UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Auditoria de uma Protocolizadora Digital de Documentos Eletrônicos

Cleyton André Pires
Marcos Aurélio Dias

Florianópolis, fevereiro de 2003.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Auditoria de uma Protocolizadora Digital de Documentos Eletrônicos

Trabalho de conclusão de curso de graduação apresentado à Universidade Federal de Santa Catarina para a obtenção do grau de Bacharel em Ciências da Computação.

Autor(es):
Cleyton André Pires
Marcos Aurélio Dias
Orientadora:
Vanessa Costa
Co-orientador:
Dr. Ricardo Felipe Custódio
Banca examinadora:
M.Sc. Marcelo Brocardo
Denise Bendo Demétrio

Florianópolis, fevereiro de 2003.

Aos nossos pais pelo apoio e incentivo durante essa longa etapa da vida, e aos nossos amigos por compartilharem conosco este momento.

"Reunir-se é um começo. Manter-se unido é um progresso. Trabalhar unido é um sucesso"

Henry Ford

SUMÁRIO

RESUMO

Esta pesquisa apresenta o modelo e a implementação de um sistema de auditoria do processo de protocolização digital de documentos eletrônicos.

Existem vários métodos para datação de documentos eletrônicos, como a datação por encadeamento linear, método da árvore sincronizada, entre outros. Seus objetivos são aumentar a confiabilidade da Protocolizadora Digital de Documentos Eletrônicos (PDDE).

A PDDE é formada por um cliente e um servidor. O processo de datação consiste das seguintes etapas: o cliente emite o resumo (*hash*) de um documento para o servidor que irá, então, datar o documento e devolver um recibo, assinado digitalmente, para o cliente contendo a data/hora do momento da protocolização anexada ao hash enviado pelo cliente.

Uma PDDE simples foi implementada para auxiliar no desenvolvimento de um sistema de auditoria sobre a mesma, sendo este o principal objetivo deste trabalho. O sistema de auditoria serve para verificar a confiabilidade da PDDE e/ou para resolver a disputa entre dois ou mais recibos identificando qual foi protocolado primeiro.

Palavras-chave: criptografia, *hash*, protocolizadora, auditoria, documentos eletrônicos.

ABSTRACT

This research presents the model and implementation of an auditing system of the digital time stamping process of electronics documents.

There are many stamping methods for electronics documents, as link, synchronized tree, more others. Their objectives are increasing Documents Electronics Digital Timestamper (PDDE) trustworthiness.

The PDDE is composed of a client and a server. The stamping process has the following stages: a client sends a hash of a document to a server that will date the document and return a receipt, digitally signed, to the client having the current date and time of the timestamp moment attached to the hash sent by client.

A simple PDDE was implemented in order to support the auditing system development, reaching the main purpose of this research. The auditing system attends to verify the PDDE trustworthiness and/or to solve a dispute among two or more receipts identifying which one was timestamped first.

Keywords: cryptography, timestamper, auditing, electronics documents

Lista de figuras

Lista de Siglas

AC - Autoridade de Certificação

AD - Autoridade de Datação

CUT - Coordinated Universal Time

DES - Data Encryption Standard

DSS - Digital Signature Standard

GMT- Greenwich Mean Time

IAT - International Atomic Time

MD5 - Algoritmo que calcula o resumo de um arquivo digital qualquer

PDDE – Protocolizadora Digital de Documentos Eletrônicos

RSA - Padrão de cifragem assimétrica

SHA1 - Algoritmo que calcula o resumo de um arquivo digital qualquer

1 INTRODUÇÃO

O uso de documentos eletrônicos está se tornando cada vez mais freqüente, apresentando uma série de vantagens em relação aos documentos em papel, tais como comunicação mais rápida e segura, economia de recursos e menor ocupação de espaço físico.

De forma a garantir a validade jurídica de tais documentos, técnicas devem ser desenvolvidas para simular os processos de autenticação utilizados no papel, ou seja, o documento eletrônico também deve ser assinado (forma de relacionamento por um código entre o conteúdo e a pessoa que o assinou).

Além de assinado, um documento eletrônico deve ser datado. A datação consiste em garantir a existência de um documento em uma determinada data e hora [LIP 99, MAS 99]. Um documento eletrônico assinado e datado possui validade legal.

A datação de documentos em papel é feita através de um carimbo que contenha data e hora confiável. Já a datação de documentos eletrônicos é realizada através de uma PDDE. Neste processo, o serviço de datação recebe o resumo (*hash*) do documento a ser datado, realiza um cálculo único referente a este documento, anexa data e hora no resultado do cálculo e o remete para o cliente [PAS 01]. A esta informação resultante dá-se o nome de recibo.

Existem dois tipos de datação: absoluta e relativa. A datação absoluta contém informações de data e hora reais, ou seja, exatamente como se é usado no mundo real. No caso da datação relativa, ela apenas indica se um documento foi datado antes ou depois de um outro [ROO 99]. A vantagem deste último tipo de datação é que não é necessário confiar cegamente na AD, já que existem mecanismos, ou métodos, que garantem que AD não possa ser maliciosa. Somente a datação relativa será abordada nesta pesquisa. Uma PDDE pode usar datação absoluta, datação relativa ou ambas.

Para garantir a validade dos documentos datados por uma PDDE, esta deve ser confiável e segura. Com base neste fato, este trabalho visa a pesquisa e implementação de um módulo de auditoria de uma PDDE.

O grande desafio na área de protocolação digital de documentos eletrônicos é desenvolver métodos de protocolação que impossibilitem, ou ao menos dificultem ao máximo, fraudes ou erros por parte da Autoridade de Datação ao se datar um determinado documento, já que este deverá ser válido juridicamente. Em 1991 Haber publicou um artigo no qual apresentava o método do encadeamento linear [HAB 91] que reduzia a necessidade de confiança na AD. A

partir daí vários métodos foram desenvolvidos ao longo dos anos, como é o caso do método da árvore sincronizada [PAS 01].

Devido à possível importância legal dos recibos assegurados pela AD devem existir mecanismos para efetuar algum tipo de auditoria no sistema. Até o momento não se tem conhecimento de trabalhos que tratem especificamente sobre auditoria de PDDE, fato este que enaltece a importância desta pesquisa.

Auditoria de PDDE é o tema de pesquisa da mestranda Vanessa Costa e deverá ser apresentada como dissertação de mestrado do curso de Ciências da Computação da Universidade Federal de Santa Catarina.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral é estudar, modelar e implementar um sistema de auditoria de uma PDDE absoluta-relativa.

1.1.2 Objetivos específicos

Este trabalho tem como objetivos específicos:

- Estudar as técnicas de criptografia;
- Estudar os métodos de datação eletrônica;
- Estudar forma de auditoria da PDDE;
- Implementar uma PDDE simples;
- Implementar um sistema de auditoria sobre a PDDE.

1.2 Conteúdo do documento

No capítulo dois será apresentado uma breve descrição das técnicas e conceitos de criptografia (este capítulo pode ser ignorado por aqueles que já possuem noções básicas de segurança computacional). No capítulo 3 serão apresentadas algumas das técnicas de datação

mais relevantes contidas na literatura para se ter uma visão geral do que se têm hoje em termos de datação digital. Já o capítulo 4 descreve a modelagem e implementação de uma PDDE simples que utiliza o método de encadeamento linear. Finalmente, o capítulo 5 trata da implementação do sistema de auditoria da PDDE.

O anexo A contém o código fonte da PDDE simples e o anexo B contém o código fonte do sistema de auditoria sobre a PDDE.

2 Fundamentos da criptografia

2.1 Introdução

Com o crescente uso das redes de computadores por organizações para conduzir seus negócios e a massificação do uso da Internet, surgiu à necessidade de se utilizar melhores mecanismos para prover a segurança das transações de informações confidenciais.

A Segurança em Informática consiste na certeza de que as informações de uso restrito não devem ser acessadas, copiadas ou codificadas por pessoas não autorizadas. Para a garantia disto, é necessário que as informações sejam cifradas.

Uma das maneiras de se evitar o acesso indevido a informações confidenciais é através da codificação ou cifragem da informação, conhecida como criptografia, fazendo com que apenas as pessoas às quais estas informações são destinadas, consigam compreendê-las. A criptografia fornece técnicas para codificar e decodificar dados, tais que os mesmos possam ser armazenados, transmitidos e recuperados sem sua alteração ou exposição. Seu principal objetivo é prover uma comunicação segura, garantindo serviços básicos de autenticação, privacidade e integridade dos dados.

A palavra criptografia tem origem grega (kriptos = escondido, oculto e grifo = grafia) e define a arte ou ciência de escrever em cifras ou em códigos, utilizando um conjunto de técnicas que torna uma mensagem incompreensível, chamada comumente de texto cifrado, através de um processo chamado cifragem, permitindo que apenas o destinatário desejado consiga decodificar e ler a mensagem com clareza, no processo inverso, a decifragem.

A criptografia pode ser classificada em duas categorias básicas, de acordo com o tipo de chave utilizada - sistema de chave simétrica (ou chave secreta), que tem como principal padrão o DES (Data Encryption Standard), e sistema de chave assimétrica (ou chave pública), que tem como principal padrão o RSA.

Para **cifrar** um texto, utiliza-se uma **chave** ou **senha** com um determinado número de bits e um algoritmo parametrizado por esta senha responsável pelo embaralhamento do texto. Quanto maior o número de bits da chave, maior é o **espaço de chaves**. Seja *b* o número de bits da chave, então o espaço de chaves é 2^b. O **ataque de força bruta** consiste em verificar todas as possíveis chaves de forma a encontrar a chave que foi utilizada para cifrar uma determinada mensagem. Desta forma, quanto maior o número de bits da chave, maior será o

esforço computacional para se descobrir a chave utilizada. Por outro lado, quanto maior for o tamanho da chave maior será o esforço computacional do processo de cifragem/decifragem.

Alguns conceitos e terminologias básicas devem ser compreendidos, segue uma lista deles:

- **Texto original** - informação sigilosa que se deseja proteger ou assinar;
- **Texto cifrado** – informação sigilosa já embaralhada, depois de ter passado por um processo de cifragem;
- **Cifragem** - processo de embaralhamento da informação;
- **Decifragem** - processo de desembaralhamento da informação;
- **Cifrador** - algoritmo responsável por cifrar a informação (texto original) com um outro objeto (chave), com a intenção de obter uma saída incompreensível (texto cifrado);
- **Decifrador** - algoritmo que realiza o processo inverso do cifrador;
- **Chave** – informação que é utilizada como parâmetro do algoritmo de cifragem, assim como o texto original.

Existem dois tipos de chaves: chaves simétricas, onde a mesma chave é utilizada para cifrar e decifrar a mensagem, e chaves assimétricas, caso em que existe uma chave para cifrar e outra para decifrar.

- **Alice** - entidade emissora
- **Beto** - entidade receptora

Cifrar significa que a mensagem legível (texto original) será transformada em uma mensagem ilegível (texto cifrado). A função de decifrar é o processo inverso da cifragem, a partir de um texto cifrado, obtém-se o texto original. Existem basicamente dois tipos de criptografia: simétrica e assimétrica.

2.2 Criptografia Simétrica

A criptografia simétrica utiliza a mesma chave para cifrar e decifrar uma mensagem. Isso significa que a chave deve ser de conhecimento tanto do emissor como do receptor da mensagem. Normalmente na criptografia simétrica, o algoritmo para cifrar e decifrar é basicamente o mesmo, mudando apenas a forma como é utilizada a chave.

O algoritmo simétrico mais conhecido hoje é o DES (*Data Encryption Standard*), o qual foi por muitos anos o padrão de cifragem simétrica. Após a definição do DES outros algoritmos, seguindo os mesmos princípios de Caixas-S e rodadas, foram surgindo, é o caso do Blowfish, IDEA, CAST128 e o RC5.

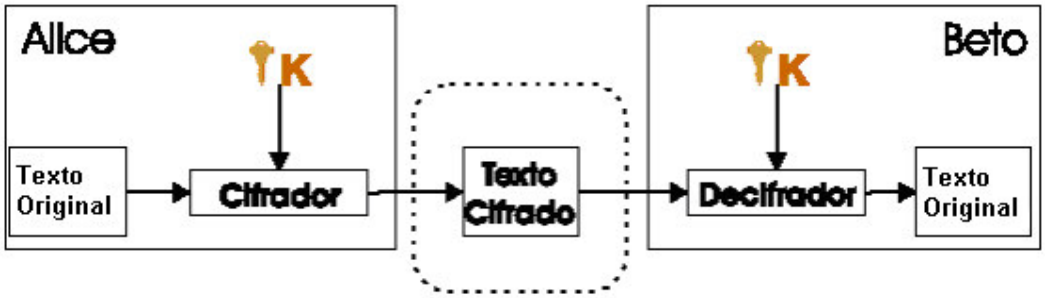

Figura 1 - Criptografia simétrica.
Alice, utilizando um cifrador e uma chave K, cifra o texto original produzindo o texto cifrado. Beto, utilizando o texto cifrado e a mesma chave K, decifra o texto cifrado obtendo o texto original.

O maior problema relacionado a esse algoritmo é com relação à distribuição de chaves. Pois, como é necessário uma chave para cada par emissor-receptor, considerando um grupo com *n* pessoas, seria necessário um número de chaves igual a n(n – 1)/2, que gera um problema de gerenciamento de uma grande quantidade de chaves.

2.3 Criptografia Assimétrica

Na criptografia assimétrica cada usuário possui uma chave pública e uma chave privada. Qualquer uma das chaves pode ser utilizada para cifrar e decifrar. Se o texto foi cifrado com a chave privada, este deverá ser decifrado com a chave pública. Se o texto foi cifrado com a chave pública, este deverá ser decifrado com a chave privada. A chave privada é mantida em segredo, enquanto a chave pública deve ser divulgada.

Em 1977 foi desenvolvido por Ron Rivest, Adi Shamir e Len Adleman o algoritmo que é o mais utilizado atualmente, o RSA.

A criptografia assimétrica pode ser utilizada para garantir a autenticação e confiabilidade. Se Alice cifra seu texto com sua chave privada, apenas sua chave pública poderá

decifrar seu texto, dessa forma fica garantido que se trata de um texto cifrado por Alice, portanto fica garantida a autenticação.

A confiabilidade fica garantida da seguinte forma: se um usuário cifra um texto com a chave pública de Beto, apenas Beto poderá decifrar esse texto, por ser o portador da chave privada.

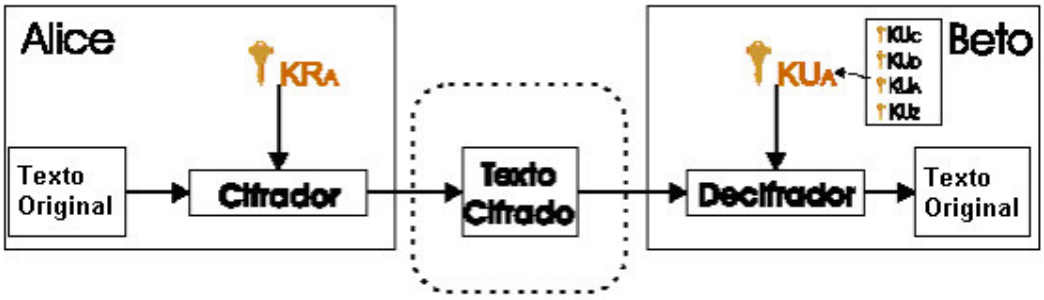

Figura 2 - Criptografia assimétrica no modo assinatura.
Alice, utilizando um cifrador e sua chave privada KR_A, cifra o texto original produzindo o texto cifrado. Beto utilizando o texto cifrado e a chave pública KU_A de Alice contida em seu repositório de chaves, decifra o texto cifrado obtendo o texto original.

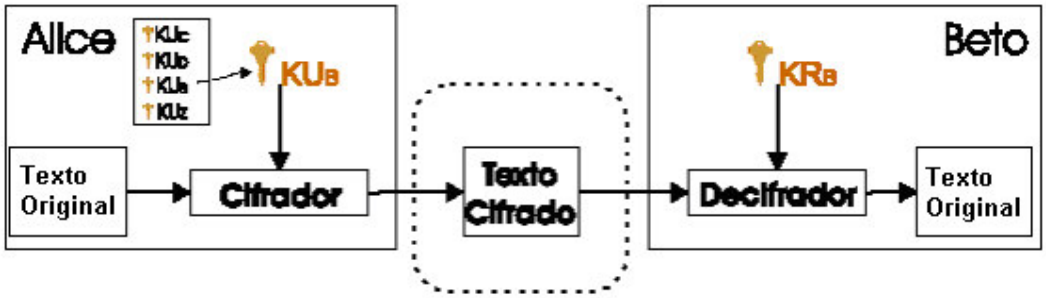

Figura 3 - Criptografia assimétrica no modo sigilo.
Alice utiliza a chave pública de Beto KU_B, contida em seu repositório de chaves, para cifrar o texto. Beto, por sua vez, decifra o texto cifrado com um decifrador utilizando sua chave privada KR_B.

2.4 Resumo (Hash)

Trata-se de uma função que gera um resumo de tamanho fixo de um arquivo qualquer. O objetivo do *hash* é gerar uma identificação (*fingerprint*) para um bloco de dados qualquer. A função *hash* deve ser aplicada a documentos de qualquer tamanho; deve produzir uma saída de tamanho fixo; possui fácil implementação; não é possível determinar o documento apartir de um hash gerado; não devem exister hash iguais.

As funções hash mais utilizadas são:

- MD5 – produz saída de 128 bits;
- SHA1 (*Secure Hash Algorithm*) - produz saída de 160 bits.

A função resumo pode ser utilizada em conjunto com a criptografia assimétrica para se obter uma forma de assinatura digital. Isso pode ser feito enviando-se para Beto a mensagem e o resumo da mensagem cifrada com a chave privada de Alice. Beto, decifra o resumo com a chave pública de Alice, calcula um novo resumo com base na mensagem recebida e compara os dois valores. Se forem iguais, a mensagem não foi alterada, garantindo-se dessa forma a sua integridade.

2.5 Assinatura Digital

Ao contrário da assinatura convencional onde se imprime no papel as bio-características de uma pessoa, a assinatura digital é um código binário que é determinado com base no documento e alguma outra informação que associa este a uma determinada pessoa. Essa associação pode ser feita por algo que se sabe (senha, por exemplo), algo que se possui (cartão magnético) ou algo que se é (medida biométrica da pessoa).

A assinatura digital tem sido implementada das seguintes formas: usando funções resumos através dos algoritmos MD5 e SHA1; utilizando o DSS (*Digital Signature Standard*); utilizando os conceitos de chave pública.

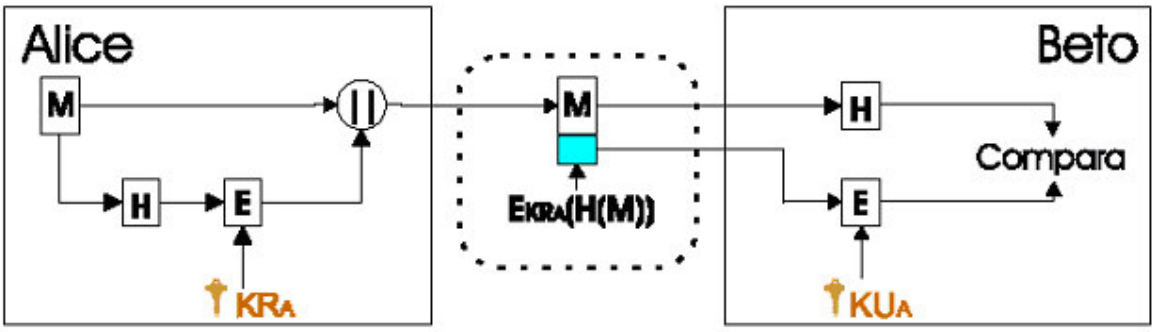

Figura 4 - Assinatura Digital.
Alice calcula o resumo da mensagem M com uma função H e cifra este resumo com sua chave privada KRA. O resultado é então concatenado com a mensagem original e enviado para Beto. Beto por sua vez decifra o pacote recebido com a chave pública KUA de Alice, calcula o resumo da mensagem M recebida e compara os dois resultados. Se os dois forem iguais, Beto tem certeza que a mensagem não foi alterada.

