Gabriel Garcia Becker, Lucas Pandolfo Perin

Integridade de Banco de Dados

Florianópolis, Santa Catarina

2013.1

Gabriel Garcia Becker, Lucas Pandolfo Perin

Integridade de Banco de Dados

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador:
Marcelo Carlomagno Carlos

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis, Santa Catarina

2013.1

Trabalho de conclusão de curso sob o título *"Integridade de Banco de Dados"*, defendido por Gabriel Garcia Becker, Lucas Pandolfo Perin e aprovado em 10 de novembro de 2013, em Florianópolis, Santa Catarina, pela banca examinadora constituída pelos professores:

Marcelo Carlomagno Carlos
Royal Holloway University of London
Orientador

Prof. Dr. Ricardo Felipe Custódio
Universidade Federal de Santa Catarina
Co-orientador

Anderson Luiz Silverio
Universidade Federal de Santa Catarina
Membro da Banca

Felipe Werlang
Universidade Federal de Santa Catarina
Membro da Banca

Agradecimentos

"Não há fatos eternos, como não há verdades absolutas."

Friedrich Nietzsche

Sumário

Lista de Siglas

SGBD Sistema de Gerenciamento de Banco de Dados

DBMS *Database Management Systems*

TPM *Trusted Platform Module*

HMAC *Hash-based Message Authentication Code*

DBA administrador de banco de dados

PKI Infraestrutura de Chaves Públucas

Lista de Figuras

Lista de Tabelas

Resumo

Modificações não autorizadas a sistemas de banco de dados podem causar prejuízos para pessoas e organizações, sendo de extrema importância a garantia de sigilo e integridade a tais sistemas. Geralmente aplicações utilizam os recursos disponibilizados por Sistema de Gerenciamento de Banco de Dados (SGBD) para assegurar o sigilo e a integridade dos dados armazenados no SGBD. Entretanto, os SGBDs que provêm os recursos necessários para garantir o sigilo e a integridade dos dados possuem um custo muito elevado, muitas vezes inviável para organizações de médio e pequeno porte. Este trabalho analisa e implementa um método de verificação de integridade de dados, baseado no uso de HMAC, independente de SGBD, aproveitando estruturas possívelmente disponíveis como *Trusted Platform Module* (TPM) e *smart cards* entre outros dispositivos criptográficos. Os testes realizados mostram a eficiência do método. Em média, a sobrecarga de processamento para o cálculo e verificação do HMAC para um registro do banco de dados não ultrapassa 100% do tempo de execução de determinada operação.

não precisa das palavras chaves?

Abstract

Cópia do artigo

Unauthorized changes of database content can result in significant losses for organizations and individuals making it is extremely important to assure the privacy and integrity of databases. The *Database Management Systems* (DBMS) provides features to encrypt the database and protect it against unauthorized access. However, open-source DBMS do not provide means to protect the database against unauthorized modifications and advanced DBMS that do provide such features are too expensive for small business enterprises. In this paper, we analyse and implement a method to protect database systems against unauthorized modifications. Furthermore we tested the analysed method, showing its efficiency.

1 *Introdução*

O uso de sistemas de bancos de dados tornou-se recorrente para os mais diversos tipos de aplicações. Com o uso extensivo da internet, as aplicações que utilizam sistemas de bancos de dados *online* são cada vez mais comuns.

1.1 Contextualização

Aplicações que utilizam bancos de dados *online* normalmente armazenam dados sensíveis, como salários e outras informaçõees pessoais (KAMEL, 2009). Devido ao conteúdo potencialmente sigiloso, o acesso ou a modificação não autorizada a tais dados pode não ser desejado. Dessa forma, o sigilo e integridade de sistemas de banco de dados tem atraído pesquisadores das áreas de banco de dados e segurança. Os sistemas de banco de dados, quando utilizados em ambientes compartilhados, possuem diversas ameaças de segurança provenientes de usuários não-autorizados, mau-uso e ameaças externas. Baseando-se nestas características, pode-se classificar as ameaças em quatro tipos principais:

1. leitura não autorizada de dados;
2. modificação não autorizada de dados;
3. remoção não autorizada de dados;
4. adição não autorizada de dados.

