REFERÊNCIAS

AMARO, G. Criptografia simétrica e criptografia de chaves públicas: Vantagens e desvantagens. *Revista Negócios e Tecnologia da Informação*, v. 2, p. 7, 2007.

ANGELES, R.; GUTIERREZ, C. Survey of graph database models. *ACM Computing Surveys (CSUR)*, v. 40, n. 1, p. 1–39, 2008.

APACHE. *The Apache Cassandra database*. 2007. Disponível em: `http://cassandra.apache.org/`. Acesso em: 30-10-2017.

ARSHAD, M. U.; KUNDU, A.; BERTINO, E.; MADHAVAN, K.; GHAFOOR, A. Security of graph data: Hashing schemes and defenitions. *University of Purdue – 2014 – Cyber Center Publications*, v. 2014, n. 1, p. 627, 2014.

BATTISTA, G. D.; PALAZZI, B. Authenticated relational tables and authenticated skip lists. *2006 ACM SIGMOD international conference on Management of Data*, p. 121 – 132, 2006.

ERVEN, G. C. G. V. *MDG-NoSQL: Modelo de Dados para Bancos NoSQL Baseados em Grafos*. Dissertação (Mestrado) — Universidade de Brasília, Brasília, Brasil, 2015.

FRANCIA, A.; FRANCIA, R. An empirical study on the performance of java/.net cryptographic apis. *Information Systems Security*, v. 16, n. 6, p. 344–354, Dec 2007. Dummy note.

GITHUB. *Guardian - Aplicação de Teste, com Neo4J*. 2017. Disponível em: `https://github.com/fmreina/neo4jGraph`. Acesso em: 30-10-2017.

GITHUB. *Guardian - Versão para o Teste de Viabilidade, com Neo4J*. 2017. Disponível em: `https://github.com/fmreina/neo4j`. Acesso em: 30-10-2017.

GOLUMBIC, M. C. *Algorithmic Graph Theory and Perfect Graphs*. Haifa, Israel: Ed. Elsevier, 2004. 340 p.

GOOGLE. *Cloud Bigtable: serviço de banco de dados de Big Data NoSQL*. 2004. Disponível em: `https://cloud.google.com/bigtable/?hl=pt`. Acesso em: 30-10-2017.

HO, L. Y.; WU, J. J.; LIU, P. Distributed graph database for large-scale social computing. *2012 IEEE Fith International Conference on Cloud Computing*, v. 2012, n. 1, p. 455–462, 2012.

JONES, P. *US Secure Hash Algorithm 1 (SHA1)*. 2001. Disponível em: `https://www.rfc-editor.org/rfc/pdfrfc/rfc3174.txt.pdf;https://www.rfc-editor.org/info/rfc3174`. Acesso em: 30-10-2017.

KRAWCZYK, H.; BELLARE, M.; CANETTI, R. Hmac: Keyed-hashing for message authentication. *Dept of Computer Science University of California at San Diego*, p. 11, 1997.

LI, F.; HADJIELEFTHERIOU, M.; KOLLIOS, G.; REYZIN, L. Dynamic authenticated index structures for outsouced databases. *21st annual IFIP WG11.3 working conference on Data and applications security*, p. 31–46, 2007.

MAO, W. *Modern Cryptography: Theory and Practice*. New Jersey, US: Prentice Hall PTR, 2003. 648 p.

MARTEL, C.; NUCKOLLS, G.; DEVANBU, P.; GERTZ, M.; KWONG, A.; STUBBLEBINE, S. G. A general model for authenticated data structures. *Algorithmica*, v. 39, p. 21–41, 2001.

MERKLE, R. C. A certified digital signature. *CRYPTO*, p. 218–238, 1989.

MIKLAU, G.; SUCIU, D. Implementing a tamper-evident database system. *10th Asian Computing Science Conference on Advances in Computer Science: Data Management on the Web*, p. 28–48, 2005.

MONGOBD, i. *MongoDB - NOSQL Database*. 2007. Disponível em: `https://www.mongodb.com/`. Acesso em: 30-10-2017.

NEO4J, i. *Neo4J - Graph Database*. 2002. Disponível em: `https://neo4j.com/`. Acesso em: 30-10-2017.

PEREIRA H P BORGES, H. R. F. S.; UNITRI, S. A. S.

Utilização de Banco de Dados NoSql em Ambientes Corporativos — Centro Universitário do Triângulo Avenida Nicomedes Alves dos Santos, Uberlândia, Brasil, 2014.

PROVOS, N.; MAZIERES, D. *Bcrypt Algorithm*. 1999. Disponível em: `http://static.usenix.org/events/usenix99/provos/provos_html/node5.html`. Acesso em: 30-10-2017.

RIVEST, R. *The MD5 Message-Digest Algorithm*. 1992. Disponível em: `https://tools.ietf.org/html/rfc1321`. Acesso em: 30-10-2017.

SCHNEIER, B. *Applied cryptography: protocols, algorithms, and source code in C - 2nd ed.* New York, NY, USAS: John Wiley and Sons, Inc., 1996. 584 p.

SILVERIO, A. L. *Utilizando Funções de Autenticação de Mensagens para a Detecção e Recuperação de Violações de Integridade de Acesso a Tabelas Relacionais*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, Brasil, 2014.

SILVERIO, A. L.; CUSTóDIO, R. F.; CARLOS, M. C.; MELLO, R. D. Efficient data integrity checking for untrusted database systems. *The Sixth International Conference on Advances in Databases, Knowledge and Data Applications*, p. 118–124, 2014.

STROZZI, C. *NoSQL, A Relational Database Management System*. 1998. Disponível em: `http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Home\%20Page`. Acesso em: 30-10-2017.

WEBBER, J. A programmatic introduction to neo4j. In: *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*. New York, NY, USA: ACM, 2012. (SPLASH '12), p. 217–218. ISBN 978-1-4503-1563-0. Disponível em: `http://doi.acm.org/10.1145/2384716.2384777`.

WOLRICH, G.; YAP, K.; GUILFORD, J.; GOPAL, V.; GULLEY, S. *Instruction set for message scheduling of SHA256 algorithm*. Google Patents, set. 16 2014. US Patent 8,838,997. Disponível em: `https://www.google.com/patents/US8838997`.