UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Anderson Luiz Silvério

UTILIZANDO FUNÇÕES DE AUTENTICAÇÃO DE MENSAGENS PARA A DETECÇÃO E RECUPERAÇÃO DE VIOLAÇÕES DE INTEGRIDADE DE ACESSO A TABELAS RELACIONAIS

Florianópolis

2014

Anderson Luiz Silvério

UTILIZANDO FUNÇÕES DE AUTENTICAÇÃO DE MENSAGENS PARA A DETECÇÃO E RECUPERAÇÃO DE VIOLAÇÕES DE INTEGRIDADE DE ACESSO A TABELAS RELACIONAIS

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Ronaldo dos Santos Mello, Dr.
Coorientador: Prof. Ricardo Felipe Custódio, Dr.

Florianópolis

2014

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

Silvério, Anderson Luiz
Utilizando funções de autenticação de mensagens para a detecção e recuperação de violações de integridade de acesso a tabelas relacionais / Anderson Luiz Silvério ; orientador, Ronaldo dos Santos Mello ; coorientador, Ricardo Felipe Custódio. - Florianópolis, SC, 2014.
101 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Integridade de dados. 3. Bancos de dados relacionais. 4. Segurança de dados. 5. Bancos de dados não confiáveis. I. Mello, Ronaldo dos Santos. II. Custódio, Ricardo Felipe. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

Anderson Luiz Silvério

UTILIZANDO FUNÇÕES DE AUTENTICAÇÃO DE MENSAGENS PARA A DETECÇÃO E RECUPERAÇÃO DE VIOLAÇÕES DE INTEGRIDADE DE ACESSO A TABELAS RELACIONAIS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 1º de setembro 2014.

Prof. Ronaldo dos Santos Mello, Dr.
Coordenador

Banca Examinadora:

Prof. Ronaldo dos Santos Mello, Dr.
Orientador
Universidade Federal de Santa Catarina

Prof. Ricardo Felipe Custódio, Dr.
Coorientador
Universidade Federal de Santa Catarina

Prof.ª Carmem Satie Hara, Dr.ª
Universidade Federal do Paraná

Prof.ª Carina Friedrich Dorneles, Dr.ª
Universidade Federal de Santa Catarina

Prof. Lau Cheuk Lung, Dr.
Universidade Federal de Santa Catarina

À minha família que tornou a realização deste trabalho possível.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a todas as pessoas que me ajudaram de forma direta ou indireta na realização deste trabalho.

À minha família, em especial à minha mãe, Jussara Zipperer, e minha namorada, Larissa Speranzini, pelo apoio e pelas oportunidades proporcionadas para eu chegar neste ponto.

Ao professor Ricardo Custódio, pela oportunidade de trabalhar com o tema deste trabalho e ao professor Ronaldo Mello, por aceitar o desafio da orientação deste trabalho e pelas inúmeras contribuições para o aprimoramento do trabalho.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

(Vince Lombardi)

RESUMO

Este trabalho propõe métodos para verificar a integridade dos dados em tempo real, simplificando e automatizando um processo de auditoria. Os métodos propostos utilizam funções criptográficas de baixo custo, são independentes do sistema de banco de dados e permitem a detecção de atualizações, remoções e inserções maliciosas, além da verificação da atualidade de registros e recuperação de tuplas modificadas de forma indevida. A detecção das alterações maliciosas dos dados é feita através de funções MAC, calculadas sobre a concatenação dos atributos de cada tupla das tabelas. Já a detecção da remoção é feita através de um novo algoritmo proposto, chamado CMAC, que faz o encadeamento de todas as linhas de uma tabela, não permitindo que novas linhas sejam inseridas ou removidas sem o conhecimento da chave utilizada para o cálculo do CMAC. Este trabalho também explora diferentes arquiteturas para a implementação dos métodos propostos, apresentando as vantagens e desvantagens de cada arquitetura. Além disso, é feita a avaliação dos métodos propostos, mostrando que o custo para calcular e armazenar os dados necessários para controlar a integridade é muito pequeno.

Palavras-chave: Integridade de dados. Bancos de dados relacionais. Segurança de dados. Bancos de dados não confiáveis.

ABSTRACT

Unauthorized changes to database content can result in significant losses for organizations and individuals. As a result, there is a need for mechanisms capable of assuring the integrity of stored data. Meanwhile, existing solutions have requirements that are difficult to meet in real world environments. These requirements can include modifications to the database engine or the usage of costly cryptographic functions. In this master's thesis, we propose a technique that uses low cost cryptographic functions and it is independent of the database engine. Our approach allows for the detection of malicious update operations, as well as insertion and deletion operations. This is achieved by the insertion of a small amount of protection data into the database. The protection data is utilized by the data owner for data access security by applying *Message Authentication Codes*. In addition, our experiments have shown that the overhead of calculating and storing the protection data is very low.

Keywords: Data Integrity. Outsourced Data. Untrusted Database.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
CMAC	*Chained-MAC*
DBAL	*Database Abstraction Layer*
FIPS	*Federal Information Processing Standard*
HMAC	*Keyed-hash Message Authentication Code*
HSM	*Hardware Security Module*
MAC	*Message Authentication Code*
MB-Tree	*Merkle B-Tree*
MHT	*Merkle Hash Tree*
NIST	*National Institute of Standards and Technology*
SHA-1	*Secure Hash Algorithm* 1
SHA-2	*Secure Hash Algorithm* 2
SQL	*Structured Query Language*
SGBD	Sistema Gerenciador de Banco de Dados

SUMÁRIO

1 INTRODUÇÃO

A terceirização de bancos de dados (BDs) está se tornando cada vez mais comum devido ao crescimento da internet e avanços nas tecnologias de redes. Através da terceirização dos BDs, as organizações podem se concentrar no seu próprio domínio de aplicação, delegando a uma entidade externa todas as necessidades para manter o seu BD, incluindo *hardware*, *software* e pessoas.

A arquitetura de BDs terceirizados traz diversos problemas de pesquisa. Um dos principais problemas é a garantia da segurança dos dados. A organização (ou indivíduo) terceiriza seus dados a uma entidade externa, que é confiável para manter a disponibilidade e prover a infraestrutura necessária para o armazenamento e recuperação dos dados. Entretanto, esta entidade pode não ser confiável em relação à manipulação dos dados, sendo fundamental a existência de meios para prover garantias da segurança dos dados, tanto de ataques internos quanto externos. Nos ataques externos, o atacante consegue acesso ao servidor, podendo realizar modificações maliciosas. Já no caso de ataques internos, alguém que já possui acesso ao servidor, mas não necessariamente permissões para alterar os dados, pode alterá-los de forma maliciosa.

Várias técnicas foram propostas nos últimos anos para tratar o sigilo de dados (HACIGUMUS et al., 2002; AGRAWAL et al., 2004; WANG; LAKSHMANAN, 2006; CESELLI et al., 2005; AGGARWAL et al., 2005; CIRIANI et al., 2009, 2010; De Capitani di Vimercati et al., 2014), sendo esta uma área já bastante explorada na literatura e com boas soluções. Por outro lado, a integridade dos dados ainda é uma área em aberto. Por integridade, nos referimos à garantia de que os dados não foram modificados, inseridos ou removidos de forma não autorizada. Muitos dos trabalhos encontrados na literatura que abordam integridade, como os de Li et al. (LI et al., 2006) e Ibrahim Kamel (KAMEL, 2009), necessitam de alterações no *kernel* dos sistemas de gerenciamento de BD (SGBDs) existentes para que possam ser implantados. Este requisito torna a utilização destes mecanismos inviável em muitos cenários reais, principalmente em ambientes que já se encontram em produção. Num ambiente de BD terceirizado (*outsourced database*), por exemplo, o cliente não tem controle sobre a infraestrutura, não podendo realizar tais modificações. Além disso, muitos SGBDs comerciais não possuem o código aberto, impossibilitando realizar alterações no seu *kernel*.

Outros trabalhos são baseados em estruturas de dados autenticadas (BATTISTA; PALAZZI, 2007; HEITZMANN et al., 2008; MIKLAU; SUCIU, 2005), como as árvores de Merkle (MERKLE, 1989) e as Skip-Lists (PUGH, 1990). Es-

tes trabalhos são mais fáceis de serem implantados em ambientes reais, pois não necessitam de mudanças no *kernel* do SGBD. Entretanto, o uso de estruturas autenticadas compromete a eficiência de BDs altamente dinâmicos, pelo fato de que a estrutura toda, ou grande parte dela, precisa ser recalculada a cada atualização no BD. Outra classe de soluções faz uso de funções criptográficas de alto custo, como criptografia assimétrica (MYKLETUN; NARASIMHA; TSUDIK, 2006; NARASIMHA; TSUDIK, 2005).

O presente trabalho propõe métodos para a verificação da integridade dos dados em tempo real, simplificando e automatizando o processo de auditoria. Além disso, pretende-se que os métodos propostos sejam eficientes (utilizando funções de baixo custo computacional), fáceis de serem implantados em ambientes reais (não necessite de alterações no *kernel* do SGBD) e que não exija que o cliente armazene os dados de controle de integridade. Para atender tais requisitos, propõe-se o uso de funções de autenticação de mensagens (do inglês, *message authentication codes* (MACs)). Com isso, é possível detectar a inserção e modificação maliciosa de dados. Para possibilitar a detecção de remoções maliciosas, propõe-se um novo algoritmo chamado "CMAC" (do inglês *Chained-MAC*), cujo objetivo é conectar linhas adjacentes.

1.1 MOTIVAÇÃO

Ataques à integridade de BDs estão se tornando cada vez mais comuns, sendo esta uma das cinco maiores preocupações das empresas (Infosecurity Europe; PRICEWATERHOUSECOOPERS, 2010; Infosecurity Europe; PWC, 2012, 2013). Além disso, um estudo feito pela *Infosecurity Europe* [1] mostrou que 27% das grandes empresas reportaram ataques que afetaram a integridade de seus dados (Infosecurity Europe; PRICEWATERHOUSECOOPERS, 2010). Tais ataques podem causar enormes fraudes financeiras que só serão descobertas muito tempo depois por processos de auditoria.

Nesse contexto, o presente trabalho almeja prover métodos para verificar a integridade dos dados em tempo real, simplificando e automatizando um processo de auditoria. Além disso, pretende-se que os métodos providos sejam eficientes e fáceis de serem implantados em ambientes reais.

[1] http://www.infosec.co.uk

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O principal objetivo deste trabalho é propor e avaliar mecanismos que permitam a detecção e recuperação de modificações não autorizadas a dados armazenados em SGBDs relacionais. Em particular, está-se interessado em mecanismos que sejam eficientes e de fácil implantação em ambientes já existentes.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Descrever as abordagens para manutenção de integridade já existentes na literatura;
- Apresentar métodos simples e de fácil implantação que permitam a detecção de modificações não autorizadas a BDs relacionais;
- Determinar a eficiência das propostas apresentadas;
- Comparar os métodos apresentados com as abordagens existentes na literatura.

1.3 CONTRIBUIÇÕES

Este trabalho é responsável diretamente pelas seguintes contribuições:

- Avaliar a viabilidade do uso e eficiência das funções MAC para garantir a integridade dos dados armazenados em BDs relacionais;
- Propor mecanismos simples e eficientes para verificar a integridade de dados armazenados em BDs relacionais;
- Propor mecanismos simples e eficientes para a recuperação dos dados;
- Apresentar um esquema para possibilitar o gerenciamento de múltiplas chaves;
- Demonstrar e comparar a eficiência dos mecanismos propostos em relação a trabalhos correlatos já existentes na literatura.

1.4 LIMITAÇÕES DO TRABALHO

O escopo deste trabalho restringe-se apenas a BDs relacionais. Ainda, para a verificação da completude das consultas, com o CMAC, é necessário que as tuplas estejam ordenadas por algum atributo (assume-se que a ordenação é sempre feita pela chave primária) e que sejam recuperadas na mesma ordem em que são inseridas.

1.5 METODOLOGIA

Para alcançar os objetivos deste trabalho, iniciou-se pelo estudo de artigos e relatórios técnicos a respeito da verificação da integridade dos dados armazenados em BDs.

Tais materiais foram classificados de acordo com a sua relevância com este trabalho e com os problemas a serem tratados. Após a escolha e classificação dos materiais pertinentes a este trabalho, selecionaram-se aqueles úteis para a fundamentação da proposta sobre integridade de dados.

Em seguida, foram elaborados métodos para tratar a integridade de dados de forma simples e eficiente. Estes métodos foram discutidos dentro de um grupo de pesquisa, o qual contribuiu para o aprimoramento dos mesmos. Para a validação da eficiência dos métodos propostos, foi implementado um protótipo e criados diversos cenários de teste. Os resultados dos testes foram analisados e utilizados para demonstrar a eficiência dos métodos propostos e também para fazer a comparação com outros métodos semelhantes encontrados na literatura. Ainda, foram submetidos artigos sobre o tema em conferências de segurança em informação para verificar a aceitação do meio científico e colher críticas que pudessem proporcionar melhorias ao trabalho.

Por fim, com base nos resultados das etapas anteriores, elaboraram-se as considerações finais. Estas contemplam a consolidação dos métodos propostos, as vantagens decorrentes de seu uso e trabalhos futuros.

1.6 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 é detalhado o conceito de integridade e suas soluções clássicas são apresentadas. No Capítulo 3 são apresentados os trabalhos relacionados. No Capítulo 4 são apresentados os métodos propostos neste trabalho para verificar a integridade e recuperar os dados. Ainda, é apresentado o problema do gerenciamento das chaves utilizadas nos métodos propostos e soluções para o problema são propostas. No Capítulo 5 são apresenta-

dos e discutidos os resultados dos experimentos realizados para verificar a eficiência dos métodos propostos neste trabalho. Por fim, as considerações finais advindas da pesquisa deste trabalho são apresentadas no Capítulo 6.

2 INTEGRIDADE DE DADOS

2.1 INTRODUÇÃO

Na criptografia moderna, integridade de dados é uma evolução de um problema clássico em sistemas de comunicação, em que erros podem ser inseridos em mensagem devido a problemas nos meios de comunicação (MAO, 2003). Por exemplo, uma interferência em um canal elétrico pode introduzir erros nos dados trafegando por este canal. Em segurança da informação, a integridade de uma mensagem está relacionada com a sua alteração de forma maliciosa ou não autorizada.

Do ponto de vista de segurança, utilizar dados que foram modificados devido a erros nos meios de comunicação ou no processamento dos dados é tão perigoso quanto utilizar dados que foram modificados de forma maliciosa. Portanto, o princípio das técnicas para prover integridade de dados é o mesmo para ambos os casos. Mais especificamente, o transmissor da mensagem codifica alguma redundância da mensagem na forma de um valor de checagem. Este valor é então utilizado pelo destinatário para verificar a integridade da mensagem.

Em sistemas de comunicação, a codificação da mensagem é feita de tal forma que garanta ao destinatário a maior probabilidade de inferir o valor original da mensagem através da codificação. Já em segurança da informação, a codificação tenta dificultar ao máximo a probabilidade de um atacante conseguir produzir uma codificação válida para uma determinada mensagem.

Adicionalmente, as técnicas criptográficas, para garantir a integridade de mensagens, são parametrizadas por uma chave, que deve ser de conhecimento somente do emissor e do destinatário da mensagem.

Para garantir a integridade dos dados em SGBD, existem diferentes aspectos da integridade a serem garantidos, sendo eles (MYKLETUN; NARASIMHA; TSUDIK, 2006; NARASIMHA; TSUDIK, 2005; PANG et al., 2005; XIE et al., 2008):

- **Corretude (*correctness*)**: Garantia de que um dado não foi modificado de forma maliciosa ou indevida;

- **Completude (*completness*)**: Garantia de que tuplas não foram omitidas de uma consulta SQL, de forma maliciosa ou indevida;

- **Atualidade (*freshness*)**: Garantia de que o dado retornado pelo SGBD não é uma cópia antiga e sim a versão mais atual do dado.

O restante deste capítulo está organizado da seguinte forma. Primeiramente, é apresentada uma definição de integridade de dados, do ponto de vista de segurança da informação, na Seção 2.2. Em seguida, são apresentadas as técnicas que utilizam criptografia simétrica, na Seção 2.3, e assimétrica, na Seção 2.4, bem como técnicas de logs, utilizadas em SGBDs, na Seção 2.5. Na Seção 2.6, é apresentado o modelo de BDs terceirizados. Por fim, é apresentado um resumo do capítulo, na Seção 2.7.

2.2 DEFINIÇÃO

Em segurança da informação, uma mensagem é dita íntegra se ela não sofreu qualquer alteração ou interferência durante o envio do emissor ao destinatário. Uma forma simples de garantir a integridade de mensagens é através do modelo de duplo canal, como esquematizado na Figura 1. Neste modelo, Alice deseja enviar uma mensagem através de um canal simples a Bob. Para que Bob possa verificar se a mensagem enviada por Alice não sofreu alteração no caminho, Alice envia uma cópia da mensagem através de um canal seguro. Quando Bob recebe as mensagens, ele verifica se ambas são idênticas. Se as mensagens forem iguais, significa que a mensagem enviada pelo canal simples não sofreu alteração. Se elas forem diferentes, Bob pode recuperar a mensagem original através da mensagem que recebeu pelo canal seguro.

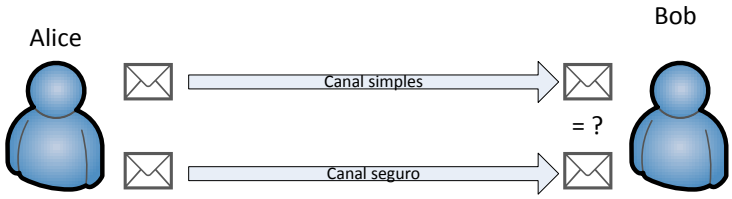


Figura 1: Modelo simples de duplo canal para verificação de integridade

Um canal é um meio de transmissão e/ou armazenamento de mensagens, sendo que o canal simples possui um baixo custo para a transmissão de dados e os dados que trafegam neste canal estão sujeitos a interferências. Um exemplo de canal simples é a internet. Já o canal seguro possui um custo elevado para a transmissão e armazenamento de dados, sendo que este canal não sofre interferências.

A implementação física de um canal seguro é inviável, portanto, o canal seguro é abstraído pelas técnicas de codificação e o valor para a verificação da integridade da mensagem é enviado juntamente com a mensagem, no mesmo canal. A Figura 2 apresenta este modelo, onde:

- $\boldsymbol{m}$ é a mensagem que se deseja enviar;
- $\boldsymbol{m'}$ é a mensagem após passar pelo canal de comunicação.
- $\boldsymbol{C}()$ é a função de codificação;
- $\boldsymbol{I}()$ é uma função de interferência que a mensagem está sujeita ao trafegar no canal de comunicação, que também está sujeita a alteração maliciosa de um atacante;
- $\boldsymbol{V}()$ é a função de verificação da integridade da mensagem;
- $\boldsymbol{k_c}$ é a chave utilizada para gerar a codificação da mensagem;
- $\boldsymbol{k_v}$ é a chave de verificação.

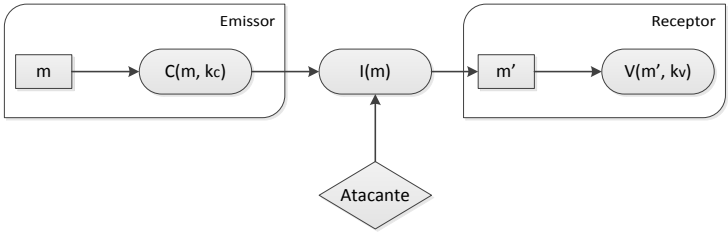


Figura 2: Modelo simples com único canal para verificação de integridade

No modelo apresentado na Figura 2, o emissor da mensagem primeiramente codifica a mensagem, através de uma função de codificação, C(), utilizando uma chave de conhecimento somente dele. Em seguida, envia a mensagem codificada para o receptor. Quando o receptor recebe a mensagem, ele submete a mensagem a uma função de verificação, V(), utilizando uma chave somente de seu conhecimento. Se a função retornar *verdadeiro*, significa que a mensagem está íntegra. Caso contrário, a mensagem foi alterada pela função de interferência, I(), durante o caminho.