2.6 Conclusão

Para se implementar um sistema de auditoria de uma PDDE é necessário utilizar conceitos de criptografia assimétrica, função resumo e assinatura digital. Desta forma esse estudo preliminar é de fundamental importância para o restante do trabalho.

3 Métodos de datação eletrônica

3.1 Introdução

Método de datação são mecanismos que servem para dificultar a ação maliciosa por parte da PDDE, aumentando sua confiabilidade.

Um bom método de datação deve atender aos seguintes requisitos de segurança:

1. **Privacidade**: Ninguém além do cliente pode ter acesso ao conteúdo do documento;
2. **Facilidade de comunicação e armazenamento**: Deve ser prático datar um documento independentemente do seu tamanho;
3. **Integridade**: Deve-se garantir a integridade dos dados e operação ininterrupta do serviço de datação;
4. **Anonimato**: Deve-se garantir o anonimato do cliente;
5. **Confiança**: Deve-se garantir que o documento será datado com data e hora correta

A privacidade é assegurada porque a AD tem acesso apenas ao resumo e não ao documento original, consegue-se comunicar e armazenar a informação facilmente pois o resumo de um documento é bem menor do que o documento original, e garante-se a integridade, pois o cliente pode verificar a assinatura digital da AD ao receber o recibo.

O requisito anonimato não é assegurado nesta forma de datação, porém ele pode ser visto como uma característica desejável, mas não obrigatória, visto que existem várias situações onde ele não é necessário.

O requisito de confiança pode ser assegurado se o equipamento de datação for lacrado e adotar padrões de segurança física, como por exemplo, os descritos na FIPS-140 [NIST, 2002]. Além disso, ele deve possuir um módulo de auditoria.

Na literatura existem vários métodos de datação de documentos eletrônicos, no entanto, somente os métodos de encadeamento linear e da árvore sincronizada, tais métodos serão apresentados a seguir.

3.2 Método do Encadeamento Linear

Trata-se de um método de datação [HAB 91] no qual a AD encadeia os resumos dos documentos que foram enviados pelos clientes e os armazena em um banco de dados interno. O encadeamento entre os recibos utiliza uma função de sentido único, como por exemplo, uma função *hash*.

O encadeamento é formado por *links*, sendo que o primeiro *link* da cadeia pode ser gerado de maneira randômica, formando o L_0. Em seguida, o segundo link L_1 é gerado utilizando o *link* anterior, no caso, L_0 e o resumo do documento enviado pelo cliente. De forma genérica, temos:

$$Ln = (t_{n-1}, ID_{n-1}, H_{n-1}, H(L_{n-1})) \qquad (1)$$

onde t_{n-1} é a data e hora em que o documento anterior foi datado, ID_{n-1} é um identificador do documento anterior, H_{n-1} é o resumo do documento anterior e $H(L_{n-1})$ é resumo do link anterior.

Após o cálculo do link, a AD gera o recibo que será enviado para o cliente que será:

$$s = SigAD(ID_n, t_n, ID_n, H_n, L_n) \qquad (2)$$

Desta forma, os resumos dos documentos que foram enviados pelos clientes ficam ordenados obedecendo à ordem de chegada.

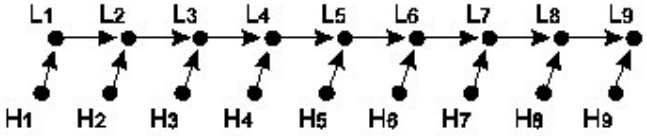

Figura 5 - Método do encadeamento linear

3.3 Método da Árvore Sincronizada

O método da árvore sincronizada também utiliza um esquema de datação relativo e, portanto, existe um encadeamento dos documentos em uma ordem temporal. No entanto, ele utiliza um conceito de rodadas, que tem como objetivo diminuir o tempo de busca [PAS 01]. Este método resolve o principal problema do método do encadeamento linear, o tempo gasto na

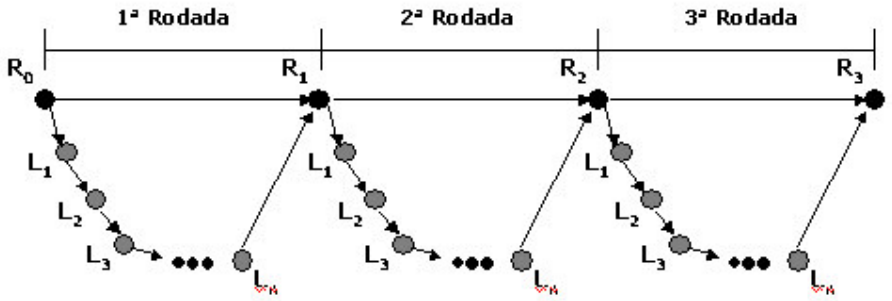

Figura 6 - Esquema da Árvore Sincronizada.

Ao receber um resumo para ser datado, a AD o encadeia com o recibo anterior Li, através de uma função F. Ao final da rodada, a AD encadeia o último Li com o recibo da rodada R_{r-1} da rodada anterior, gerando o R_{ratual}

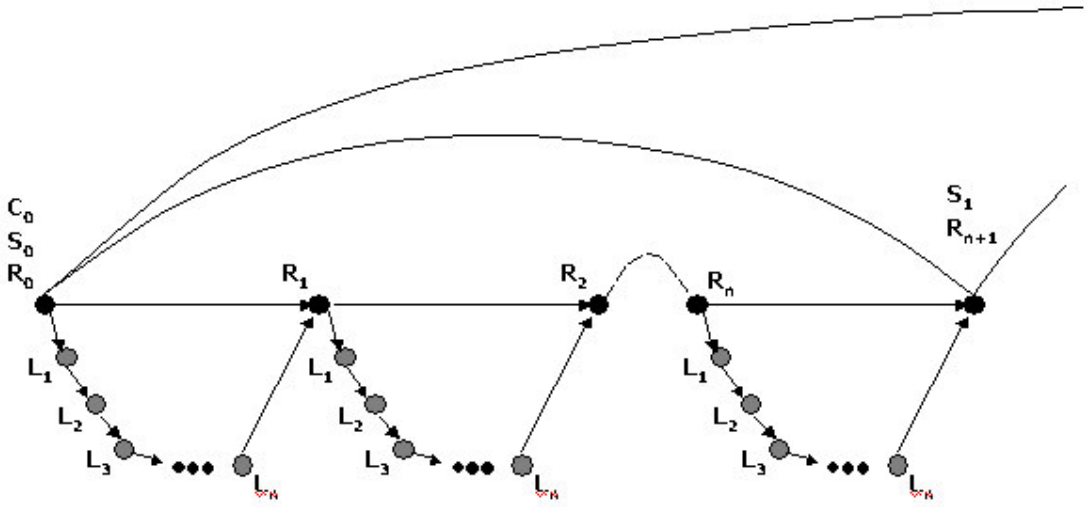

Figura 7 - Generalização do Método da Árvore Sincronizada.

Como o método utiliza pontos de sincronismo, pode-se definir políticas de encadeamentos maiores do que uma rodada, criando um ambiente onde o verificador possa buscar uma determinada informação com muito mais agilidade.

verificação do encadeamento que é diretamente proporcional ao número de recibos encadeados.

Antes de prosseguir, é necessário definir alguns conceitos que são utilizados neste método:

- Rodada: Período de tempo bem definido ou quantidade máxima de solicitações de protocolização.
- Ponto de Confiança: Ponto onde ocorreu a última publicação de um recibo que representa todos os encadeamentos desde o último ponto de confiança. Após a publicação, todos os encadeamentos podem ser exportados para um meio de armazenamento externo, para uma eventual auditoria, e a base de dados da AD passa a ter apenas o ponto de confiança como o primeiro link do encadeamento. Os pontos de confiança são representados como Ci.
- Ponto de Sincronismo: Ponto que representa todas as rodadas presentes em um determinado tempo. Os pontos de sincronismo são representados por Si.
- Recibo: Contém os encadeamentos de sua rodada, bem como as informações das rodadas dos pontos de sincronismo anteriores até o último ponto de confiança publicado.

Todos os resumos que chegam para ser datados são encadeados, e ao final de um determinado tempo, é aplicada uma função F em todos os resumos para gerar um único recibo que os represente.

A figura 6 descreve um exemplo de encadeamento da árvore sincronizada com três rodadas.

Para diminuir o tempo de busca por um recibo no encadeamento, saltos podem ser realizados. Um salto gera um ponto de sincronismo que representa todos os recibos que pertencem ao intervalo a ser saltado.

A figura 7 descreve exemplos de saltos que podem ser realizados.

3.4 Conclusão

Este capítulo apresentou de forma didática dois métodos de datação: encadeamento linear e árvore sincronizada. O primeiro foi a primeira técnica a ser desenvolvida para evitar a necessidade da confiança total na AD. Tem a vantagem de ser um método simples, porém possui um tempo elevado de verificação da cadeia de encadeamento (diretamente proporcional ao tamanho da cadeia). Já o segundo método, árvore sincronizada, é uma evolução do primeiro no sentido da melhoria de sua eficiência, já que permite dar saltos dentro da cadeia de encadeamento agilizando o processo.

4 Um modelo simplificado de PDDE

4.1 Introdução

O sistema proposto a seguir é na realidade apenas uma parte daquilo que seria uma PDDE real. Uma PDDE para ser completa exige ainda uma série de mecanismos de segurança que estão fora do escopo deste trabalho. Uma PDDE consiste, além do software que realiza a protocolação em si (o qual será apresentado no decorrer deste capítulo), de outras entidades como:

- Hardware específico (figura 8);
- Servidor de tempo confiável: responsável por fornecer data e hora confiáveis à PDDE;
- Mecanismos de autodestruição de seu certificado digital em caso de violação física (tentativa de acesso interno à PDDE) e/ou lógica (tentativa de ataque através de interfaces de comunicação).

Figura 8 - Imagem da Bry PDDE.

4.2 Arquitetura do sistema

O sistema consiste de duas partes:

1. Cliente: Software que roda em ambiente Windows e que permite ao usuário escolher um arquivo a ser protocolado e o servidor em que este irá protocolar o documento virtual.
2. Servidor: roda em ambiente UNIX e é responsável por atender as requisições dos clientes.

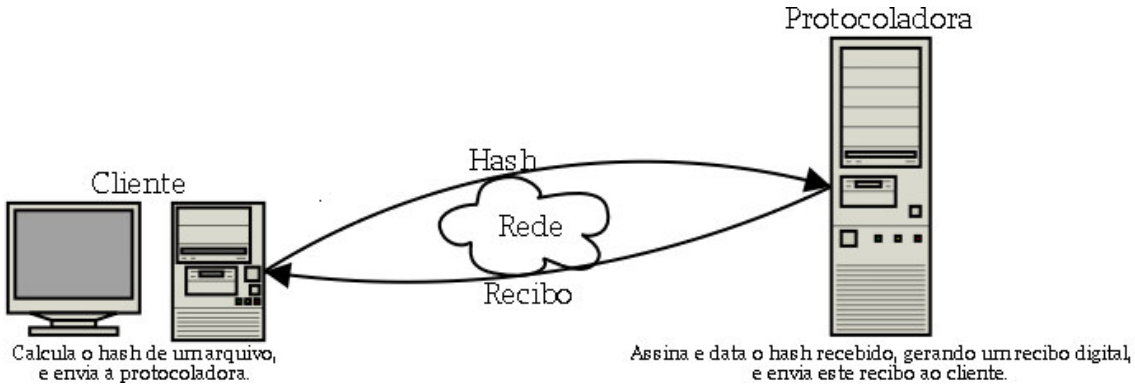

Figura 9 - Processo de protocolação.

Ambos serão descritos com mais detalhes a seguir.

4.2.1 Cliente

Foi criado um programa cliente escrito em C++ utilizando o Visual C++ 6.0 como ambiente de programação, devido principalmente ao fato de sua confiabilidade e no desejo de aperfeiçoar os conhecimentos nesta ferramenta.

O cliente tem por objetivo principal enviar os documentos a serem protocolados para o servidor. Para isso o cliente escolhe o documento a ser protocolado, gera o resumo do mesmo, conecta ao servidor e envia o resumo. Então recebe o recibo do servidor e salva localmente o mesmo com o nome desejado pelo usuário.

a) Estrutura de classes

A figura 10 descreve o diagrama de classes do programa. As classes *Socket* e *ClientSocket*, responsáveis pelo estabelecimento de uma conexão entre cliente e servidor, são as mesmas utilizadas no programa servidor.

A classe *Crypto* fornece funções relacionadas à criptografia, como o cálculo da função resumo e a verificação da assinatura digital. Para tanto utiliza a biblioteca *CryptoAPI* do Windows que é equivalente à *OpenSSL* do Linux, utilizada no lado do servidor.

A classe *Protocol* é responsável pelo processamento de envio e recebimento do recibo, enquanto a classe *ProtocoladoraDlg* serve como interface do sistema e é responsável pela ativação dos métodos da classe *Protocol*.

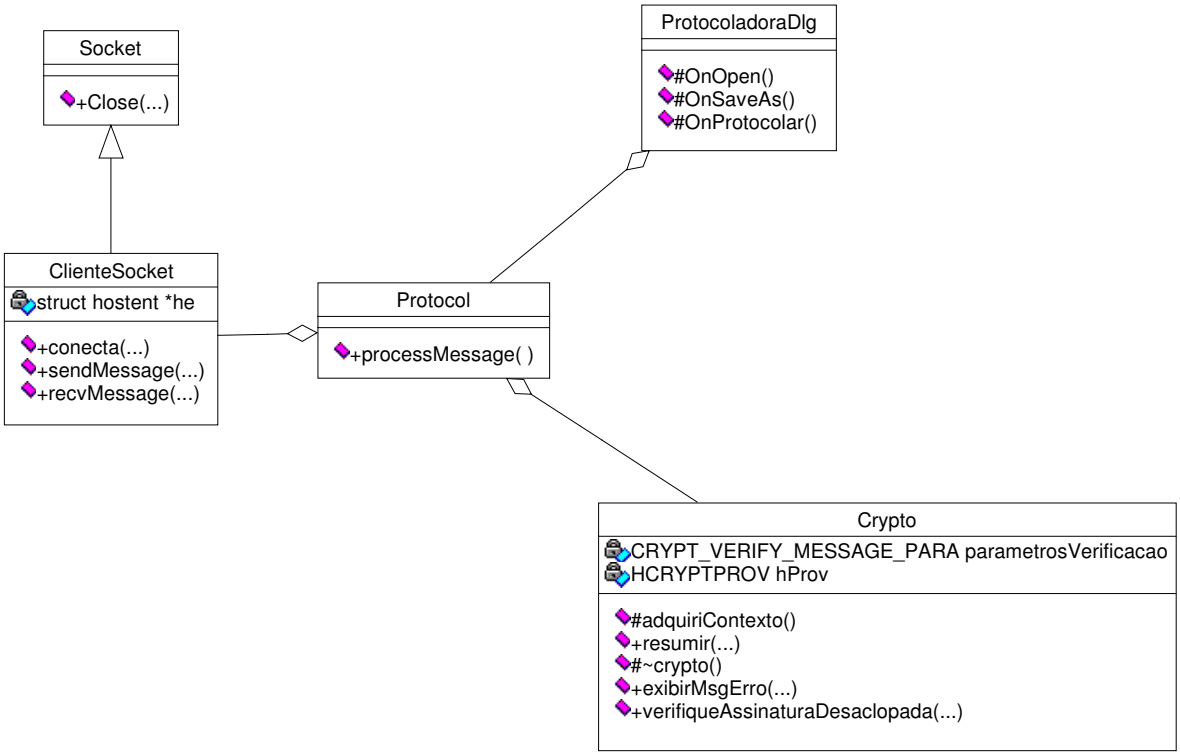

Figura 10 – Diagrama de classes do cliente PDDE.

b) Demonstração do sistema cliente

Abaixo é mostrada a interface do software cliente:

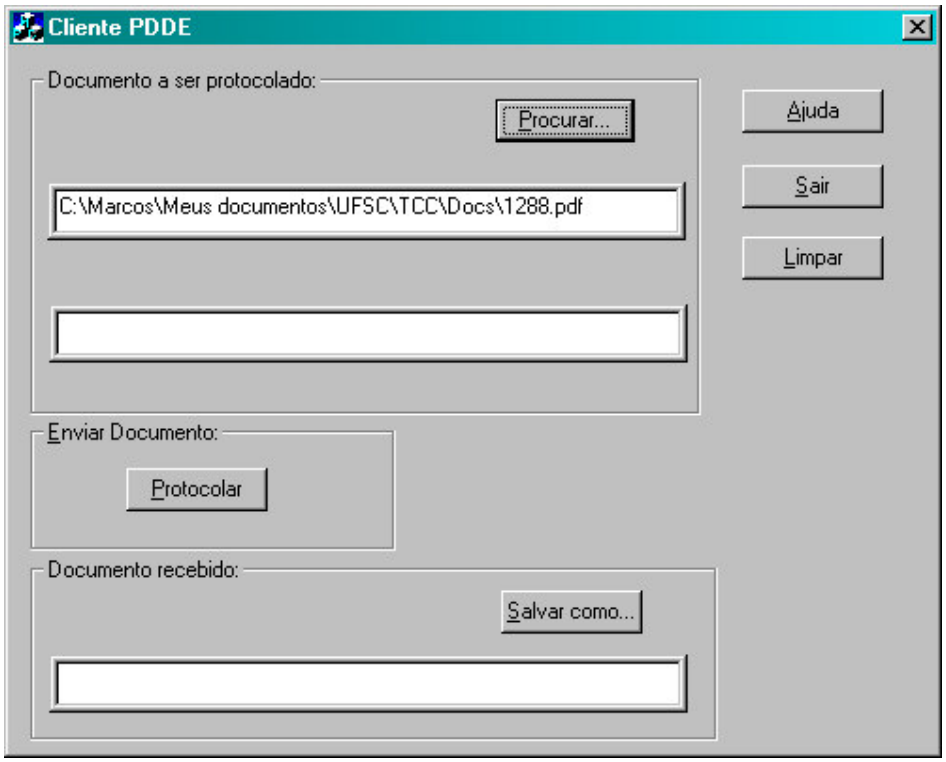

Figura 11 - Software cliente PDDE.

4.2.2 Servidor

Esta, que é a principal parte do sistema, foi escrita em C++ (compilador g++) para rodar em máquinas UNIX/Linux. Para estas escolhas foram levados em conta a confiabilidade e performance do sistema.

A figura abaixo mostra, de forma simplificada, todas as operações que são realizadas pela PDDE desde a recepção de uma requisição de datação até a emissão do recibo para o cliente.