Está OK para TCC1, adicionar mais dados para a monografia. Dados com datas, marcos históricos etc..

1.2 Objetivos

1.2.1 Gerais

Este trabalho tem como objetivo geral apresentar soluções plausíveis para garantir a integridade de bancos de dados.

1.2.2 Específicos

- Elaborar uma biblioteca em Java de com a finalidade de prover suporte para as soluções apresentadas neste trabalho;
- Providenciar suporte aos dispositivos criptográficos existentes no mercado com a biblioteca, incrementando novas camadas de segurança para as aplicações;
- Facilitar o desenvolvimento de aplicações de *software* com garantia de integridade de bancos de dados;
- Manter as soluções apresentadas dentro de uma faixa de desempenho atraente para aplicações que possam vir a utilizar a biblioteca criada.

1.3 Justificativa

O problema da leitura não autorizada de dados (sigilo dos dados) já foi pesquisado extensivamente e normalmente é resolvido através da cifração do banco de dados (CESELLI et al., 2005; SAMARATI; VIMERCATI, 2010), em conjunto com métodos de indexação (CESELLI et al., 2005; DAMIANI et al., 2003), para agilizar as consultas. Entretanto, os problemas de modificações não autorizadas de dados (integridade dos dados) ainda necessitam de soluções eficientes (SAMARATI; VIMERCATI, 2010). Os métodos existentes para tal problema geralmente requerem o desenvolvimento de um novo Sistema de Gerenciamento de Banco de Dados (SGBD) ou alterações significantes nos SGBDs já existentes (XIE; WANG; YIN, 2007). Adicionalmente, soluções disponíveis no mercado possuem custos elevados de implantação e manutenção, inviáveis para empresas de pequeno e médio porte que necessitam de tais serviços de segurança.

1.4 Metodologia

Neste trabalho, será estudado toda a fundamentação teórica necessária para compreender as soluções propostas. Será demonstrado como as ameaças citadas anteriormente podem ser combatidas utilizando *Hash-based Message Authentication Code* (HMAC) e cifração de dados sigilosos. Em seguida, serão comentadas e justificadas as tecnologias utilizadas para o desenvolvimento da biblioteca. Em posse desse conhecimento, pode-se finalmente elaborar mais sobre a solução. O próximo passo, será demonstração da fundamentação da biblioteca. Como ela está construida e como que seus serviços são acessados pelas aplicações. É importante levantar que uma biblioteca de *software* não constitui uma aplicação por si só. Ela apenas contribui com os recursos necessários para soluções. Em um último momento, as soluções criadas neste trabalho serão testadas em divérsos cenários. Dessa forma, pode-se demonstrar o custo de desempenho para a utilização da biblioteca de integridade de bancos de dados.

1.5 Limitações do Trabalho

Verificar a necessidade. Usar TCC Dexter como exemplo

2 *Fundamentação Teórica*

2.1 Criptografia

Criptografia é o estudo e a arte de transformar informação de forma que fique ilégivel ou incompreensível para todos exceto a quem ela for destinada. Utilizada quase que exclusivamente nos seus dias mais atuais em circunstâncias diplomaticas e militares, hoje é mais do que um simples meio de trocar informação secreta (LUICIANO; PRICHETT, 1987). Existe a necessidade urgente de proteger a vasta quantidade de dados existêntes em meios digitais, mesmo aqueles em ambientes supostamente seguros. Continuar explorando o artigo do Luciano para relatar mais sobre história da criptografia (TCC2)

