

**Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística  
Curso de Bacharelado em Ciência da Computação**

# **Sistema para Validação e Visualização de Certificados Digitais**

**Bruno Maluche Neto  
Iomani Engelmann Gomes**

Florianópolis, 03 de fevereiro 2003

**Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística  
Curso de Bacharelado em Ciência da Computação**

# **Sistema para Validação e Visualização de Certificados Digitais**

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

Autores: **Bruno Maluche Neto**  
**Iomani Engelmann Gomes**

Orientadora: **Denise Demétrio**  
Co-Orientador: **Pro.º Ricardo F. Custódio**  
Banca Examinadora: **Fabiano Castro Pereira**  
**Júlio da Silva Dias**

Florianópolis, 03 de fevereiro 2003

## **Agradecimentos**

Agradecemos a nossa família, principalmente nossos pais, que sempre nos deram muito apoio para a conquista deste trabalho.

Agradecemos também nossos amigos que sempre nos mantiveram animados e confiantes no sucesso deste trabalho.

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

— Albert Einstein

# Sumário

<i>Lista de Figuras</i> .....	<i>vii</i>
<i>Lista de Tabelas</i> .....	<i>viii</i>
<i>Lista de Abreviaturas</i> .....	<i>ix</i>
<i>Resumo</i> .....	<i>x</i>
<i>Abstract</i> .....	<i>xi</i>
<i>Justificativa</i> .....	<i>xii</i>
<b>Capítulo 1</b> .....	<b>13</b>
<i>Introdução</i> .....	<b>13</b>
<b>1.1</b> <b>Objetivos</b> .....	<b>14</b>
<b>1.2</b> <b>Motivação</b> .....	<b>14</b>
<b>Capítulo 2</b> .....	<b>16</b>
<i>Fundamentação Teórica</i> .....	<b>16</b>
<b>2.3</b> <b>Criptografia simétrica</b> .....	<b>18</b>
<b>2.4</b> <b>Criptografia assimétrica</b> .....	<b>20</b>
<b>2.5</b> <b>Funções Resumo (Hash)</b> .....	<b>21</b>
<b>2.6</b> <b>Assinatura Digital</b> .....	<b>22</b>
2.6.1    Descrição .....	<b>24</b>
2.6.2    RSA (Rivest, Shamir e Adleman) aplicado a Assinatura Digital .....	<b>26</b>
<b>2.7</b> <b>Padrões de Criptografia de Chave Pública</b> .....	<b>26</b>
2.7.1    PKCS#7 v1.6- Padrão de Criptografia de Mensagem .....	<b>27</b>
2.7.2    PKCS#10 v1.7- Padrão de Requisição de Certificado .....	<b>27</b>
<b>2.8</b> <b>ASN.1</b> .....	<b>28</b>
Tabela1. Tipos com rótulos da classe Universal. ....	<b>29</b>
2.8.1    BER .....	<b>31</b>
2.8.2    DER .....	<b>32</b>
<b>Capítulo 3</b> .....	<b>34</b>
<i>Infra-estrutura de Chaves-Públicas</i> .....	<b>34</b>
<b>3.2</b> <b>Autoridades Certificadoras (ACs)</b> .....	<b>35</b>
<b>3.3</b> <b>Certificados Digitais</b> .....	<b>37</b>
3.3.1    Descrição .....	<b>37</b>
<b>3.4</b> <b>Lista de Certificados Revogados (LCR)</b> .....	<b>38</b>
<b>3.5</b> <b>Componentes ICP</b> .....	<b>40</b>
3.5.1    Servidor de Certificados .....	<b>41</b>
3.5.2    Servidor de diretórios .....	<b>41</b>