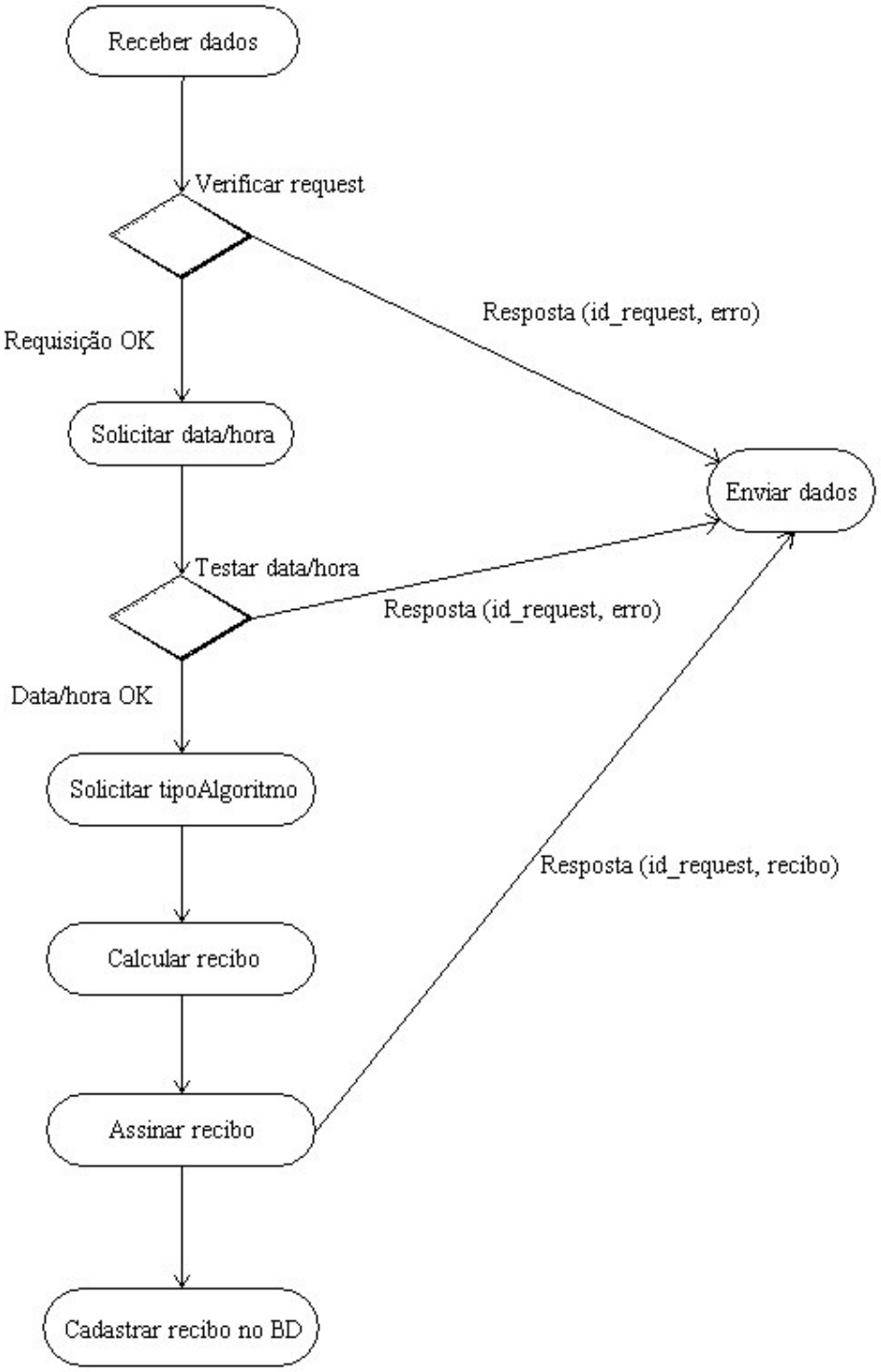

Figura 12 - Fluxo de dados da PDDE.

O sistema foi projetado de forma a aceitar várias protocolações ao mesmo tempo, utilizando-se do conceito de *multithreads*. A cada cliente que se conecta ao servidor, uma *thread* é criada para tratar exclusivamente desta conexão, enquanto outra fica "ouvindo" por novas.

Um banco de dados é utilizado para guardar as informações sobre o sistema, deixando-o mais flexível e facilitando a manutenção. O banco de dados da empresa Borland Interbase foi escolhido para esta tarefa por ser um banco de dados *freeware* e por ter boa aceitação no mercado.

O algoritmo de protocolação suportado é o encadeamento linear.

A PDDE gera assinatura digital no padrão PKCS#7 que descreve uma sintaxe para dados que utilizam criptografia, como assinaturas digitais. Para tanto, utiliza um certificado digital no padrão X.509 emitido pelo LabSec (Laboratório de Segurança em Computação da UFSC).

Quando um documento é assinado segundo o padrão de assinatura PKCS#7, algumas informações devem ser anexadas ao documento: o resumo do documento (*hash*) cifrado com a chave privada do usuário e o certificado digital do usuário. Neste caso em particular, a PDDE seria o usuário. A figura a seguir mostra a assinatura digital neste formato.

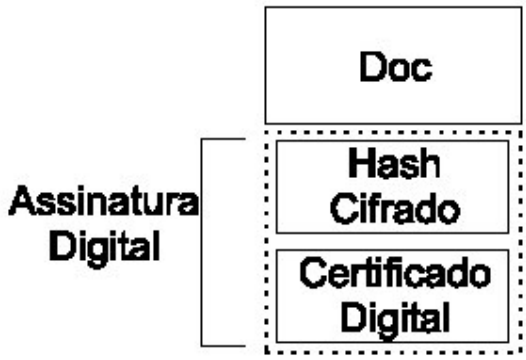

Figura 13 - Assinatura digital no formato PKCS#7.

a) Estrutura de classes

A estrutura de classe foi pensada, principalmente, de forma a tornar o sistema robusto e escalável. Além disso, procurou-se desenvolver módulos, ou classes, que pudessem ser facilmente reutilizados, fato este que foi alcançado com o reuso da classe *Database* no sistema de auditoria descrito no capítulo 6. Outras classes como a classe *Thread* também pode ser facilmente reutilizada em outros sistemas que necessitem de operações *multithreads*, por exemplo. Outro bom exemplo é a classe *Crypto* que, através de chamadas a métodos simples,

encapsula funções da biblioteca OPENSSL do Linux que são de difícil utilização, como a assinatura de uma mensagem no formato PKCS#7. A figura a seguir mostra o diagrama de classes do servidor PDDE.

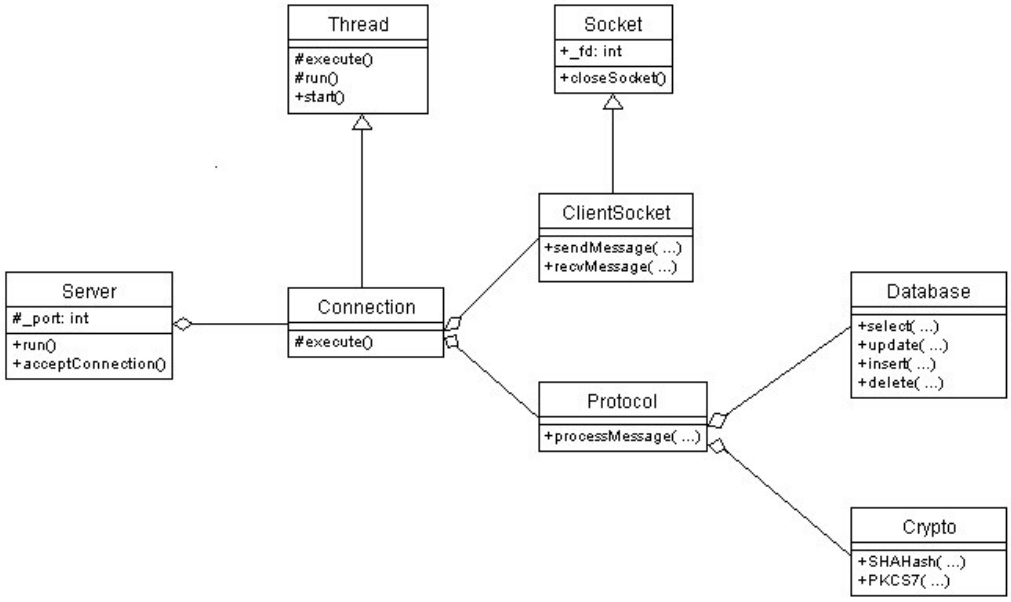

Figura 14 – Diagrama de classes do servidor PDDE.

b) Descrição das classes

A seguir são apresentadas as descrições de cada classe do sistema:

- **Thread** – Esta classe permite a criação de uma nova *thread* de execução no programa, bastando que se tenha uma classe que seja subclasse desta. Esta subclasse deve sobrescrever o método run da classe Thread.
- **Socket** – contém informações básicas que permitem o estabelecimento de uma comunicação fim-a-fim entre duas máquinas.
- **ClientSocket** – Especialização da classe Socket que atua como o cliente na comunicação entre um par de máquinas.
- **Server** – Especialização da classe Socket que atua como o servidor na comunicação entre um par de máquinas. Sua função é esperar por requisições provenientes da rede.
- **Connection** – Subclasse de Thread que serve para tratar com uma conexão específica, deixando livre o servidor (classe Server) para esperar por outras requisições da rede, enquanto um documento é protocolado.

- **Protocol** – Classe responsável por realizar o processo de protocolação de um documento.
- **Database** – Classe com funções que agrupam e simplificam os métodos para realizar operações sobre banco de dados.
- **Crypto** – Possui métodos relacionados à segurança do processo de protocolação (criptografia, hash, certificados digitais, etc). Para tanto utiliza a biblioteca OpenSSL do Linux.

c) Banco de dados

O servidor utiliza um banco de dados para guardar informações referentes às protocolações dos documentos. Para este propósito foi utilizado o Interbase por ser um banco de dados *freeware* e largamente utilizado, fato que comprova sua qualidade.

A seguir estão descritas as tabelas usadas:

- **Tabela config**

Esta tabela contém informações de configuração do sistema, como por exemplo a localização (diretório e nome do arquivo) do certificado e da chave pública da PDDE, o tipo de algoritmo de protocolação que está sendo utilizado (link, árvore sincronizada, etc), entre outros.

```
SQL> show table config;
COD_CONFIG                      INTEGER Not Null
RECIBO                          INTEGER Not Null
TIPOALG                         INTEGER Not Null
MOD_CONFIANCA                   INTEGER Not Null
PATH_CERT                       VARCHAR(50) Not Null
PATH_KEY                        VARCHAR(50) Not Null
PASS_PHRASE                     VARCHAR(20) Not Null
CONSTRAINT CONFIGPRIMARYKEY:
  Primary key (COD_CONFIG)

Triggers on Table CONFIG:
CONFIGTRIGGER, Sequence: 0, Type: BEFORE INSERT, Active
```

- **Tabela log**

Responsável por guardar informações de log do sistema. Qualquer erro que acontece no sistema é registrado nesta tabela.

```
SQL> show table log;
COD_LOG                         INTEGER Not Null
```

```
DATA                            INTEGER Not Null
ID_REQUEST                      INTEGER Nullable
DESCRICAO                       VARCHAR(100) Not Null
CONSTRAINT LOGPRIMARYKEY:
  Primary key (COD_LOG)

Triggers on Table LOG:
LOGTRIGGER, Sequence: 0, Type: BEFORE INSERT, Active
```

- **Tabela tipoalgoritmo**

  Contém informações sobre os algoritmos de protocolação suportados pela PDDE.

```
SQL> show table tipoalgoritmo;
COD_TIPO                        INTEGER Not Null
ALGORITMO                       VARCHAR(50) Nullable
TABELA                          VARCHAR(10) Not Null
CONSTRAINT TIPOALGORITMOPRIMARYKEY:
  Primary key (COD_TIPO)

Triggers on Table TIPOALGORITMO:
TIPOALGTRIGGER, Sequence: 0, Type: BEFORE INSERT, Active
```

- **Tabela link**

  Guarda informações a respeito dos *links* que compõem o encadeamento linear.

```
SQL> show table link;
COD_LINK                        INTEGER Not Null
RECIBO                          INTEGER Not Null
DATA                            INTEGER Not Null
HASH                            VARCHAR(200) Not Null
LINK                            VARCHAR(200) Not Null
CONSTRAINT LINKPRIMARYKEY:
  Primary key (COD_LINK)

Triggers on Table LINK:
LINKTRIGGER, Sequence: 0, Type: BEFORE INSERT, Active
```

- **Tabela requisição**

  Esta tabela guarda informações sobre cada requisição de protocolação que é feita.

```
SQL> show table requisicao;
COD_REQ                         INTEGER Not Null
RECIBO                          INTEGER Not Null
DATASIG                         INTEGER Not Null
HASH                            VARCHAR(200) Not Null
ID_REQUEST                      DOUBLE PRECISION Not Null
CONSTRAINT REQUISICAOPRIMARYKEY:
  Primary key (COD_REQ)

Triggers on Table REQUISICAO:
REQUISICAOTRIGGER, Sequence: 0, Type: BEFORE INSERT, Active
```

4.3 Conclusão

Para se realizar uma auditoria de um determinado sistema é necessário um profundo conhecimento sobre o mesmo, de modo que seja possível identificar indícios de mau funcionamento, como falhas de segurança ou mesmo erros de projeto, no sistema auditado.

A implementação desta PDDE foi importante para este entendimento e será útil na concepção de um sistema de auditoria sobre a mesma.

5 Auditoria sobre a PDDE

5.1 Introdução

Devido a possível importância legal dos recibos emitidos pela PDDE, devem existir mecanismos para efetuar algum tipo de auditoria no sistema. Um modo fácil de se fazer isto é periodicamente verificar de forma aleatória alguns recibos da PDDE. Se eles estiverem encadeados inconsistentemente, então a PDDE agiu maliciosamente e não pode ser considerada confiável.

Define-se auditoria como um conjunto de técnicas de observações e exames, aplicados de forma sistemática, que no contexto do auditado, visa opinar sobre sua situação, sobre sua riqueza, quando este for o caso, ou sobre funções ou áreas específicas componentes do patrimônio do auditado.

Auditoria de sistemas é o ramo da auditoria que revisa e avalia os controles internos informatizados e que tem como objetivos:

- **Verificar a eficiência**: Verifica-se a utilização de recursos de computação alocados ao sistema.
- **Constatar eficácia**: Compreende a validação dos resultados gerados pelos sistemas.
- **Atestar a segurança**:
    1. *Segurança física*: instalações, equipamentos, suprimentos, documentação, dados, pessoal, entre outros.
    2. *Segurança lógica*: Sistemas e informações.
    3. *Segurança em comunicação*: veiculação dos dados por meios de comunicação.

A auditoria de sistemas adotada na PDDE tem como objetivo atestar a segurança e consiste em:

1. Verificar a validade de um recibo;
2. Determinar a seqüência entre dois recibos (quem protocolou primeiro);
3. Analisar se existe ramo malicioso no método de encadeamento;

Os detalhes de funcionamento da auditoria devem estar previstos nas Diretrizes de Práticas de Protocolizações (DPP). A auditoria pode ser realizada de duas maneiras:

- **Auditoria externa**: Utiliza somente as informações contidas nos recibos e nas informações publicadas pela AD em um diretório público;
- **Auditoria interna**: utiliza informações internas armazenadas na AD como os logs e o próprio encadeamento. Devido a isto, a AD deve ser protegida com lacres e deve estar em um lugar seguro para que ninguém não autorizado tenha acesso a máquina [PAS 01].

A auditoria tem por objetivo verificar:

1. *Integridade*: através da verificação da assinatura digital da PDDE.
2. *Irretroatividade*: possibilidade de verificar uma eventual alteração da data em que um documento foi protocolado.

É importante ressaltar que, para cada método de datação, deve existir um método de auditoria correspondente capaz de verificar a confiabilidade de uma determinada PDDE.

A idéia deste trabalho é implementar um método de auditoria para uma PDDE. Métodos de auditoria estão sendo desenvolvidos por Vanessa Costa [COS 02].

5.2 Auditoria para o método de encadeamento linear

A seguir é descrito quais são os procedimentos necessários para realizar os três tipos de verificações na auditoria para o método do encadeamento linear:

1. *Verificar a validade de um recibo*: Para verificar se um determinado documento foi protocolado por uma PDDE específica, deve-se percorrer os *links* do encadeamento e verificar se o resumo do documento está presente na lista. Note que devido ao fato de que o tempo de busca no método do encadeamento linear ser diretamente proporcional ao número de protocolizações, se o encadeamento na base de dados interna da PDDE for muito grande, este tipo de verificação pode ser muito lenta.
2. *Resolver disputa entre dois recibos*: Para verificar qual entre dois documentos foi protocolado primeiro em uma PDDE, é necessário percorrer o encadeamento e verificar em qual posição o primeiro documento se encontra. Em seguida, deve-se percorrer o encadeamento novamente e procurar em que posição o segundo

documento se encontra. De posse das posições em que os documentos foram protocolizados, respectivamente, basta compará-las para saber qual deles foi protocolado primeiro.

3. *Verificar a autenticidade do Banco de dados*: Para verificar se existem ramos maliciosos na AD, deve-se calcular os *links* a partir do ponto de confiança até o final do encadeamento e verificar se os *links* calculados são equivalentes aos *links* constantes nos recibos.

5.3 Auditoria para o método da árvore sincronizada

A seguir é descrito quais são os procedimentos necessários para realizar os três tipos de verificações na auditoria para o método da árvore sincronizada:

1. *Verificar a validade de um recibo:* é feita através do recálculo das informações constantes no recibo, checagem do Ponto de Confiança C_0, verificação da assinatura digital da PDDE e consulta a Lista de Certificados Revogados;
2. *Resolver disputa entre dois recibos para saber qual protocolou primeiro*: Este tipo de auditoria leva em consideração a rodada a que pertencem os recibos. Se os dois recibos estão na mesma rodada, a verificação é direta, basta voltar até o último ponto de confiança comum aos dois documentos e refazer o caminho. Não há a necessidade de pesquisa em banco de dados. Se os dois recibos estão em espaços de sincronismo diferentes, ou seja, se estão em rodadas diferentes, é possível checar os dois somente em relação ao Ponto de Confiança¸ o C0. Neste caso, para poder checar um em relação ao outro deve-se realizar uma consulta ao Diretório Público e solicitar a cadeia de protocolização existente entre os dois.
3. *Verificar a autenticidade do Banco de dados:* Consiste na tarefa de analisar se o banco de dados pertence à mesma PDDE. Devem ser disponibilizadas as seguintes informações:

    (a) Ponto de Confiança;

    (b) Data e hora da publicação do Ponto de Confiança;

    (c) Quantidade de recibos;

    (d) Tamanho do arquivo;

    (e) Outras informações.

5.4 Conclusão

O método a ser utilizado em uma auditoria de PDDE dependerá do método de datação suportado pela própria PDDE. Para cada método de datação há um método de auditoria correspondente. Neste capítulo foram vistos em detalhes dois métodos de auditoria: encadeamento linear e árvore sincronizada. As operações de verificação são as mesmas em ambos, o que difere é a maneira como cada uma é realizada.

6 Implementação de um sistema de auditoria

6.1 Introdução

O sistema de auditoria foi implementado para o método de datação do encadeamento linear devido à sua maior simplicidade em relação a outros métodos de datação e, principalmente, à disponibilidade de um banco de dados fornecido pela empresa Bry contendo exemplos reais de protocolações.

6.2 Requisitos

Para uma melhor compreensão dos requisitos do sistema foram desenvolvidos casos de uso de forma a detalhar cada funcionalidade. A seguir o diagrama de casos de uso é mostrado.