> *"A palavra criptografia significa escondido ou escrita secreta. Criptografia é conhecida geralmente como o embaralhamento e o desembaralhamento de mensagens privadas. Uma mensagem é embaralhada para manter sua privacidade ou para proteger sua confidencialidade. Técnicas modernas de criptografia são também usadas para determinar caso uma mensagem foi alterada após o momento de ter sido criada e para identificar a origem da mesma. Uma mensagem não adulterada tem integridade. O conhecimento da origem de uma mensagem significa autenticidade."* (HOUSLEY; POLK, 2001, p.-5)

Existem diversos algoritmos criptográficos para cifragem e decifragem de mensagens. Em sua maioria, faz-se uso de duas entradas para gerar uma mensagem cifrada. Uma entrada é o conteudo da informação sigilosa e a outra entrada é um valor secreto denominado chave. Depedendo se o algorítmo é dito simétrico ou asimétrico, existirão uma ou mais chaves funcionando de maneiras diferentes.

2.1.1 Criptografia Simétrica

Em *cripotografia simétrica*, usa-se apenas uma chave secreta entre quem envia e quem recebe a mensagem cifrada. É por este motivo que sistemas que fazem uso de criptografia simétrica podem também ser chamados de sistemas de *chaves secretas compartilhadas* (HOUSLEY; POLK, 2001). Em um cenário onde deseja-se enviar uma mensagem cifrada para alguém, deve-se garantir que essa chave seja uma segredo entre destino e destinatário. Ela será utilizada no processo de cifragem por quem envia e também no processo de decifragem por quem recebe.

O uso de chaves simétricas não garante segurança em todos os casos. Quando existe a necessidade de trocar mensagem com alguém que não possui a chave usada para cifragem, é necessário enviar a mesma de forma que o destino. Este é um momento bastante vulnerável onde terceiros podem obter a chave secreta e maliciosamente decifrar a mensagem sigilosa. Outra grande vulnerabilidade do uso de chaves simétricas é confiar em quem quer que seja que possui a sua chave de forma que ela seja armazenada de forma segura. Pouco adianta compartilhar de uma chave secreta caso ela não esteja devidamente protegida de terceiros. Mencionar tecnicas de evitar essas vulnerabilidades com citações

2.1.2 Criptografia Assimétrica

Enquanto o uso de chaves simétricas pode ser bastante conveniênte, o gerenciamento das mesmas pode se tornar bastante complexo. Em alguns casos, deseja-se manter uma vasta quantidade de chaves para comunicação segura entre diversos periféricos. Nesses casos algoritmos de comunicação, por exemplo, podem se tornar pouco eficiêntes devido a necessidade de gerenciar cada chave para um destino diferente.

Criptografia assimétrica ou *criptografia de chaves públicas*, utiliza duas chaves distintas no lugar de uma. Uma delas chamada de *chave privada* e a outra chamada de *chave pública*. Ambas chaves são complementares, porém, nunca deve ser possível obter-se a chave privada através da chave pública (HOUSLEY; POLK, 2001). O uso de chaves públicas simplifica bastante o processo de gerenciamente de chaves pelo fato de reduzir drasticamente o número de chaves que serão gerenciadas. Por outro lado, algorítmos de cifração de dados através do uso de chaves públicas não são tão eficiêntes quando os de chave simétrica. Adicionalmente, pelo fato de existir uma chave pública capaz de decifrar a informação protegida, esta informação não possui mais confidencialidade, apenas autenticidade.

Existem diversos usos para criptografia de chaves públicas. Entre eles, a troca de chaves públicas para gerar uma chave simétrica conhecida entre apenas dois periféricos (DIFFIE; HELL-

MAN, 1976). Outro exemplo é a Infraestrutura de Chaves Públucas (PKI), mantendo uma cadeia de entidades com posse de chaves privadas, é capaz de garantir autenticidade de documentos eletrônicos para usuários (HOUSLEY; POLK, 2001).

2.1.3 Resumo criptográfico

precisa disso? usamos o hmac, talvez só falar dele Chave de seção?