3.5.3	Servidor para recuperação de chaves	42
3.5.4	Aplicações ICP	42
3.5.4.1	S/Mine	43
3.5.4.2	SSL	43
3.5.4.3	IPSEC	43
3.5.4.4	SET	44
<b>Capítulo 4</b>		<b>45</b>
<b>Recomendação X.509</b>		<b>45</b>
4.1	<b>Introdução</b>	<b>45</b>
4.2	<b>Certificados X509</b>	<b>45</b>
4.2.1	<b>Estrutura Interna</b>	<b>46</b>
4.2.2	<b>Extensões</b>	<b>49</b>
	Tabela 2. Estrutura de uma LCR	51
4.4	<b>Construção e Validação do Caminho de Certificação</b>	<b>51</b>
4.4.1	Modelo hierárquico	52
4.4.2	Validação de um certificado digital X.509	54
<b>Capítulo 5</b>		<b>59</b>
<b>Sistema de Validação e Visualização de um Certificado Digital</b>		<b>59</b>
5.2	<b>Detalhes do Projeto</b>	<b>59</b>
5.3	<b>Persistência de dados</b>	<b>63</b>
5.4	<b>Use Cases</b>	<b>64</b>
5.5	<b>Modelo conceitual</b>	<b>65</b>
5.6	<b>Visualização dos campos do Certificado</b>	<b>66</b>
5.7	<b>Exemplos</b>	<b>67</b>
5.7.1	Caminho de Certificação Válido	68
5.7.2	Caminho de Certificação Inválido	70
<b>Capítulo 6</b>		<b>72</b>
6.1	<b>Conclusão</b>	<b>72</b>
<b>Bibliografia</b>		<b>74</b>
<b>Anexos</b>		<b>76</b>
	<b>ANEXO 1: Artigo do Trabalho</b>	<b>77</b>
	<b>ANEXO 2: SENTENÇAS SQL PARA CRIAÇÃO DO BANCO DE DADOS</b>	<b>86</b>
	<b>ANEXO 3: CÓDIGO FONTE</b>	<b>88</b>

## Lista de Figuras

Figura 1-Criptografia Simétrica .....	19
Figura 2-Criptografia Assimétrica .....	21
Figura 3-Como funciona a função Hash .....	22
Figura 4-Função Hash com a assinatura digital .....	25
Figura 5-Requisitos para Internet segura .....	34
Figura 6-Arquitetura de uma AC .....	36
Figura 7-Estrutura básica de uma LCR.....	39
Figura 8-Arquitetura ICP .....	40
Figura 9-Caminho de certificação em uma arquitetura hierárquica.....	53
Figura 10- Exemplo de árvore de certificação .....	54
Figura 11-Etapas da validação de um certificado digital.....	55
Figura 12- Estruturas(classes) utilizadas para criação de um certificado X.509 .....	60
Figura 13-Tela Inicial .....	61
Figura 14- Classes utilizadas para a montagem e validação do caminho de certificação. 62	
Figura 15-Caminho de Certificação.....	63
Figura 16 -Use Cases .....	64
Figura 17 –Modelo Conceitual .....	66
Figura 18 –Visualização dos atributos do certificado.....	67
Figura 19–Inserindo um certificado.....	68
Figura 20 –Informações básica do certificado .....	69
Figura 21 –Visualização de um caminho de certificação válido .....	70
Figura 22 –Visualização de um caminho de certificação inválido .....	71

## **Lista de Tabelas**

Tabela1. Tipos com rótulos da classe Universal.....	29
Tabela 2. Estrutura de uma LCR.....	51



## **Lista de Abreviaturas**

<b>AC</b>	<i>-Autoridade Certificadora</i>
<b>ANSI</b>	<i>-American National Standards Institute</i>
<b>ASN.1</b>	<i>-Abstract Syntax Notation One</i>
<b>BER</b>	<i>- Basic Encoding Rules</i>
<b>BLOB</b>	<i>-Binary Large Objects</i>
<b>DER</b>	<i>-Distinguished Encoding Rules</i>
<b>DES</b>	<i>-Data Encryption Standard</i>
<b>IETF</b>	<i>- Internet Engineering Task Force</i>
<b>IPSEC</b>	<i>-Internet Protocol Security</i>
<b>ISO</b>	<i>-International Standards Organization</i>
<b>KDC</b>	<i>-Key Distribution Center</i>
<b>LCR</b>	<i>-Lista de Certificados Revogados</i>
<b>LDAP</b>	<i>-Lightweight Directory Access Protocol</i>
<b>PEM</b>	<i>-Privacy-Enhanced Mail</i>
<b>PGP</b>	<i>-Pretty Good Privacy</i>
<b>PKCS</b>	<i>-Public Key Cryptography Standard</i>
<b>PKI</b>	<i>-Public key Infrastructure</i>
<b>RFC</b>	<i>-Request For Comments</i>
<b>RSA</b>	<i>-Rivest, Shamir e Adleman</i>
<b>SET</b>	<i>-Security Electronic Transaction</i>
<b>SSL</b>	<i>-Secure Socket Layer</i>
<b>S/MIME</b>	<i>-Secure/Multipurpose Internet Mail Extension</i>
<b>TCP/IP</b>	<i>-Transfer Control Protocol/Internet Protocol</i>

# Resumo

O trabalho tem como intenção desenvolver um sistema para validação e visualização de Certificados Digitais X.509.