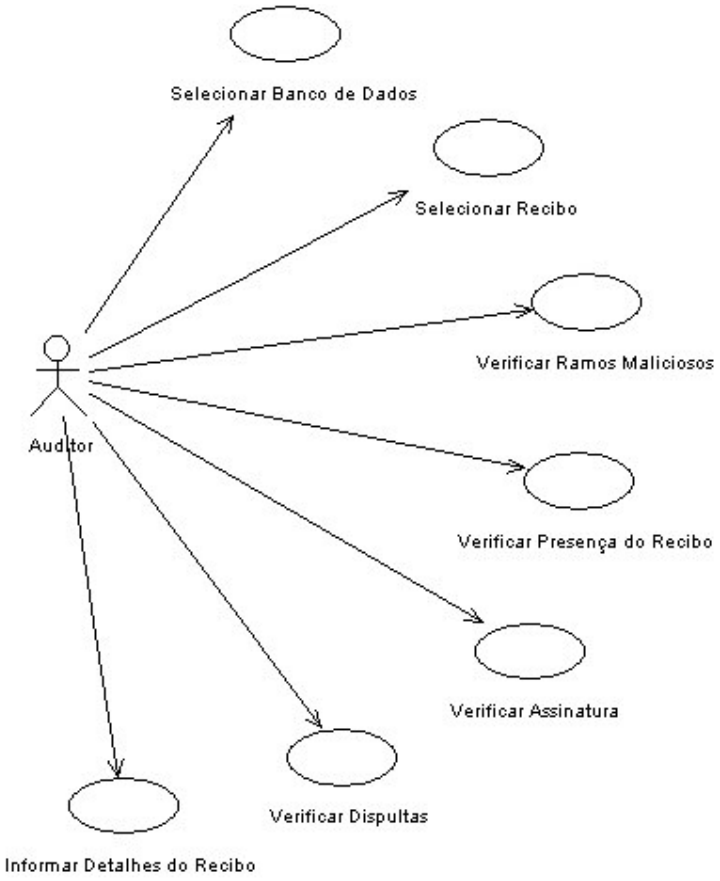

Figura 15 - diagrama de casos de uso.

A seguir cada caso de uso é descrito em detalhes.

- **Selecionar banco de dados**

Use Case #1	Selecionar Banco de Dados
Objetivo dentro do contexto	Auditor conseguir selecionar o banco de dados ao qual deseja realizar as verificações desejadas.
Escopo	Auditoria
Pré - Condições	----------
Término com sucesso	O auditor seleciona o banco de dados desejado.
Término com falha	O auditor não conseguirá selecionar um banco de dados
Atores Primários Atores Secundários	Auditor ---------
Gatilho	---------

Descrição	Passo	Ação
	1	Auditor seleciona o arquivo de banco de dados
	2	Sistema recebe o arquivo selecionado
	3	Sistema exibe o caminho completo do arquivo selecionado

- **Selecionar recibo**

Use Case #2	Selecionar Recibo
Objetivo dentro do contexto	Auditor conseguir selecionar o recibo ao qual deseja realizar as verificações desejadas.
Escopo	Auditoria
Pré - Condições	----------
Término com sucesso	O auditor seleciona o recibo desejado.
Término com falha	O auditor não conseguirá selecionar o recibo
Atores Primários Atores Secundários	Auditor ---------
Gatilho	---------

Descrição	Passo	Ação
	1	Auditor seleciona o arquivo do recibo
	2	Sistema recebe o arquivo selecionado
	3	Sistema exibe o caminho completo do arquivo selecionado

- **Verificar ramos maliciosos**

Use Case #3	Verificar Ramos Maliciosos
Objetivo dentro do contexto	Auditor conseguir verificar a confiabilidade da protocolizadora, ou seja, se existe ramos maliciosos.
Escopo	Auditoria
Pré - Condições	O auditor deve ter selecionado um banco de dados
Término com sucesso	O auditor recebe uma mensagem dizendo que a protocolizadora está insenta de ramos maliciosos.
Término com falha	O auditor recebe uma mensagem dizendo que a protocolizadora possui ramos maliciosos.
Atores Primários Atores Secundários	Auditor ---------
Gatilho	---------

Descrição	Passo	Ação

	1	Auditor habilita a verificação de ramos maliciosos
	2	Sistema verifica se existe algum banco de dados selecionado
	3	Sistema exibe uma mensagem com o resultado da verificação

- **Verificar presença do recibo**

Use Case #4	Verificar Presença do Recibo
Objetivo dentro do contexto	Auditor conseguir verificar a presença de um determinado recibo dentro de um banco de dados selecionado.
Escopo	Auditoria
Pré - Condições	O auditor deve ter selecionado um banco de dados e um recibo
Término com sucesso	O auditor recebe uma mensagem dizendo que o recibo consta no banco de dados selecionado.
Término com falha	O auditor recebe uma mensagem dizendo que o recibo não consta no banco de dados selecionado.
Atores Primários Atores Secundários	Auditor ---------
Gatilho	---------

Descrição	Passo	Ação
	1	Auditor habilita a verificação da presença de recibos
	2	Auditor seleciona um recibo para verificação
	3	Sistema verifica se existe algum banco de dados e recibo selecionados
	4	Sistema exibe uma mensagem com o resultado da verificação

- **Verificar Assinatura**

Use Case #5	Verificar Assinatura
Objetivo dentro do contexto	Auditor conseguir verificar se a assinatura é válida.
Escopo	Auditoria
Pré - Condições	O auditor deve ter selecionado um banco de dados e um recibo
Término com sucesso	O auditor recebe uma mensagem dizendo que a assinatura é válida
Término com falha	O auditor recebe uma mensagem dizendo que a assinatura não é válida
Atores Primários Atores Secundários	Auditor ---------
Gatilho	--------

Descrição	Passo	Ação
	1	Auditor habilita a verificação da assinatura
	2	Sistema verifica se um banco de dados foi selecionado
	3	Sistema verifica se a assinatura é compatível com a assinatura acoplada ao recibo

- **Resolver disputa**

Use Case #6	Resolver Disputa
Objetivo dentro do contexto	Auditor conseguir verificar entre dois recibos selecionados, qual foi protocolado primeiro.
Escopo	Auditoria
Pré - Condições	O auditor deve ter selecionado um banco de dados e dois recibos
Término com sucesso	O auditor recebe uma mensagem dizendo qual recibo foi protocolado primeiro
Término com falha	O auditor recebe uma mensagem dizendo que o recibo não consta no banco de dados selecionado.

Atores Primários Atores Secundários	Auditor ---------
Gatilho	Habilita a seleção de um novo recibo

Descrição	Passo	Ação
	1	Auditor habilita a comparação de recibos
	2	Sistema habilita a seleção de um segundo recibo
	3	Auditor verifica a disputa de recibos
	4	Sistema verifica se um banco de dados foi selecionado
	5	Sistema verifica se os dois recibos existem no banco de dados
	6	Sistema verifica qual foi protocolado primeiro
	7	Sistema informa qual recibo foi protocolado primeiro
Extensão	Passo	Ação alternativa
	4a	Sistema exibe uma mensagem de ausência de banco de dados selecionado
	5a	Sistema exibe uma mensagem de ausência de recibo selecionado

- **Informar detalhes do recibo**

Use Case #7	Informar detalhes do recibo
Objetivo dentro do contexto	Auditor conseguir verificar detalhes do recibo.
Escopo	Auditoria
Pré - Condições	O auditor deve ter selecionado um recibo
Término com sucesso	O auditor recebe as informações do recibo.
Término com falha	O auditor não recebe as informações do recibo.
Atores Primários Atores Secundários	Auditor ---------
Gatilho	---------

Descrição	Passo	Ação
	1	Auditor seleciona um recibo
	2	Sistema verifica se existe algum banco de dados selecionado
	3	Auditor seleciona detalhes do recibo
	4	Sistema informa detalhes do recibo selecionado

6.3 Conteúdo do banco de dados

O BD foi projetado para armazenar e manipular recibos emitidos pela *Bry PDDE* (PDDE desenvolvida pela empresa Bry que já opera comercialmente) que faz suas datações utilizando o método do encadeamento linear. A utilização deste BD fornecido pela Bry dá um caráter prático à pesquisa, já que este sistema de auditoria pode ser utilizado para auditar um produto comercial.

Para a implementação deste BD foi utilizado o Interbase 6.

Tabela LINK	
RECIBO (Primery key)	INTEGER Not Null
HASH	VARCHAR(40) Not Null
LINK	VARCHAR(40) Not Null
DATA	VARCHAR(40) Not Null

Tabela 1- Conteúdo do BD utilizado no processo de auditoria.

A seguir é descrito cada campo da tabela:

- **Recibo:** número seqüencial incrementado a cada protocolação.
- **Hash:** contém o hash do documento que será protocolado.
- **Link:** contém o hash da concatenação entre o link anterior e o campo hash atual.
- **Data:** armazena a data e hora da momento da protocolação no seguinte formato:

YYYYMMDDHHMMSS.LLLZ

Onde:

- **YYYYMMDD** corresponde à data da protocolação (ex: 20030121 – 21/01/2003);
- **HHMMSS** equivale à hora da protocolação (ex: 095532 – 09:55:32);
- **LLL** é a informação dos milissegundos da protocolação (esta informação é relevante já que a PDDE pode realizar mais de uma protocolação no intervalo de um segundo).
- **'.'** e **'Z'** são caracteres auxiliares que constam em todas as datas.

6.4 Formato do recibo

O recibo emitido pela PDDE utiliza o padrão definido na RFC 3161 onde é especificado o protocolo de comunicação TSP – *Time Stamping Protocol*.

A seguir é mostrado, em formato ASN.1, o conteúdo do recibo emitido pela PDDE após uma requisição de protocolação seguindo o padrão descrito no RFC 3161.

```
TimeStampResp ::= SEQUENCE {
          status                  PKIStatusInfo,
          timeStampToken          TimeStampToken     OPTIONAL  }


PKIStatusInfo ::= SEQUENCE {
        status      PKIStatus,
        statusString  PKIFreeText    OPTIONAL,
        failInfo    PKIFailureInfo   OPTIONAL  }


PKIStatus ::= INTEGER {
      granted                (0),
      grantedWithMods        (1),
      rejection              (2),
      waiting                (3),
      revocationWarning      (4),
      revocationNotification (5)   }


PKIFailureInfo ::= BIT STRING {
       badAlg               (0),
       badRequest           (2),
       badDataFormat        (5),
       timeNotAvailable    (14),
       unacceptedPolicy    (15),
       unacceptedExtension (16),
       addInfoNotAvailable (17)
       systemFailure       (25)


TimeStampToken ::= SEQUENCE {
       tstInfo   TSTInfo
       signature Signature
}


TSTInfo ::= SEQUENCE {
          version                    INTEGER  { v1(1) },
          policy                     TSAPolicyId,
          messageImprint             MessageImprint,
          serialNumber               INTEGER,
          genTime                    GeneralizedTime,
          accuracy                   Accuracy                OPTIONAL,
          ordering                   BOOLEAN             DEFAULT FALSE,
          nonce                      INTEGER                 OPTIONAL,
          tsa                        [0] GeneralName         OPTIONAL,
          extensions                 [1] IMPLICIT Extensions OPTIONAL   }

Accuracy ::= SEQUENCE {
                     seconds        INTEGER           OPTIONAL,
                     millis     [0] INTEGER  (1..999) OPTIONAL,
                     micros     [1] INTEGER  (1..999) OPTIONAL  }
```

Para simplificar a programação/implementação, o protocolo especificado acima foi mapeado nas seguintes estruturas em linguagem C:

```
typedef struct sTimeStampResp_OK
{
      PKIStatusInfo status;
       TimeStampToken timeStampToken;
} TimeStampResp_OK;


typedef struct sTimeStampToken
{
      TSTInfo tstInfo;
      Signature signature;
}TimeStampToken;


typedef struct sPKIStatusInfo
{
  PKIStatus status;
  PKIFailureInfo failInfo;
}PKIStatusInfo;


typedef INT16 PKIStatus;


typedef INT16 PKIFailureInfo;


typedef struct sTSTInfo
{
     char genTime[DATE_LEN];
     INT16 version;
     OID policy;
     UINT64 serialNumber;
     MessageImprint messageImprint;
      MessageImprint link;
     UINT32 nonce;
     char tsa[TSA_LEN];
}TSTInfo;


typedef struct sMessageImprint
{
     OID algorithmIdentifier;
     char hashedMessage[SHA_HASH_LEN];
}MessageImprint;


typedef struct sLinkProtocolStruct
{
      INT32 sign_len;
      BYTE signature[MAX_SIGNATURE_LEN];
}LinkProtocolStruct;


typedef struct sSignature
{
     OID algorithmIdentifier;
     LinkProtocolStruct detachedSignature;
}Signature;
```

Sendo assim, o arquivo de recibo da datação terá exatamente o tamanho da estrutura *TimeStampResp_OK*. Para preencher esta estrutura com os dados do recibo basta simplesmente ler todos os bytes do arquivo e realizar um *type casting* para esta estrutura em seguida. Desta forma é extremamente prático acessar cada informação específica a respeito do recibo como, por exemplo, a data da protocolação (campo *genTime*) ou o tamanho da assinatura digital (campo *sign_len*).

6.5 Função resumo (hash)

A função resumo utilizada nesta pesquisa é a SHA1 que produz saída de 160 bits, sendo mais segura que o MD5 que possui saída de 128 bits. Além disso é um padrão bastante popular e de fácil implementação.

Sendo a saída de do SHA1 de 160 bits, ou 20 bytes, o leitor mais atento pode se perguntar: porque o banco de dados, descrito na seção anterior, reserva um espaço de 40 bytes nos campos *hash* e *link* da tabela ao invés de 20 bytes apenas? A resposta é que se optou por guardar esta informação no formato hexadecimal. Desta forma, cada byte do hash SHA1 gerado (ex: 00101111) precisa de 2 bytes para representa-lo no formato hexadecimal (ex: 2F).

6.6 Arquitetura do sistema

O sistema de auditoria consiste basicamente em um software capaz de ler o arquivo do banco de dados de uma PDDE, interpretar os recibos a serem verificados (emitidos pela PDDE) e utilizar funções de verificação, como verificar a assinatura digital de um recibo, para assegurar a confiabilidade da PDDE e resolver disputas. O software foi desenvolvido na linguagem C++ utilizando o ambiente Visual C++ 6.0. A justificativa para a escolha do Visual C++ é que é um ambiente que permite o desenvolvimento de interfaces gráficas, ainda que um pouco limitado se comparado ao Borland C++ por exemplo, mas principalmente pelo tamanho reduzido do arquivo executável e da familiaridade com esta ferramenta por parte dos pesquisadores.

6.7 Diagrama de classes

A figura 16 descreve o diagrama de classes desenvolvido para o sistema de auditoria.

A utilização da modelagem orientada a objetos nos permite agrupar funções/características comuns em classes de forma a facilitar a manutenção e a escalabilidade do sistema. A seguir cada classe será estuda em mais detalhes.

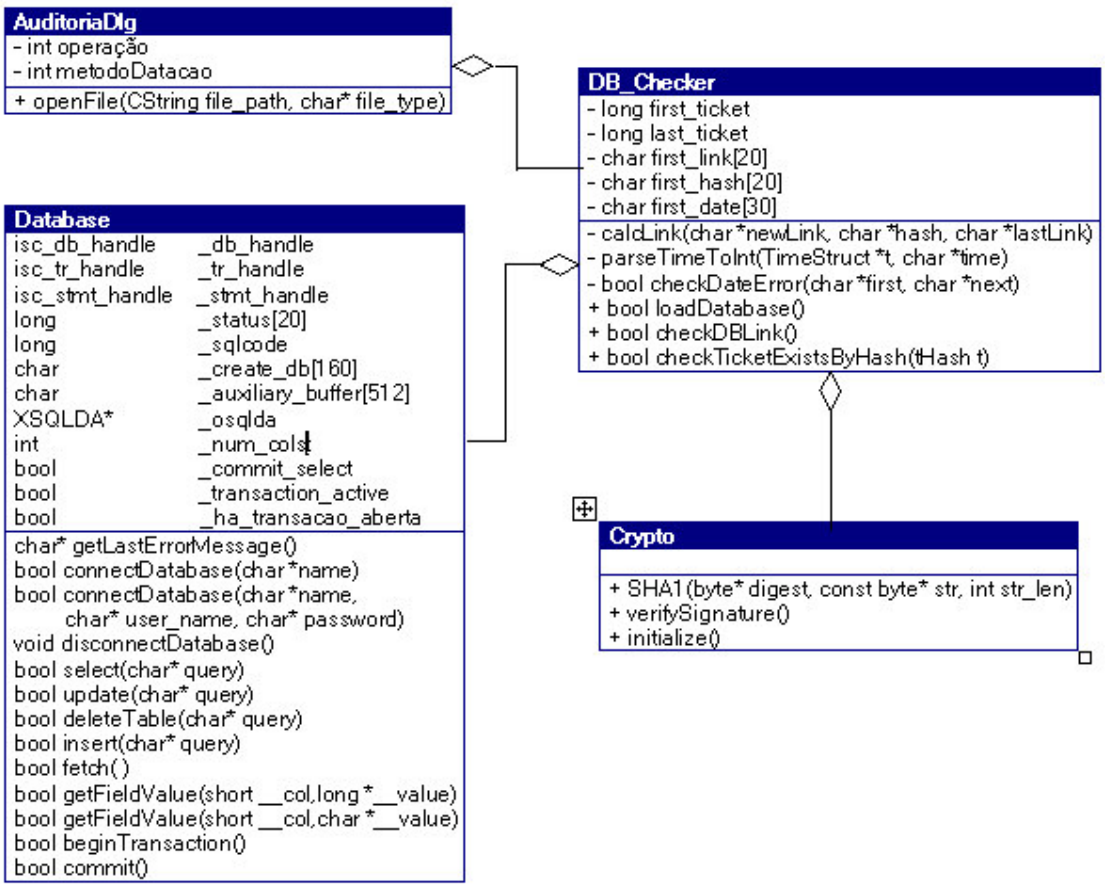

Figura 16 - Diagrama de classes do sistema de auditoria.

- **AuditoriaDlg**: é a interface do sistema com o usuário. Possui funções que tratam da interação usuário-sistema. Apesar de não estar expresso no diagrama acima, para cada ação que o usuário pode tomar (exemplo o clique de um botão) há um método que descreve o procedimento que o programa deve seguir para a ação específica.

- **DB_Checker**: esta é a principal classe do sistema já que possui as funções de verificação propriamente ditas. Uma das funções mais importantes desta classe é a de verificar se existe ramo malicioso no método de encadeamento (*checkDBLink*). A seguir é mostrado o fluxo básico de controle para esta função em linguagem de alto nível:

    1. Ler os campos $hash_0$, $link_0$ *e* $data_0$ do BD;
    2. Para cada linha da tabela *LINK*, ou seja para cada protocolação:
        2.1 Calcular o hash da concatenação entre $link_{n-1}$ e $hash_n$

2.2 Verificar se o hash calculado no passo anterior é igual ao $link_n$

2.3 Verificar se o campo data da protocolação atual ($data_n$) é menor do que a data da próxima protocolação ($data_{n+1}$)

Outra função importante desta classe é a que verifica a validade de um recibo através dos seguintes passos:

1. *recálculo das informações constantes no mesmo;*
2. *verificação da assinatura digital da PDDE;*
3. *consulta a lista de certificados revogados.*

Por último, tem-se a função que verifica a pertinência de um recibo na cadeia de datação. Esta função simplesmente percorre seqüencialmente todas as linhas da tabela LINK a procura de um determinado hash de um documento (*checkTicketExistsByHash*) .

- **Database:** esta classe abstrai detalhes de implementação relativos ao uso do Interbase 6.0. Desta forma se for necessária a utilização de outro banco de dados (exemplo Oracle) basta que se altere somente esta classe, mantendo a mesma interface, para que o sistema continue funcionando corretamente. Outra vantagem é que ela agrupa e simplifica a utilização de funções relativas a BD. Esta classe foi reutilizada da PDDE descrita no capítulo 4.
- **Crypto:** da mesma forma que a classe anterior, esta classe também agrupa e simplifica a utilização de funções relacionadas à criptografia, como o cálculo de funções resumo e a verificação de assinaturas digitais. Vale ressaltar que esta é exatamente a mesma classe *Crypto* da PDDE descrita na seção 4.1.1.2.