2.2 Bancos de Dados

2.3 Tecnologias usadas

Lucas Martins aconselhou não utilizar as subsections aqui. Resumir todo o conteudo em alguns parágrafos. Talvez cortar algumas tecnologias como Java, xml, Junit, etc..

2.3.1 Bouncy Castle

2.3.2 LibCryptodev

2.3.3 Java

2.3.4 SQLite

2.3.5 Subversion

2.3.6 JUnit

2.3.7 XML

2.3.8 TPMJ

foi usado o tpmj porque o modulo pkcs#11 para o acesso ao tpm era bastante instável

teste testi //só no comentario?

foi usado mais alguma coisa, mais alguma coisa precisa ser colocada aqui?

Keystore, vai na criptografia?

O PKCS#11 vai nisso?, e PKCS#12

falar o que é um provider?

chaves PEM e DER, falar?, deu problemas com isso no TPMJ

Ant? Foi usado isso para fazer os builds das versões

3 *Integridade e Sigilo*

ainda é só cópia do artigo

sera que essa parte antes da primeira seção não é melhor ir para introdução?

Um dos problemas mais comuns quando se trata do sigilo em base de dados é a leitura não autorizada de informações sensíveis presentes nas tabelas. A proteção contra este tipo de acesso pode ser ser feita a nível de aplicação, o que é de fato realizada na maioria dos casos. Para tal, sugere-se a cifração de dados em colunas das tabelas. Deve-se selecionar colunas onde existem informações sigilosas nas tabelas e mante-las cifradas. A cifração destes dados pode ser feita diretamente pela aplicação ou pelo SGBD.

Para não afetar muito o desempenho das consultas, é comum o uso de métodos de indexação juntamente com a cifração dos dados (CESELLI et al., 2005; DAMIANI et al., 2003; HACIGÜMÜS; IYER; MEHROTRA, 2004).

Para evitar a modificação não autorizada de registros contidos na base de dados, a aplicação deve restringir e definir as permissões de cada usuário. O SGBD também deve conter um mapeamento correto de usuários e seus respectivos privilégios a fim de impedir a execução de ações não autorizadas. Entretanto, existem perfis de usuários, como os administradores do servidor, administrador de banco de dados (DBA) e programadores, que podem acessar o SGBD sem o conhecimento da aplicação, podendo ocorrer situações onde, embora o usuário não seja autorizado a realizar determinada modificação, ele tenha permissões suficientes no sistema que permitem que ele o faça, mesmo que não intencionalmente. A cifração de colunas que contém dados sensíveis já reduz consideravelmente este problema, uma vez que um usuário mal-intencionado não conseguirá modificar o registro por não possuir a chave, e possivelmente sequer será capaz de identificar qual registro ele deve modificar para obter os efeitos que ele deseja. Embora as chances do atacante sejam reduzidas, ainda há a possibilidade de realizar modificação de registros a fim de corromper o sistema.

Infelizmente, os SGBDs mais comuns, como MySQL, PostreSQL, Firebird, etc, não possuem mecanismos para evitar tais problemas. Outros sistemas mais específicos possuem opções

mais avançadas, porém também possuem custo bastante elevado. Para prover tal funcionalidade, foi estudada uma proposta que faz uso de HMAC (BELLARE; CANETTI; KRAWCZYK, 1996; **??**).

3.1 Adição do cálculo do HMAC nos registros da base de dados

Para se implementar este mecanismo, a aplicação será a única responsável pela manipulação dos registros da base de dados, e, para prover tal garantia, a aplicação terá uma chave simétrica conhecida somente por ela. Esta chave, por sua vez, será utilizada para gerar o HMAC dos registros.

Através da utilização do HMAC, é possível detectar qualquer modificação não autorizada realizada nos registros. Isto deve-se ao fato de que o atacante não tem conhecimento da chave necessária para gerar o HMAC, impossibilitando-o que consiga calcular um valor de HMAC válido. Este mecanismo também soluciona a questão de adição de registros pelo atacante, pois novamente, este necessitará da chave da aplicação para realizar o cálculo do HMAC de forma correta.