2.3 TÉCNICAS COM CRIPTOGRAFIA SIMÉTRICA

Nas técnicas de criptografia simétrica, as chaves k_c e k_v, utilizadas nas funções de C() e V(), respectivamente, são chaves simétricas, isto é, são as mesmas (MAO, 2003).

Geralmente são utilizadas funções MAC em técnicas simétricas. Uma forma de calcular o MAC é parametrizando uma função de hash com uma chave simétrica. Esta técnica é chamada de HMAC (do inglês, *Keyed-hash Message Authentication Code*).

2.3.1 Funções de *Hash*

Funções de *hash* são funções matemáticas que, para cada valor de entrada, produzem uma saída diferente de tamanho fixo. Esta saída é comumente chamada de impressão digital, pois identifica unicamente a mensagem de entrada (SCHNEIER, 1996).

Uma função de *hash* F deve satisfazer as seguintes propriedades (SCHNEIER, 1996):

- **Eficiência**: o cálculo do *hash* deve ser relativamente simples de se calcular. Mais especificamente, o cálculo deve ser feito em tempo polinomial;
- **Resistência à pré-imagem**: deve ser computacionalmente inviável recriar uma mensagem a partir do seu *hash*, ou seja, dado um valor de *hash* h, deve ser computacionalmente inviável encontrar um valor x tal que $h = F(x)$;
- **Resistência à colisão**: deve ser computacionalmente inviável produzir duas mensagens distintas que, quando aplicadas a F, resultem no mesmo *hash*.

Devido a estas características, funções de *hash* tem diversos usos na computação. Por exemplo, em sistemas de autenticação, as funções de *hash* são comumente utilizadas para que o servidor não precise guardar a senha do usuário. Neste tipo de sistema, o servidor guarda apenas o *hash* da senha e durante a autenticação, o cliente computa o *hash* de sua senha e o envia para o servidor. Outro uso é para a verificação de integridade. Por exemplo, uma mensagem pode ser enviada pela rede juntamente com o seu *hash*. Quando o destinatário recebe a mensagem, ele recalcula o *hash* a partir da mensagem recebida. Se os *hashes* forem iguais, a mensagem está íntegra. Se forem diferentes, indica que houve alguma alteração durante a transmissão dos dados

(da mensagem e/ou do seu *hash*), por alguma interferência do canal de transmissão ou de forma proposital por um atacante. Nestes casos, a mensagem deve ser considerada como não íntegra, apesar de não ser possível identificar se a modificação foi na mensagem ou no *hash*.

É importante ressaltar que o avanço das técnicas de criptoanálise e do poder de processamento dos computadores tem sido capaz de comprometer as propriedades dos algoritmos de *hash*. Quando isso acontece, estes algoritmos passam a ser considerados inseguros. Por exemplo, Yajima et al. (YAJIMA et al., 2009) descrevem ataques de colisão sobre o algoritmo SHA-1, enquanto Aumasson et al. (AUMASSON; MEIER; MENDEL, 2009) e Sasaki et al. (SASAKI; AOKI, 2008) apresentam o comprometimento à primeira pré-imagem do algoritmo MD5.

O NIST (do inglês, *National Institute of Standards and Technolog*) periodicamente publica recomendações acerca da segurança do uso dos diferentes algoritmos de *hash*. Na edição atual (BARKER et al., 2012) recomenda-se o uso dos algoritmos de *hash* da família SHA-2 (National Institute of Standards and Technology, 2012). Porém, muitos sistemas ainda utilizam o algoritmo SHA-1 (National Institute of Standards and Technology, 2012).

2.3.2 *Keyed-hash Message Authentication Code*

As funções de *hash* são a base para a verificação da integridade de dados. Porém, em sistemas onde, além da integridade, é necessária a verificação da autenticidade da mensagem, as funções de *hash* já não são mais adequadas. Para este caso, existem as funções MAC. Estas funções utilizam as funções de *hash* e adicionam um novo parâmetro: uma chave secreta. A Equação 2.1 esquematiza de forma simplificada o cálculo de uma função MAC, onde k é a chave secreta, h uma função de *hash* qualquer, m é a mensagem e $||$ representa a operação de concatenação.

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

Pelas propriedades das funções de *hash* listadas na Seção 2.3.1, é possível verificar que, para calcular um MAC válido utilizando uma função de *hash* h e uma chave k para uma mensagem m, é necessário conhecer a chave k e a mensagem m.

As funções MAC, quando utilizam uma função de *hash* são também chamadas de HMAC, e são geralmente calculadas como definido pela RFC2104 (KRAWCZYK; BELLARE; CANETTI, 1997) e mostrado na Equação 2.2, onde:

- **h** é uma 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.

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

2.4 TÉCNICAS COM CRIPTOGRAFIA ASSIMÉTRICA

Nas técnicas de criptografia assimétrica, as chaves k_c e k_v, utilizadas nas funções de C() e V(), respectivamente, são chaves assimétricas, isto é, são diferentes. Geralmente, sendo uma secreta e uma pública (MAO, 2003). A Figura 3 esquematiza o funcionamento da criptografia assimétrica.

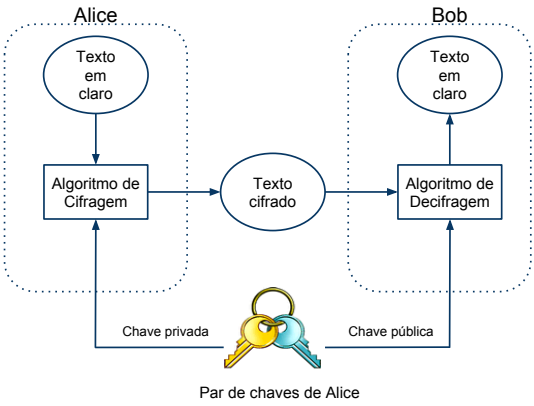


Figura 3: Criptografia Assimétrica

No modelo de criptografia assimétrica, se Alice deseja enviar uma mensagem ao Bob e quer ter certeza de que a mensagem não foi modificada no caminho, ela envia para Bob a mensagem em claro e a mesma mensagem cifrada com a sua chave privada, como esquematizado na Figura 4. Bob, que

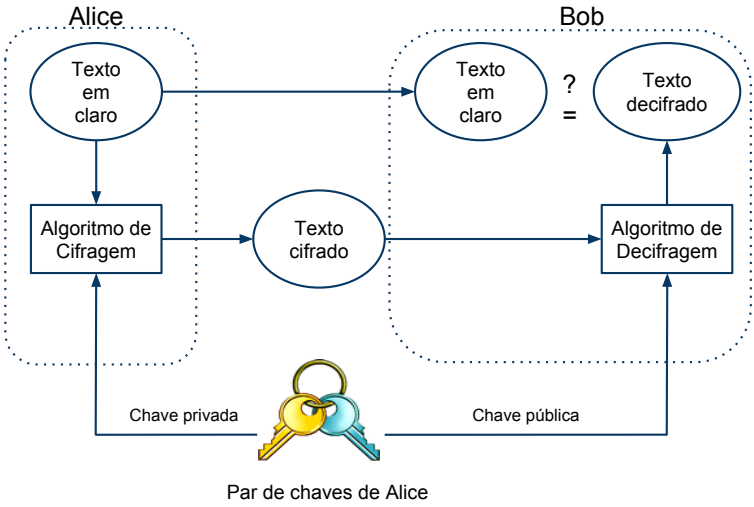


Figura 4: Verificação de integridade com criptografia assimétrica

tem conhecimento da chave pública de Alice (pois a chave é pública e pode ser distribuída livremente), a utiliza para decifrar a mensagem e compara a mensagem decifrada com a mensagem em claro também recebida de Alice. Se as duas mensagens foram iguais, Bob tem certeza que a mensagem não sofreu alterações no caminho.

Este esquema melhora, com relação à criptografia simétrica, o gerenciamento e distribuição das chaves. Na criptografia simétrica, se Alice precisa se comunicar com muitas pessoas, será necessário uma chave diferente para cada pessoa e cada chave terá que ser distribuída e armazenada de forma segura. Já na criptografia assimétrica, cada pessoa possui o seu próprio par de chaves, sendo que apenas a chave privada deve ser armazenada de forma segura (e nunca é distribuída), enquanto que a chave pública pode ser distribuída livremente, como por exemplo, disponibilizada em um repositório. Entretanto, este esquema é ineficiente, pois é necessário que a mensagem seja enviada duas vezes (uma em claro e outra cifrada) e a criptografia assimétrica também é ineficiente em termos de tempo de processamento. Uma forma de

resolver estes problemas é com a assinatura digital, ilustrada na Figura 5. Na

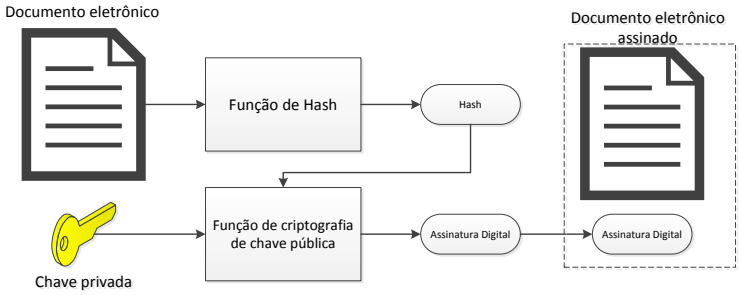


Figura 5: Criação de assinatura digital

assinatura digital, ao invés de cifrar a mensagem, é cifrado apenas o seu *hash*, também conhecido como impressão digital da mensagem, que possui um tamanho fixo e é pequeno. Para verificar a integridade da mensagem, decifra-se a assinatura digital e a compara com o *hash* calculado sobre o documento, conforme ilustrado na Figura 6.

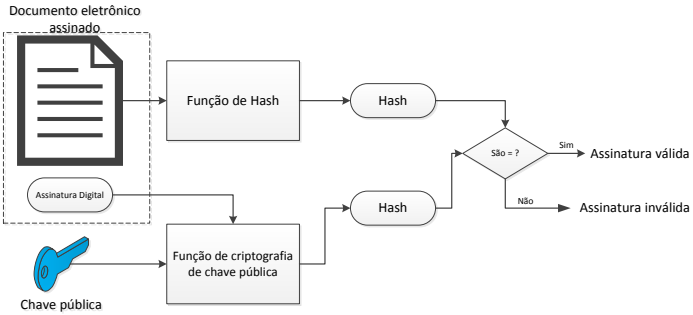


Figura 6: Verificação de assinatura digital

2.5 LOGS

Os SGBDs mantêm registro de todas as operações realizadas que afetam os dados nele armazenados (DATE, 2003). O conjunto desses registros armazenados é determinado log, um arquivo sequencial. Existem diferentes tipos de entradas no log, referentes a cada operação que pode ser realizada no BD. Cada entrada refere-se a uma transação T específica, que é identificada por um *id de transação*. Os tipos mais comuns de entradas do registro de log são (ELMASRI; NAVATHE, 2010):

- (**start_transaction, T**): Indica que a transação T iniciou sua execução;
- (**write_item, T, X, *valor_antigo*, *valor_novo***): Indica que o item X do BD foi atualizado pela transação T;
- (**read_item, T, X**): Indica que o item X do BD foi lido pela transação T;
- (**commit, T**): Indica que a transação T foi concluída com sucesso e que seu efeito pode ser propagado ao BD de forma permanente;
- (**abort, T**): Indica que a transação T foi abortada.

Apesar de manter um log das operações realizadas, os SGBDs não tratam da integridade dos dados, como definida neste trabalho. Em geral, os SGBDs tratam a integridade dos dados do ponto de vista semântico através de regras pré-programadas, ou a integridade de acesso através de autenticação e permissões pré-concedidas. Os logs são mantidos para que o sistema possa se recuperar de falhas, dentre elas (SILBERSCHATZ; KORTH; SUDARSHAN, 2006):

- **Erro lógico**: A transação pode falhar devido a uma divisão por zero, estouro de inteiro, erro de programação, etc;
- **Falha do sistema**: Quedas de energia, falhas de *hardware* ou *software* que causem uma falha do sistema;
- **Falha de disco**: Alguns blocos do disco ou a cabeça de leitura e gravação pode estar com defeito, ocasionando perdas de dados e/ou problemas na leitura/gravação dos dados.

Recuperar o BD de alguma dessas falhas significa que ele é restaurado ao seu estado consistente mais recente antes do momento da falha. Geralmente, para a recuperação há duas estratégias (ELMASRI; NAVATHE, 2010):

- **Falha catastrófica**: Se houver uma falha catastrófica, como uma falha no disco, o método de recuperação restaura uma cópia antiga (*backup*) do BD e reconstrói um estado mais recente refazendo as operações de transações confirmadas que foram salvas no log;
- **Falha não catastrófica**: Quando uma falha não catastrófica ocorre, o método de recuperação identifica mudanças que podem causar inconsistências no BD e, em seguida, desfaz ou refaz operações de transações conforme necessário para voltar o BD a um estado consistente.

Apesar dos logs poderem ser utilizados também para verificar a integridade dos dados, esta não é uma prática comum na literatura. Como os logs registram todas as operações realizadas, eles tendem a crescer de forma exponencial com relação ao número de registros no BD, principalmente para BDs muito dinâmicos, isto é, com muitas consultas e atualizações de dados. Além disso, a verificação da integridade através dos logs também não é trivial. Uma forma de verificar se o BD está íntegro é refazendo todas as operações do log sob uma versão antiga do BD (cuja integridade foi verificada anteriormente). Se, após este processo, os BDs forem iguais, o BD está íntegro. Porém, esta abordagem é estática. Para realizar este procedimento é necessário o congelamento do BD no seu estado atual e, enquanto as operações do log estão sendo refeitas na versão antiga, o BD atual não pode processar nenhuma nova operação.

Outro problema dos logs é que, como eles geralmente são mantidos no SGBD, se um atacante pode modificar o BD ele pode também modificar os logs. Dessa forma, os logs precisam ser protegidos por uma senha de conhecimento e posse somente do cliente ou devem ser mantidos pelo cliente, o que pode ser inviável uma vez que os logs podem crescer em tamanho tanto quanto ou até mais que o próprio BD.

2.6 BANCOS DE DADOS TERCEIRIZADOS

O modelo de BDs terceirizados (*outsourced databases*) é um exemplo típico da arquitetura cliente-servidor. Neste modelo, o servidor possui toda a infraestrutura para armazenar os dados de diversos clientes, além de prover mecanismos eficientes para os clientes executarem todas as operações básicas de um SGBD, isto é, criar, atualizar e consultar os dados. Um exemplo deste modelo é apresentado na Figura 7.

O cliente e o servidor são definidos neste trabalho da seguinte forma:

- **Cliente**: No modelo de BDs terceirizados o cliente não é necessariamente uma entidade única, como um usuário. O cliente pode ser uma

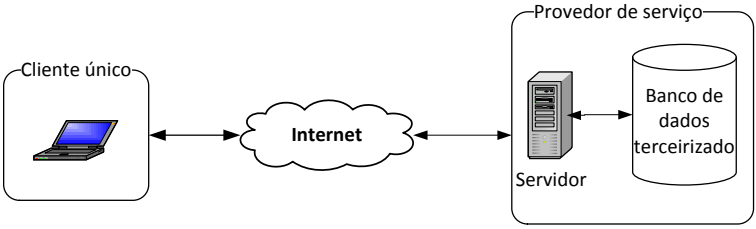


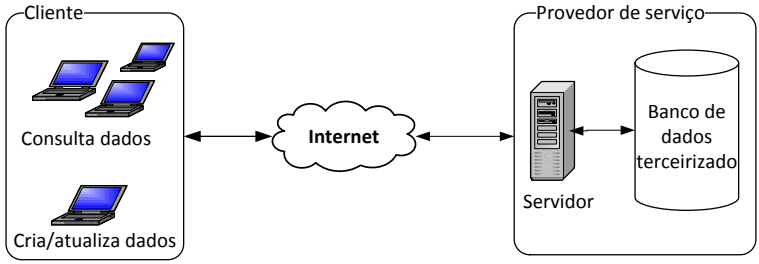
Figura 7: Visão geral de um banco de dados terceirizado

organização e/ou um conjunto de usuários e aplicações que se conectam ao BD;

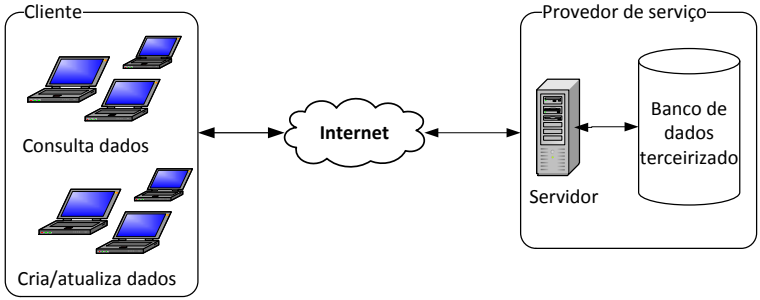
- **Servidor**: O servidor é a entidade responsável por armazenar os dados. Novamente, ele pode não ser uma entidade única, pois os dados podem estar difundidos em diferentes máquinas para balanceamento de carga e redundância.

Neste modelo, assume-se que o cliente confia no servidor para armazenar os seus dados. Mais especificamente, o servidor é responsável pelo disponibilidade, replicação (para balanceamento de carga) e backup dos dados. Entretanto, o servidor pode não ser confiável em relação ao conteúdo dos dados armazenados por ele. Isto é, o servidor pode não ser confiável com relação à confidencialidade e integridade dos dados.

E. Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) apresentam três modelos diferentes de BDs terceirizados. No modelo mais simples, apresentado na Figura 7, há apenas um cliente se comunicando com o servidor. Este cliente cria, atualiza, remove e consulta os seus dados. Nos modelos apresentados na Figura 8, vários clientes se comunicam com o servidor. No primeiro modelo (Figura 8a), apenas um usuário é responsável por criar e atualizar os dados. Os demais usuários apenas consultam os dados. Um exemplo deste modelo são BDs públicos, onde apenas uma entidade mantém os dados, mas qualquer um pode acessar e consultar os dados. No modelo mais genérico (Figura 8b), vários clientes podem criar e atualizar o mesmo conjunto de dados e vários clientes podem consultar os dados. Por exemplo, um sistema de colaboração entre diferentes organizações.



(a)



(b)

Figura 8: Arquitetura de bancos de dados terceirizados com múltiplos clientes

2.7 RESUMO DO CAPÍTULO

Este capítulo descreve as técnicas clássicas para tratar a verificação da integridade de dados e os conceitos necessários para compreendê-las. Inicialmente, foi apresentado o conceito de integridade, como é utilizado neste trabalho. Em seguida, descreveram-se as funções de *hash*, como o HMAC, e a criptografia simétrica, que são amplamente utilizadas em vários sistemas comuns ao nosso dia-a-dia. Por exemplo, repositórios de dados que geralmente, além do dado em si, armazenam também o *hash* do dado para que se possa verificar a integridade após fazer o *download*. Posteriormente, descreveram-se a criptografia assimétrica e a assinatura digital, que são utilizadas, por exemplo, em *websites* seguros. Neste caso, a assinatura digital é utilizada para, além da integridade, garantir a autenticidade. Em seguida, descreveram-se os siste-

mas de logs dos SGBDs. Estes, por sua vez, são voltados para a recuperação dos dados em caso de falhas. Por fim, descreveram-se os modelos de BDs terceirizados, necessários para o entendimento das premissas adotadas no desenvolvimento dos métodos propostos neste trabalho.

3 TRABALHOS RELACIONADOS

3.1 INTRODUÇÃO

Segurança em BDs tem sido estudada extensivamente pelas comunidades de BD e criptografia. Recentemente, algumas técnicas foram propostas para verificar a integridade dos dados salvos em um BD. Estas técnicas geralmente abordam a corretude e/ou completude dos dados e consultas SQL. Do ponto de vista de integridade, corretude está relacionada à alteração não autorizada ou indevida de um dado. Por outro lado, completude refere-se ao resultado dos comandos SQL. Quando um cliente envia uma consulta SQL a um servidor de BD, ele espera como resposta um conjunto de dados. Este conjunto de dados é dito completo se ele contém todos os dados que satisfazem a consulta SQL enviada pelo cliente.