6.8 Demonstração do sistema de auditoria

A seguir é mostrada a interface do sistema com o usuário.

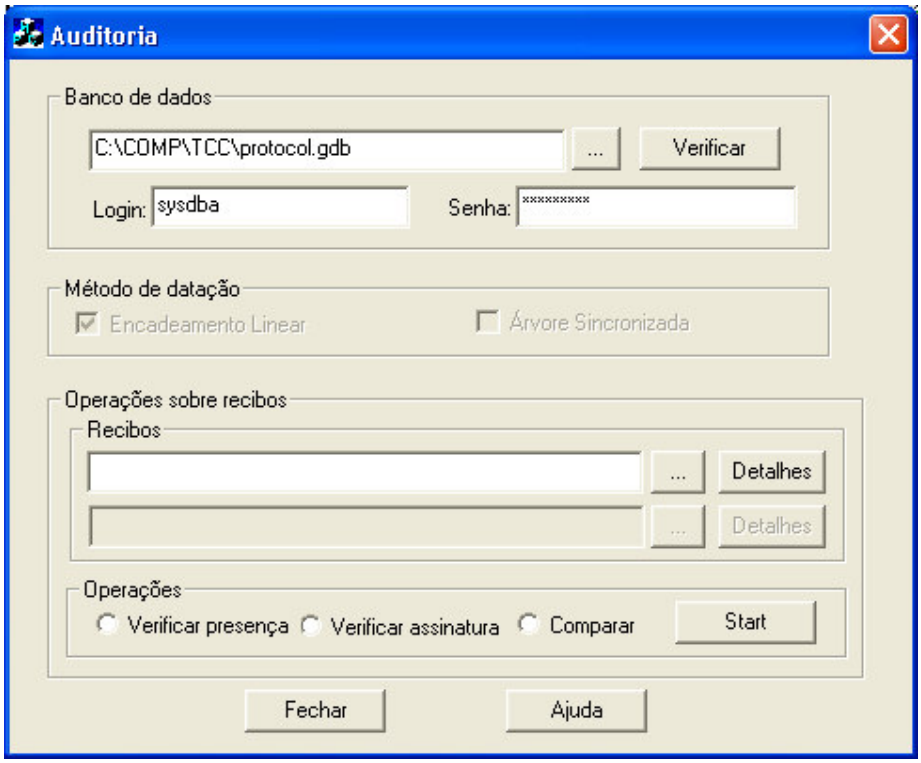

Figura 17 - Interface do programa de auditoria.

Algumas considerações sobre a interface são importantes para que o usuário não tenha problemas em utilizar o software.

a) Verificação do banco de dados

Quando o usuário pressionar o botão Verificar, após ter selecionado o arquivo do banco de dados e informado o *login* e a senha, a seguinte mensagem aparecerá indicando o sucesso da operação de verificação, caso realmente o banco de dados esteja consistente:

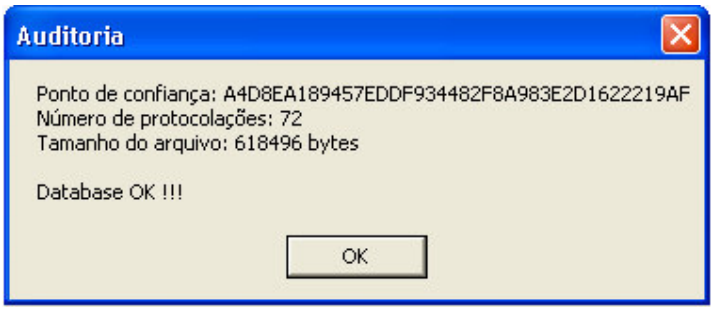

Figura 18 - Verificação do BD da PDDE.

O ponto de confiança nada mais é do que o primeiro link do BD.

b) Operações sobre o recibo

O usuário, ao selecionar um recibo, tem a opção de visualizar detalhes do mesmo, como a data e hora da datação (campo genTime). Todos estes campos estão descritos no RFC 3161.

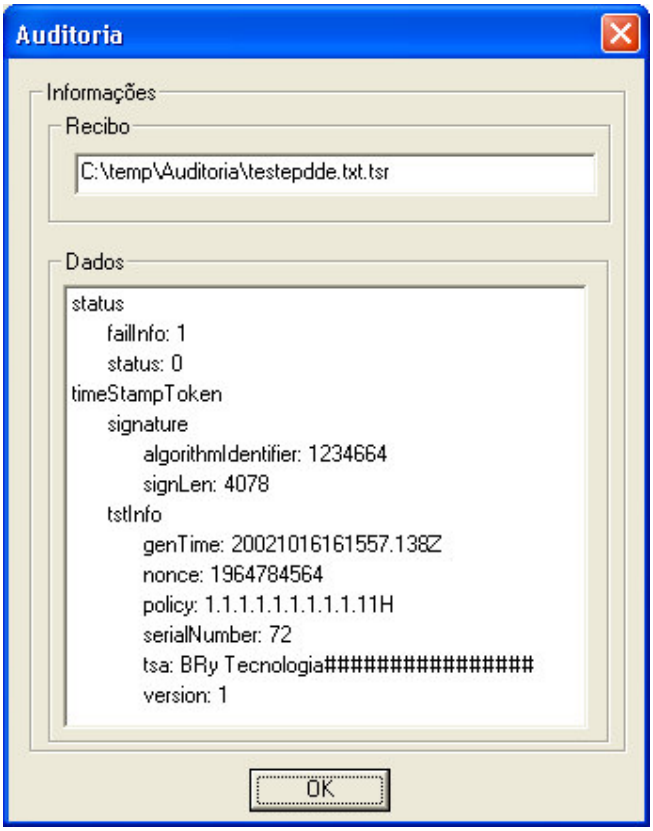

Figura 19 - Detalhes do recibo emitido pela PDDE.

Para realizar qualquer verificação sobre um recibo basta selecionar a operação desejada (verificar presença, verificar assinatura ou comparar) e pressionar o botão *Start*. Uma mensagem irá indicar o resultado da operação, como no exemplo abaixo:

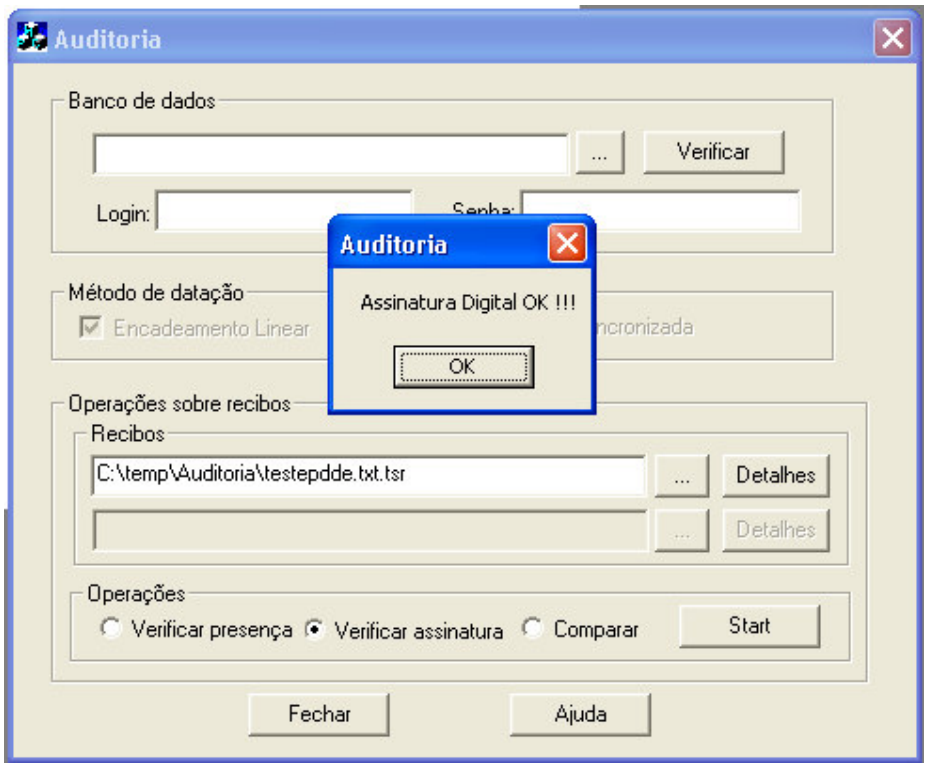

Figura 20 - Verificação da assinatura digital de um recibo.

6.9 Conclusão

O sistema desenvolvido conseguiu cumprir todos os requisitos básicos de auditoria conforme descrito no capítulo 5. Requisitos não funcionais, que dizem respeito à qualidade do sistema, como o tamanho do arquivo executável e o desempenho do software também foram atingidos, na medida em que a versão final do software possui apenas 344 Kbytes e todas as verificações executam extremamente rápido.

A seguir são descritas algumas sugestões de melhorias que podem ser feitas no software de auditoria:

- Suporte ao método da árvore sincronizada;
- Possibilidade de se comparar recibos que não estejam no mesmo BD;
- Na verificação da assinatura digital da PDDE, averiguar se, no momento da datação do recibo, o certificado digital da PDDE constava na lista de certificados revogados;

7 Considerações finais

A motivação para o desenvolvimento deste trabalho foi a necessidade observada de se ter maneiras de auditar uma AD de forma a garantir o sigilo e a segurança do processo de protocolação de documentos eletrônicos.

O primeiro objetivo específico do trabalho foi estudar as técnicas de criptografia. Este objetivo foi alcançado e está descrito no capítulo 2. Ele é importante para o entendimento das técnicas básicas de criptografia. Essas técnicas são utilizadas nos métodos de datação, através da criptografia simétrica e assimétrica, de forma a garantir o sigilo na comunicação. A criptografia assimétrica e a função resumo (*hash*) são utilizadas para a assinatura digital.

O segundo objetivo específico foi estudar os métodos de datação de documentos eletrônicos existentes. Foram estudados os métodos de datação de encadeamento linear, árvore e árvore sincronizada, que estão descritos no capítulo 3. Com isso esse objetivo foi alcançado. Através desse estudo é possível identificar como deve funcionar um sistema de datação de documentos eletrônicos de forma segura e confiável. Além disso, são analisadas as vantagens e desvantagens de cada método de datação, como por exemplo, com relação a eficiência do sistema utilizando cada método.

O terceiro objetivo específico foi estudar formas de auditoria de PDDEs. Este objetivo está descrito no capítulo 4. Esta etapa da pesquisa é importante pois fornece o embasamento teórico dos métodos que irão dar suporte à implementação de um sistema de auditoria. Foram estudados métodos de auditoria que podem ser aplicados a PDDEs que utilizem encadeamento linear e árvore sincronizada como métodos de datação.

O quarto objetivo específico foi implementar uma PDDE simples. Esse objetivo foi alcançado e está descrito no capítulo 5. Foi implementada uma PDDE que utiliza como método de datação o encadeamento linear, pelo fato de ser um método mais simples de implementar. A implementação de uma PDDE permitiu uma melhor compreensão do processo de datação, além de ter ajudado na identificação de possíveis ações "maliciosas" por parte da PDDE que deveriam ser verificadas no processo de auditoria.

O último objetivo específico foi implementar um sistema de auditoria capaz de auditar PDDEs que utilizassem tanto o método do encadeamento linear como o da árvore sincronizada como métodos de datação. A implementação deste sistema foi desenvolvida baseada no banco de dados e no formato do recibo da *Bry PDDE* (PDDE desenvolvida pela empresa Bry Tecnologia que trabalha em parceria com o LabSec). Esta PDDE suporta apenas o método do encadeamento

linear. Dessa forma, pela indisponibilidade de um banco de dados que suportasse o método de datação da árvore sincronizada, foi implementada apenas auditoria para o encadeamento linear, fazendo com que este objetivo específico não fosse alcançado em sua totalidade como se era esperado.

Dentro do escopo e dos resultados obtidos descritos anteriormente, pode-se observar as seguintes contribuições:

- Proporcionou o estudo e elaboração de textos explicando de forma didática os principais métodos de datação;
- Este trabalho proporcionou o desenvolvimento de uma PDDE que utiliza o encadeamento linear como método de datação, mas que pode ser facilmente estendida para utilizar outros métodos de datação sem a necessidade de grandes alterações;
- Elaboração de textos explicativos descrevendo métodos para auditoria de PDDEs que utilizam o encadeamento linear ou árvore sincronizada como métodos de datação;
- Permitiu o desenvolvimento de um sistema de auditoria aplicável a uma PDDE que opera comercialmente;

Os trabalhos desenvolvidos no contexto deste relatório fornecem subsídios para o desenvolvimento de uma PDDE que utilize árvore sincronizada como método de datação e para ampliação do sistema de auditoria para que possa ser capaz de auditar PDDEs que protocolam usando o método da árvore sincronizada.

8 REFERÊNCIAS BIBLIOGRÁFICAS

[BEN 92] BENALOH, J. Efficient broadcast time-stamping. **Clarkson University, Department of Math and Computer Science, [S.l.],** Abril, 1992.

[BEN 94] BENALOH, J. One-way accumulators: **A decentralized alternative to digital signatures.** Advances in Cryptology - Proceedeings of Eurocrypt 93, LNCS 756, [S.l.], p.274–285, Berlin, 1994.

[COS 02] COSTA, V., PIRES, C. A., AURÉLIO, M. A. **Auditoria do Processo de Protocolização de Documento Eletrônicos**. Florianópolis, 2002.

[HAB 91] HABER, S. **How to time-stamp a digital document**. Journal of Cryptology, [S.l.], v.3, p.99–112, 1991.

[JUS 98] JUST, M. K. **On the Temporal Authentication of Digital Data**. School of Computer Science - Carleton University, December, 1998. Ph.d.

[LIP 99] LIPMAA, H. **SECURE AND EFFICIENT TIME-STAMPING SYSTEMS**. University of Tartu - Estonia, July, 1999. Ph.d.

[MAS 99] MASSIAS, H.; AVILA, X. S. **Timestamps: Main issues on their use and implementation**. IEEE 8th InternationalWorkshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, [S.l.], 1999.

[PAS 01] PASQUAL, E. S. **IDDE – Uma infra-estrutura para datação de documentos eletrônicos**. Florianópolis, 2002.

[ROO 99] ROOS, M. **Integrating time-stamping and notarization**. University of Tartu - Estonia, 1999. Masterthesis.

[SCH 95] SCHNEIER, B. **Applied Cryptography: Protocols, Algorithms, and Source Code in C 2nd Edition**. John Wiley and Sons, 2. ed., 1995.

[STA 98] STALLINGS, W. **Cryptography and Network Security**. Prentice Hall, 2. ed., 1998.

[STI 95] STINSON, D. R. **Cryptography : Theory and Practice**. CRC Press, 1995.

APÊNDICE 1 - Artigo

Auditoria de uma Protocolizadora Digital de Documentos Eletrônicos

Cleyton André Pires, Marcos Aurélio Dias

Departamento de Informática e Estatística - INE

Universidade Federal de Santa Catarina

88040-900 Florianópolis-SC

cleyton@inf.ufsc.br, aurelio@inf.ufsc.br

Resumo. *O processo de protocolização digital de documentos eletrônicos não garante que Autoridade de Datação - entidade responsável por anexar data e hora ao documento - seja confiável. Com objetivo de tornar este processo mais confiável, este artigo descreve o modelo e a implementação de um sistema de auditoria de uma PDDE – Protocolizadora Digital de Documentos Eletrônicos - que utiliza como método de datação o encadeamento linear .*

Palavras-chaves: auditoria, datação digital, documentos eletrônicos.

Abstract: *The digital time stamping process of electronic documents does not assure that the Time Server Authority, that is responsible for attach date and time to the document, be trustworthy. With the purpose of becoming this process more reliable, this article describes the model and implementation of an auditing system of a Electronic Documents Digital Timestamper, that uses the link method.*

Keywords: auditing, digital time stamping, electronic document.

1 Introdução

Atualmente o uso de documentos eletrônicos aumentou consideravelmente em relação aos documentos em papel. Isto se deve principalmente a velocidade de comunicação mais rápida e segura, provendo economia de recursos, menor ocupação de espaço físico.

Para que o documento eletrônico tenha a mesma validade jurídica do documento em papel, é preciso que o mesmo seja assinado e datado. A assinatura permite que características da pessoa que está assinando sejam adicionadas ao documento de forma a garantir que esse foi assinado pela pessoa como está tivesse assinado a mão. A assinatura é conseguida através da criptografia. A datação garante que o documento existe em uma determinada data e hora [LIP 99]. A data e a hora anexadas ao documento no processo de datação devem condizer com a data e a hora corrente, de modo a garantir que o documento existiu em um determinado momento no tempo. Sem o processo de datação não se pode afirmar que um documento é ou não intempestivo, ou seja, decidir se ele está fora do seu prazo legal [COS 02]. A datação é conseguida por uma Autoridade de Datação (AD), que deve ser uma entidade confiável.

Existem vários métodos de datação de documentos eletrônicos, entre eles destaca-se o método encadeamento linear por sua simplicidade. Este método é descrito na seção 2. A seção 3 descreve o processo de auditoria de uma PDDE e, finalmente, a seção 4 apresenta a implementação de um sistema de auditoria para tornar o processo de protocolização o mais seguro e confiavél possível.

2 Métodos de datação

Métodos de datação são mecanismos que servem para aumentar a confiabilidade de uma PDDE, dificultando uma ação maliciosa por parte da mesma.

Um bom método de datação deve atender aos seguintes requisitos de segurança:

1. **Privacidade**: Apenas o cliente pode ter acesso ao conteúdo do documento;
2. **Facilidade de comunicação e armazenamento**: Deve ser prático datar o documento independentemente de seu tamanho;
3. **Integridade**: Deve-se garantir que os dados não são alterados e a operação ininterrupta do serviço de datação;
4. **Anonimato**: Deve-se garantir o anonimato do cliente.
5. **Confiança**: Deve-se garantir que um documento será datado com a data e hora correta;

A AD acrescenta data e hora ao documento ao recebe-lo do cliente, armazena uma cópia que será utilizada em caso de disputa e devolve um recibo indicando que o documento foi datado. Este não é um bom procedimento já que não atende aos requisitos de privacidade uma vez que a AD fica conhecendo o documento e uso do canal de comunicação e armazenamento uma vez que todo o documento deve ser transmitido e armazenado Este será um problema caso o documento seja muito grande. Para resolver este problema pode-se utilizar um resumo do documento, conhecido como *hash*. O *hash* representa de forma única um documento. Os mais conhecidos são o MD5 e o SHA-1 com tamanhos de 128 e 160 bits respectivamente.

Assim, ao invés de se transmitir o documento, o cliente transmite o resumo, atendendo aos requisitos de privacidade, uso do canal de comunicação e espaço de armazenamento, pois o tamanho do resumo é muito menor que o do documento.

A integridade pode ser garantida, pois quando a AD anexa data e hora ao resumo, a mesma assina e o envia ao cliente do serviço. O cliente por sua vez verifica a assinatura e tem certeza que o resumo que ele enviou foi realmente o resumo que foi enviado para a AD.

Existem várias formas de resolver o requisito de anonimato. Este artigo não trata deste requisito em particular. O anonimato não invalida o cumprimento dos outros requisitos de segurança. O anonimato pode ser visto como um complemento desejável, existindo muitas situações onde não há a sua necessidade.