3.2 Exemplo de implementação

Para exemplificar o funcionamento do método utilizando HMAC, considere uma tabela de pessoas, com as colunas id, nome, email e a coluna para armazenar o valor do HMAC. Os dados são apresentados na Tabela 1.

Tabela 1: Tabela de exemplo de dados.

id	**nome**	**email**	**hmac**
1	Joao	joao@foo.com.br	abcd
2	Maria	maria@foo.com.br	qwer
3	Ana	ana@foo.com.br	kjhd
4	Roberto	roberto@foo.com.br	vpiu

3.2.1 Adicionando registros

Para adicionar um novo registro, a aplicação deve primeiramente calcular o valor do HMAC para o novo registro. Esse cálculo é feito através da concatenação dos valores de todas as colunas da tabela. Para adicionar uma nova pessoa com o nome “Jose” e email “jose@foo.com.br”,

o cálculo do HMAC é apresentado na expressão (3.1).

$$HMAC(chave, Josejose@foo.com.br) = ohn4 \tag{3.1}$$

Após o calculo do HMAC, a aplicação envia o comando de insert ao banco de dados, conforme ilustrado no trecho de código 3.1.

Listing 3.1: Código SQL para inserir um registro com HMAC

```
INSERT INTO exemplo (nome, email, hmac)
VALUES ('Jose',
        'jose@foo.com.br',
        'ohn4');
```

3.2.2 Modificando registros

A modificação de um registro é semelhante à adição. Por exemplo, para a alteração do registro número 3, da Tabela 1, primeiramente deve-se consultar o banco de dados para obter o registro, conforme ilustrado no trecho de código 3.2. O próximo passo é calcular o HMAC, apresentado na expressão (3.2). Por fim, atualiza-se o registro com o comando update ilustrado no trecho de código 3.3.

Listing 3.2: Código SQL para selecionar um registro

```
SELECT nome, email FROM exemplo WHERE id='3';
```

$$HMAC(chave, Anaana.new@foo.com.br) = m3cx \tag{3.2}$$

Listing 3.3: Código SQL para atualizar um registro com HMAC

```
UPDATE exemplo SET email='ana.new@foo.com.br', hmac='m3cx
    '
    WHERE id='3';
```

3.2.3 Verificando a integridade de registros

Por fim, para verificação da integridade de um registro, deve ser feita uma consulta ao banco de dados pelo registro. Calcula-se o HMAC para o registro e compara o HMAC calculado com o HMAC obtido do banco de dados. A igualdade desses valores indica que o registro está íntegro.

3.3 Medidas de proteção contra remoção não autorizada

Através da utilização do HMAC, já é possível detectar qualquer modificação e adição não autorizada de registros. Entretanto, não é possível detectar a remoção não autorizada de registros.

Para o problema de remoção não autorizada de registros, sugere-se a criação de uma nova coluna para as tabelas do banco de dados, chamada de "Histórico cifrado". O histórico cifrado possui as seguintes características:

- permite relacionar dois ou mais registros de forma que possa se detectar a ausência de um deles, caso este seja removido;
- não permitir que uma terceira parte possa calcular o "histórico cifrado" sem conhecer as chaves de cifração;
- utilização de operações de baixo custo computacional: criptografia simétrica e a operação lógica "ou exclusivo" (XOR);
- requer pouco espaço de armazenamento;
- não permite que sejam detectadas deleções dos n últimos registros.