A maior parte dos trabalhos publicados, relacionados com integridade de dados, baseiam-se no uso de estruturas de dados autenticadas, como as árvores de *Merkle* (MERKLE, 1989) e as *SkipLists* (PUGH, 1990). Estas estruturas são comumente utilizadas quando é necessário dar garantias ao usuário que um dado ou um conjunto de dados está íntegro ou não.

O restante deste capítulo está organizado da seguinte forma. A Seção 3.2 apresenta os trabalhos correlatos baseados em estruturas de autenticação. Os trabalhos baseados em assinatura digital são apresentados na Seção 3.3. A Seção 3.4 apresenta alguns trabalhos que não se baseiam em técnicas criptográficas para a verificação da integridade dos dados. A Seção 3.5 apresenta os trabalhos que tratam da recuperação das tuplas. Por fim, a Seção 3.6 conclui este capítulo, fazendo uma análise comparativa entre os trabalhos apresentados.

3.2 TÉCNICAS BASEADAS EM ESTRUTURAS AUTENTICADAS

A árvore de *Merkle* (MHT, do inglês *Merkle Hash Tree*) é uma árvore binária que contêm em suas folhas os resumos criptográficos dos dados, ou seja, a folha associada ao dado x contém o valor de $h(x)$, onde $h()$ é uma função de *hash*, como o SHA-1 (National Institute of Standards and Technology, 2012). Os nodos intermediários da árvore contêm os resumos criptográficos da concatenação dos resumos criptográficos dos nodos filhos. Por exemplo, um nodo com os filhos c_1 e c_2 é calculado pela função $h(c_1||c_2)$, onde $||$ representa a concatenação de dois valores. A Figura 9 mostra um exemplo de

uma árvore de *Merkle*. É importante ressaltar que os nodos expressos por "DADO n" não fazem parte da árvore, são apenas ilustrativos e representam quais os dados que a árvore contém.

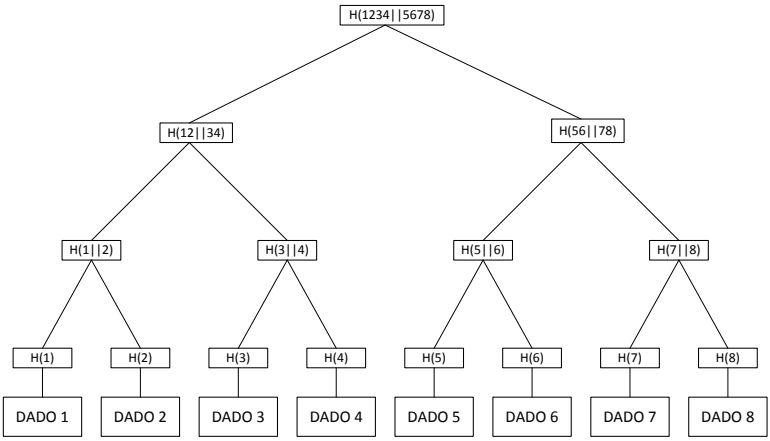


Figura 9: Árvore de *Merkle*

Li et al. (LI et al., 2006) apresenta a *Merkle B-Tree* (MB-Tree), que é uma extensão da árvore B^+-tree de BDs relacionais para armazenar os valores dos resumos criptográficos dos dados, assim como na árvore de *Merkle*. Esta árvore é então utilizada para prover provas de corretude e completude das consultas executadas no servidor de BD. Os experimentos apresentados mostram que a solução é eficiente. Entretanto, há a necessidade de estender a B^+-tree utilizada pelos SGBDs para uma MB-Tree. Este requisito pode dificultar a implantação da proposta em sistemas reais, sobretudo em sistemas já existentes.

A *SkipList* (apresentada na Figura 10) é uma estrutura de dados que armazena, de forma eficiente, um conjunto de dados ordenados e dá suporte às operações de busca, inserção e remoção. A *SkipList* possui um conjunto de elementos que é armazenado como uma sequência de listas ligadas $S_0, S_1, ..., S_t$. Estas listas são os níveis e os elementos de cada lista são os nodos, sendo que o primeiro nível S_0 armazena todos os elementos de S de forma ordenada. Os próximos níveis S_i, com $0 < i <= t$, armazenam um subconjunto dos elementos de S_{i-1}. A forma como estes subconjuntos são definidos determina o tipo da *SkipList*. O método mais comum para definir um subconjunto é escolhendo de forma randômica, com probabilidade

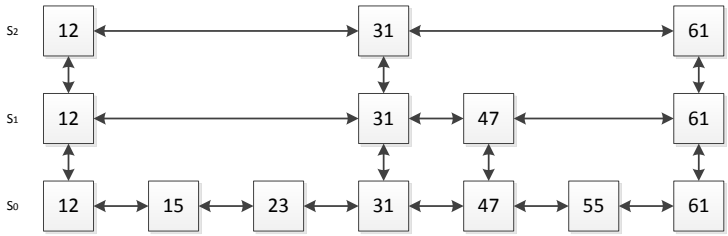


Figura 10: *SkipList*

de $1/2$, os elementos do nível anterior. Porém, existem também métodos determinísticos, como o apresentado por Munro et al. (MUNRO; PAPADAKIS; SEDGEWICK, 1992).

Di Battista e Palazzi (BATTISTA; PALAZZI, 2007) propõem a implementação de uma *SkipList* em uma tabela relacional. Neste esquema uma nova tabela, chamada de *security table*, é adicionada ao esquema do DB. Esta tabela armazena uma skipList autenticada e é utilizada para prover garantias da autenticidade e completude das consultas executadas pelo servidor de BD. Ao contrário da proposta de Li et al., a proposta de Di Battista e Palazzi não requer alterações no *kernel* do SGBD. Como apenas uma nova tabela é adiciona ao esquema, a implementação da proposta pode ser feita como um *plug-in* para o SGBD.

Semelhante ao trabalho de Di Battista e Palazzi, Miklau e Suciu (MIKLAU; SUCIU, 2005) propõem a implementação de uma árvore de *hash* em uma tabela relacional. Porém, neste esquema o cliente deve armazenar uma pequena porção dos dados localmente, que é utilizada para fornecer as provas de integridade. Para verificar a integridade, o cliente precisa recalcular a árvore e comparar o nodo raiz da árvore com o dado que está armazenado localmente. Apesar desta abordagem também poder ser implementada como um *plug-in* para o SGBD, o uso de uma árvore de *hash* compromete a sua eficiência. Isto deve-se ao fato de que, para cada inserção no BD, a árvore precisa ser reconstruída.

3.3 TÉCNICAS BASEADAS EM ASSINATURA DIGITAL

Diferente dos trabalhos apresentados anteriormente, E. Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) propõem o uso de assinaturas digitais para prover integridade dos dados. Em seu trabalho, ao invés de criar uma nova tabela, eles criam uma nova coluna em cada tabela, que armazena o valor da assinatura da concatenação das demais colunas de cada linha da tabela. Para verificar a integridade de uma determinada linha, o cliente faz a verificação da assinatura. Este trabalho foi estendido por Narasimha e Tsudik (NARASIMHA; TSUDIK, 2005) para também prover provas da completude das consultas.

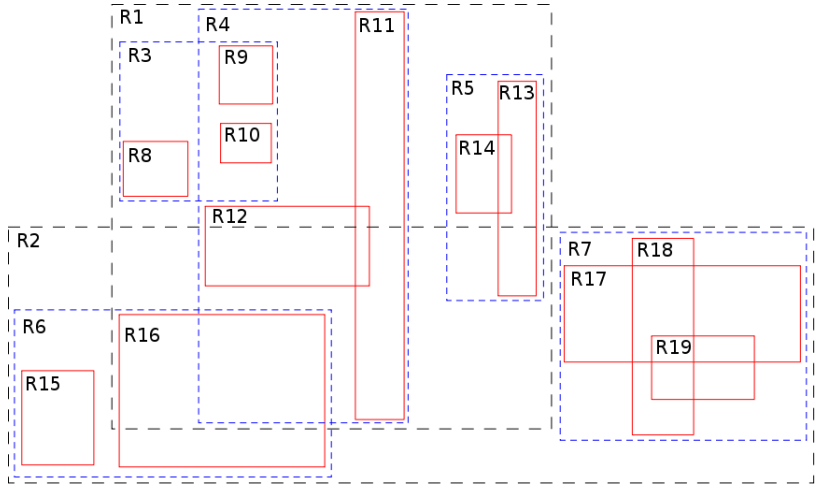
A motivação de E. Mykletun et al. para o uso de assinaturas é permitir a verificação de integridade em ambientes onde vários usuários acessam a mesma base de dados. De fato, a proposta dos autores se mostra compatível e condizente com tais ambientes, porém, para cenários monousuário, a solução não se mostra tão eficiente. Neste caso, é preferível o uso de criptografia simétrica, em que é possível atingir um mesmo nível de segurança com chaves menores, além dos algoritmos serem mais eficientes.

3.4 OUTRAS TÉCNICAS

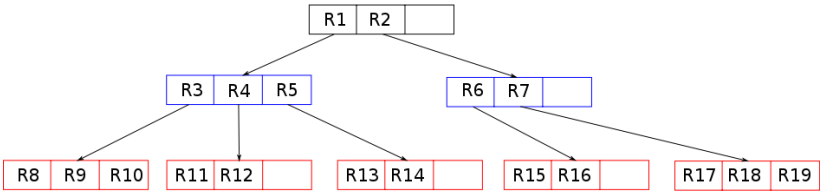
Existem também alguns trabalhos que não são baseadas em criptografia (KAMEL, 2009; XIE et al., 2007). Eles empregam técnicas probabilísticas que, de modo geral, inserem uma quantidade de dados de controle no BD para depois serem utilizadas para verificar se todos os dados que satisfazem a uma consulta foram ou não retornados pelo SGBD. A principal desvantagem destas técnicas é que elas verificam apenas se os dados foram removidos das consultas, mas não conseguem verificar a integridade dos dados em si.

Ibrahim Kamel (KAMEL, 2009) propõe uma técnica para a detecção de alterações maliciosas baseada em marcas d'água. As marcas d'água são inseridas em árvores R (GUTTMAN, 1984), que são uma extensão das árvores B para objetos multidimensionais. A Figura 11 mostra um exemplo de uma árvore R. A marca d'água consiste em rearranjar as entradas dos nodos da árvore com base em um valor inicial secreto. Por exemplo, as entradas poderiam ser rearranjadas com base no valor de x do canto inferior esquerdo de cada entrada, chamado de *Lowx*.

A Figura 12 mostra um conjunto de entradas de um nodo da árvore R. As entradas destacadas, A e B, estão ordenadas de acordo com o seu *Lowx*, de forma que a entrada B está mais a esquerda pois $L_b < L_a$. A linha pontilhada demonstra um possível ataque que pode ser feito à base de dados, oca-



(a) Ordenação espacial



(b) Representação da árvore

Figura 11: Exemplo de árvore R

sionando a alteração do *Lowx* da entrada A de L_a para L'_a. Neste cenário, ao verificar a integridade do nodo da árvore, o SGBD tentará recriar a ordenação das entradas, com base no exemplo de valor inicial secreto apresentado anteriormente. Ao recriar a ordenação, notar-se-á que, no nodo, a entrada A agora está antes da entrada B, significando que houve um ataque.

Xie et. al. (XIE et al., 2007) propõe um método probabilístico para verificar a integridade dos dados, inserindo uma quantidade de tuplas falsas no BD. Dessa forma, há uma probabilidade de que as consultas realizadas no SGBD retornem algumas das tuplas inseridas propositalmente. Os autores afirmam que, analisando as tuplas retornadas, é possível verificar se todas as tuplas que satisfazem a consulta foram ou não retornadas. Uma vantagem deste trabalho é o suporte a diferentes tipos de consulta, como consultas com *join*. Os trabalhos baseados em criptografia codificam todos ou um subconjunto dos atributos de cada linha das tabelas. Dessa forma, todos os atributos

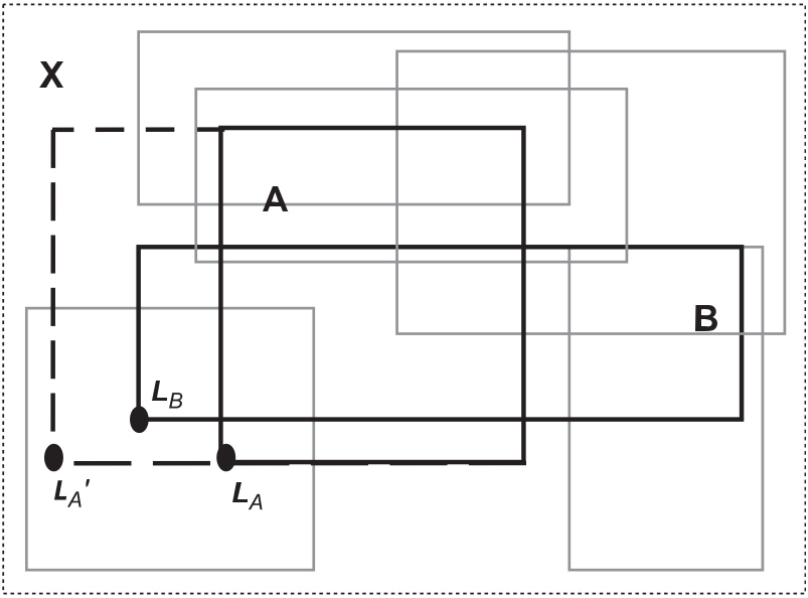


Figura 12: Exemplo de ataque, alterando o tamanho da entrada A (KAMEL, 2009)

codificados devem ser retornados, mesmo que não façam parte da consulta, para ser possível a verificação da integridade. Porém, como o trabalho de Xie et. al. não verifica a integridade dos dados, mas apenas a completude da consulta, não é necessário que todos os atributos sejam retornados, flexibilizando o tipo de consultas que o SGBD pode receber. A principal desvantagem deste trabalho é que ele limita-se a verificar apenas a completude das consultas, não provendo formas de verificar se os dados em si estão corretos.

3.5 RECUPERAÇÃO DE TUPLAS

Como apresentado no Capítulo 2, as técnicas de recuperação implementadas nos SGBDs são voltadas para a recuperação de falhas, ou seja, não tratam o caso de ataques ao BD. Entretanto, a recuperação do BD em casos de ataque também é feita através de logs (LIU; AMMANN; JAJODIA, 2000; AMMANN; JAJODIA; LIU, 2002; ZHU et al., 2008; CHAKRABORTY; MAJUMDAR; SURAL, 2010). Além disso, estes trabalhos requerem um terceiro agente para

identificar as alterações maliciosas, chamada de detector de intrusão, presente nas arquiteturas de BDs tolerantes a intrusões.

A Figura 13 apresenta uma arquitetura genérica de um sistema de BD tolerante a intrusões. Nesta arquitetura tem-se os seguintes elementos (LIU, 2002):

- **Mediador**: Funciona como um *proxy* para as transações dos usuários e do reparador de danos;
- **Detector de intrusões**: Monitora e analisa o BD em busca de transações maliciosas;
- **Gerenciador de reparos**: É composto por dois módulos: avaliador de danos e reparador de danos. O avaliador de danos identifica as transações maliciosas e quais transações legítimas foram afetadas; O reparador de danos tenta reparar as transações legítimas que foram afetadas pelas transações maliciosas.

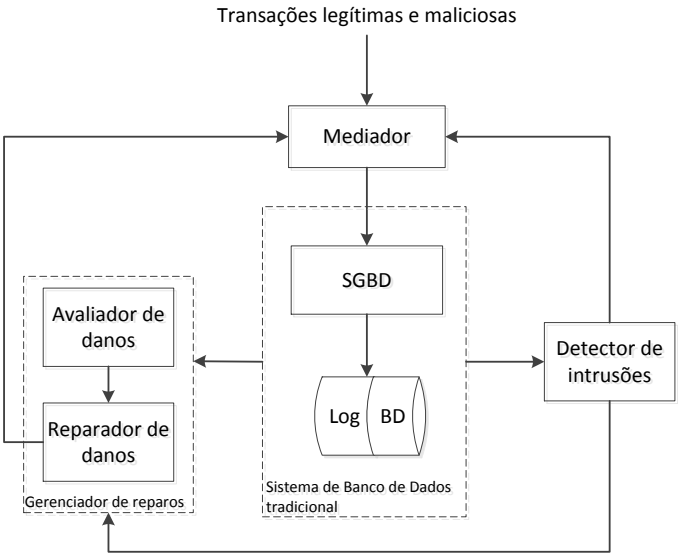


Figura 13: Arquitetura de um BD tolerante a intrusões

Na arquitetura apresentada na Figura 13, o detector de intrusões fica constantemente monitorando o BD por transações maliciosas. Quando o detector de intrusões identifica uma transação maliciosa, são disparados alarmes para o avaliador de danos que, através dos logs, irá identificar o dano causado. Com base nos logs e sabendo qual parte do BD foi afetada, o reparador de danos envia comandos SQL para o mediador para reverter os danos causados pela transação maliciosa.

Um dos grandes problemas desta arquitetura é que, mesmo o monitoramento e reparo dos danos sendo feito em tempo real, uma transação maliciosa pode se espalhar rapidamente, causando sérios danos ao BD. Por exemplo, quando uma transação T_i lê um dado x atualizado por uma transação maliciosa T_j, T_i é diretamente afetada por T_j. Uma terceira transação, T_k, pode ser diretamente afetada por T_i, mas não por T_j. Ainda assim, T_k é indiretamente afetada por T_j, e assim sucessivamente.

Em decorrência do problema da propagação dos efeitos de transações maliciosas, os logs precisam armazenar todas as transações realizadas. Isto pode acarretar um crescimento exponencial do tamanho do arquivo de logs, principalmente em sistemas dinâmicos, com uma alta frequência de consultas e atualizações dos dados. Além disso, no modelo de BDs terceirizados, considera-se que o servidor não é confiável. Portanto, os logs precisam ser armazenados pelo cliente, ou protegidos de alguma forma no servidor.

3.6 RESUMO DO CAPÍTULO

Este capítulo apresenta os trabalhos relacionados ao escopo desta dissertação. Os trabalhos correlatos podem ser divididos em duas principais vertentes: trabalhos baseados em estruturas autenticadas, como a árvore de Merkle; e trabalhos baseados em assinatura digital. A tabela 1 apresenta um comparativo destes trabalhos. As características consideradas para a comparação são as seguintes:

- **Implementação**: *Flexível* significa que a proposta não requer alterações no SGBD e pode ser implementada em diferentes níveis, como na aplicação ou um *plugin* para o SGBD; *No SGBD* significa que a proposta requer alterações na estrutura interna do SGBD, usualmente nas suas estruturas de dados;

- **Criptografia**: *Baixo custo* significa apenas o uso de criptografia simétrica; *médio custo* significa o uso de criptografia simétrica em conjunto com criptografia assimétrica; *alto custo* significa o uso de apenas criptografia assimétrica. As propostas que não utilizam criptografia tem

um custo igual ou menor à criptografia simétrica;

- **Corretude**: Se a proposta consegue identificar alterações maliciosas nos dados;

- **Completude**: Se a proposta consegue verificar se todos os dados que satisfazem a uma consulta foram retornados pelo SGBD.