O requisito de confiança pode ser atendido se a AD é considerada confiável. Isso pode ser obtido, na prática, tendo-se um equipamento lacrado e passível de auditoria, como por exemplo, os descritos na FIPS-140[NIS 02]. Contudo, o lacre e a auditoria implicam em custos e possibilidade de fraudes, provocando uma desconfiança por parte do cliente. Na realidade, o uso de auditoria só transfere a necessidade de confiança a uma terceira entidade, neste caso o auditor. O ideal seria que a AD não pudesse ser maliciosa, mesmo que seu administrador o fosse. Isso já é possível utilizando-se alguns dos métodos descritos a seguir.

Estes métodos levam em consideração a questão temporal, ou seja, como o documento recebe a data e hora. Ela pode ser **absoluta** ou **relativa**. A autenticação temporal absoluta contém informações de data e hora igual a usada no mundo real. Já a autenticação temporal relativa contém informações que somente verificam se um documento foi datado antes ou depois de um outro documento.

Os dois métodos temporais de autenticação podem ser usados para se datar documentos, mas o método absoluto pressupõe que a AD seja uma entidade confiável. Para o método relativo

não é necessária a existência de entidade confiável, pois existem mecanismos que garantem que mesmo que a AD seja maliciosa, o documento sempre será datado com data e hora real. A seguir são apresentados os principais métodos que trabalham com o método de autenticação temporal relativa.

A figura 1 mostra como seria uma datação utilizando uma AD confiável.

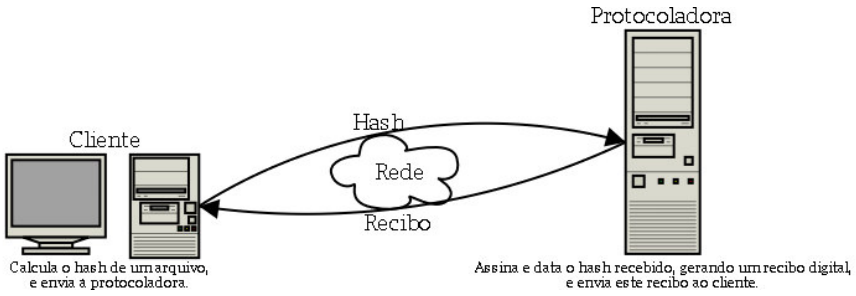

Figura 1 - Datação de documentos eletrônicos

2.1 Método do Encadeamento linear

Trata-se de um método de datação [HAB 91] no qual a AD encadeia os resumos dos documentos que foram enviados pelos clientes e os armazena em um banco de dados interno. O encadeamento entre os recibos utiliza uma função de sentido único, como por exemplo, uma função *hash*.

O encadeamento é formado por *links*, sendo que o primeiro *link* da cadeia pode ser gerado de maneira randômica, formando o L_0. Em seguida, o segundo link L_1 é gerado utilizando o *link* anterior, no caso, L_0 e o resumo do documento enviado pelo cliente. De forma genérica, temos:

$$Ln = (t_{n-1}, ID_{n-1}, H_{n-1}, H(L_{n-1})) \quad (1)$$

onde t_{n-1} é a data e hora em que o documento anterior foi datado, ID_{n-1} é um identificador do documento anterior, H_{n-1} é o resumo do documento anterior e $H(L_{n-1})$ é resumo do link anterior.

Após o cálculo do link, a AD gera o recibo que será enviado para o cliente que será:

$$s = SigAD(ID_n, t_n, ID_n, H_n, L_n) \quad (2)$$

Desta forma, os resumos dos documentos que foram enviados pelos clientes ficam ordenados obedecendo à ordem de chegada.

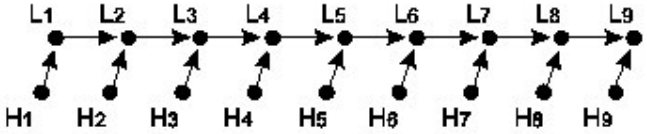

Figura 2 - Método do encadeamento linear

3 Auditoria de sistemas

Define-se auditoria como ”um conjunto de técnicas de observações e exames, aplicados de forma sistemática, que no contexto do auditado, visa opinar sobre sua situação, sobre sua riqueza, quando for o caso, ou sobre funções ou áreas específicas de componentes do patrimônio do auditado”.

Auditoria de sistemas é o ramo da auditoria que revisa e avalia os controles internos informatizados e que tem como objetivos:

- **Verificar a eficiência**: Verifica-se a utilização de recursos de computação alocados ao sistema.
- **Constatar eficácia**: Compreende a validação dos resultados gerados pelos sistemas.
- **Atestar a segurança**:
    1. *Segurança física*: instalações, equipamentos, suprimentos, documentação, dados, pessoal, entre outros.
    2. *Segurança lógica*: Sistemas e informações.
    3. *Segurança em comunicação*: veiculação dos dados por meios de comunicação.

A auditoria de sistemas no contexto da PDDE tem como objetivo atestar a confiabilidade e consiste em garantir a:

1. *Integridade*: através da verificação da assinatura digital da PDDE.
2. *Irretroatividade*: possibilidade de verificar uma eventual alteração da data em que um documento foi protocolado.

Para tanto as seguintes verificações devem ser feitas:

1. Verificar a validade de um recibo;
2. Determinar a seqüência entre dois recibos (quem protocolou primeiro);
3. Analisar se existe ramo malicioso no método de encadeamento;

Os detalhes de funcionamento da auditoria devem estar previstos nas Diretrizes de Práticas e Protocolizações (DPP). A auditoria pode ser realizada de duas maneiras:

- **Auditoria externa**: Utiliza somente as informações contidas nos recibos e nas informações publicadas pela AD em um diretório público;
- **Auditoria interna**: utiliza informações internas armazenadas na AD como os logs e o próprio encadeamento. Devido a isto, a AD deve ser protegida com lacres e deve estar em um lugar seguro para que ninguém não autorizado tenha acesso a máquina [PAS 02].

Somente a auditoria externa é tratada neste trabalho.

3.1. Auditoria para o método de encadeamento linear

A seguir é descrito quais são os procedimentos necessários para realizar os três tipos de verificações na auditoria para o método do encadeamento linear:

1. *Verificar a validade de um recibo*: Para verificar se um determinado documento foi protocolado por uma PDDE específica, deve-se percorrer os links do encadeamento e verificar se o resumo do documento está presente na lista. Note que devido ao fato de que o tempo de busca no método do encadeamento linear ser diretamente proporcional ao número de protocolizações, se o encadeamento na base de dados internos da PDDE for muito grande, este tipo de verificação pode ser muito lenta.
2. *Resolver disputa entre dois recibos para saber qual protocolou primeiro*: Para verificar quais de dois documentos foram protocolados primeiro em uma PDDE, é necessário percorrer o encadeamento e verificar em qual posição o primeiro documento se encontra. Em seguida, deve-se percorrer o encadeamento novamente e procurar em que posição o segundo documento se encontra. De posse das posições em que os documentos foram protocolizados, respectivamente, basta compará-las para saber qual deles foi protocolado primeiro.
3. *Verificar a autenticidade do Banco de dados*: Para verificar se existem ramos maliciosos na AD, deve-se calcular os links a partir do ponto de confiança até o final do

encadeamento e verificar se os links calculados são equivalentes aos links constantes nos recibos.

4 Implementação do sistema de auditoria

O sistema de auditoria foi implementado para o método de datação do encadeamento linear devido à sua maior simplicidade em relação a outros métodos de datação e, principalmente, à disponibilidade de um banco de dados fornecido pela empresa Bry contendo exemplos reais de protocolações.

O sistema de auditoria consiste basicamente em um software capaz de ler o arquivo do banco de dados de uma PDDE, interpretar os recibos a serem verificados (emitidos pela PDDE) e utilizar funções de verificação. O software foi desenvolvido em Visual C++ 6.0 por apresentar arquivo executável menor, comparado ao Borland C++.

A função resumo utilizada nesta pesquisa é a SHA1 que produz saída de 160 bits, sendo mais segura que o MD5 que possui saída de 128 bits. Além disso é um padrão bastante popular e de fácil implementação.

O BD foi projetado para armazenar e manipular recibos emitidos pela *Bry PDDE* (PDDE desenvolvida pela empresa Bry que já opera comercialmente) que faz suas datações utilizando o método do encadeamento linear. O BD foi desenvolvido no Interbase 6.0.

4.1Demonstração do sistema de auditoria

A seguir é mostrada a interface do sistema com o usuário.

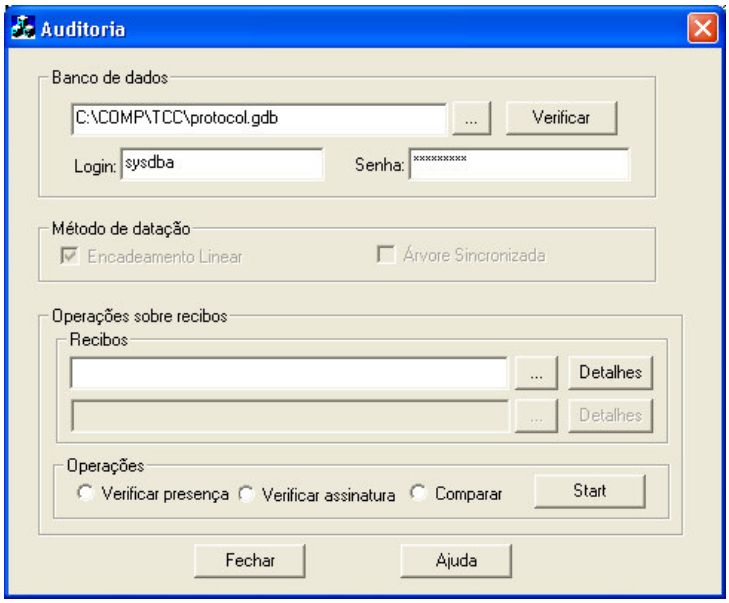

Figura 21 - Interface do programa de auditoria.

Algumas considerações sobre a interface são importantes para que o usuário não tenha problemas em utilizar o software.

a) Verificação do banco de dados

Quando o usuário pressionar o botão Verificar, após ter selecionado o arquivo do banco de dados e informado o *login* e a senha, a seguinte mensagem aparecerá indicando o sucesso da operação de verificação, caso realmente o banco de dados esteja consistente:

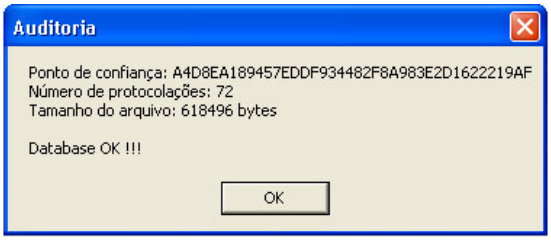

Figura 22 - Verificação do BD da PDDE.

O ponto de confiança nada mais é do que o primeiro link do BD.

b) Operações sobre o recibo

O usuário, ao selecionar um recibo, tem a opção de visualizar detalhes do mesmo, como a data e hora da datação (campo genTime). Todos estes campos estão descritos no RFC 3161.

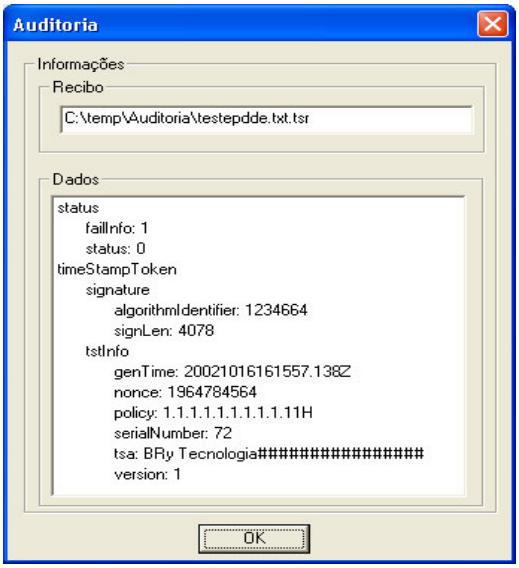

Figura 23 - Detalhes do recibo emitido pela PDDE.

Para realizar qualquer verificação sobre um recibo basta selecionar a operação desejada (verificar presença, verificar assinatura ou comparar) e pressionar o botão *Start*. Uma mensagem irá indicar o resultado da operação, como no exemplo abaixo:

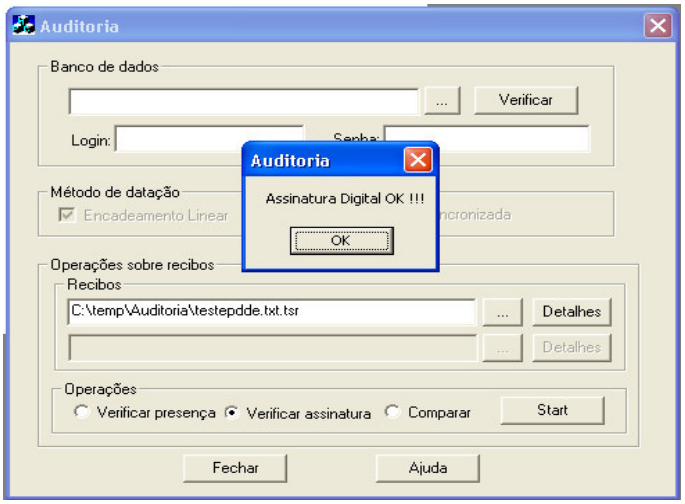

Figura 24 - Verificação da assinatura digital de um recibo.

5 Considerações finais

A datação de documentos eletrônicos é extremamente importante para resolver situações de disputa de documentos datados digitalmente. A AD possui extrema importância na datação, pois se trata da melhor forma de datação de documentos eletrônicos. Porém o principal problema da AD é confiar nela, ou seja, acreditar que a mesma não possua ramos maliciosos. Uma forma de aumentar a confiança em uma AD é submeter seu banco de dados ao processo de auditoria. Este artigo apresentou um método para realizar auditoria em uma protocoladora digital de documentos eletrônicos que utiliza como método de datação o de encadeamento linear, visando aumentar a confiança no processo de datação.

Se um método de auditoria for utilizado em um sistema de protocolização digital de documentos eletrônicos, o processo terá uma maior confiabilidade e com isso, documentos eletrônicos poderão ser considerados válidos juridicamente.

Referências

[BEN 92] BENALOH, J. Efficient broadcast time-stamping. **Clarkson University, Department of Math and Computer Science, [S.l.],** Abril, 1992.

[COS 02] COSTA, V., PIRES, C. A., DIAS, M. A. **Auditoria do Processo de Protocolização de Documento Eletrônicos**. Florianópolis, 2002.

[HAB 91] HABER, S. How to time-stamp a digital document. Journal of Cryptology, [S.l.], v.3, p.99–112, 1991.

[LIP 99] LIPMAA, H. SECURE AND EFFICIENT TIME-STAMPING SYSTEMS. University of Tartu - Estonia, July, 1999. Ph.d.

[PAS 01] PASQUAL, E. S. **IDDE – Uma infra-estrutura para datação de documentos eletrônicos**. Florianópolis, 2002.

ANEXO A – código fonte da PDDE

- **Protocol.cpp (cliente PDDE)**

```
BYTE* protocol::processMessage(BYTE *message, INT32 *len,TimeStampReq *req,char* documento)
{
	TimeStampResp_OK *resp; //estrutura com as informacoes da resposta do servidor

	resp = (TimeStampResp_OK *) message;
	if ( ! isValidResponse (resp, *len) )
	{
		*len = sizeof(TimeStampResp_FAIL);
		return badRequestResponse();
	}
	if(req->messageImprint.hashedMessage == resp->timeStampToken.tstInfo.messageImprint.hashedMessage)
	{
		if(verificaReqServer((BYTE*)documento,resp))
		{
			if(resp->timeStampToken.tstInfo.nonce == req->nonce)
			{
				return (BYTE*)resp;
			}
			else
			{
				cout << "O nounce recebido eh diferente do enviado" << endl;
				return (BYTE*)0;
			}
		}
		else
		{
			cout << "A assinatura esta incorreta !!!" << endl;
			return (BYTE*)0;
		}
	}
	else
	{
		cout << "Hash diferente do enviado" << endl;
		return (BYTE*)0;
	}
	return (BYTE*)resp;

}
```

- **ProtocoladoraDlg.cpp (cliente PDDE)**

```
void CProtocoladoraDlg::OnProtocolar()
{
	// TODO: Add your control notification handler code here
	char *resposta;
	BYTE *resp;
	protocol *newProtocol = new protocol;
	int *tamDoc,*tamHash;
	TimeStampReq *novoReq;

	newProtocol->estabelecaConexao(""150.162.66.115"",3000);
	char* req = newProtocol->processReq(documento,tamDoc);
	newProtocol->enviaReq((char*)req);
	resposta = newProtocol->respostaServidor();
	if(resposta == "")
	{
		this->MessageBox("A protocolação falhou !!!",NULL,MB_OK);
	}
	else
	{
		novoReq = (TimeStampReq*)req;
		resp = newProtocol->processMessage((BYTE*)resposta,tamHash,novoReq,documento);
		if(strcmp((const char*)resp,(const char*)0))
		{
			this->MessageBox("A protocolação falhou !!!",NULL,MB_OK);
		}
		else
		{
			this->MessageBox("A protocolação ocorreu corretamente !!!",NULL,MB_OK);
			FILE *arqRecibo;
			m_path_recevied = "recibo.pdde";
			if( (arqRecibo = fopen(m_path_recevied , "w+" )) != NULL )
			{
				if( fputs( (const char*)resposta, arqRecibo ) != NULL)
				fclose( arqRecibo );
			}
			this->OnSaveAs();
		}
	}

}
```

- **Arquivo crypto.h**

```
/*************************************************************************/
/*************************************************************************/
#ifndef CRYPTO_H
#define CRYPTO_H

#include "common.h"
#include <openssl/sha.h>
#include <openssl/rsa.h>
#include <openssl/pem.h>

/*************************************************************************/
class Crypto
{
        public:
                Crypto(char *__kr_file_name, char *__cer_file_name);
                ~Crypto();
                void set_kr(char *__kr_file_name);
                void set_cer(char *__cer_file_name);
                BYTE *SHAHash(BYTE *__msg, UINT64 __len);
                BYTE *RSASign(BYTE *__msg, UINT64 __len, UINT32 *__sign_len) throw
(Exception *);
                BYTE *PKCS7(BYTE *__msg, INT32 __msg_len, INT32 *__sign_pkcs7_len) throw
(Exception *);
                INT32 deleteFile(char *__path);
        protected:
                //Atributos
                char *_kr_file_name; //name of the file that contains the private key of thre...
                char *_cer_file_name; //name of the file that contains the certificate..
                //Metodos
                BYTE* parsePKCS7FromSmime(char *__smime_file_name, INT32 *__len) throw
(Exception *);
                void createSmimeFile(char *__hash_file_name, char *__smime_file_name) throw
(Exception *);
                void createFile1(char *__file_name, BYTE *__msg, INT32 __msg_len) throw
(Exception *);
                void createFile2(char *__file_name, BYTE *__msg, INT32 __msg_len) throw
(Exception *);
                BYTE* PKCS7ToPER(char *__pkcs7_file_name, INT32 *__ret_len) throw(Exception
*);
};

#endif
```