Para calcular o histórico cifrado de um registro n, obtém-se o HMAC desse registro e do registro anterior a ele. Em seguida é aplicada a função XOR nestes HMACs e o resultado é cifrado com uma chave simétrica. Esse cálculo é apresentado na expressão (3.3).

$$HistoricoCifrado(chave, HMAC_n, HMAC_{n-1}) = Cifrao(k, (HMAC_n \oplus HMAC_{n-1})) \tag{3.3}$$

3.3.1 Removendo um registro

Para remover um registro n, deve-se excluir o registro e atualizar o histórico cifrado do registro $n+1$. Para atualizar o histórico cifrado, utiliza-se o HMAC registro $n+1$ e $n-1$.

3.3.2 Verificando se um registro foi removido

Para verificar se um registro t_n, de uma tabela T foi removido, a seguinte propriedade deve ser satisfeita:

$$\forall t_n, t_{n+1} \in T : t_n.HistoricoCifrado = HistoricoCifrado(chave, t_n, t_{n+1}) \tag{3.4}$$

3.3.3 Remoção do ultimo registro

Esta proposta possui uma vulnerabilidade, o método apresentado não detecta quando os últimos registros são removidos indevidamente, uma vez que a verificação é feita com base no registo anterior. Uma possível solução para o problema consiste em adicionar ao final da tabela uma enupla com valores aleatórios conhecidos. Dessa forma, se o último registro for removido, poderá ser identificado, uma vez que os valores de controle não estarão mais presentes. Desenvolver melhor a ideia explicando como serão afetadas as operações CRUD. Mencionar também solução circular com citação do artigo do Anderson

Começar explicando a ideia? E depois o que foi feito para ela existir? Tem que falar da biblioteca, dos provideres para os cada dispositivo (e do callback remoto?), do gerenciador de chaves pkcs#12, e outros subprojetos

4 Testes de Desempenho

Os testes de desempenho vão antes da implementação, não?

Nós temos a ideia, testamos o custo dela, e como foi bom, implementamos.

Quais testes vamos usar? O seu feito em C perin?

4.1 Testes

4.2 Análise de Resultados

5 *Implementação*

5.1 Biblioteca

Para o cálculo do HMAC há uma classe responsável pela geração das chaves (HMACKeyGenerator), uma pelo cálculo do HMAC (HMACCalculator), além das classes que representam a chave para ser utilizada no cálculo do HMAC (HMACKey) e o HMAC em si (HMAC). A partir de uma chave HMAC é possível inicializar o HMACCalculator. Nele, é possível adicionar dados para o calculo do HMAC e finalizar a operação. O trecho de código 5.1 demonstra como é feito o cálculo do HMAC de uma mensagem qualquer.

Listing 5.1: Exemplo do cálculo do HMAC

```
HMacKey key = HMacKey::generate("SHA1");
HMACCalculator hmacCalc = new HMACCalculator(key);
hmacCalc.update("mensagem");
HMAC hmac = hmacCalc.doFinal();
```

Para o histórico cifrado, tem-se a classe responsável pelo cálculo (EncryptedHistoryCalculator) e a classe que representa o histórico cifrado (EncryptedHistory). Para iniciar o cálculo do histórico, é necessário uma chave simétrica (SymmetricKey), que pode ser gerada de forma semelhante à chave utilizada pelo HMAC, pela classe SymmetricKeyGenerator . Para o cálculo do histórico, é necessário fornecer uma lista de HMACs e uma chave simétrica, coforme ilustrado no trecho de código 5.2.

Listing 5.2: Exemplo do cálculo do HMAC

```
public EncryptedHistory generate(List<HMac> hMacs) throws
    SymmetricCipherException, EncryptionException {
        List<HMac> h = new ArrayList<HMac>();
        for(HMac hMac : hMacs) h.add(hMac.clone());
        byte[] xor = h.get(0).getBytes();
        for (int i = 1; i < h.size(); i ++) {
                byte[] hmac = h.get(i).getBytes();
```

```
                for (int j = 0; j < hmac.length; j++)
                    xor[j] = (byte) (xor[j] ^ hmac[j]);
            }
            byte[] result = cipher.doFinal(xor);
            return new EncryptedHistory(result);
        }
```