Tabela 1: Comparação entre os trabalhos relacionados

	Implementação	Criptografia	Corretude	Completude
(KAMEL, 2009)	No SGBD	Não utiliza	Sim	Não
(LI et al., 2006)	No SGBD	Médio custo	Sim	Sim
(BATTISTA; PALAZZI, 2007)	Flexível	Médio custo	Sim	Sim
(MIKLAU; SUCIU, 2005)	Flexível	Médio custo	Sim	Sim
(MYKLETUN; NARASIMHA; TSUDIK, 2006)	Flexível	Alto custo	Sim	Não
(NARASIMHA; TSUDIK, 2005)	Flexível	Alto custo	Sim	Sim
(XIE et al., 2007)	Flexível	Não utiliza	Não	Sim

Os trabalhos baseados em estruturas autenticadas utilizam diferentes tipos de estruturas de dados. Li et al. (LI et al., 2006) propõe uma adaptação da árvore de *Merkle* para ser implementada na B+-tree dos BDs relacionais. Como apresentado na tabela 1, podemos ver que a proposta de Li et al. necessita de alterações no SGBD e suporta a verificação da corretude dos dados, bem como da completude de consultas. Como utiliza criptografia simétrica em conjunto com criptografia assimétrica, o custo foi definido como médio. Ao contrário de Li et al., Di Battista e Palazzi (BATTISTA; PALAZZI, 2007) propõem a implementação da estrutura autenticada em uma tabela. Além disso, Di Battista e Palazzi utilizam a *SkipList*. Com relação à tabela 1, pode-se notar que a grande vantagem em relação ao trabalho de Li el at. é a flexibilidade de implementação. De forma semelhante, Miklau e Suciu (MIKLAU; SUCIU, 2005) também propõem a implementação de uma estrutura autenticada em uma tabela relacional, com a diferença que a estrutura proposta é a árvore de *hash*. Dessa forma, a principal diferença entre os dois trabalhos está na eficiência das estruturas escolhidas.

Os trabalhos baseados em assinatura digital têm como maior desvantagem a eficiência. O custo de uma assinatura digital é de 100 a 1000 vezes maior que o custo de uma operação de *hash* ou de criptografia simétrica. Entretanto, dentre os trabalhos estudados, o trabalho de E. Mykletun et al. (MYKLETUN; NARASIMHA; TSUDIK, 2006) é o único que suporta ambientes mais complexos, onde há diversos usuários ou aplicações atualizando e verificando a integridade do mesmo conjunto de dados. Apesar disso, o trabalho não deixa claro como é feito o gerenciamento das chaves criptográficas. Isto é, num ambiente onde x usuários podem atualizar os dados, quando um usuário atualiza um determinado dado, ele assina este dado com sua chave privada. Quando algum outro usuário for verificar a assinatura, ele precisa saber qual chave pública utilizar, dentre os x usuários do sistema.

Por fim, analisando a Tabela 1 conclui-se que os trabalhos existentes, que tratam tanto da corretude quanto da completude dos dados, possuem duas principais limitações. Uma delas é a necessidade de alterar o *kernel* do SGBD. Esta limitação pode impactar na aplicabilidade da solução para diferentes cenários, limitando o seu uso. Exemplos destes cenários são os de BDs terceirizados, onde o cliente não controla o SGBD e, portanto, não pode fazer tais modificações e os ambientes de produção que utilizam SGBDs de código fechado, com licenças que não possibilitem tais modificações. A outra limitação está no uso de funções criptográficas e/ou estruturas de dados de alto custo de processamento.

O presente trabalho supera tais limitações, apresentando métodos independentes de SGBD e que utilizam funções criptográficas de baixo custo. Estes métodos são apresentados no próximo capítulo.

4 MÉTODOS PROPOSTOS

4.1 INTRODUÇÃO

Para obter um método eficiente para prover integridade de dados, este trabalho propõe o uso de funções criptográficas de baixo custo, no caso, as funções MAC (BELLARE; CANETTI; KRAWCZYK, 1996; KRAWCZYK; BELLARE; CANETTI, 1997). Basicamente, a proposta consiste em adicionar uma nova coluna em cada tabela do BD. Esta coluna armazena o resultado da função MAC aplicada sobre as demais colunas da tabela. Esta função também necessita de uma chave criptográfica, que deve ser de conhecimento apenas do cliente.

Como exemplo, suponha que se deseja prover integridade para uma tabela chamada "Pessoa". Esta tabela tem três colunas, *Id*, *Nome* e *Email*, mais a coluna para armazenar o resultado da função MAC. Considerando um cliente utilizando uma chave k e $||$ representando a concatenação de valores, o cálculo do MAC é feito da seguinte forma:

$$MAC(k, m) = MAC(k, Nome||Email) \tag{4.1}$$

Como resultado, a tabela de exemplo, com a coluna para o MAC, é mostrada na Tabela 2. É importante ressaltar que o cálculo do MAC pode ser feito sobre todas as colunas da tabela ou apenas um subconjunto de colunas. Entretanto, apenas as colunas que entrarem no cálculo do MAC podem ser autenticadas quanto a sua integridade.

Tabela 2: Exemplo de uma tabela com uma coluna para o MAC

Id	Nome	Email	MAC
1	João	joao@mail.com	9c35...022b
2	Maria	maria@mail.com	89a1...a4bf
3	José	jose@mail.com	cd61...352b

O simples uso de uma função MAC garante a integridade das operações de *INSERT* e *UPDATE*. Entretanto, o BD ainda é vulnerável à remoção maliciosa de linhas das tabelas. Neste trabalho, isto é tratado através do encadeamento dos MACs. Este novo algoritmo, chamado neste trabalho de CMAC, do inglês *Chained-MAC*, é calculado como mostra a Equação 4.2, onde n é a linha atual, k a chave utilizada pelo cliente e MAC_n o valor MAC da linha n. Para armazenar o valor do CMAC, uma nova coluna é criada para

cada tabela do BD.

$$CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n)) \tag{4.2}$$

Como há o encadeamento de linhas, não é possível para um atacante remover uma linha de forma maliciosa, pois seria necessário recalcular o CMAC da linha posterior. Como o atacante não possui a chave para o cálculo do MAC, ele não consegue atualizar o CMAC. Entretanto, ainda há um problema: a remoção da última linha não é detectada. Para resolver este problema, o encadeamento das linhas é feito de forma circular. Dessa forma, a última linha é ligada com a primeira. Com isso, o cálculo do CMAC permanece da mesma forma. Apenas há uma pequena modificação na operação *INSERT*. Após a inserção da nova linha, deve-se recalcular e atualizar o CMAC da primeira linha.

A Figura 14a mostra um exemplo de uma tabela com as colunas MAC e CMAC. Nesta figura, os círculos indicam os valores dos MAC e CMAC de cada linha e as setas indicam o MAC que foi utilizado para calcular o CMAC. Por exemplo, para a linha com $id = 1$ foi utilizado o MAC da própria linha e da linha anterior ($id = 0$). Como pode-se notar, o CMAC só precisa ser atualizado quando o MAC da própria linha ou da linha anterior é atualizado, ou seja, não há um efeito cascata na atualização dos CMACs, pois quando o CMAC é atualizado, os MACs permanecem inalterados. A Figura 14b mostra um possível ataque que poderia ser feito à tabela. Neste exemplo, um atacante removeu a linha destacada (linha do meio). Numa remoção autorizada via aplicação, seria necessário ligar as linhas com $id = 1$ e $id = n - 1$. Porém, como o atacante não possui a chave secreta para recalcular o CMAC da linha com $id = n - 1$, o encadeamento fica quebrado. Quando a aplicação for validar o CMAC da linha com $id = n - 1$, ela utilizará o MAC da linha com $id = 1$ e a comparação não irá fechar, indicando que pelo menos uma linha foi removida entre as linhas com $id = 1$ e $id = n - 1$.

O restante deste capítulo está organizado da seguinte forma. A Seção 4.2 apresenta a arquitetura geral do sistema, além das premissas e características levadas em conta para a proposta da solução. A Seção 4.3 apresenta os algoritmos para adicionar e atualizar linhas, calculando o MAC e CMAC. A Seção 4.4 apresenta o algoritmo para remover linhas, calculando o MAC e CMAC e as Seções 4.5 e 4.6 apresentam os algoritmos para verificar a integridade de dados e consultas, respectivamente. A Seção 4.7 apresenta a proposta deste trabalho para recuperar o valor original dos dados, enquanto que a seção 4.8 discute o problema do gerenciamento das chaves utilizadas nos métodos propostos e apresenta soluções para estes problemas. A Seção 4.9 apresenta a proposta para verificar se os dados retornados pelo servidor correspondem a atualização mais recente destes dados. A Seção 4.10 apresenta e analisa a

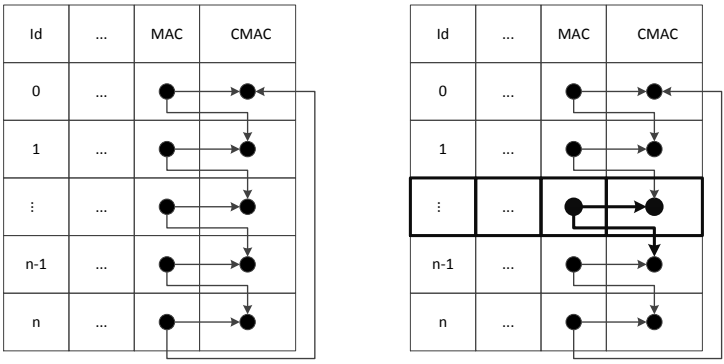


Figura 14: Exemplo de uma tabela com CMAC

flexibilidade e as diferentes formas de implementar as técnicas apresentadas neste trabalho. Por fim, a Seção 4.11 faz um resumo do capítulo.

4.2 ARQUITETURA GERAL DO SISTEMA

Com base nos modelos de BDs terceirizados apresentados no Capítulo 2, alguns aspectos foram levados em consideração para o desenvolvimento da proposta para o controle da integridade de dados e para a análise/comparação dos trabalhos relacionados:

- **Ataque**: O ponto principal de ataque no modelo de BDs terceirizados é o servidor e os ataques podem ser externos e internos. Um ataque externo se caracteriza quando alguém não autorizado consegue acesso ao BD, sendo capaz de ler, inserir, remover e alterar os dados. Já um ataque interno se caracteriza quando alguém tem autorização para acessar o BD, porém não deveria realizar modificações nem ler os dados salvos. Por exemplo, o servidor tem total controle sobre os dados armazenados pelo cliente, porém não deveria ler nem alterar os dados salvos pelo cliente. Um exemplo de ataque que poderia ser feito em um BD terceirizado é a venda de informações estratégicas de uma organização para terceiros. Apesar da dificuldade em se evitar tais tipos de ataques, uma boa solução para controle de integridade deve ser capaz de detectar

qualquer modificação não autorizada realizada nos dados;

- **Flexibilidade**: O método deve ser flexível o suficiente para dar suporte aos diferentes modelos apresentados, além da arquitetura clássica, onde o próprio cliente mantém os seus dados localmente em um SGBD. Por exemplo, o método pode ser implementado diretamente no cliente (suportando os modelos das Figuras 8a e 8b; pode ser implementado em um *fron-end*, conforme apresentado na Figura 15; ou pode ser implementado diretamente no servidor, como um *plugin* para o SGBD. Também é desejável que o controle da integridade possa ser feito em diferentes níveis de granularidade. Por exemplo, a nível de tabela, de tupla, etc.

- **Eficiência**: O método deve causar a menor sobrecarga possível no processamento, como por exemplo, através do uso de funções criptográficas de baixo custo. Como em qualquer sistema de segurança, um ou mais aspectos do sistema são degradados para se obter algum nível de segurança. Portanto, deve-se manter um balanceamento entre a eficiência e o nível de controle de integridade desejado;

- **Armazenamento**: Assim como na eficiência, o método deve causar a menor sobrecarga no armazenamento possível. Entretanto, este ponto é menos crítico que a eficiência, pois ao terceirizar o BD, o custo de armazenamento diminui consideravelmente. Isto é, o cliente pode armazenar mais dados com um custo menor que se manter os dados localmente;

- **Compatibilidade**: O método deve ser compatível com as tecnologias já existentes. Isto é, é desejável que o método não necessite realizar alterações drásticas no cliente nem no servidor. Por exemplo, um método que requer alterações no SGBD pode ser inviável de ser implementado no modelo de BD terceirizado, pois o cliente não tem controle na infraestrutura do servidor. Além disso, é improvável que o servidor faça alterações em sua infraestrutura para atender apenas um cliente.

Além disso, neste trabalho também é inserido um terceiro elemento, o *front-end*, como mostrado na figura 15. O *front-end* controla todos os aspectos da integridade dos dados. Isto é, ele é responsável por fazer os cálculos do MAC e CMAC, gerenciar as chaves criptográficas para tais cálculos, verificar a integridade dos dados e, caso o dado tenha sido alterado maliciosamente, recuperar o valor original do dado. Para isso, o *front-end* é dividido da seguinte forma (como apresentado na Figura 16):

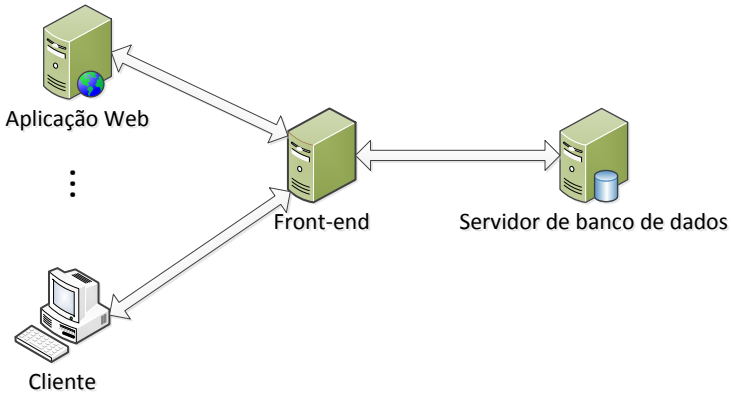


Figura 15: Visão geral da arquitetura do sistema

- **Base de chaves**: Base de dados responsável por armazenar e gerenciar as chaves criptográficas utilizadas para o cálculo do MAC e CMAC. A estrutura desta base e como ela é gerenciada são descritos na Seção 4.8;
- ***Cache***: Um sistema simples de *cache* para tratar das otimizações apresentadas no Capítulo 5.
- **Logs**: Estrutura de logs para possibilitar a recuperação do valor original dos dados, quando estes sofrerem alguma alteração não autorizada. Esta estrutura e como ela é gerenciada são descritos na Seção 4.7.

4.3 ADICIONANDO E ATUALIZANDO LINHAS

Adicionar novas linhas na tabela, utilizando a abordagem deste trabalho é bastante simples. Usando a Tabela 2 como exemplo, para adicionar uma nova entrada com os dados Nome = Pedro, Email = pedro@mail.com, é necessário calcular o MAC e executar o comando *SQL* abaixo, onde 'de65...7988' é o valor do MAC calculado:

```
INSERT INTO Pessoa (Nome, Email, mac)
VALUES ('Pedro',
```

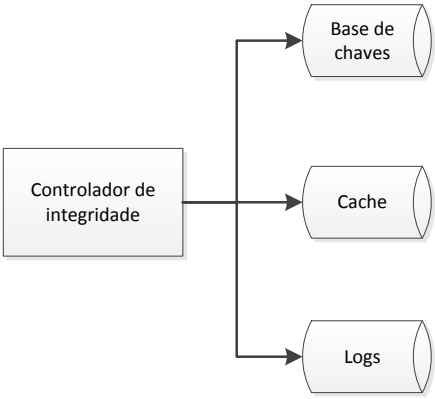


Figura 16: Visão detalhada do *front-end*

```
'pedro@mail.com',
'de65...7988');
```

O CMAC deve ser calculado utilizando o MAC da linha anterior e o MAC da linha que está sendo adicionada. Continuando com o exemplo anterior, o CMAC seria calculado como $CMAC(k, MAC(k,' cd61...352b' \oplus' de65...7988'))$. A operação de *INSERT*, utilizando MAC e CMAC, é detalhada no Algoritmo 1.

A atualização de linhas é similar à operação de *INSERT*. As colunas *MAC* e *CMAC* devem ser recalculadas, considerando os valores a serem atualizados. Entretanto, há um passo extra, que consiste na atualização do CMAC da linha $n + 1$. Este passo é necessário pelo fato de que a linha n está sendo atualizada. A operação de *UPDATE*, utilizando MAC e CMAC, é detalhada no Algoritmo 2.

4.4 REMOVENDO LINHAS

Para remover uma linha de uma tabela com a coluna de MAC, nenhuma operação adicional se faz necessária. Isto se deve ao fato de que,

Algoritmo 1: Algoritmo para a inserção de uma nova linha, calculando os valores de MAC e CMAC.

Entrada: Uma tupla T e seus atributos $t_0, \ldots, t_i$, de acordo com o esquema de uma tabela R e uma chave criptográfica k, utilizada para calcular o MAC.
Passo 1: Calcular $MAC_{n+1} = MAC(k, t_0 || \ldots || t_i)$
Passo 2: Calcular o CMAC
Passo 2.1: Obter o MAC da última linha da tabela R, MAC_n
Passo 2.2: Calcular
$CMAC_{n+1} = MAC(k, (MAC_n \oplus MAC_{n+1}))$
Passo 3: Inserir a tupla T, juntamente com os valores calculados de MAC_{n+1} e $CMAC_{n+1}$
Passo 4: Calcular o CMAC da primeira linha
Passo 4.1: Obter o MAC da primeira linha de R, MAC_1
Passo 4.2: Calcular $CMAC_1 = MAC(k, (MAC_n \oplus MAC_1))$
Passo 4.3: Atualizar o valor calculado de $CMAC_1$

Algoritmo 2: Algoritmo para a atualização de uma linha, calculando os valores de MAC e CMAC.

Entrada: Uma tupla T e seus atributos $t_0, \ldots, t_i$, de acordo com o esquema de uma tabela R e uma chave criptográfica k, utilizada para calcular o MAC.
Passo 1: Calcular $MAC_n = MAC(k, t_0 || \ldots || t_i)$
Passo 2: Calcular o CMAC
Passo 2.1: Obter o MAC da linha anterior de R, MAC_{n-1}
Passo 2.2: Calcular
$CMAC_n = MAC(k, (MAC_{n-1} \oplus MAC_n))$
Passo 3: Realizar um *UPDATE* na tupla T, juntamente com os valores calculados de MAC_n e $CMAC_n$
Passo 4: Calcular o CMAC da linha $n+1$. Se a enésima linha da tabela for a última linha, então a linha $n+1$ a ser considerada será a primeira linha da tabela.
Passo 4.1: Obter o MAC da linha $n+1$ de R, MAC_{n+1}
Passo 4.2: Calcular
$CMAC_{n+1} = MAC(k, (MAC_n \oplus MAC_{n+1}))$
Passo 4.3: Atualizar o valor calculado de $CMAC_{n+1}$

utilizando apenas a coluna para o MAC, não é possível verificar a integridade da tabela contra remoções maliciosas de linhas. Com o uso do CMAC, é necessário recalcular o valor do CMAC da próxima linha. A operação de *DELETE*, utilizando MAC e CMAC, é detalhada no Algoritmo 3.

Algoritmo 3: Algoritmo para a remoção de uma linha, recalculando os valores de MAC e CMAC.

Entrada: Uma tupla T, de acordo com o esquema de uma tabela R e uma chave criptográfica k, utilizada para calcular o MAC.
Passo 1: Remover a tupla T
Passo 2: Calcular o CMAC da linha $n + 1$. Se a enésima linha da tabela for a última linha, então a linha $n + 1$ a ser considerada será a primeira linha da tabela.
Passo 2.1: Obter o MAC da linha $n + 1$ de R, MAC_{n+1} e o MAC da linha $n - 1$, MAC_{n-1}
Passo 2.2: Calcular
$CMAC_{n+1} = MAC(k, (MAC_{n-1} \oplus MAC_{n+1}))$
Passo 2.3: Atualizar o valor calculado de $CMAC_{n+1}$

4.5 VERIFICANDO A INTEGRIDADE DE UMA TABELA

A integridade dos dados pode ser provida em diferentes níveis de granularidade. É possível verificar a integridade de uma tabela, de uma linha ou de um atributo. Verificar a integridade de uma tabela significa verificar se todos os MAC e CMAC de todas as linhas da tabela estão corretos. Verificar a integridade de uma única linha significa verificar se o MAC daquela linha em específico está correto (considerando o seu cálculo sob todos os seus atributos). E, por último, verificar a integridade de uma única coluna significa verificar se o MAC de cada linha, calculado considerando apenas a coluna em questão, está correto. É importante ressaltar que, neste trabalho, nem todos os níveis de granularidade podem ser providos ao mesmo tempo. Por exemplo, não é possível prover a integridade de uma linha inteira e de apenas um único atributo ao mesmo tempo. Isto se deve ao fato de que o MAC sempre deve ser calculado sob o mesmo conjunto de atributos. Por outro lado, é possível verificar a integridade de uma única linha ou de uma tabela inteira, pois estas verificações estão separadas entre as colunas MAC e CMAC.