A validação será feita através da construção da árvore de certificação digital. O sistema permitirá que o usuário tenha uma base de dados com suas entidades certificadoras de confiança. Também será possível a visualização gráfica do caminho de certificação digital, assim como seu conteúdo.

Seu diferencial sobre outros sistemas existentes, é que este se trata de um sistema com código fonte aberto, sujeito a análises e alterações por parte de seus usuários. Por este motivo pode ser auditado e saber se realmente faz o que diz que faz.

Palavras-Chaves: Certificação Digital, Código Aberto, Criptografia, ASN.1.

# Abstract

The work was done with a view to develop a system to validation and visualization digital certificates X.509.

The validation will be done through building a digital certification's tree. The system will allow that the user has database with his secure certifier's entity. It will also be possible a graphic visualization of digital certification's path and its contents too.

The differential, in comparison with others existing systems, is that this one has open source code, which is possible that the users make analysis and alterations. So, for this reason, you can analyse it and know if it really do what it is exposed.

Key-words: Digital Certification, Open Source Code, Cryptography, ASN.1.

# Justificativa

Atualmente não existe no Brasil um software que garanta a validação e a integridade de um certificado digital, somado a isso, os softwares estrangeiros existentes para este propósito não possuem código aberto, o que impossibilita a garantia da sua integridade.

A segurança está baseada no conhecimento do programa fonte. Se o código fonte é conhecido, pode-se aceitá-lo e saber se faz o que diz que faz.

O projeto está sendo desenvolvido em parceria com o LABSEC/UFSC, e tem como objetivo final garantir a autonomia brasileira em relação às ferramentas para criptografia e certificação digital. Sendo assim nosso sistema será utilizado por todos que desejem validar e visualizar a árvore de certificação dos certificados digitais no Brasil.

A área de segurança digital está em franco crescimento no Brasil, e não existem muitos profissionais qualificados a atender a demanda do mercado, sendo assim, está é uma ótima oportunidade para aprendermos muitos aspectos essenciais e nos especializarmos em outros.

# Capítulo 1

## Introdução

Com a atual “explosão” da Internet, surgiram uma infinidade de facilidades para o acesso a informação. Entre essas facilidades podemos citar o comércio eletrônico e as transações bancárias através dos *home-bankings*. Contudo, essa facilidade de acesso a informação trouxe um grande problema, pois a Internet não possuía meios eficientes para garantir a segurança eletrônica.

Nesse ambiente, tornou-se indispensável o desenvolvimento de novas tecnologias que garantam segurança nas transações *on-line*. A tecnologia que surgiu com maior êxito foi a criptografia.

Suponha que alguém deseje enviar uma mensagem para um amigo, e deseja ter a certeza que ninguém terá acesso a ela. Entretanto existe a possibilidade de alguém interceptar a mensagem e abri-la. A mensagem enviada é chamada de **texto plano**. Um modo de esconder o conteúdo da mensagem é denominado de **cifragem**, e o resultado da cifragem é o **texto cifrado**. O processo inverso, para se obter o texto plano novamente, é denominado **decifragem**. Criptografia então, pode ser definida como a arte ou ciências de esconder informações.

Em 1976, *Diffie e Hellman* inventaram a criptografia de chaves públicas, que introduziu o conceito da **assinatura digital** em documentos eletrônicos. Então, tornou-se necessário algum mecanismo confiável para o armazenamento das chaves públicas. Esse mecanismo surgiu em 1978, **os Certificados Digitais**. Estes, trazem informações sobre o “dono” da chave pública e a própria chave pública. Para se garantir a autenticidade e

integridade do certificado foi necessário que alguma entidade confiável assinasse-o digitalmente. Essa entidade foi denominada de AC – **Autoridade de Certificação**.

Contudo, para que um certificado digital seja válido, ele precisa passar por uma série de testes de validação. Sem a aplicação deste processo torna-se inviável a utilização de qualquer certificado digital.

## **1.1 Objetivos**

### **1.1.1 Objetivo Geral**

Temos como objetivo neste trabalho elaborar um sistema capaz de visualizar e validar um certificado digital X.509 no modelo hierárquico.

### **1.1.2 Objetivos Específicos**

- Fazer uma revisão criptográfica dos principais conceitos utilizados na certificação digital e ICP.
- Estudar a estrutura do certificado X.509 e de uma LCR.
- Estudar o processo de validação de um certificado X.509.

## **1.2 Motivação**

É interessante notar que muitas aplicações tecnológicas não são efetuadas hoje no mundo digital por falta de segurança e privacidade. Por exemplo, tecnologicamente hoje não seria difícil ou mesmo custoso fazer a implementação de um sistema de laudos médicos via Internet. O problema está, em como garantir que o laudo será efetuado pelo médico responsável pelo paciente. Problemas como este nos levam a criar expiração em prover uma tecnologia segura para prover maior uso e conforto das ferramentas hoje já disponível para grande parte da população.