5.1.1 Provider

Um nome melhor para essa subseção, ja que vai ter uma seção para a implementação do provider

A classe SecurityService possibilita efetuar diretamente o cálculo de HMAC, Histórico Cifrado e cifração, sendo preciso apenas usar os métodos correspondentes diretamente, realizando toda a operação em apenas uma função, por exemplo, para realizar o cálculo de um HMAC, devemos apenas chamar o método *getHMAC(fields:List<objs>)* passando a lista de objetos que sera utilizada no cálculo. As chaves simétricas e assimétricas usadas são definidas pela clase SecurityServiceSpi. A classe SecurityServiceSpi deve ser implementada por um provider, com a finalidade de suportar os mais diversos dispositivos criptográficos através de uma mesma interface. Desta forma, pode-se adicionar diferentes dispositivos criptográficos simplesmente implementando uma interface, não sendo necessários realizar grandes alterações na aplicação. O diagrama de classes da Figura 1 apresenta esta abstração das classes SecurityService e SecurityServiceSpi.

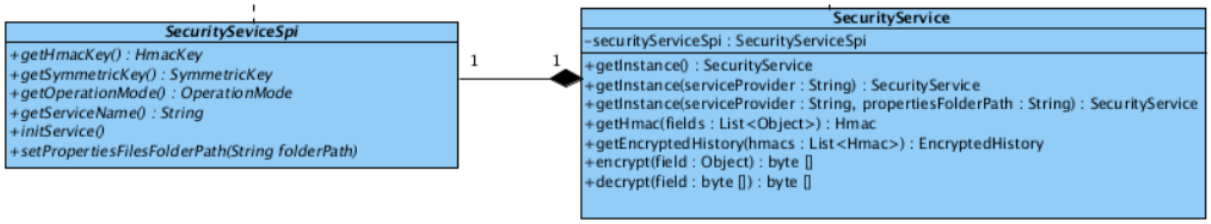

Figura 1: Representação do *provider* da biblioteca

5.2 Provider

5.2.1 Remote Callback

uma subseção para isso?

5.3 PKCS#12 Manager

Ou a seção vai com um nome diferente?

5.4 KeyStore Manager

6 Conclusão

TODO

6.1 Trabalhos Futuros

TODO

Referências

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.

DAMIANI, E. et al. Balancing confidentiality and efficiency in untrusted relational DBMSs. In: *Proc. of the 10th ACM Conference on Computer and Communications Security*. Washington, DC, USA: [s.n.], 2003.

DIFFIE, B. W.; HELLMAN, M. E. New directions in criptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976.

HACIGÜMÜS, H.; IYER, B. R.; MEHROTRA, S. Efficient execution of aggregation queries over encrypted relational databases. In: *DASFAA*. [S.l.: s.n.], 2004. p. 125–136.

HOUSLEY, R.; POLK, T. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. [S.l.]: Wiley Computer Publishing, 2001.

KAMEL, I. A schema for protecting the integrity of databases. *Computers & Security*, Elsevier Ltd, v. 28, n. 7, p. 698–709, out. 2009. ISSN 01674048. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0167404809000297>.

LUICIANO, D.; PRICHETT, G. From caesar ciphers to public-key cryptosystems. *The College Mathematics Journal*, v. 18, n. 1, p. 2–17, January 1987.

SAMARATI, P.; VIMERCATI, S. D. C. di. Data protection in outsourcing scenarios: issues and directions. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. New York, NY, USA: ACM, 2010. (ASIACCS '10), p. 1–14. ISBN 978-1-60558-936-7. Disponível em: <http://doi.acm.org/10.1145/1755688.1755690>.

XIE, M.; WANG, H.; YIN, J. Integrity auditing of outsourced data. *Very large data bases*, p. 782–793, 2007. Disponível em: <http://dl.acm.org/citation.cfm?id=1325940>.