Para verificar a integridade de uma linha, é necessário calcular o MAC da linha e comparar com o MAC que está armazenado no BD. Se os dois valores forem iguais, pode-se assegurar que a linha não foi alterada indevida-

mente (observando o fato de que ambos os MACs devem ser calculados sob o mesmo conjunto de atributos). Se esta mesma verificação para todas as linhas da tabela for aplicada, podemos assegurar a integridade da tabela contra inserções e modificações. Entretanto, como apresentado anteriormente, somente o MAC não garante a integridade da tabela contra remoções. Para isso, utilizamos o CMAC, verificando cada par de linhas consecutivas. De forma mais geral, pode-se assegurar que uma tabela T não teve nenhuma linha removida de forma maliciosa se satisfazer a Equação 4.3, isto é, se para cada par de tuplas t_{n-1}, t_n, o CMAC da tupla t_n for igual ao CMAC calculado com os valores do MAC obtido das tuplas t_{n-1} e t_n.

$$\forall t_{n-1}, t_n \in T : t_n.CMAC = CMAC(k, t_{n-1} \oplus t_n) \tag{4.3}$$

4.6 VERIFICANDO A COMPLETUDE (*COMPLETENESS*) DE CONSULTAS

O CMAC também pode ser utilizado para verificar a completude de buscas por intervalo dentro da ordenação utilizada para o CMAC. Por exemplo, se o CMAC está ordenado por uma chave primária com auto incremento, então o intervalo da busca deve ser definido pela chave primária.

A verificação da completude é possível devido ao encadeamento de linhas adjacentes. Verificando apenas o CMAC das linhas dentro do intervalo de busca garante que nenhuma tupla intermediária foi omitida. Entretanto, apenas com esta verificação, não é possível garantir que as tuplas das bordas não foram omitidas. Para obter a garantia de que as tuplas das bordas também não foram omitidas, é necessário buscar algumas tuplas fora do intervalo da busca. Por exemplo, para uma busca das tuplas i a j (onde $j > i$), é necessário retornar também as tuplas $i - 1$ e $j - 1$, para garantir que as tuplas i e j não foram omitidas.

Se o cliente não confiar nas respostas do SGBD à suas consultas, e precisar de garantias que nenhum registro foi omitido, ele deve proceder de acordo com o Algoritmo 4.

4.7 RECUPERANDO O VALOR ORIGINAL DE UMA TUPLA

Ao contrário dos trabalhos que tratam da recuperação de tuplas, apresentados no Capítulo 3, neste trabalho os ataques são identificados através de tuplas específicas ao invés de transações. Além disso, a verificação da integri-

Algoritmo 4: Algoritmo para verificar a completude de consultas

Entrada: Uma consulta Q
Passo 1: Enviar a consulta Q ao servidor, que retornará um conjunto T de valores $t_i, \ldots, t_j$, onde $i <= j$
Passo 2: Obter as linhas t_{i-1} e t_{j+1}
Passo 3: Verificar a integridade dos valores $t_{i-1}, t_i, \ldots, t_j, t_{j+1}$, como apresentado na seção 4.5

dade é feita sempre antes da utilização de uma determinada tupla. Portanto, é possível manter uma versão simplificada dos logs.

O arquivo de logs proposto possui três tipos de entradas, uma para cada operação SQL que altera o BD fisicamente. São elas:

- (***insert_row*, T, X, Id**): Indica que X (conjunto de atributos $x_0, ..., x_n$, respeitando o esquema de T) foi adicionado na tabela T, na linha identificada por Id;
- (***update_row*, T, X, Id**): Indica que a linha identificada por Id, da tabela T, foi atualizada com os valores de X (conjunto de atributos $x_0, ..., x_n$, respeitando o esquema de T);
- (***delete_row*, T, Id**): Indica que a linha identificada por Id foi removida da tabela T.

Quando um dado é inserido no BD, é adicionada uma entrada no log do tipo *insert_row* para este dado. Sempre que este dado é alterado, a entrada antiga é apagada, sendo adicionada uma nova entrada referente a operação realizada (*update_row* ou *delete_row*). Uma vez que um determinado dado é removido, a entrada do log para este dado não é mais modificada, ou seja, ele chega no estado final, conforme apresentado na Figura 17.

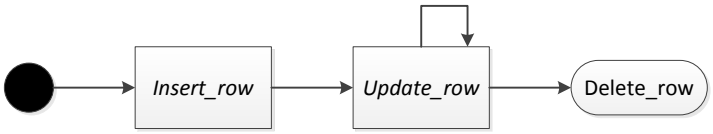


Figura 17: Máquina de estados de entradas no log para um determinado dado

Como é feita uma verificação da integridade dos dados a cada acesso a ele no BD, é necessário apenas manter no log a entrada referente a última operação que manipulou determinada linha, mantendo o log com um baixo número de entradas e, consequentemente, ocupando pouco espaço em disco. Além disso, não é necessário adicionar no log as operações de leitura no BD, uma vez que, se a verificação de integridade falhar, o dado não será utilizado (será recuperado), não havendo a propagação de dados não íntegros. Com isso, é possível estimar a quantidade de entradas que o log terá e, consequentemente, o tamanho que irá ocupar.

A quantidade de entradas que o log terá é a soma da quantidade de linhas de cada tabela do BD (contabilizando também as linhas removidas). Se a chave primária das tabelas for um número serial com *auto-increment*, a quantidade de entradas do log pode ser expressa pelo somatório $\sum_{t \in T} \max(t.serial)$, onde T é o conjunto de todas as tabelas do BD. O tamanho do arquivo de logs em *bytes* pode ser expresso pelo somatório $\sum_{i=1}^{n} 1 + 30 + 8 + size(X)$, onde:

- 1 byte é utilizado para identificar o tipo de entrada: *insert_row*, *update_row* ou *delete_row*;
- 30 bytes é utilizado para identificar o nome da tabela[1];
- 8 bytes é utilizado para identificar a linha da tabela;
- $size(X)$ é o tamanho em bytes do conjunto X dos dados inseridos ou atualizados no BD (se for uma operação de *delete*, será 0).

Se, durante a verificação do MAC e CMAC, for detectada alguma inconsistência com os dados de uma determinada linha, para recuperar os dados com os valores da última atualização realizada, deve buscar no log a entrada correspondente à linha e à tabela em que a verificação falhou e executar um *UPDATE* com os valores recuperados do log. O valor retornado ao cliente também deve ser o valor recuperado do log.

Vale ressaltar que, além da recuperação dos dados, é importante que o responsável pelos dados seja notificado que houve uma modificação não autorizada e que haja um processo de auditoria para detectar a origem do ataque e proceder as ações legais cabíveis para a punição dos responsáveis, além de correções de possíveis vulnerabilidades de segurança que possibilitaram o ataque à base de dados. Tal processo, entretanto, está fora do escopo deste trabalho.

[1] 30 bytes é o valor máximo para o nome de uma tabela no SGBD *Oracle*

4.8 GERENCIAMENTO DE CHAVES

A segurança de todo método criptográfico depende da confidencialidade das chaves criptográficas, sendo que o correto gerenciamento das chaves tem uma importância fundamental. Os trabalhos relacionados à integridade de dados não abordam o gerenciamento das chaves criptográficas. Além disso, eles consideram o uso de uma chave única. Entretanto, o uso de uma única chave não é o ideal, uma vez que se a chave é comprometida, a base de dados inteira também é comprometida. Portanto, o caso ideal é o uso de múltiplas chaves (por exemplo, uma chave por tabela).

Apesar do uso de múltiplas chaves reduzir o impacto do comprometimento de uma chave específica, ainda é necessário que as chaves sejam geradas e armazenadas de forma segura, de preferência em dispositivos criptográficos, como o HSM (do inglês, *hardware security module*) (MARTINA; SOUZA; CUSTODIO, 2007). Além de aumentar a segurança das chaves, o uso de dispositivos criptográficos facilita o compartilhamento das chaves entre diferentes aplicações que necessitem atualizar ou recuperar informações do BD ou em ambientes multiusuários.

4.8.1 Armazenamento das chaves

Para gerenciar diversas chaves, Hacigümüş e Mehrotra (HACIGUMUS; MEHROTRA, 2005) apresentam uma tabela simples, chamada de *Key registry*, apresentada na tabela 3. Apesar do foco deste trabalho ser em bases criptografadas, a mesma idéia pode também ser utilizada para armazenar as chaves criptográficas para o controle de integridade. Esta tabela consiste de cinco colunas:

- **ID da chave (KID)**: Esta coluna representa a chave primária e identifica unicamente a chave.
- **Correspondência**: Indica em quais tuplas a chave foi utilizada. Pela notação utilizada na Tabela 3, indica-se a tabela e o intervalo de tuplas em que a chave foi utilizada. Por exemplo, na primeira entrada da tabela, a chave foi utilizada na tabela Empregado, para todas as tuplas no intervalo especificado.
- **Expiração**: Indica a data de expiração da chave. O valor desta coluna pode ser uma data, como mostrado na Tabela 3, mas também pode ser um intervalo de tempo menor, como horas, minutos, etc. Especificar uma data de expiração para a chave é importante para limitar o tempo

de vida da chave, reduzindo a probabilidade de um comprometimento da chave.

- **Cardinalidade**: Especifica quantas vezes a chave já foi utilizada. Esta coluna pode ser utilizada para limitar o número de vezes que uma chave pode ser utilizada. Dessa forma, além da data de expiração, uma chave pode ser atualizada por exceder um número determinado de usos.

- **Chave**: Contém a chave propriamente dita.

Tabela 3: *Key Registry*

KID	Correspondência	Expiração	Card.	Chave
45	Empregado:[1,250]	06/12/2013	250	*a chave*
92	Empregado:[250,500]	07/12/2013	120	*a chave*
52	Pessoa:[1,500]	06/08/2013	500	*a chave*

Como a tabela *Key registry* é uma tabela conceitual, a busca pelas chaves através da coluna de correspondência pode ser ineficiente. Para tornar a busca mais eficiente, este trabalho propõe dividir esta coluna em três novas colunas, como mostrado na Tabela 4.

Ao invés da coluna de correspondência, esta nova tabela contém as seguintes colunas:

- **Tabela**: O nome da tabela em que a chave está sendo utilizada.

- **Início do Intervalo (II)**:A primeira linha da tabela em que a chave deve ser utilizada.

- **Fim do Intervalo (FI)**: A última linha da tabela em que a chave deve ser utilizada.

Tabela 4: Novo *key Registry*

KID	Tabela	II	FI	Expiração	Card.	Chave
45	Empregado	1	250	06/07/2014	350	*a chave*
92	Empregado	251	500	07/07/2014	250	*a chave*
52	Pessoa	1	500	06/08/2014	500	*a chave*

As colunas de expiração e cardinalidade podem ser utilizadas para definir políticas para a atualização das chaves. Como cada chave será utilizada em apenas uma pequena parte do BD, elas podem ser atualizadas com maior

frequência, reduzindo as chances de um comprometimento de chave e sem uma sobrecarga muito grande para a aplicação. Entretanto, não há um valor ideal genérico para estas colunas. Por exemplo, para clientes que necessitam de um alto desempenho, a expiração da chave deve ser suficientemente longa de forma a não sobrecarregar o BD com a atualização constante dos MACs e consequentemente degradar o desempenho do sistema. Por outro lado, a expiração não pode ser muito longa de forma a não criar brechas para ataques.

Outro fator que pode afetar os valores da expiração e cardinalidade das chaves é a semântica dos dados armazenados em cada tabela. Por exemplo, uma tabela que armazena os salários de funcionários de uma empresa é mais crítica que uma tabela listando todos os departamentos da empresa, devido ao fato que uma alteração no salário de um funcionário causará a empresa maiores danos do que a alteração do nome de um departamento. Portanto, estas tabelas podem ter valores diferentes para a expiração e cardinalidade das chaves. Para a tabela que armazena os salários, é desejável que a expiração e a cardinalidade sejam menores, enquanto que as chaves referentes a tabela contendo os nomes dos departamentos podem ser atualizadas com menos frequência.

Também é importante ressaltar que os algoritmos apresentados neste capítulo não consideram o uso de múltiplas chaves para o BD como um todo. Entretanto, considera-se que a sobrecarga para o uso de múltiplas chaves é mínimo, se comparado com o ganho de segurança. A tabela *key registry* é uma estrutura bastante simples e não requer um SGBD para armazená-la. Esta tabela pode ser mantida por um BD simples e eficiente, como o SQLite, e por ser pequena pode ser mantida em memória. Para possibilitar o uso de múltiplas chaves é necessário adicionar nos algoritmos uma instrução para a busca da chave correta a ser utilizada, como apresentado abaixo.

```
SELECT Chave
FROM Key_Registry
WHERE Tabela='Nome_Tabela' AND II <= Pos_Tupla
    AND FI >= Pos_Tupla;
```

4.8.2 Geração e atualização das chaves

A geração de novas chaves é um processo bastante simples. Primeiramente, o cliente deve definir os valores de expiração e cardinalidade da chave, qual será a tabela e o intervalo de tuplas em que a chave será utilizada, além do tamanho da chave. Com estes valores definidos, a chave propriamente dita

deve ser gerada. Isto pode ser feito através de um biblioteca criptográfica, como o OpenSSL (OPENSSL, 2013). Com a chave gerada, ela é inserida na tabela *key registry*.

Após a chave ter sido salva na tabela *key registry*, se já existirem tuplas para o intervalo definido, estas tuplas devem ser recuperadas do BD para que seja calculado o MAC e CMAC de cada linha. Este cenário pode acontecer se, por exemplo, o cliente já possuir a sua base de dados em produção e deseja começar a utilizar mecanismos de verificação de integridade (vale ressaltar que este cenário é diferente da atualização de uma chave, onde os valores do MAC e CMAC são recalculados). Assim sendo, deve-se obter as tuplas no intervalo correspondente à chave gerada, calcular o MAC e CMAC para todas as linhas retornadas pela consulta e, por fim, atualizar as linhas no BD.

Para a atualização da chave o processo é semelhante à geração. Uma nova chave deve ser gerada (com os mesmos ou com novos valores de cardinalidade, expiração e tamanho da chave) e atualizada na tabela *key registry*. Após a geração, todos os valores de MAC e CMAC das tuplas existentes no intervalo correspondente da chave devem ser recalculados e atualizados no BD. O Algoritmo 5 apresenta os passos descritos para a geração e atualização das chaves de forma resumida. Nos passos 2 e 3, devem ser executadas as operações *INSERT* e *SELECT*, respectivamente, com exceção da atualização da chave, onde o *INSERT* deve ser substituído por um *UPDATE*.

Algoritmo 5: Algoritmo para a geração e atualização de uma chave

Entrada: Os atributos para a geração da chave: nome_tabela, ii, fi, cardinalidade, expiração, tamanho_chave
Passo 1: Gerar a chave
Passo 2: Inserir a chave na tabela *key registry*
Passo 3: Buscar as tuplas da tabela *nome_tabela* dentro do intervalo ii e fi
Passo 3.1: Para cada tupla retornada calcular o MAC e CMAC
Passo 3.2: Atualizar os valores calculados no BD

4.9 VERIFICAÇÃO DA ATUALIDADE (*FRESHNESS*) DOS DADOS

Os SGBDs geralmente realizam cópias de segurança periódicas para possibilitar a recuperação do BD em caso de falhas catastróficas. No contexto de BDs terceirizados, isto traz um novo problema de segurança. Um atacante de posse destas cópias de segurança pode voltar o estado do BD a um estado

anterior válido. Como neste estado anterior os dados foram enviados pelo cliente, as verificações do MAC e CMAC podem indicar que os dados estão íntegros, mas não podem garantir que os dados são atuais.

Na literatura, poucos trabalhos abordam este problema (DANG, 2008; XIE et al., 2008). De forma geral, estes trabalhos apresentam as mesmas limitações apresentadas pelos trabalhos apresentados no Capítulo 3, para verificar a corretude e completude. O trabalho de Dang (DANG, 2008) utiliza criptografia assimétrica, cifrando o *timestamp* da última atualização de cada tupla. Além disso, é necessário que o cliente estabeleça um período de validade para o par de chaves utilizado para realizar a assinatura. Já o trabalho de Xie et. al. (XIE et al., 2008) baseia-se no trabalho anterior dos mesmos autores para verificar a corretude e completude (XIE et al., 2007). É um método determinístico que baseia-se na inserção e manutenção de tuplas falsas no BD ao longo do tempo.

Neste trabalho, propomos uma alternativa simples para verificar se os dados retornados pelo servidor correspondem à última atualização feita pelo cliente.

O valor do MAC de cada linha é atualizado sempre que a linha correspondente é atualizada. Portanto, é possível verificar se os dados retornados correspondem à última atualização mantendo uma base local de MACs. Sempre que uma operação que altera o BD for realizada, uma cópia do MAC calculado é mantida localmente. Quando o cliente recupera uma linha do BD, após a verificação de integridade, o cliente verifica se o MAC existe nesta cópia local. Se existir, significa que o servidor retornou a versão mais recente dos dados. Se o MAC não existir nesta base local, significa que foi retornada uma versão antiga do dado. Neste caso, pode-se utilizar os logs, descritos na seção 4.7, para retornar os dados à sua última versão.

A base de MACs deve ser implementada em uma estrutura eficiente para as operações de consulta, inserção, remoção e atualização. Um exemplo de tal estrutura são as tabelas de *hash*, onde as operações têm complexidade $O(1)$ (CORMEN et al., 2009). Além disso, esta estrutura é também eficiente em termos de armazenamento, com complexidade $O(n)$. Por exemplo, se for utilizado o HMAC, com a função de *hash* SHA-1, como algoritmo MAC, para uma base de dados com 1 milhão de registros, a base de MACs ocuparia menos de 20 *Megabytes*.

Uma alternativa para a base de MACs, é utilizar os campos de expiração e cardinalidade das chaves (conceitos apresentados na Seção 4.8) da seguinte forma:

- **Expiração da chave**: As chaves podem ter um período de vida. Após a expiração de uma chave, uma nova chave é gerada e todos os MACs calculados com a chave antiga são recalculados com a nova chave e

atualizados no BD. Dessa forma, se uma determinada linha for retornada a uma versão antiga, em que o MAC foi calculado com uma chave expirada, a verificação do MAC acusará um problema de atualização;

- **Cardinalidade da chave**: Além do período de vida, as chaves podem ser atualizadas após um determinado número de usos. Da mesma forma que na expiração, quando a chave atinge o número de uso determinado, uma nova chave é gerada e todos os MACs são atualizados. Portanto, se uma determinada linha for retornada a uma versão antiga, em que o MAC foi calculado com uma chave que já atingiu o limite de usos, a verificação do MAC também acusará um problema de atualização;

Entretanto, esta técnica é suscetível a um ataque. Para chaves com um período de expiração muito longo (ou uma quantidade de usos elevadas), um atacante pode retornar os dados para uma versão antiga, porém ainda dentro do período de vida da chave atual. Desta forma, a verificação do MAC será feita com a mesma chave que o gerou e, portanto, os valores serão os mesmos. Além disso, se forem estabelecidos valores muito curtos para a expiração e cardinalidade das chaves, a atualização frequente das chaves diminuirá a eficiência do sistema.

4.10 FLEXIBILIDADE DE IMPLEMENTAÇÃO

Um dos diferenciais deste trabalho é a flexibilidade de implantação das técnicas de verificação e recuperação de integridade. Nesta seção são analisadas as vantagens e desvantagens de implantação das técnicas propostas em três diferentes níveis: como um *plug-in* para o SGBD (Seção 4.10.1), diretamente no cliente (Seção 4.10.2) e em um *front-end* (Seção 4.10.3).

4.10.1 Implementação no SGBD

A implantação das técnicas propostas diretamente no SGBD é recomendada quando os dados são armazenados *in-house*. Isto é, quando o próprio cliente gerencia e mantém o SGBD. Neste cenário, o cliente tem total controle do SGBD e pode facilmente criar um *plug-in* para controlar a integridade de seus dados.