- **Arquivo crypto.cpp**

```
/*************************************************************************/
/*************************************************************************/
#include "crypto.h"
#include "common.h"
#include <unistd.h>
#include <fstream.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <string>
#include <iostream.h>

/*************************************************************************/
Crypto::Crypto(char *__kr_file_name, char *__cer_file_name)
{
        INT16 kr_size = strlen(__kr_file_name);
        INT16 cer_size = strlen(__cer_file_name);
        _kr_file_name = new char[kr_size];
        _cer_file_name = new char[cer_size];
        strcpy(_cer_file_name, __cer_file_name);
        strcpy(_kr_file_name, __kr_file_name);
}

/*************************************************************************/
Crypto::~Crypto()
{
         delete(_kr_file_name);
         delete(_cer_file_name);
}

/*************************************************************************/
void Crypto::set_kr(char *__kr_file_name)
{
        delete[] _kr_file_name;
        INT16 kr_size = strlen(__kr_file_name);
        _kr_file_name = new char[kr_size];
        strcpy(_kr_file_name, __kr_file_name);
}

/*************************************************************************/
void Crypto::set_cer(char *__cer_file_name)
{
        delete[] _cer_file_name;
        INT16 cer_size = strlen(__cer_file_name);
        _cer_file_name = new char[cer_size];
        strcpy(_cer_file_name, __cer_file_name);
}

/*************************************************************************/
```

```
BYTE* Crypto::SHAHash(BYTE *__msg, UINT64 __len)
{
        BYTE *ret_value = new BYTE[SHA_DIGEST_LENGTH + 1];
        SHA1(__msg, __len, ret_value);
        ret_value[SHA_DIGEST_LENGTH] = '\0';
        return ret_value;
}


/*************************************************************************/
BYTE* Crypto::RSASign(BYTE *__msg, UINT64 __len, UINT32 *__sign_len) throw (Exception *)
{
        RSA *rsa;
        FILE *fp = fopen(_kr_file_name,"rb");
        if(!fp)
          throw new Exception(IO_ERROR,"Crypto::RSASign(BYTE *, UINT64, char *)");

        rsa = PEM_read_RSAPrivateKey(fp, NULL,NULL,NULL);
        if(rsa == NULL) {
            fclose(fp);
            throw new Exception(OPENSSL_ERROR,"Crypto::RSASign(BYTE *, UINT64, char *)");
         }

        INT32 tam = RSA_size(rsa);
        BYTE *sigret = new BYTE[tam];
        INT32 assina  = RSA_sign(NID_sha1, __msg, __len, sigret,__sign_len,rsa);
        if(assina != 1) {
                 fclose(fp);
                throw  new  Exception(OPENSSL_ERROR,"Crypto::RSASign(BYTE  *,  UINT64,  char
*)");
        }

        fclose(fp);
        return sigret;
}

/*************************************************************************/
void Crypto::createFile1(char *__file_name, BYTE *__msg, INT32 __msg_len) throw (Exception *)
{
        FILE *out = fopen(__file_name,"wb");
        if(!out)
                throw new Exception(IO_ERROR,"Crypto::createFile(char *,BYTE *, INT32)");
        INT32 status = fwrite((char *)__msg,__msg_len,1,out);
        fclose(out);
        if(status == EOF)
                throw new Exception(IO_ERROR,"Crypto::createFile(char *,BYTE *, INT32)");
}

/*************************************************************************/
void Crypto::createFile2(char *__file_name, BYTE *__msg, INT32 __msg_len) throw (Exception *)
{
                 FILE *out = fopen(__file_name,"wb");
```

```
        if(!out)
                throw new Exception(IO_ERROR,"Crypto::createFile(char
*,BYTE *, INT32)");
        INT32 status = fputs((char *)__msg,out);
        fclose(out);
        if(status == EOF)
                throw new Exception(IO_ERROR,"Crypto::createFile(char
*,BYTE *, INT32)");
}


/*********************************************************************/
void Crypto::createSmimeFile(char *__hash_file_name, char *__smime_file_name) throw
(Exception *)
{
        if(fork() == 0)
        {
                execlp("openssl", "openssl","smime","-sign","-in",__hash_file_name,"-
out",__smime_file_name,"-signer", _cer_file_name,"-inkey",_kr_file_name, NULL);
                throw new Exception(OPENSSL_ERROR,"Crypto::PKCS7(BYTE *)");
        }
        else
                wait(0);
}

/*********************************************************************/
BYTE* Crypto::PKCS7(BYTE *__msg, INT32 __msg_len, INT32 *__sign_pkcs7_len) throw
(Exception *)
{
        BYTE *ret;

        //Gerar os nomes dos arquivos HASH, SMIME, PKCS7
        //*************************************************//
        //verificar se msg < 256 ...
        string hash_file_name = byteArrayToHexString(20, (BYTE *)__msg);
        string smime_file_name = hash_file_name;
        string pkcs7_file_name =  hash_file_name;
        hash_file_name.append(".hash");
        smime_file_name.append(".smime");
        pkcs7_file_name.append(".pkcs7");
        //*************************************************//
        createFile1((char *)hash_file_name.c_str(),__msg, __msg_len); //se acontecer uma
Excecao aqui, o fluxo eh desviado para a funcao chamadora...
        createSmimeFile((char *)hash_file_name.c_str(),(char *)smime_file_name.c_str()); // idem
        INT32 ret_len = 0;
        ret = parsePKCS7FromSmime((char *)smime_file_name.c_str(), &ret_len); //idem
        createFile2((char *)pkcs7_file_name.c_str(),ret, ret_len);
        if(ret) delete ret;
        ret = PKCS7ToPER((char *)pkcs7_file_name.c_str(), __sign_pkcs7_len);
        //Deletar arquivos criados
        //***********************//
        if(deleteFile((char *)hash_file_name.c_str()) == -1 )
```

```
                cout << "FALHA EM DELETAR ARQUIVO" << endl;
        if(deleteFile((char *)smime_file_name.c_str()) == -1)
                cout << "FALHA EM DELETAR ARQUIVO" << endl;
        if(deleteFile((char *)pkcs7_file_name.c_str()) == -1)
                cout << "FALHA EM DELETAR ARQUIVO" << endl;
        //**********************//
        return ret;
}

/*************************************************************************/
BYTE *Crypto::PKCS7ToPER(char *__pkcs7_file_name, INT32 *__ret_len) throw(Exception *)
{
        BYTE *ret_value;
        char temp_file_name[256];
        strcpy(temp_file_name,"sign");
        strcat(temp_file_name,__pkcs7_file_name);
        if(fork() == 0)
        {
                execlp("openssl",     "openssl","asn1parse","-inform","PER","-in",__pkcs7_file_name,"-
i","-dump","-out", temp_file_name,"-noout", NULL);
                throw new Exception(OPENSSL_ERROR,"Crypto::PKCS7ToPER(char *, INT32)");
        }
        else
        {
                wait(0);
                string str = "";
                char buffer[4000];
                                        int i=0;
                FILE *fd = fopen(temp_file_name,"rb");
                if( ! fd)
                        throw new Exception(IO_ERROR,"Crypto::parsePKCS7FromSmime(char *)");

                while(!feof(fd)){
                  fscanf(fd,"%c",&buffer[i]);
                                                i++;
                                        }

                *__ret_len = i-1;
                ret_value = new BYTE[*__ret_len];
                memcpy(ret_value,(char *)buffer, *__ret_len);
                fclose(fd);
                if(deleteFile(temp_file_name) == -1)
                   cout << "FALHA EM DELETAR ARQUIVO" << endl;
                return ret_value;
        }
}

/*************************************************************************/
INT32 Crypto::deleteFile(char *__path)
{
         return unlink(__path);
}
```

```
/*************************************************************************/
BYTE* Crypto::parsePKCS7FromSmime(char *__smime_file_name, INT32 *__len) throw (Exception
*)
{
        BYTE *ret_value;
        string str = "";
        char buffer[200];
        FILE *fd = fopen(__smime_file_name,"rb");
        if( ! fd)
                throw new Exception(IO_ERROR,"Crypto::parsePKCS7FromSmime(char *)");

        while(fgets(buffer, 199, fd))
                str += buffer;

        INT32 pos = str.rfind("smime.p7s");
         str = str.substr(pos,str.size() - pos);
         pos = str.find("\n\n");
         str = str.substr(pos+2,str.size() - pos);
         pos = str.rfind("\n\n-");
         str = str.substr(0,pos);
        *__len = str.size() + 1;
        ret_value = new BYTE[*__len];
        memcpy(ret_value, (char*)str.c_str(), *__len);
        fclose(fd);

         return ret_value;
}
```

- **Arquivo protocol.h**

```
/*************************************************************************/
/*************************************************************************/
#ifndef PROTOCOL_H
#define PROTOCOL_H

#include "database.h"
#include "crypto.h"
#include "common.h"
#include <time.h>
#include <string>

#define ALG_LINK 1
#define ALG_ARV_SINCRO 4

/*************************************************************************/
class Protocol
{
 public:
        BYTE* processMessage(BYTE *__message, INT32 *__len, bool *__closeConnection);
        Protocol( );
```

```
        ~Protocol( );
 protected:
        Database _db;
        Crypto *_crypto;

                bool isValidRequest(TimeStampReq *__req,  INT32 __len);
        time_t getTime(char *__parsedTime);
        BYTE* badRequestResponse();
        BYTE* systemFailureResponse();
        BYTE* systemFailureResponse(char *__cause);
        bool isValidTime(time_t __time);
        bool getAlgType(INT32 * __alg_type);
        bool getTable(INT32, char *);
        bool isSecurityModule();
  //    string byteArrayToHexString(INT32 __data_len, BYTE *__data);
};

#endif
```

- **Arquivo protocol.cpp**

```
/*****************************************************************************/
/*****************************************************************************/
#include "protocol.h"
#include <string.h>
#include <sys/timeb.h>

/*****************************************************************************/
Protocol::Protocol(){
  char path_cert[50];
  char path_key[50];
  _db.connectDatabase(Database::_path, Database::_user, Database::_pass);

  if( ! _db.select("select path_cert from config")) {
          cout << "erro no path_cert" << _db.getLastErrorMessage() << endl;
  }
  if(_db.fetch())
          _db.getFieldValue(0,path_cert);

  if( ! _db.select("select path_key from config")) {
          cout << "erro no path_key" << _db.getLastErrorMessage() << endl;
  }
  if(_db.fetch())
          _db.getFieldValue(0,path_key);

  _crypto = new Crypto(path_key,path_cert);
}

/*****************************************************************************/
Protocol::~Protocol(){
```

```
 _db.disconnectDatabase();
 if(_crypto) delete (_crypto);
}

/*************************************************************************/
bool Protocol::isValidRequest(TimeStampReq *__req, INT32 __len){
 if(__len != sizeof(TimeStampReq))
        return false;
 if(__req->version == 1)
        return true;
 return false;
}

/*************************************************************************/
BYTE* Protocol::badRequestResponse(){
 TimeStampResp_FAIL *resp = new TimeStampResp_FAIL();
 resp->status.status = REQUEST_REJECTED;
 resp->status.failInfo = BAD_DATA_FORMAT;
 BYTE *result = (BYTE *) resp;
 return result;
}

/*************************************************************************/
BYTE* Protocol::systemFailureResponse(char *__cause){
 TimeStampResp_FAIL *resp = new TimeStampResp_FAIL();
 resp->status.status = REQUEST_REJECTED;
 resp->status.failInfo = SYSTEM_FAILURE;
 strcpy(resp->cause, __cause);
 return (BYTE *) resp;
}

/*************************************************************************/
int retorneNumeroMes(char *__mes_str){
 if(strcmp(__mes_str,"Jan") == 0){
        return 1;
 }
 else if(strcmp(__mes_str,"Feb") == 0){
        return 2;
 }
 else if(strcmp(__mes_str,"Mar") == 0){
        return 3;
 }
 else if(strcmp(__mes_str,"Apr") == 0){
        return 4;
 }
 else if(strcmp(__mes_str,"May") == 0){
        return 5;
 }
 else if(strcmp(__mes_str,"Jun") == 0){
        return 6;
 }
 else if(strcmp(__mes_str,"Jul") == 0){
```

```
        return 7;
 }
 else if(strcmp(__mes_str,"Aug") == 0){
        return 8;
 }
 else if(strcmp(__mes_str,"Sep") == 0){
        return 9;
 }
 else if(strcmp(__mes_str,"Oct") == 0){
        return 10;
 }
 else if(strcmp(__mes_str,"Nov") == 0){
        return 11;
 }
 else if(strcmp(__mes_str,"Dec") == 0){
        return 12;
 }
 return 0; // nunca deve chegar aki
}

/*************************************************************************/
void parseTime(char *__time, UINT16 __mil, char *__return){
 char dia_sem[5];
 int ano;
 int mes;
 int dia;
 char mes_str[5];
 int hora;
 int min;
 int seg;
 sscanf(__time,"%s %s %d %d:%d:%d %d",dia_sem,mes_str,&dia,&hora, &min,&seg,&ano);
 mes = retorneNumeroMes(mes_str);
 sprintf(__return, "%d%.2d%.2d%.2d%.2d%.2d.%.3d%s",ano,mes,dia,hora,min,seg,__mil,"Z");
}


/*************************************************************************/
time_t Protocol::getTime(char *__parsedTime){
 struct timeb tp;
 ftime(&tp);
 char* time_str =  ctime(&tp.time);
 parseTime(time_str, tp.millitm, __parsedTime);
 return tp.time;
}

/*************************************************************************/
bool Protocol::isValidTime(time_t __time){
 char buf[100];
 sprintf(buf, "Select cod_req from requisicao where datasig > '%d' ",__time);
 _db.select(buf);
 if(_db.fetch()){
        cout <<"not isValidDate : " <<  _db.getLastErrorMessage() << endl;
```

```
        return false;
 }
 return true;
}

/*************************************************************************/
bool Protocol::getAlgType(INT32 * __alg_type){
 if( ! _db.select("select tipoalg from config")){
        cout <<"Erro no select tipoAlg : " <<  _db.getLastErrorMessage() << endl;
        return false;
 }
 if( _db.fetch()){
        _db.getFieldValue(0,(INT64 * ) __alg_type);

        return true;
 }
 cout << "Nao retornou nada do select tipo alg " << endl;
 return false;
}

/*************************************************************************/
bool Protocol::getTable(INT32 __alg_type, char * __table){
 char query[50];
 sprintf(query, "select tabela from tipoalgoritmo where cod_tipo = '%d'",__alg_type);
 if( ! _db.select(query)){
        cout <<"erro no select do getTable : " <<  _db.getLastErrorMessage() << endl;
        return false;
 }
 if( _db.fetch()) {
        cout << "Sucesso em getTable" << endl;
        _db.getFieldValue(0,__table);
        return true;
 }
 return false;
}

/*************************************************************************/
bool Protocol::isSecurityModule(){
 INT64 aux = 0;
 _db.select("select mod_confianca from config");
 if(_db.fetch())
        _db.getFieldValue(0, &aux);
 if(aux == 0)
        return false;
 else
        return true;
}

/*************************************************************************/
BYTE* Protocol::processMessage(BYTE *__message, INT32 *__len, bool *__closeConnection){
 BYTE *sign_pkcs7 = NULL;
 TimeStampReq *req;
```

```
 TimeStampResp_OK *resp;
 INT32 sign_pkcs7_len;
 INT64 recibo;
 char query[200];

 *__closeConnection = true;
 req = (TimeStampReq *) __message;


/*************************************************************************/
 if ( ! isValidRequest (req, *__len) ) {
        *__len = sizeof(TimeStampResp_FAIL);
        cout << "BAD REQUEST  !!! len: " << *__len << endl;
        return badRequestResponse();
 }

 *__len = sizeof(TimeStampResp_FAIL);
/*************************************************************************/
 char parsedTime[30];
 int i;
 for(i=0; i<30; i++)
        parsedTime[i]= '0';
 time_t timeInSeconds = getTime(parsedTime);
 i= 0;
 while (parsedTime[i]!= '\0')
        i++;
 parsedTime[i]= '0';

/*************************************************************************/
 if ( ! isValidTime(timeInSeconds)){
        cout << " ! isValidTime " << endl;
        return systemFailureResponse("Not valid time");
 }

/*************************************************************************/
 INT32 alg_type;
 char table[50];
 if( ! getAlgType(&alg_type)){
        string x = "getAlgType\n";
        x.append(_db.getLastErrorMessage());
        return systemFailureResponse((char*)x.c_str());
 }

/*************************************************************************/
 if(! _db.beginTransaction()) {
        cout <<"BEGIN TRANSACTION: " << _db.getLastErrorMessage() << endl;
        string x = "BeginTransaction\n";
        x.append(_db.getLastErrorMessage());
        return systemFailureResponse((char*)x.c_str());
 }

/*************************************************************************/
```

```
// switch case dos algoritmos
 switch(alg_type){
/*************************************************************************/
        case ALG_LINK :
                char last_link[41];
                BYTE *new_link;

/*************************************************************************/
                if( ! _db.select("Select recibo from config"))  {
                  string x = "Select recibo from config\n";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }
                if( ! _db.fetch() ) {
                  string x = "fetch Select recibo from config\n";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }
                _db.getFieldValue(0,&recibo);

/*************************************************************************/
                if(isSecurityModule()){
                  sprintf(query,"Select recibo from requisicao where recibo > %d",recibo);
                  if( ! _db.select(query) ) {
                          string x = "Select recibo from req\n";
                          x.append(_db.getLastErrorMessage());
                          return systemFailureResponse((char*) x.c_str());
                  }
                  if(_db.fetch()){
                          string x = "fetch Select recibo from req\n";
                          x.append(_db.getLastErrorMessage());
                          return systemFailureResponse((char*) x.c_str());
                  }
                }

/*************************************************************************/
                if( ! _db.select("select link from link order by cod_link desc")) {
                  string x = "Select link\n";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }
                if(!_db.fetch()){
                  string x = "fetch Select link\n";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }
                _db.getFieldValue(0,last_link);

/*************************************************************************/
                string hash_recebido_hex = req->messageImprint.hashedMessage;
                string temp = hash_recebido_hex;
                temp = temp + last_link;
```

```
                new_link = _crypto->SHAHash((BYTE*) temp.c_str(), temp.size());

/*************************************************************************/
                string new_link_hex = byteArrayToHexString(20, new_link);
                sprintf(query,"insert into link (recibo,data,hash,link) values
(%d,%d,'%s','%s')",recibo,timeInSeconds,(char*)hash_recebido_hex.c_str(),(char
*)new_link_hex.c_str() );
                if( ! _db.insert(query) ) {
                  cout << "insert into link" << _db.getLastErrorMessage() << endl;
                  string x = "insert into link\n";
                  x.append(_db.getLastErrorMessage());
                  delete[] new_link;
                  return systemFailureResponse((char*) x.c_str());
                }

/*************************************************************************/
                sprintf(query,"insert into requisicao (recibo,datasig,hash,id_request) values
(%d,%d,'%s',%d)",recibo,timeInSeconds,(char *)hash_recebido_hex.c_str(), req->nonce);
                if( ! _db.insert(query) ) {
                  string x = "insert into req\n";
                  x.append(_db.getLastErrorMessage());
                  delete[] new_link;
                  return systemFailureResponse((char*) x.c_str());
                }

/*************************************************************************/
                sprintf(query,"update config set recibo = %d", recibo + 1);
                if(! _db.update(query)) {
                  string x = "Update config ";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }

/*************************************************************************/
                resp = new TimeStampResp_OK();
                resp->status.status = PKI_GRANTED;
                strcpy(resp->timeStampToken.tstInfo.genTime, parsedTime);
                resp->timeStampToken.tstInfo.version = 1;
                strcpy(resp->timeStampToken.tstInfo.policy, OID_DEFAULT);
                resp->timeStampToken.tstInfo.serialNumber = recibo;
                resp->timeStampToken.tstInfo.messageImprint  = req->messageImprint;
                strcpy(resp->timeStampToken.tstInfo.link.algorithmIdentifier ,OID_SHA1);
                memcpy(resp-
>timeStampToken.tstInfo.link.hashedMessage,(char*)new_link_hex.c_str(), 40);
                resp->timeStampToken.tstInfo.nonce = req->nonce;
                strcpy(resp->timeStampToken.tstInfo.tsa, TSA_DEFAUT);

/*************************************************************************/
                char temp_version[2], temp_serial[10], temp_nonce[10];
                sprintf(temp_version,"%d",resp->timeStampToken.tstInfo.version);
                sprintf(temp_serial,"%d",recibo);
                sprintf(temp_nonce,"%d",resp->timeStampToken.tstInfo.nonce);
```

```
                int tam_array[9];
                tam_array[0]= sizeof(resp->timeStampToken.tstInfo.genTime);
                tam_array[1]= strlen(temp_version) + tam_array[0];
                tam_array[2]= sizeof(resp->timeStampToken.tstInfo.policy) + tam_array[1];
                tam_array[3]= strlen(temp_serial) + tam_array[2];
                tam_array[4]=                                                    sizeof(resp-
>timeStampToken.tstInfo.messageImprint.algorithmIdentifier) + tam_array[3];
                tam_array[5]=                                                    sizeof(resp-
>timeStampToken.tstInfo.messageImprint.hashedMessage) + tam_array[4];
                tam_array[6]=    sizeof(resp->timeStampToken.tstInfo.link.algorithmIdentifier)    +
tam_array[5];
                tam_array[7]=       sizeof(resp->timeStampToken.tstInfo.link.hashedMessage)       +
tam_array[6];
                tam_array[8]= strlen(temp_nonce) + tam_array[7];
                tam_array[9]= sizeof(resp->timeStampToken.tstInfo.tsa) + tam_array[8];

                INT16 tam_pacote = tam_array[9];
                char* pacote  = new char[tam_pacote];
                memcpy(pacote,          resp->timeStampToken.tstInfo.genTime,          sizeof(resp-
>timeStampToken.tstInfo.genTime));
                memcpy(&pacote[tam_array[0]], temp_version, sizeof(temp_version));
                memcpy(&pacote[tam_array[1]],  resp->timeStampToken.tstInfo.policy,  sizeof(resp-
>timeStampToken.tstInfo.policy));
                memcpy(&pacote[tam_array[2]], temp_serial, sizeof(temp_serial));
                memcpy(&pacote[tam_array[3]],                                          resp-
>timeStampToken.tstInfo.messageImprint.algorithmIdentifier,                    sizeof(resp-
>timeStampToken.tstInfo.messageImprint.algorithmIdentifier));
                memcpy(&pacote[tam_array[4]],                                          resp-
>timeStampToken.tstInfo.messageImprint.hashedMessage,                          sizeof(resp-
>timeStampToken.tstInfo.messageImprint.hashedMessage));
                memcpy(&pacote[tam_array[5]],                                          resp-
>timeStampToken.tstInfo.link.algorithmIdentifier,                              sizeof(resp-
>timeStampToken.tstInfo.link.algorithmIdentifier));
                memcpy(&pacote[tam_array[6]], resp->timeStampToken.tstInfo.link.hashedMessage,
sizeof(resp->timeStampToken.tstInfo.link.hashedMessage));
                memcpy(&pacote[tam_array[7]], temp_nonce, sizeof(temp_nonce));
                memcpy(&pacote[tam_array[8]],    resp->timeStampToken.tstInfo.tsa,    sizeof(resp-
>timeStampToken.tstInfo.tsa));
                try{
                  sign_pkcs7 = _crypto->PKCS7((BYTE*) pacote , tam_pacote, &sign_pkcs7_len);
                }
                catch (Exception *e){
                  if(sign_pkcs7) delete sign_pkcs7;
                  delete[] pacote;
                  delete[] new_link;
                  delete e;
                  string x = "PKCS7\n";
                  x.append(_db.getLastErrorMessage());
                  return systemFailureResponse((char*) x.c_str());
                }

/*****************************************************************/
```

```
                strcpy(resp->timeStampToken.signature.algorithmIdentifier ,OID_SHA1);
                resp->timeStampToken.signature.detachedSignature.sign_len = sign_pkcs7_len;
                BYTE                *sign                =(BYTE*)                &resp-
>timeStampToken.signature.detachedSignature.signature;
                memcpy(sign, sign_pkcs7, sign_pkcs7_len);

/*************************************************************************/
                delete[] pacote;
                delete[] sign_pkcs7;
                delete[] new_link;
                *__len = sizeof(TimeStampResp_OK);
                break;
/*************************************************************************/

 } //end Switch

 _db.commit();
 return (BYTE *) resp;
}
```