Com esta intenção é que varias pesquisas são realizadas hoje com o objetivo de assegurar maior confiabilidades nos sistemas que fazem uso de informações sigilosas.

### **1.3 Metodologia e Ferramentas**

Para a realização deste trabalho foi realizada primeiramente uma ampla pesquisa bibliográfica em livros, documentos eletrônicos, dissertações de mestrado e artigos sobre certificados digitais e ICP.

Após essa fase inicial de estudos, partimos para uma leitura mais técnica – os RFCs. Onde encontramos as informações necessárias para a elaboração de um método de validação para certificados X.509.

Então elaboramos os casos de uso e o modelo conceitual do sistema e partimos para a implementação, utilizando a ferramenta Builder C++ 6.0 para codificação e Telelogic Tau 4.1 para a modelagem de dados.

### **1.4 Conteúdo do Documento**

O primeiro capítulo apresenta a introdução, os objetivos gerais e específicos, a motivação, a metodologia e ferramentas utilizadas no trabalho e o conteúdo do documento. O segundo capítulo apresenta uma revisão dos principais conceitos criptográficos, necessários para o posterior entendimento do conteúdo. O capítulo 3 apresenta a ICP (infra-estrutura de chaves públicas). O capítulo 4 nos fala das recomendações X.509. O capítulo 5 apresenta o sistema desenvolvido de Visualização e Validação de Certificados Digitais X.509.

# Capítulo 2

## Fundamentação Teórica

### 2.1 Introdução

Este capítulo aborda todos os conceitos criptográficos necessários para o entendimento do processo de validação de um certificado digital, abordando as técnicas e algoritmos utilizadas na construção do mesmo.

Abordaremos um breve histórico da criptografia, criptografia simétrica e assimétrica, função resumo, assinatura digital, certificados digitais, asn.1 e codificação BER e DER.



## 2.2 Criptografia

A segurança em sistemas computacionais está baseada na privacidade das informações e no restrito acesso às pessoas ou mesmo outros agentes computacionais, mantendo o sigilo e a autenticidade da informação. Para que isto seja possível, é necessário o uso da criptografia entre as transmissões de dados mencionadas.

Com o avanço da tecnologia da informação (TI), esta necessidade cresceu imensamente, e em vários pontos é uma barreira a ser vencida para que sistemas contribuam de melhor forma para a sociedade como um todo. Vários exemplos ainda hoje podem ser mencionados, que não são implantados pela falta de segurança e auditoria dos mesmos, apesar de que tecnologicamente os sistemas estariam aptos para fazer o que se propõe. Um exemplo simples: imagine um sistema onde um médico pudesse verificar on-line tomografias computadorizadas de seus pacientes, e submeter os laudos ao hospital, sem o deslocamento físico. Apesar de tecnologicamente possível, sem a garantia deste sistema, nada pode provar que o laudo passado ao hospital é mesmo do devido médico, somado a isso, provar que as informações transmitidas dos pacientes passarão pela rede sem que ninguém, por algum meio indevido tivesse acesso a elas.

Vendo então a grande necessidade de “*esconder informações*” nos sistemas computacionais, criaram-se algumas maneiras para que isto fosse possível, uma delas seria através da codificação, também conhecida como criptografia, que é uma técnica que tem como objetivo final que somente as pessoas às quais as informações são verdadeiramente destinadas tenham acesso. Assim como garantir a codificação para a transmissão dos dados dentro de um canal de comunicação não seguro, esta técnica deve garantir o caminho inverso do processo realizado, que seria a decodificação da informação pela pessoa devidamente autorizada, a fim de garantir a recuperação, compreensão e armazenamento dos dados.

Podemos definir criptografia, com sendo a técnica ou arte em escrever em código (ou cifra), através de técnicas que deixem a informação transmitida ou armazenada ilegível a agentes indevidos (um agente pode ser um sistema computacional ou uma pessoa). Como os recursos computacionais vêm crescendo vertiginosamente, e isto possibilita uma gama de ataques às técnicas que antes eram consideradas totalmente

seguras, esta arte hoje se torna complexa e deve estar em constante desenvolvimento e análise.

Para criptografar um texto, utiliza-se uma chave, da qual depende grande parte da segurança do processo. O número de bits de uma chave criptográfica é uma variável fundamental no sistema de segurança. Seu tamanho indica o nível de esforço necessário para realizar ataques para determinar a chave e, conseqüentemente, ter acesso aos dados protegidos.