A Figura 18 esquematiza a arquitetura com o controle de integridade sendo feito dentro do servidor de BD. Neste cenário, o controlador de integridade é o *plug-in*. Ele é responsável por receber as operações SQL do cliente, fazer os cálculos do MAC e CMAC, verificar a integridade e, quando

necessário, recuperar o valor original de um dado (em caso de uma alteração indevida).

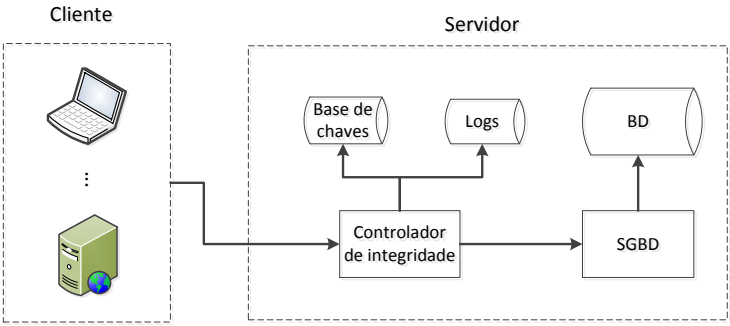


Figura 18: Implementação como um plugin para o SGBD

Apesar da base de chaves e dos logs serem módulos separados na Figura 18, pode-se assumir que estes estejam incorporados ao SGBD. A base de chaves pode ser uma nova tabela do BD e os logs podem ser os próprios logs do SGBD (desde que seja possível configurar o log de acordo com o que foi apresentado na seção 4.7) ou uma nova tabela no BD.

A grande vantagem em implantar o controle de integridade como um *plug-in* para o SGBD é o ganho em eficiência. Como apresentado no Capítulo 5, o principal fator que degrada a eficiência das técnicas propostas é a necessidade constante de buscar dados adicionais para verificar a integridade das tuplas. Com o controlador de integridade sendo um *plug-in* estas consultas são feitas apenas internamente no SGBD, além de não haver a sobrecarga nos dados trafegados entre o cliente e o servidor (neste modelo, os dados de controle de integridade não precisam sair do servidor, sendo isso transparente para o cliente).

Entretanto, como desvantagem, toda a segurança do sistema é comprometida se o servidor for comprometido. Como as chaves criptográficas para calcular o MAC e CMAC estão no próprio servidor, se o servidor for invadido o atacante pode gerar valores válidos para as colunas MAC e CMAC e toda a base de dados, não sendo mais possível detectar modificações feitas pelo atacante.

4.10.2 Implementação no cliente

No caso em que o cliente terceiriza os seus dados, fica inviável fazer a implementação diretamente no servidor, tanto por questões de segurança quanto pelo fato de o cliente não ter permissões para fazer alterações no SGBD. Neste caso, a implementação pode ser feita no lado do cliente, diretamente na aplicação que utiliza os dados, conforme apresentado na Figura 19.

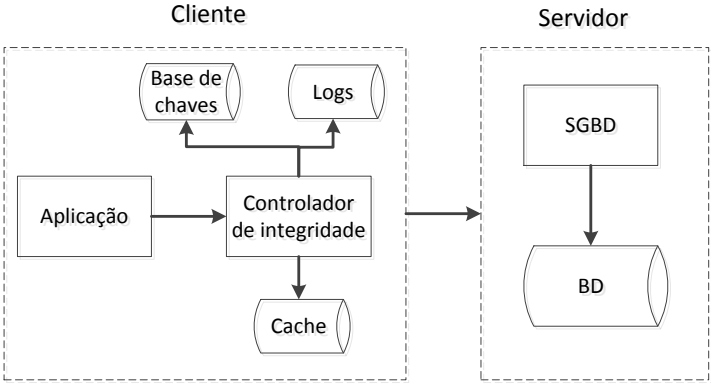


Figura 19: Implementação no cliente

Neste cenário, o controlador de integridade é implementado como um módulo adicional da aplicação. A base de chaves, os logs e a *cache* devem ficar em um BD controlado pela aplicação. Entretanto, não é necessário um SGBD completo para gerenciar estas bases, pois a estrutura é simples e a quantidade de dados é pequena. Estas bases podem ser armazenadas em um arquivo e permanecerem carregadas em memória enquanto a aplicação estiver executando, para aumentar a eficiência. Além disso, estes dados podem ser manipulados por um BD simples, como o *SQLite* (SQLITE, 2014).

A maior desvantagem desta arquitetura é que se o cliente possuir diversas aplicações, todas elas terão que ter este módulo adicional e terão que compartilhar as chaves criptográficas e os logs para poderem calcular o MAC e CMAC, além de verificar e corrigir problemas de integridade. Outro agravante é que, se as aplicações estiverem implementadas em diferentes lingua-

gens de programação, as técnicas propostas terão que ser implementadas em cada uma das linguagens das aplicações.

Portanto, é recomendado o uso desta arquitetura apenas quando o cliente tem uma aplicação única. Se o cliente for uma empresa, por exemplo, isto não implica necessariamente que apenas um funcionário da empresa pode acessar os dados. Entretanto, para que seja possível que diversos usuários tenham acesso aos dados, a aplicação deve ser centralizada (uma aplicação *web*, por exemplo).

A grande vantagem desta arquitetura é o baixo custo de implantação, mantendo uma eficiência condizente com as necessidades de muitos sistemas. O cliente não precisa implantar ou manter nenhum servidor adicional, bastando apenas adicionar um novo módulo na sua aplicação para realizar o controle da integridade. A incorporação do controle de integridade é particularmente simples em aplicações modulares, em específico, aplicações que utilizam alguma camada de abstração do bando de dados (do inglês, *database abstraction layer* (DBAL)).

4.10.3 Implementação no *front-end*

A arquitetura proposta neste trabalho é a baseada em um *front-end*. O *front-end* age como um intermediário entre o cliente e o servidor, como apresentado na Figura 20. O cliente se comunica com o *front-end* como se

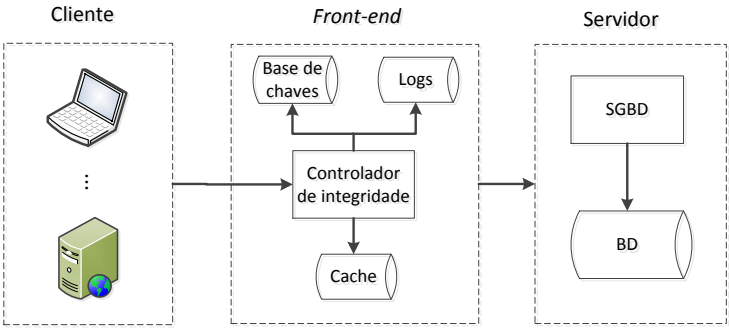


Figura 20: Implementação como um *front-end*

estivesse se comunicando com o servidor. O *front-end* recebe as requisições

do cliente, realiza as operações necessárias para garantir o controle da integridade e encaminha para o servidor. Por exemplo, numa operação *INSERT*, o *front-end* calcula os valores do MAC e CMAC, acrescentando estes valores ao *INSERT* original e o enviando ao servidor. Numa operação de *SELECT*, o *front-end* solicita os dados necessários para verificar a integridade ao servidor, verifica a integridade dos dados (se for detectado uma alteração maliciosa recupera o valor original do(s) dado(s)) e retorna os dados para o cliente.

Nesta arquitetura, todo o controle de integridade fica transparente para o cliente, sendo possível o acesso de diversas aplicações ao mesmo conjunto de dados sem a necessidade do compartilhamento de chaves criptográficas. De forma geral, o fluxo de comunicação entre as partes procede da seguinte forma:

1. O cliente envia suas consultas normalmente ao *front-end*, como se estivesse enviando ao SGBD;

2. Se for uma operação de modificação de dados (*INSERT*, *UPDATE* ou *DELETE*), o *front-end* solicita os valores MAC e CMAC necessários ao SGBD e calcula o MAC e CMAC dos dados enviados pelo cliente. Em seguida, o *front-end* modifica a operação SQL para incluir os valores MAC e CMAC calculados e envia a operação para o servidor;

3. Se for uma operação de consulta (*SELECT*), o *front-end* adiciona à consulta os valores do MAC e CMAC necessários para verificar a integridade e envia a consulta ao servidor. Em seguida, verifica a integridade dos dados e, se a verificação for positiva, retorna os dados solicitados ao cliente. Se for detectada uma modificação maliciosa, o *front-end* utiliza os logs para recuperar o valor original dos dados e retorna os dados corrigidos para o cliente.

Esta arquitetura é a recomendada quando o cliente deseja terceirizar os seus dados, mantendo um controle da integridade dos mesmos, e possui diversas aplicações que precisam acessar os dados. A grande vantagem é que não é necessário realizar alterações tanto nas aplicações quanto no servidor. O *front-end* também não precisa ser um servidor de alto desempenho. Para o armazenamento das chaves, logs e cálculo do MAC e CMAC, recomenda-se o uso de um dispositivo criptográfico, como um HSM.

Os HSMs são dispositivos criptográficos projetados para proteger os dados neles armazenados contra qualquer tipo de interferência lógica ou física, além de prevenir a extração dos dados (geralmente chaves criptográficas) (MARTINA; SOUZA; CUSTODIO, 2007). Os HSMs normalmente são submetidos a vários testes para a homologação de seu *hardware*, como o FIPS 140-2 (National Institute of Standards and Technology, 2002). Além disso,

normalmente os HSMs possuem aceleradores criptográficos, aumentando a eficiência para operações criptográficas (podem ser capazes de realizar mais de oito mil operações simétricas por segundo, por exemplo).

4.11 RESUMO DO CAPÍTULO

Este capítulo apresentou as técnicas para verificação e recuperação da integridade de dados armazenados em BDs relacionais. Inicialmente, foram apresentados os algoritmos para o controle da integridade: MAC e CMAC. O MAC é responsável por controlar a integridade de tuplas individuais, enquanto que o CMAC é responsável por controlar se alguma tupla, dentre um conjunto de tuplas, foi removida. Ainda, foi identificado que o CMAC não detecta a remoção de tuplas nas extremidades das tabelas, sendo apresentada uma variação do algoritmo para tratar esta vulnerabilidade.

A seção 4.2 apresentou a arquitetura geral do sistema proposta neste trabalho. Nesta arquitetura, temos o cliente, que é o dono e responsável pelos dados, o servidor, responsável por armazenar e garantir a disponibilidade dos dados e o *front-end*, que é uma camada entre o cliente e o servidor e que é responsável por fazer todo o controle da integridade dos dados.

Em seguida, foram apresentados os algoritmos para o cálculo do MAC e CMAC para as operações SQL: *INSERT*, *UPDATE* e *DELETE*. Também foi apresentado o algoritmo para verificar a integridade dos dados retornados pela operação *SELECT* e também para verificar se todas as tuplas que satisfazem a consulta foram retornados.

A seção 4.7 apresentou a técnica proposta neste trabalho para recuperar o valor original dos dados, quando uma alteração maliciosa é detectada através da verificação do MAC e CMAC. Esta técnica também permite recuperar uma tupla que foi removida de forma maliciosa.

Este capítulo apresentou os problemas do gerenciamento das chaves utilizadas para o cálculo do MAC e CMAC. Inicialmente, discorreu-se sobre o problema de utilizar uma única chave para um BD inteiro e o impacto de um comprometimento da chave. Se apenas uma chave é utilizada para proteger o BD, o comprometimento desta chave significa o comprometimento de todos os dados, ou seja, não será mais possível verificar a integridade dos dados.

A Seção 4.8 apresentou a tabela *key registry*, uma tabela simples para possibilitar o uso de múltiplas chaves para diferentes tabelas ou conjunto de tuplas do BD. Além disso, nesta tabela é também possível definir valores para a quantidade de usos e expiração de cada chave. Com uma atualização periódica das chaves, diminui-se ainda mais as chances e o impacto de um comprometimento das chaves. Ainda, a Seção 4.8 apresentou como deve ser

feita a geração e atualização das chaves na tabela *key registry*. Também apresentou as alterações necessárias nos algoritmos apresentados neste capítulo, para inserção, atualização, remoção e busca de tuplas do BD. Para possibilitar o uso de múltiplas chaves, a chave correta deve ser buscada durante a execução destes algoritmos. Além disso, na geração e atualização de chaves, pode ser necessário calcular ou recalcular os valores MAC e CMAC das tuplas correspondentes, caso existam.

A seção 4.9 apresentou a técnica proposta para verificar se os dados retornados pelo servidor são os dados referentes à última atualização efetuada através da comparação dos MACs. Por fim, a seção 4.10 discorreu sobre a flexibilidade de implementação da proposta deste trabalho. Mostrou-se que a implementação pode ser feita tanto no cliente quando no servidor, e também em um *front-end*. Foram analisadas as vantagens e desvantagens da implementação em cada entidade e indicada qual arquitetura foi adotada neste trabalho.

5 AVALIAÇÃO DOS MÉTODOS PROPOSTOS

5.1 INTRODUÇÃO

Diferentes cenários foram considerados para avaliar a eficiência das técnicas propostas. Para cada cenário, foi executada uma bateria de mil testes sobre uma tabela com dez mil registros contendo valores randômicos. Todos os resultados apresentados representam a média de cada bateria de testes. Para cada cenário, foi avaliado o tempo gasto nas operações de *INSERT*, *UPDATE*, *DELETE* e *SELECT*. Além disso, foram considerados quatro casos distintos:

1. A execução sem mecanismo de integridade algum, utilizado como o valor base para as comparações;
2. A execução utilizando apenas a coluna com o *MAC*;
3. A execução utilizando tanto a coluna do *MAC* quanto a do *CMAC*;
4. A execução utilizando a coluna do *MAC* e o *CMAC* no modo circular.

Além das operações *SQL*, também foram avaliadas as propostas para a verificação da atualidade de um registro e da recuperação de um registro não íntegro, além de avaliar o impacto de se utilizar múltiplas chaves para o BD.

O restante deste capítulo está organizado da seguinte forma. A seção 5.2 apresenta detalhes de como foi feita a implementação do protótipo para os testes e do esquema do BD utilizado. A seção 5.3 apresenta os resultados dos testes para a operação *INSERT*, enquanto que a seção 5.4 apresenta os resultados dos testes para a operação *UPDATE*. Os resultados dos testes para a operação *DELETE* são apresentados na seção 5.5 e os resultados para a operação *SELECT* na seção 5.6. Já a seção 5.7 apresenta os resultados dos testes para a verificação da atualidade de registros. A seção 5.8 apresenta os resultados dos testes para a recuperação de um registro não íntegro e a seção 5.9 apresenta os resultados dos testes utilizando múltiplas chaves. Por fim, a seção 5.10 apresenta um resumo do capítulo.

5.2 IMPLEMENTAÇÃO

Para verificar a eficiência das técnicas propostas foi implementado um protótipo. Para a execução dos testes, foi escolhido o HMAC (BELLARE;

CANETTI; KRAWCZYK, 1996; KRAWCZYK; BELLARE; CANETTI, 1997), como a função MAC. O protótipo para a avaliação do uso do MAC e CMAC foi implementado na linguagem *C*. O HMAC foi calculado pela biblioteca criptográfica *OpenSSL* (OPENSSL, 2013), e o SGBD utilizado foi o *MySQL* (MYSQL, 2014). Os testes para a avaliação do uso do MAC e CMAC foram realizados em uma máquina com processador *Intel Core 2 Quad CPU Q8400*, com 4Mb de *cache* e *clock* de 2.66GHz, 4GB de memória RAM à 800Mz e disco rígido *SATAII* de 320Gb, com 16Mb de *cache* à 7200RPM. O sistema operacional utilizado foi o Ubuntu 11.04 32-bit, com o *OpenSSL* versão 0.9.8d e o *MySQL* versão 5.1. Além disso, para calcular o HMAC foi utilizada a função de resumo criptográfico *SHA-1* (National Institute of Standards and Technology, 2012) e uma chave criptográfica de 256 *bits*.

Já para a avaliação da verificação de atualidade, recuperação de registros não íntegros e uso de múltiplas chaves, foi utilizada a arquitetura com *front-end*, apresentada no Capítulo 4. O protótipo foi implementado na linguagem *PHP*. O *front-end* é uma máquina com processador *Intel Core 2 Quad CPU Q8400*, com 4Mb de *cache* e *clock* de 2.66GHz, 4GB de memória RAM à 800Mz e disco rígido *SATAII* de 320Gb, com 16Mb de *cache* à 7200RPM. O sistema operacional utilizado foi o Ubuntu 12.10 32-bit, com o *OpenSSL* versão 1.0.1c e PHP 5.4.6. Já o servidor MySQL está em uma máquina com processador *Intel Core i5-3360M 2.8GHz*, 4GB de memória RAM à 1600Mz e disco rígido de 320Gb à 7200RPM. O sistema operacional utilizado foi o Ubuntu 12.10 32-bit, com o *MySQL* versão 5.5.

Para a realização dos testes, foi criada uma tabela com cinco colunas, além das colunas para armazenar o MAC e CMAC. As colunas são as seguintes:

- **Id**: Campo inteiro e chave primária;
- **Nome**: Campo de texto, do tipo *varchar*, preenchida com valores de tamanho randômico de zero a cem bytes;
- **Email**: Campo de texto, do tipo *varchar*, preenchida com valores de tamanho randômico de zero a cem bytes;
- **Senha**: Campo de texto, do tipo *varchar*, preenchida com valores de tamanho randômico de zero a cem bytes;
- **Descrição**: Campo de texto, do tipo *longtext*, preenchida com valores de tamanho randômico de zero a mil bytes.

Além disso, antes de cada teste a tabela é removida e repopulada com dez mil entradas. Também foram desabilitadas as opções de gerenciamento de

cache do MySQL para evitar discrepâncias nos testes. Não foram utilizadas várias tabelas, com tipos mais complexos, como chaves estrangeiras, pois o objetivo dos experimentos é avaliar cada operação SQL básica de forma independente. Além disso, para o método proposto não há diferença se a tabela possui ou não chave estrangeira, pois para o cálculo do MAC o que é levado em consideração é apenas o valor do campo e não a sua semântica.

Os cenários de otimização apresentados nas seções seguintes foram simulados de diferentes formas. Para o caso do *INSERT*, o MAC da primeira linha necessita sempre estar disponível do lado do cliente. Para simular esta condição, o MAC foi pré-carregado antes do início do teste. No teste sem otimização, foi feito um *SELECT* para buscar o MAC da primeira linha, sendo que o tempo deste *SELECT* foi computado no total de tempo do teste. Já nos cenários de otimização das demais operações *SQL*, o MAC e CMAC necessários variam de acordo com qual linha é manipulada (é necessário o MAC e CMAC da linha anterior e posterior à linha que está sendo manipulada). Neste caso, é impraticável manter os valores sempre no lado do cliente. Para as operações de *UPDATE* e *SELECT* sempre é feito um *SELECT* para buscar a linha n que será manipulada. Nos cenários sem otimização são feitos dois *SELECT*s. O primeiro para buscar a linha n que será manipulada e, em sequencia, um outro *SELECT* para buscar os MACs e CMACs necessários das linhas $n-1$ e $n+1$. No caso otimizado, os dois *SELECT*s foram juntados em um só que busca todos os valores da linha n, e o MAC e CMAC das linhas $n-1$ e $n+1$. Por fim, para o *DELETE*, assim como no *INSERT*, no teste sem otimização foi feito um *SELECT* adicional para obter os MACs das linhas $n-1$ e $n+1$. No caso otimizado, estes valores foram pré-carregados antes do início do teste.

5.3 AVALIAÇÃO DA OPERAÇÃO *INSERT*

No primeiro cenário, o foco é a avaliação e comparação de tempos de execução da operação *INSERT* para cada um dos casos especificados anteriormente.

Os resultados, apresentados na Figura 21, mostram que o tempo de execução utilizando apenas a coluna com o HMAC aumentou 11%, sendo que 90% do tempo foi gasto pelo servidor e apenas 10% pelo cliente. O uso do CMAC aumenta o tempo de execução em 180%, com 91% do tempo gasto no servidor e apenas 9% no cliente. Por fim, o CMAC no modo circular aumenta o tempo de execução em 684%, sendo 72% do tempo gasto no servidor e 28% no cliente.