- **Arquivo common.h**

```
/*************************************************************************/
/*************************************************************************/
#include <stdlib.h>
#include <string>

#ifndef COMMONFUNCTIONSH
#define COMMONFUNCTIONSH
/*=======================================================================
                                                    CONSTANTS
=======================================================================*/
#define SUCCESS 1
#define MAX_TRANSFERED_DATA_SIZE 10000 // Número máximo de bytes transferidos
#define CONNECT_ERROR -1
#define IO_ERROR -2
#define HOST_UNKNOWN -3
#define _SOCKET_ERROR -4
#define CLOSE_SOCKET_ERROR -5
#define WINSOCK_ERROR -6
#define REQUEST_REJECTED 2
#define BAD_DATA_FORMAT 5
#define SYSTEM_FAILURE 25
#define OPENSSL_ERROR -98
#define MALLOC 1 // qdo um ponteiro eh alocado com malloc
#define NEW 2   // qdo um ponteiro eh alocado com new

//LINUX

#define THREAD_ERROR -9
```

```
#define OID_DEFAULT "1.1.1.1.1.1.1.1.11"
#define OID_SHA1 "00.0.0.1.3.14.3.2.26"
#define MAX_SIGNATURE_LEN 5000
#define PKI_GRANTED 0
#define TSA_LEN 10
#define TSA_DEFAUT "PDDELabSEC"
#define SHA_HASH_LEN 40
#define DATE_LEN 30
#define INVALID_SOCKET -1
#define SOCKET_ERROR -1
#define BIND_ERROR -9
#define BACKLOG 128
#define LISTEN_ERROR -17
#define ACCEPT_ERROR -23
#define OPENSSL_ERROR -78
/*====================================================================*/

/*====================================================================
                                                        TYPEDEFs
====================================================================*/
typedef unsigned char BYTE;
typedef unsigned long UINT64;
typedef short INT16;
typedef unsigned short UINT16;
typedef int INT32;
typedef long INT64;
typedef unsigned int UINT32;
typedef INT16 AlgorithmIdentifier;
typedef INT16 PKIStatus;
typedef INT16 PKIFailureInfo;
typedef char OID[20];

typedef struct sMessageImprint
{
        OID algorithmIdentifier;
        char hashedMessage[SHA_HASH_LEN];
}MessageImprint;

typedef struct sLinkProtocolStruct
{
        INT32 sign_len;
        BYTE signature[MAX_SIGNATURE_LEN];
}LinkProtocolStruct;

typedef struct sSignature
{
        OID algorithmIdentifier;
        LinkProtocolStruct detachedSignature;
}Signature;

typedef struct sTSTInfo
{
```

```
        char genTime[DATE_LEN];
        INT16 version;
        OID policy;
        UINT32 serialNumber;
        MessageImprint messageImprint;
    MessageImprint link;
        UINT64 nonce;
        char tsa[TSA_LEN];
}TSTInfo;

typedef struct sPKIStatusInfo
{
        PKIStatus status;
        PKIFailureInfo failInfo;
}PKIStatusInfo;

typedef struct sTimeStampToken
{
        TSTInfo tstInfo;
        Signature signature;
}TimeStampToken;

typedef struct sTimeStampReq
{
        INT16 version;
        OID reqPolicy;
        MessageImprint messageImprint;
        bool certReq;
        UINT64 nonce;
}TimeStampReq;

typedef struct sTimeStampResp_OK
{
        PKIStatusInfo status;
        TimeStampToken timeStampToken;
}TimeStampResp_OK;

typedef struct sTimeStampResp_FAIL
{
        PKIStatusInfo status;
        char cause[300];
}TimeStampResp_FAIL;

/*=========================================================================*/

class Exception
{
protected:
        int codError;
        char *where;
public:
        Exception(int error, char* local);
```

```
        int getErrorCode();
        char* getWhere();
};


//Funções úteis que não pertencem a nenhuma classe...
void initApp() throw (Exception *);
void endApp();
void freePointer(void *__pointer);
string byteArrayToHexString(int __len, BYTE *__data);
int HexStringToByteArray(char * input_text, char * output_buffer);
```

ANEXO B – Código fonte do sistema de auditoria

- **Arquivo checkdb.cpp**

```
#include "checkdb.h"
#include "sha1.c"
//------------------------------------------------------


DB_Checker::DB_Checker() {

   db_loader = new Database();
 //  crypto    = new Crypto();

};

//------------------------------------------------------


DB_Checker::DB_Checker(char* db_path) {
   db_loader = new Database();
   db_loader->connectDatabase(db_path,"SYSDBA","masterkey");
};

//------------------------------------------------------


DB_Checker::~DB_Checker() {
   db_loader->disconnectDatabase();
   delete(db_loader);

};

//------------------------------------------------------


bool DB_Checker::loadDataBase() {
```

```
   char ret = 0;
   char query[200];
   char hash[SHA_HASH_LEN];
   char hex_hash[(2*SHA_HASH_LEN) + 1];


   if( db_loader->select("select recibo from link order by recibo desc") ) {
         if(db_loader->fetch()) {
            last_ticket=0;
            db_loader->getFieldValue(0,&last_ticket);
            ret++;
         }
   }

   if( db_loader->select("select recibo from link order by recibo asc") ) {
         if(db_loader->fetch()) {
            first_ticket=0;
            db_loader->getFieldValue(0,&first_ticket);
            ret++;
         }
   }


   sprintf(query,"select hash from link where recibo=%d",last_ticket);
   if( db_loader->select(query) )
{
         if(db_loader->fetch())
{
            db_loader->getFieldValue(0,hex_hash);
            HexStringToByteArray(hex_hash,hash);
            memcpy(last_hash,hash,SHA_HASH_LEN);
            ret++;
         }
   }

   sprintf(query,"select hash from link where recibo=%d",first_ticket);
   if( db_loader->select(query) )
{
         if(db_loader->fetch())
{
            db_loader->getFieldValue(0,hex_hash);
            HexStringToByteArray(hex_hash,hash);
            memcpy(first_hash,hash,SHA_HASH_LEN);
            ret++;
         }
   }

   sprintf(query,"select link from link where recibo=%d",last_ticket);
   if( db_loader->select(query) )
{
         if(db_loader->fetch())
```

```
{
            db_loader->getFieldValue(0,hex_hash);
            HexStringToByteArray(hex_hash,hash);
            memcpy(last_link,hash,SHA_HASH_LEN);
            ret++;
        }
    }

    sprintf(query,"select link from link where recibo=%d",first_ticket);
    if( db_loader->select(query) )
{
        if(db_loader->fetch())
{
            db_loader->getFieldValue(0,hex_hash);
            HexStringToByteArray(hex_hash,hash);
            memcpy(first_link,hash,SHA_HASH_LEN);
            ret++;
        }
    }

    sprintf(query,"select data from link where recibo=%d",last_ticket);
    if( db_loader->select(query) )
{
        if(db_loader->fetch())
{
            db_loader->getFieldValue(0,last_time);

            ret++;
        }
    }

    sprintf(query,"select data from link where recibo=%d",first_ticket);
    if( db_loader->select(query) )
{
        if(db_loader->fetch())
{
            db_loader->getFieldValue(0,first_time);

            ret++;
        }
    }

    return (ret == 8);

};

//-----------------------------------------------------


void DB_Checker::calcLink(char* new_link, char* hash, char* last_link) {

    BYTE  aux[2*SHA_HASH_LEN];
```

```
   BYTE link[SHA_HASH_LEN];

   memcpy(aux,hash,SHA_HASH_LEN);
   memcpy(&aux[SHA_HASH_LEN],last_link,SHA_HASH_LEN);
         calcSHA(link,aux,(2*SHA_HASH_LEN));
   //link = crypto->SHAHash(aux, (2*SHA_HASH_LEN));
   memcpy(new_link,link,SHA_HASH_LEN);
  // delete[] link;

};

//-----------------------------------------------------


void DB_Checker::parseTimeToInt(Time_Struct* T, char* time) {

   char ano[5]; char mes[3]; char dia[3];
   char hor[3]; char min[3]; char sec[3];
   char mse[4];

   memcpy(ano, time,       4);   ano[4]='\0';
   memcpy(mes, &(time[4]),  2);         mes[2]='\0';
   memcpy(dia, &(time[6]),  2);         mes[2]='\0';
   memcpy(hor, &(time[8]),  2);         mes[2]='\0';
   memcpy(min, &(time[10]), 2);         mes[2]='\0';
   memcpy(sec, &(time[12]), 2);         mes[2]='\0';
   memcpy(mse, &(time[15]), 3);         mse[3]='\0';

   T->ano = ((unsigned short int) atoi(ano));
   T->mes = ((unsigned short int) atoi(mes));
   T->dia = ((unsigned short int) atoi(dia));
   T->hor = ((unsigned short int) atoi(hor));
   T->min = ((unsigned short int) atoi(min));
   T->sec = ((unsigned short int) atoi(sec));
   T->mse = ((unsigned short int) atoi(mse));

};

//-----------------------------------------------------


bool DB_Checker::checkDateError(char* first, char* next) {

   Time_Struct T1,T2;

   this->parseTimeToInt(&T1,first);
   this->parseTimeToInt(&T2,next);

   if (T1.ano > T2.ano) return(true);
   if (T1.ano < T2.ano) return(false);

   if (T1.mes > T2.mes) return(true);
```

```
    if (T1.mes < T2.mes) return(false);

    if (T1.dia > T2.dia) return(true);
    if (T1.dia < T2.dia) return(false);

    if (T1.hor > T2.hor) return(true);
    if (T1.hor < T2.hor) return(false);

    if (T1.min > T2.min) return(true);
    if (T1.min < T2.min) return(false);

    if (T1.sec > T2.sec) return(true);
    if (T1.sec < T2.sec) return(false);

    if (T1.mse > T2.mse) return(true);
    if (T1.mse < T2.mse) return(false);

    return(false);

};

//-----------------------------------------------------


bool DB_Checker::checkDBLink() {

    bool ret = true;
    char query[200];


    char hash [SHA_HASH_LEN]; char next_hash [SHA_HASH_LEN];
    char next_hash_hex[(2*SHA_HASH_LEN) + 1];

    char cLink[SHA_HASH_LEN];
    BYTE link [SHA_HASH_LEN]; char next_link [SHA_HASH_LEN];
    char next_link_hex[(2*SHA_HASH_LEN) + 1];

    char time [DATE_LEN];     char next_time [DATE_LEN];

    memcpy(hash, first_hash, SHA_HASH_LEN);
    memcpy(link, first_link, SHA_HASH_LEN);
    memcpy(time, first_time, DATE_LEN);

    for (unsigned int i = (first_ticket + 1); i < last_ticket; i++) {

        sprintf(query,"select hash, link, data from link where recibo=%d",i);
        if( db_loader->select(query) )
 {
            if(db_loader->fetch())
{
                db_loader->getFieldValue(0,next_hash_hex);
                HexStringToByteArray(next_hash_hex,next_hash);
```

```
                db_loader->getFieldValue(1,next_link_hex);
                HexStringToByteArray(next_link_hex,next_link);
                db_loader->getFieldValue(2,next_time);

                calcLink(cLink,next_hash,(char*)link);
                if ( (!(memcmp(cLink,next_link,SHA_HASH_LEN) == 0)) ) {
                   ret = false;
                   printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
                            next_hash_hex,
                            "++++++++++++++++++++++++++++++++++++++++",
                            (byteArrayToHexString(SHA_HASH_LEN,(BYTE*) link)).c_str(),
                            "----------------------------------------",
                            (byteArrayToHexString(SHA_HASH_LEN,(BYTE*) cLink)).c_str(),
                            "<><><><><><><><><><><><><><><><><><><>",
                            (byteArrayToHexString(SHA_HASH_LEN,(BYTE*) next_link)).c_str()
                           );
                   break;
                }

                if  ( checkDateError(time,next_time) ) {
                   ret = false;
                   printf("%s > %s\n\n", time, next_time);
                   break;
                }

                memcpy(hash,next_hash,SHA_HASH_LEN);
                memcpy(link,next_link,SHA_HASH_LEN);
                memcpy(time,next_time,DATE_LEN);
             }
          }

   }

   return(ret);

};

int DB_Checker::checkTicketExistsByHash(tHash  h)
{
   char query[200];
   char hash[41];
   int pos = 0;
   sprintf(query,"select hash from link order by recibo asc");
   if( db_loader->select(query) )
   {
           while(db_loader->fetch())
      {
              db_loader->getFieldValue(0,hash);
              if(strcmp(hash,h) == 0)
         {
            return pos;
         }
```

```
        pos++;
          }
   }
   return -1;
}

bool DB_Checker::getDBInfo(char *info)
{

   info[0] = '\0';
   strcat(info,"Ponto de confiança: ");
   strcat(info,(byteArrayToHexString(SHA_HASH_LEN,(BYTE*) first_link)).c_str());
   strcat(info,"\nNúmero de protocolações: ");
   char temp[10];
   sprintf(temp,"%d",last_ticket - first_ticket + 1);
   strcat(info,temp);
   return true;
}

//-------------------------------------------------
```

- **Arquivo checkdb.h**

```
#ifndef CHECKDBPROT_H
#define CHECKDBPROT_H

#include <time.h>
#include <string.h>
#include <fcntl.h>
//#include <unistd.h>
#include <sys/timeb.h>

#include "common.h"
#include "database.h"
//#include "crypto.h"

//-------------------------------------------------


typedef _INT64    tNumber;
typedef char*   tHash;
typedef char*    tLink;
typedef char*    tTime;

//-------------------------------------------------


class DB_Checker {

   protected:
```

```
//Database parameters------------------------------------------------

        _INT64 first_ticket;
        char  first_hash  [SHA_HASH_LEN];
        char  first_link  [SHA_HASH_LEN];
        char  first_time  [DATE_LEN];

        _INT64 last_ticket;
        char  last_hash   [SHA_HASH_LEN];
        char  last_link   [SHA_HASH_LEN];
        char  last_time           [DATE_LEN];

//--------------------------------------------------------------------------

   private:

        void calcLink(char* new_link, char* hash, char* last_link);
        void parseTimeToInt(Time_Struct* T, char* time);
        bool checkDateError(char* first, char* next);

   public:

        Database* db_loader;
        bool loadDataBase();

        DB_Checker ( );
        DB_Checker (char* db_path);
        ~DB_Checker( );

        bool checkDBLink();

        bool checkTicketExists              (tNumber n);
        int checkTicketExistsByHash(tHash  h);
        bool checkTicketExistsByLink(tLink  l);
        bool checkTicketExistsByDate(tTime  t);
        bool checkTicketExists      (tNumber n, tHash h, tLink l, tTime t);

        bool checkTicketRecover     (tNumber n);
        bool checkTicketRecoverByHash(tHash  h);
        bool checkTicketRecoverByLink(tLink  l);
        bool checkTicketRecoverByDate(tTime  t);
        bool checkTicketRecover     (tNumber n, tHash h, tLink l, tTime t);

        bool checkTicketOrder     (tNumber n1, tNumber n2);
        bool checkTicketOrderByHash(tHash   h1, tHash h2);
        bool checkTicketOrderByLink(tLink   l1, tLink l2);
        bool checkTicketOrder     (tNumber n1, tHash h1, tLink l1,
                                   tNumber n2, tHash h2, tLink l2 );

   bool getDBInfo(char *info);
};
```