Como pode ser observado, o maior impacto ocorre com o uso do

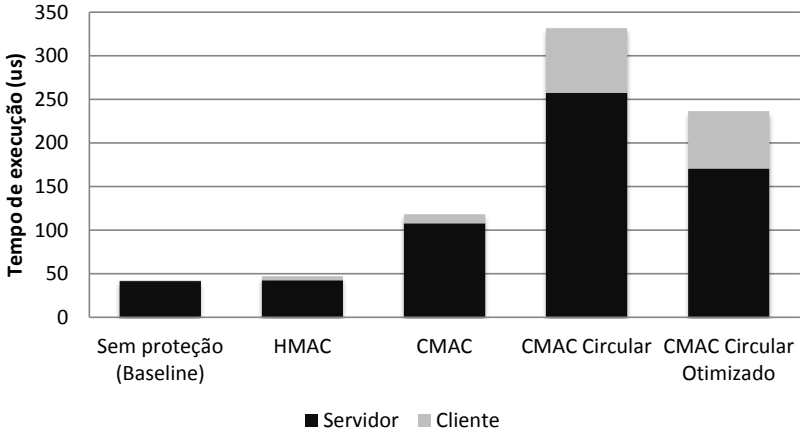


Figura 21: Comparação dos tempos de execução para a operação *INSERT*

CMAC no modo circular. Um aumento de 684% no tempo de execução pode não ser compatível com as necessidades de ambientes que precisam de maior desempenho. No entanto, o desempenho do CMAC no modo circular pode ser melhorado se acrescentarmos um requisito à técnica. Avaliando os resultados dos testes e o algoritmo do modo circular fica claro que este aumento no tempo de execução se dá pelo fato de existir uma operação *SQL* adicional para buscar a primeira linha da tabela. Na arquitetura com *front-end*, por exemplo, há a tabela dos MACs e, portanto, não é necessário a busca da primeira linha. Portanto, se considerarmos que o MAC desta linha está sempre disponível no lado do cliente, a sobrecarga cai de 684% para 460%. Apesar do aumento relativo ser grande, é importante ressaltar que os tempos de execução estão em μs. Portanto, em termos absolutos, o custo computacional é baixo, principalmente para o cliente. O último cenário, por exemplo, demorou 236μs, sendo que 170μs foram gastos no servidor e apenas 66μs no cliente.

5.4 AVALIAÇÃO DA OPERAÇÃO *UPDATE*

No segundo cenário, o foco é a avaliação e comparação de tempos de execução da operação *UPDATE* para cada um dos casos especificados anteriormente.

Os resultados, apresentados na Figura 22, mostram que o tempo de execução utilizando apenas a coluna com o HMAC aumentou em apenas 5%,

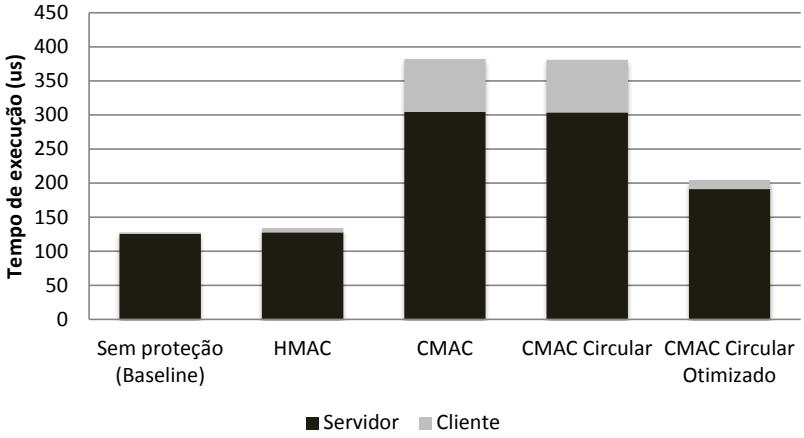


Figura 22: Comparação dos tempos de execução para a operação *UPDATE*

sendo que 95% do tempo foi gasto do servidor e apenas 5% no cliente. Já o uso do CMAC, tanto no modo normal quanto no modo circular, aumentou o tempo de execução em 199%, dos quais 80% foram gastos no servidor e 20% no cliente. O motivo para o tempo de execução ser o mesmo para o CMAC no modo normal e no modo circular é que ambos os modos executam as mesmas operações.

Assim como para a operação *INSERT*, a operação de *UPDATE* pode ser otimizada adicionando um nova condição para o método. Neste caso a condição é que a consulta para obter a linha a ser atualizada retorne, além da própria linha, as linhas adjacentes a ela. Por exemplo, para uma consulta a linha n, seriam retornadas também as linhas $n - 1$ e $n + 1$ (ou apenas a coluna MAC delas). A sobrecarga desta condição em relação ao volume de dados trafegados é mínima, se comparado com o ganho em desempenho. Por exemplo, nos testes realizados, como foi utilizada a função de resumo criptográfico SHA-1, a sobrecarga é de 40 bytes (20 bytes para o HMAC de cada linha). Por outro lado, a sobrecarga no tempo de execução cai de 199% para apenas 60%.

5.5 AVALIAÇÃO DA OPERAÇÃO *DELETE*

No terceiro cenário, o foco é a avaliação e comparação de tempos de execução da operação *DELETE* para cada um dos casos especificados anteri-

ormente.

Utilizando apenas a coluna do HMAC, não há custo adicional para a operação de *DELETE*, pois não há operações extras executadas. Em contrapartida, o uso do CMAC, tanto no modo normal quanto no modo circular, aumenta o tempo de execução em 265%, sendo que 96% do tempo é gasto no servidor e apenas 4% no cliente. Como mostrado anteriormente para a operação de *UPDATE*, o tempo de execução para o CMAC em seus dois modos é o mesmo, pois executam-se as mesmas operações. Estes resultados

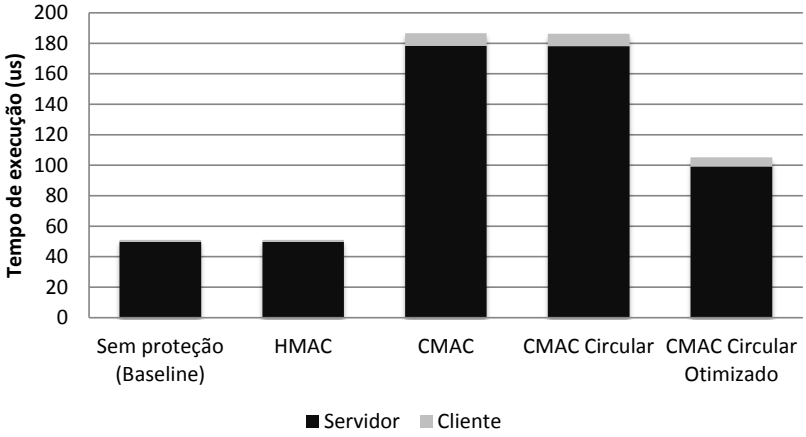


Figura 23: Comparação dos tempos de execução para a operação *DELETE*

podem ser observados na Figura 23.

Assim como nas operações anteriores, o CMAC pode ser otimizado para a operação de *DELETE*. Isto é feito aproveitando a mesma ideia apresentada para a operação de *UPDATE*, ou seja, a mesma consulta que retorna a linha n a ser deletada retorna também o HMAC das linhas $n-1$ e $n+1$. Com isso, a sobrecarga no tempo de execução cai de 265% para 106%.

5.6 AVALIAÇÃO DA OPERAÇÃO *SELECT*

Por fim, no último cenário, o foco é a avaliação e comparação de tempos de execução da operação *SELECT* para cada um dos casos especificados anteriormente.

Para verificar a integridade utilizando apenas a coluna do HMAC, é necessário recalcular os HMACs e comparar com o valor obtido do servi-

dor de BD. Esta operação tem uma sobrecarga no tempo de execução de 22%. Por outro lado, para verificar a integridade utilizando o CMAC, é necessário, além do HMAC, recalcular o CMAC. Além disso, para recalcular o CMAC é necessário o HMAC da linha anterior e posterior. Estas operações a mais aumentam a sobrecarga para 192%, conforme apresentado na Figura 24. Entretanto, se considerarmos que os CMACs das linhas anterior e posterior

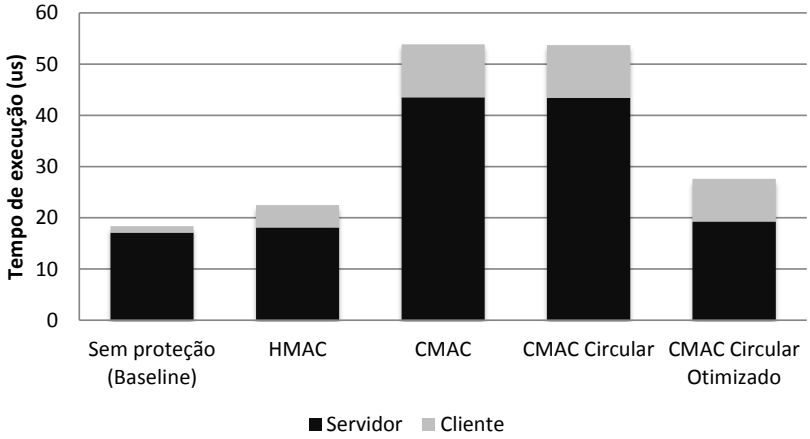


Figura 24: Comparação dos tempos de execução para a operação *SELECT*

são retornados na mesma consulta *SQL*, a sobrecarga cai para 50%.

5.7 AVALIAÇÃO DA VERIFICAÇÃO DA ATUALIDADE

Para a verificação da atualidade de um registro, é comparado o MAC do registro armazenado no servidor com o MAC do registro armazenado no *front-end*. No lado do *front-end*, este MAC fica armazenado em uma tabela de *hash* carregada em memória, onde a chave da tabela é o MAC e o seu valor é a chave primária do registro.

Como pode ser visto na Figura 25, o custo, em termos de tempo de execução, é irrelevante. Por exemplo, para uma tabela com 1 milhão de registros, a verificação da atualidade demorou, em média, 0,007ms, enquanto que a busca do registro no servidor demorou, em média, 87ms. Por conta deste pequeno custo, as linhas para os dois cenários (com e sem verificação) estão sobrepostas, não sendo possível visualizar ambas as linhas. O custo de armazenamento dos MACs no *front-end* também é pequeno. Para a mesma tabela

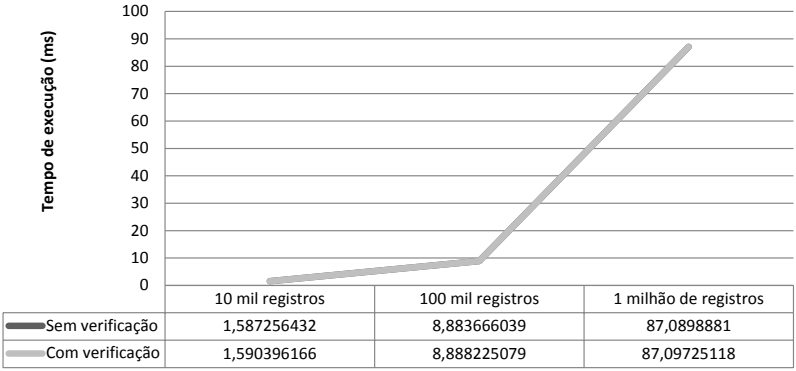


Figura 25: Comparação dos tempos de execução para a verificação de atualidade

com 1 milhão de registros são necessários menos de 10Mb para armazenar a tabela de *hash* contendo todos os MACs.

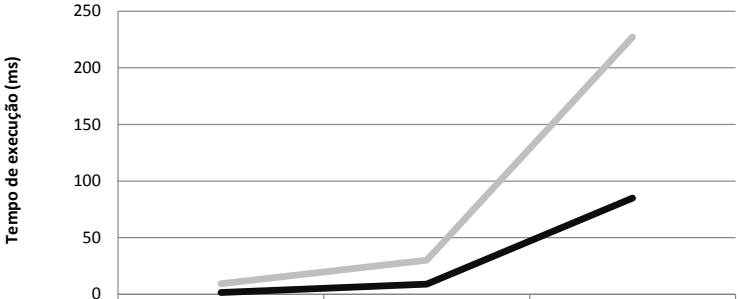
5.8 AVALIAÇÃO DA RECUPERAÇÃO DE UMA LINHA

Quando a verificação do MAC falhar, a recuperação do registro é feita através da base de logs mantida no *front-end*. A tupla original é recuperada do log e então é feita uma atualização no BD com os valores recuperados do log.

Como pode ser visto na Figura 26, há um custo adicional considerável quando se deseja recuperar uma tupla para o seu valor original. Este custo está associado diretamente a dois fatores: (i) busca da tupla na base de logs e (ii) atualização do BD com os valores recuperados do log. Entretanto, o custo relativo diminui de acordo com o aumento do tamanho do BD. Para a tabela com 10 mil registros, a sobrecarga no tempo de execução é de 83%. Já na tabela com 100 mil registros, cai para 70% e, por fim, na tabela com 1 milhão de registros a sobrecarga é de 63%.

5.9 AVALIAÇÃO DO USO DE MÚLTIPLAS CHAVES

Para a avaliação do uso de múltiplas chaves para o BD, foi comparado o tempo de execução da operação *SELECT* para buscar dez registros

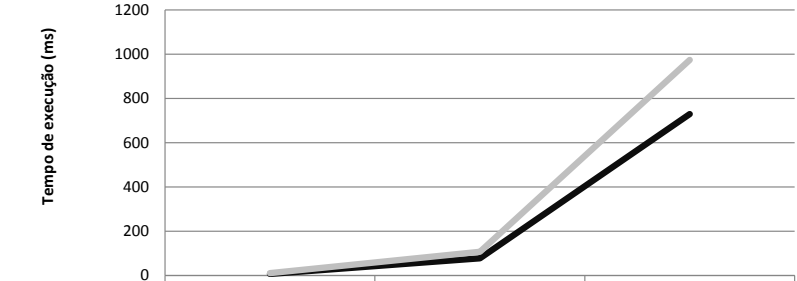


	10 mil registros	100 mil registros	1 milhão de registros
Sem recuperação	1,54660368	8,92047596	85,01344895
Com recuperação	9,259181976	30,1240623	227,1851668

Figura 26: Comparação dos tempos de execução para a recuperação de uma linha

aleatórios do BD, verificando a integridade (MAC e CMAC) dos mesmos. No caso com chave única, a chave está explícita no código, como um atributo da classe que realiza a verificação da integridade. Já no caso de múltiplas chaves, elas estão armazenadas em um BD *SQLite* carregado em memória. Além disso, o ambiente foi criado de forma que cada chave é utilizada para um intervalo de 5 registros. Isto é, a primeira chave é utilizada para os registros de 1 a 5, a segunda para os registros de 6 a 10 e assim por diante. A operação *SELECT* busca dez registros aleatórios de forma que, para os cenários de múltiplas chaves pode ser necessária a utilização (e consequentemente a busca) de 2 ou 3 chaves distintas.

Como pode ser visto na Figura 27, há um custo razoável para realizar a busca das chaves. Entretanto, assim como na recuperação de tuplas, o custo relativo diminui conforme o tamanho da tabela aumenta. Para a tabela com 10 mil registros, a sobrecarga no tempo de execução é de 48%. Já na tabela com 100 mil registros, cai para 37% e, por fim, na tabela com 1 milhão de registros a sobrecarga é de 34%. É importante ressaltar que esta sobrecarga pode variar muito dependendo de como a base de chaves for configurada. Por exemplo, se o intervalo de tuplas for grande, haverá menos acessos à base de chaves e, portanto, menor sobrecarga no tempo de execução.



	10 mil registros	100 mil registros	1 milhão de registros
Chave única	6,69726261	77,56039166	728,9059858
Múltiplas chaves	9,882750335	106,528789	973,7437389

Figura 27: Comparação dos tempos de execução entre o uso de chave única e múltiuplas chaves

5.10 RESUMO DO CAPÍTULO

Este capítulo apresentou os experimentos realizados para avaliar os métodos propostos no capítulo 4. Os resultados, em termos de tempo de execução, foram apresentados para as operações básicas *SQL*: *INSERT*, *UPDATE*, *DELETE* e *SELECT*. Como pode ser observado nos testes, o impacto do uso de cada método é afetado por dois principais fatores:

1. o número de operações *SQL* executadas;
2. o número de operações criptográficas executadas.

Além disso, pode-se notar que o número de operações *SQL* adicionais necessárias para cada método (por exemplo, a atualização de uma linha utilizando CMAC requer duas operações *SQL*) tem um impacto muito maior na sobrecarga de execução do que as operações criptográficas. Esta diferença pode ser notada nos cenários onde são necessárias apenas operações criptográficas (cenários apenas com o uso do MAC), onde houve um impacto consideravelmente menor.

Apesar dos valores relativos apresentados para as operações *SQL*, em muitos casos, serem altos, é importante ressaltar que, em termos de valores absolutos, o tempo de execução sempre foi baixo. Isto é, os resultados apresentados para as operações *SQL* estão em μs. A maior sobrecarga foi para o cenário do CMAC na operação de *INSERT*. Neste cenário, houve um aumento de 459%. Entretanto, em termos de tempo de execução, o aumento

foi de apenas 194μs. Já no caso da operação *SELECT*, onde houve o menor impacto, o aumento foi de apenas 9μs.

A seção 5.2 mostra como os cenários de otimização foram simulados nos experimentos. Além do método apresentado na seção 5.2, os casos onde os valores de MACs e CMACs de linhas adjacentes necessitam estar disponíveis no lado do cliente, para poder ser possível a verificação da integridade e/ou atualização dos dados da tabela, também podem ser implementados através de uma *cache* local do lado do cliente. Esta *cache* poderia ser mantida em memória e de um tamanho fixo, armazenando apenas os MACs e CMACs das linhas mais acessadas, diminuindo a probabilidade de serem necessárias consultas adicionais ao BD para buscar tais valores. Esta abordagem é especificamente interessante para a arquitetura com *front-end*, apresentada na seção 4.2.

Os experimentos realizados para avaliar as técnicas propostas para a arquitetura com *front-end* também foram apresentados, sendo eles: verificação da atualidade de registros, recuperação do valor original de um registro não íntegro e utilização de múltiplas chaves para o BD. Constatou-se que o custo da verificação da atualidade é irrelevante, tanto em termos de tempo de execução quanto em memória necessária para armazenar a tabela de *hash* com os MACs. Já para a recuperação de um registro há uma sobrecarga considerável, entretanto, parte dessa sobrecarga é intrínseca à própria recuperação, que é o custo em atualizar a tupla no BD e, portanto, estará presente em qualquer técnica que não altere o SGBD. Também pode-se notar que a sobrecarga diminui conforme o tamanho do BD aumenta. Além disso, assim como apresentado na avaliação das operações *SQL*, é importante ressaltar que o tempo de execução sempre foi baixo. Para uma tabela com 1 milhão de registros, o custo adicional para recuperar uma tupla foi de apenas 142ms.

Por fim, os experimentos com múltiplas chaves mostram que a sobrecarga em se utilizar uma base de chaves é aceitável e, assim como na recuperação de tuplas, diminui conforme o tamanho do BD aumenta. Além disso, esse custo pode ser controlado através da configuração da base de chaves. Por exemplo, os valores *II* e *FI*, apresentados na seção 4.8.1, podem ser ajustados para reduzir a quantidade de chaves utilizadas e, consequentemente, melhorar a eficiência da busca.

6 CONSIDERAÇÕES FINAIS

Este trabalho tem como objetivo propor e avaliar mecanismos que permitam a detecção e recuperação de modificações não autorizadas a dados presentes em BDs relacionais. Os mecanismos propostos fazem uso de MACs para possibilitar a verificação da integridade dos dados. Um MAC é calculado para cada linha de cada tabela do BD a partir da concatenação dos atributos das linhas e o resultado é salvo em uma nova coluna. Este mecanismo garante a corretude (*correctness)* das linhas, isto é, garante que o conteúdo das linhas não foi modificado de forma maliciosa.

Para prover garantias da completude (*completeness*) das consultas, e garantir que nenhum linha foi removida maliciosamente, foi proposto um novo algoritmo, chamado CMAC. O objetivo deste algoritmo é encadear as linhas, de forma que não seja possível adicionar ou remover novas linhas sem o conhecimento da chave secreta, de conhecimento apenas do cliente.

As técnicas clássicas para tratar a verificação da integridade de dados e os conceitos necessários para compreendê-las são abordadas no Capítulo 2. Estas técnicas, entretanto, não atendem as necessidades de BDs relacionais, sendo necessário novas abordagens para o tratamento da verificação da integridade de dados.

Nesse contexto, o Capítulo 3 apresenta os trabalhos existentes na literatura para tratar do problema de integridade para BDs terceirizados. Foi possível notar que as abordagens existentes na literatura dividem-se em duas categorias principais: abordagens baseadas em assinatura digital e baseadas em estruturas autenticadas. Verificou-se que tais abordagens possuem limitações e desvantagens. Por fim, foi apresentado um comparativo entre estas abordagens.

Buscando eliminar as limitações dos trabalhos existentes na literatura, o Capítulo 4 apresentou os mecanismos para tratar a verificação da integridade dos dados propostos neste trabalho. Para obter um mecanismo eficiente, foram empregadas funções MAC, além de proposto um novo algoritmo, o CMAC. Os mecanismos apresentados também são independentes de SGBD, pois os valores do MAC e CMAC são armazenados em colunas adicionais. Dessa forma, alcançou-se um mecanismo flexível, que pode ser implementado tanto no servidor quando no cliente, ou ainda em um *front-end*. As vantagens e desvantagens da implementação em cada uma destas partes são descritas na seção 4.10.

O Capítulo 4 dedicou-se também a descrever propostas iniciais para tratar da recuperação dos dados e da verificação da atualidade (*freshness*) dos dados. Para possibilitar a recuperação dos dados, é proposto manter um regis-

tro da última operação de modificação no BD para cada linha. Como é feita a verificação da integridade antes da utilização do dado, não é necessário manter um registro completo de todas as operações SQL executadas, diminuindo drasticamente o tamanho do arquivo de log, se comparado com outros trabalhos. A verificação da atualidade do dado é feita mantendo localmente um pequeno registro de todos os MACs existentes no BD. Quando o MAC é calculado para a verificação da integridade, compara-se o MAC calculado com o MAC armazenado localmente. Se eles forem iguais significa que o SGBD retornou o dado referente a última atualização realizada. Ainda, descreveu-se o problema do gerenciamento das chaves utilizadas para o cálculo do MAC e CMAC. Para aumentar a segurança e diminuir o impacto de um comprometimento de chave, discorreu-se sobre a importância do uso de múltiplas chaves. Para possibilitar o uso de múltiplas chaves, apresentou-se uma tabela simples que correlaciona uma determinada chave com um conjunto de tuplas. Desta forma, se uma chave for comprometida, apenas um conjunto de tuplas é afetado, ao invés de todo o BD.

Para avaliar os mecanismos de verificação de integridade, foram realizados experimentos, descritos no Capítulo 5. Foi testado a sobrecarga de processamento do cálculo do MAC e CMAC para cada operação básica SQL (*INSERT*, *UPDATE*, *DELETE* e *SELECT*). Ainda, para cada operação SQL, criaram-se diferentes cenários de teste: sem mecanismo de verificação de integridade; utilizando apenas o MAC; utilizando o MAC e CMAC; e utilizando o MAC e CMAC no modo circular. Os resultados se mostraram condizentes com as necessidades de um ambiente de BD dinâmico, sendo que a operação *SELECT*, comumente a mais executada, foi a que apresentou a menor sobrecarga: 22% para a verificação do MAC e 50% para a verificação do MAC e CMAC. É importante salientar que a verificação do CMAC, a mais custosa, nem sempre é necessária. Só se faz necessário verificar o CMAC quando a operação *SELECT* retorna um conjunto de tuplas sequenciais e deseja-se verificar se alguma tupla no intervalo da busca foi removida maliciosamente ou omitida pelo SGBD.

Por fim, conclui-se que os objetivos deste trabalho foram atingidos. Os mecanismos propostos apresentam vantagens sobre todos os trabalhos correlatos apresentados no Capítulo 3, como pode ser visto na tabela 5. Do ponto de vista de eficiência, o nosso trabalho não é superior a todos os trabalhos correlatos, entretanto, os trabalhos que podem ser superiores em eficiência apresentam outras desvantagens: requerem modificação no *kernel* do SGBD ou não provêm verificação da completude de consultas. Por outro lado, as abordagens que são independentes de SGBD e/ou provêm tanto a corretude quanto a completude fazem uso de funções criptográficas de baixo desempenho, como a criptografia assimétrica, ou utilizam árvores para armazenar

Tabela 5: Comparação entre os trabalhos relacionados e este trabalho

	Implementação	Criptografia	Corretude	Completude
Este trabalho	Flexível	Baixo custo	Sim	Sim
(KAMEL, 2009)	No SGBD	Não utiliza	Sim	Não
(LI et al., 2006)	No SGBD	Médio custo	Sim	Sim
(BATTISTA; PALAZZI, 2007)	Flexível	Médio custo	Sim	Sim
(MIKLAU; SUCIU, 2005)	Flexível	Médio custo	Sim	Sim
(MYKLETUN; NARASIMHA; TSUDIK, 2006)	Flexível	Alto custo	Sim	Não
(NARASIMHA; TSUDIK, 2005)	Flexível	Alto custo	Sim	Sim
(XIE et al., 2007)	Flexível	Não utiliza	Não	Sim

os dados de controle de integridade. De forma geral, estas árvores possuem complexidade $O(log(n))$ para manipulação, isto é, para a consulta, inserção e remoção de dados, enquanto que os nossos mecanismos possuem complexidade $O(1)$.

6.1 PUBLICAÇÕES

Esta dissertação gerou as seguintes publicações:

1. Um artigo no 28º Workshop de Teses e Dissertações em Banco de Dados (WTDBD'13) (SILVéRIO; MELLO; CUSTóDIO, 2013)
2. Um artigo completo no *Sixth International Conference on Advances in Databases, Knowledge, and Data Applications* (DBKDA'14) (SILVéRIO et al., 2014)

6.2 TRABALHOS FUTUROS

Como sugestão de ponto de partida para trabalhos futuros tem-se a extensão deste trabalho para outros tipos de BDs, como os *NoSQL*. No modelo *NoSQL*, ao invés de tuplas, temos objetos complexos cujos atributos podem ser valores simples (*string*, inteiro, etc) ou valores complexos (outros obje-

tos). Dessa forma, o cálculo do MAC e CMAC precisa ser repensado de forma a garantir o mesmo nível de granularidade.

Outra possibilidade de trabalho futuro é a eliminação da necessidade de ordenação das tuplas pela chave primária. Esta necessidade pode gerar problemas caso a tabela necessite ser reordenada por algum outro atributo. Uma alternativa para este problema é utilizar o ordenamento da tabela de índices do próprio SGBD (árvore B), pois este ordenamento geralmente não é alterado. Uma cópia desta árvore poderia ser armazenada no *front-end* e, ao invés dos dados, os nodos folha da árvore conteriam o CMAC. Outra alternativa é inserir alguma referência que indique qual foi a tupla $n - 1$ utilizada para o cálculo do CMAC no próprio CMAC.

REFERÊNCIAS

AGGARWAL, G. et al. Two can keep a secret: A distributed architecture for secure database services. In: **In Proc. CIDR**. [S.l.: s.n.], 2005.

AGRAWAL, R. et al. Order preserving encryption for numeric data. In: **Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2004. (SIGMOD '04), p. 563–574. ISBN 1-58113-859-8. Disponível em: <http://doi.acm.org/10.1145/1007568.1007632>.

AMMANN, P.; JAJODIA, S.; LIU, P. Recovery from malicious transactions. **Knowledge and Data Engineering, IEEE Transactions on**, v. 14, n. 5, p. 1167–1185, Sep 2002. ISSN 1041-4347.

AUMASSON, J.-P.; MEIER, W.; MENDEL, F. Preimage attacks on 3-pass haval and step-reduced md5. In: AVANZI, R.; KELIHER, L.; SICA, F. (Ed.). **Selected Areas in Cryptography**. Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5381). p. 120–135. ISBN 978-3-642-04158-7. Disponível em: <http://dx.doi.org/10.1007/978-3-642-04159-4_8>.

BARKER, E. et al. **Recommendation for Key Management - Pat1: General (Revision 3)**. [S.l.], Jul 2012. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf>.

BATTISTA, G. D.; PALAZZI, B. Authenticated relational tables and authenticated skip lists. In: **Proceedings of the 21st annual IFIP WG 11.3 working conference on Data and applications security**. Berlin, Heidelberg: Springer-Verlag, 2007. p. 31–46. ISBN 978-3-540-73533-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1770560.1770564>.

BELLARE, M.; CANETTI, R.; KRAWCZYK, H. Keying hash functions for message authentication. In: **Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology**. London, UK, UK: Springer-Verlag, 1996. (CRYPTO '96), p. 1–15. ISBN 3-540-61512-1. Disponível em: <http://dl.acm.org/citation.cfm?id=646761.706031>.

CESELLI, A. et al. Modeling and assessing inference exposure in encrypted databases. **ACM Transactions on Information and System Security (TISSEC)**, v. 8, n. 1, February 2005.

CHAKRABORTY, A.; MAJUMDAR, A. K.; SURAL, S. A column dependency-based approach for static and dynamic recovery of databases from malicious transactions. **Int. J. Inf. Sec.**, v. 9, n. 1, p. 51–67, 2010.

CIRIANI, V. et al. Keep a few: Outsourcing data while maintaining confidentiality. In: **Proc. of the 14th European Symposium On Research In Computer Security (ESORICS 2009)**. Saint Malo, France: [s.n.], 2009.

CIRIANI, V. et al. Combining fragmentation and encryption to protect privacy in data storage. **ACM Transactions on Information and System Security**, v. 13, n. 3, p. 1–33, jul. 2010. ISSN 10949224. Disponível em: <http://portal.acm.org/citation.cfm?doid=1805974.1805978>.

CORMEN, T. H. et al. **Introduction to Algorithms, Third Edition**. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262033844, 9780262033848.

DANG, T. K. Ensuring correctness, completeness, and freshness for outsourced tree-indexed data. **Inf. Resour. Manage. J.**, IGI Global, Hershey, PA, USA, v. 21, n. 1, p. 59–76, jan. 2008. ISSN 1040-1628. Disponível em: <http://dx.doi.org/10.4018/irmj.2008010104>.

DATE, C. **An Introduction to Database Systems**. 8. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321197844.

De Capitani di Vimercati, S. et al. Fragmentation in presence of data dependencies. **IEEE Transactions on Dependable and Secure Computing (TDSC)**, 2014. To appear.

ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems**. 6th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0136086209, 9780136086208.

GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. **SIGMOD Rec.**, ACM, New York, NY, USA, v. 14, n. 2, p. 47–57, jun. 1984. ISSN 0163-5808. Disponível em: <http://doi.acm.org/10.1145/971697.602266>.

HACIGUMUS, H. et al. Executing sql over encrypted data in the database-service-provider model. In: **Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data**. New York,

NY, USA: ACM, 2002. (SIGMOD '02), p. 216–227. ISBN 1-58113-497-5. Disponível em: <http://doi.acm.org/10.1145/564691.564717>.

HACIGUMUS, H.; MEHROTRA, S. Efficient key updates in encrypted database systems. In: **Proceedings of the Second VDLB international conference on Secure Data Management**. Berlin, Heidelberg: Springer-Verlag, 2005. (SDM'05), p. 1–15. ISBN 3-540-28798-1, 978-3-540-28798-8. Disponível em: <http://dx.doi.org/10.1007/11552338_1>.

HEITZMANN, A. et al. Efficient integrity checking of untrusted network storage. In: **Proceedings of the 4th ACM international workshop on Storage security and survivability**. New York, NY, USA: ACM, 2008. (StorageSS '08), p. 43–54. ISBN 978-1-60558-299-3. Disponível em: <http://doi.acm.org/10.1145/1456469.1456479>.

Infosecurity Europe; PRICEWATERHOUSECOOPERS. **Information security breaches survey**. Earl's Court, London, April 2010. Disponível em: <http://pdf.pwc.co.uk/isbs_survey_2010_technical_report.pdf>.

Infosecurity Europe; PWC. **Information security breaches survey**. Earl's Court, London, April 2012. Disponível em: <http://www.pwc.co.uk/en_UK/uk/assets/pdf/olpapp/uk-information-security-breaches-survey-technical-report.pdf>.

Infosecurity Europe; PWC. **Information security breaches survey**. Earl's Court, London, April 2013. Disponível em: <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/200455/bis-13-p184-2013-information-security-breaches-survey-technical-report.pdf>.

KAMEL, I. A schema for protecting the integrity of databases. **Computers & Security**, v. 28, n. 7, p. 698–709, 2009.

KRAWCZYK, H.; BELLARE, M.; CANETTI, R. **HMAC: Keyed-Hashing for Message Authentication**. IETF, fev. 1997. RFC 2104 (Informational). (Request for Comments, 2104). Disponível em: <http://www.ietf.org/rfc/rfc2104.txt>.

LI, F. et al. Dynamic authenticated index structures for outsourced databases. In: **Proceedings of the 2006 ACM SIGMOD international**

conference on Management of data. New York, NY, USA: ACM, 2006. (SIGMOD '06), p. 121–132. ISBN 1-59593-434-0. Disponível em: <http://doi.acm.org/10.1145/1142473.1142488>.

LIU, P. Architectures for intrusion tolerant database systems. In: **Computer Security Applications Conference, 2002. Proceedings. 18th Annual**. [S.l.: s.n.], 2002. p. 311–320. ISSN 1063-9527.

LIU, P.; AMMANN, P.; JAJODIA, S. Rewriting histories: Recovering from malicious transactions. In: ATLURI, V.; SAMARATI, P. (Ed.). **Security of Data and Transaction Processing**. [S.l.]: Springer US, 2000. p. 7–40. ISBN 978-1-4613-7009-3.

MAO, W. **Modern Cryptography: Theory and Practice**. [S.l.]: Prentice Hall Professional Technical Reference, 2003. ISBN 0130669431.

MARTINA, J.; SOUZA, T.; CUSTODIO, R. Openhsm: An open key life cycle protocol for public key infrastructure's hardware security modules. In: LOPEZ, J.; SAMARATI, P.; FERRER, J. (Ed.). **Public Key Infrastructure**. Springer Berlin Heidelberg, 2007, (Lecture Notes in Computer Science, v. 4582). p. 220–235. ISBN 978-3-540-73407-9. Disponível em: <http://dx.doi.org/10.1007/978-3-540-73408-6_16>.

MERKLE, R. C. A certified digital signature. In: BRASSARD, G. (Ed.). **CRYPTO**. [S.l.]: Springer, 1989. (Lecture Notes in Computer Science, v. 435), p. 218–238. ISBN 3-540-97317-6.

MIKLAU, G.; SUCIU, D. Implementing a tamper-evident database system. In: **Proceedings of the 10th Asian Computing Science conference on Advances in computer science: data management on the web**. Berlin, Heidelberg: Springer-Verlag, 2005. (ASIAN'05), p. 28–48. ISBN 3-540-30767-2, 978-3-540-30767-9. Disponível em: <http://dl.acm.org/citation.cfm?id=2074944.2074951>.

MUNRO, J. I.; PAPADAKIS, T.; SEDGEWICK, R. Deterministic skip lists. In: **Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms**. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992. (SODA '92), p. 367–375. ISBN 0-89791-466-X. Disponível em: <http://dl.acm.org/citation.cfm?id=139404.139478>.

MYKLETUN, E.; NARASIMHA, M.; TSUDIK, G. Authentication and integrity in outsourced databases. **ACM Transactions on Storage**, v. 2, n. 2, p. 107–138, maio 2006. ISSN 15533077. Disponível em: <http://portal.acm.org/citation.cfm?doid=1149976.1149977>.

MYSQL. **MySQL: The world's most popular open source database**. 2014. Disponível em: <http://www.mysql.com>. Acesso em: 2014.

NARASIMHA, M.; TSUDIK, G. Dsac: integrity for outsourced databases with signature aggregation and chaining. In: **Proceedings of the 14th ACM international conference on Information and knowledge management**. New York, NY, USA: ACM, 2005. (CIKM '05), p. 235–236. ISBN 1-59593-140-6. Disponível em: <http://doi.acm.org/10.1145/1099554.1099604>.

National Institute of Standards and Technology. **FIPS PUB 140-2: Security Requirements for Cryptographic Modules**. National Institute of Standards and Technology, 2002. Disponível em: <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>.

National Institute of Standards and Technology. **FIPS PUB 180-4: Secure Hash Standard**. National Institute of Standards and Technology, 2012. Disponível em: <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

OPENSSL. **OpenSSL: The Open Source toolkit for SSL/TLS**. 2013. Disponível em: <http://www.openssl.org>. Acesso em: 2014.

PANG, H. et al. Verifying completeness of relational query results in data publishing. In: **Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 2005. (SIGMOD '05), p. 407–418. ISBN 1-59593-060-4. Disponível em: <http://doi.acm.org/10.1145/1066157.1066204>.

PUGH, W. Skip lists: a probabilistic alternative to balanced trees. **Commun. ACM**, ACM, New York, NY, USA, v. 33, n. 6, p. 668–676, jun. 1990. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/78973.78977>.

SASAKI, Y.; AOKI, K. Preimage attacks on step-reduced md5. In: MU, Y.; SUSILO, W.; SEBERRY, J. (Ed.). **Information Security and Privacy**. Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 5107). p. 282–296. ISBN 978-3-540-69971-2. Disponível em: <http://dx.doi.org/10.1007/978-3-540-70500-0_21>.

SCHNEIER, B. **Applied cryptography (2nd ed.): protocols, algorithms, and source code in C**. New York, NY, USA: John Wiley & Sons, Inc., 1996. ISBN 0-471-11709-9.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Database Systems Concepts**. 5. ed. New York, NY, USA: McGraw-Hill, Inc., 2006. ISBN 0072958863, 9780072958867.

SILVéRIO, A. L. et al. Efficient data integrity checking for untrusted database systems. In: **Proceedings of the 2014 Sixth International Conference on Advances in Databases, Knowledge, and Data Applications**. IAIRA, 2014. (DBKDA'14), p. 118–124. ISBN 978-1-61208-334-6. Disponível em: <http://www.thinkmind.org/index.php?view=article&articleid=dbkda_2014_5_20_50068>.

SILVéRIO, A. L.; MELLO, R. S.; CUSTóDIO, R. F. Efficient integrity checking for untrusted database systems. In: **Proceedings of the Workshop of Thesis and Master Dissertations in Databases (WTDB**. [S.l.]: Sociedade Brasileira de Computação, 2013. (WTDBD'13), p. 36–42.

SQLITE. **SQLite**. 2014. Disponível em: <https://sqlite.org/>. Acesso em: 2014.

WANG, H.; LAKSHMANAN, L. V. S. Efficient secure query evaluation over encrypted xml databases. In: **Proceedings of the 32Nd International Conference on Very Large Data Bases**. VLDB Endowment, 2006. (VLDB '06), p. 127–138. Disponível em: <http://dl.acm.org/citation.cfm?id=1182635.1164140>.

XIE, M. et al. Integrity auditing of outsourced data. In: **Proceedings of the 33rd International Conference on Very Large Data Bases**. VLDB Endowment, 2007. (VLDB '07), p. 782–793. ISBN 978-1-59593-649-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1325851.1325940>.

XIE, M. et al. Providing freshness guarantees for outsourced databases. In: **Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology**. New York, NY, USA: ACM, 2008. (EDBT '08), p. 323–332. ISBN 978-1-59593-926-5. Disponível em: <http://doi.acm.org/10.1145/1353343.1353384>.

YAJIMA, J. et al. A strict evaluation on the number of conditions for sha-1 collision search. **IEICE Transactions**, v. 92-A, n. 1, p. 87–95, 2009.

ZHU, H. et al. Dynamic data recovery for database systems based on fine grained transaction log. In: **Proceedings of the 2008 International**

Symposium on Database Engineering & Applications. New York, NY, USA: ACM, 2008. (IDEAS '08), p. 249–253. ISBN 978-1-60558-188-0. Disponível em: <http://doi.acm.org/10.1145/1451940.1451975>.