A criptografia é um mecanismo de segurança que permite a implementação de diversos serviços (autenticação, não-repúdio, integridade, confidencialidade).

- ✓ **Autenticação:** verificar se o emissor ou usuário A é realmente quem diz ser.
- ✓ **Não-repúdio:** O emissor de uma mensagem não pode negar que enviou a mesma
- ✓ **Integridade:** Garantir que a mensagem inicial não sofreu alterações até chegar ao seu destino
- ✓ **Confidencialidade:** Garantia que somente o receptor desejado tenha acesso à mensagem transmitida.

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

## 2.3 Criptografia simétrica

Existem dois grandes grupos de técnicas para o uso da criptografia. A convencional, que é baseada em uma chave privada (secreta), onde todos que devem ter acesso à informação requerida devem conhecer a chave. Adicionado isso, outra grande

característica desta técnica é que a mesma chave é usada para a cifragem (ou codificação) da informação e para decodificação. O outro grupo seria a criptografia assimétrica.

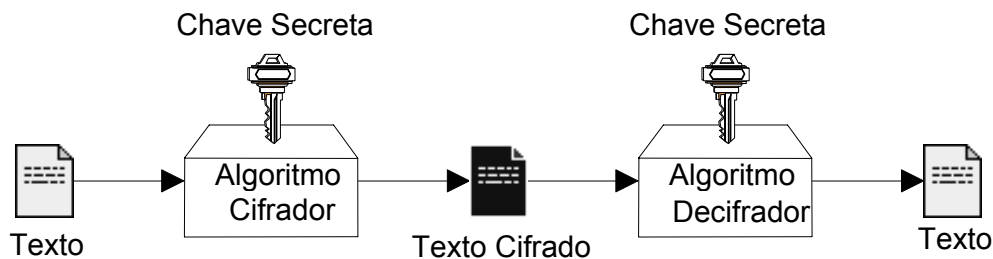


Figura 1-Criptografia Simétrica

Na criptografia simétrica, se uma pessoa quer se comunicar com outra com segurança, ela deve passar primeiramente a chave utilizada para cifrar a mensagem. Este processo é chamado distribuição de chaves e como a chave é o principal elemento de segurança para o algoritmo, ela deve ser transmitida por um meio seguro. Porém, se existe um meio seguro de se enviar a chave porque não enviar a própria mensagem por este meio? A resposta para esta questão é que meios seguros de comunicação são geralmente caros e mais difíceis de serem obtidos e utilizados, sendo então razoável sua utilização uma única vez, mas não continuamente.

Existe outro problema na distribuição das chaves. Imaginando-se o caso de três pessoas – A,B e C – que queiram se comunicar com chaves secretas (criptografia simétrica). Serão necessárias 3(três) chaves: uma compartilhada entre A e B, outra entre A e C, e a última entre B e C. Se mais pessoas forem inclusas neste sistema de comunicação, mais chaves seriam necessárias. Em geral, se  $n$  pessoas querem se comunicar utilizando chave secreta serão necessárias  $(n)*(n-1)/2$  chaves, gerando um grande problema para o gerenciamento de chaves entre grandes grupos de usuários.

Uma das tentativas de solucionar o problema da distribuição de chaves foi a criação de um centro de distribuição de chaves (Key Distribution Center – KDC), que seria responsável pela comunicação entre pessoas aos pares. Para isto, o KDC deve ter

consigo todas as chaves secretas dos usuários que usam seus serviços. Por exemplo, imagine que A deseja mandar uma mensagem secreta para B. Para isto ele manda a mensagem ao KDC usando sua chave secreta. O KDC recebe esta mensagem, decifrando com a chave secreta de A, depois o KDC a cifra novamente usando agora a chave secreta de B, e envia para o mesmo. O maior problema em torno do KDC, é que ele se torna um componente centralizado, além de ser gerenciado por pessoas que podem casualmente ser corrompida.

## **2.4 Criptografia assimétrica**

Esta técnica de criptografia também é conhecida como criptografia por chaves públicas. Ela se baseia no uso de uma par de chaves, uma pública e outra privada, para a cifragem das informações. Diferentemente da técnica convencional, nesta técnica a informação cifrada com a chave pública, só poderá ser decodificada com o uso da chave privada, e de maneira contrária também, ou seja, uma informação cifrada ou codificada com a chave privada, só poderá ser lida com o uso da chave pública. Outra característica importante desta técnica de criptografia, é que o conhecimento da chave pública não permite a descoberta da chave privada.

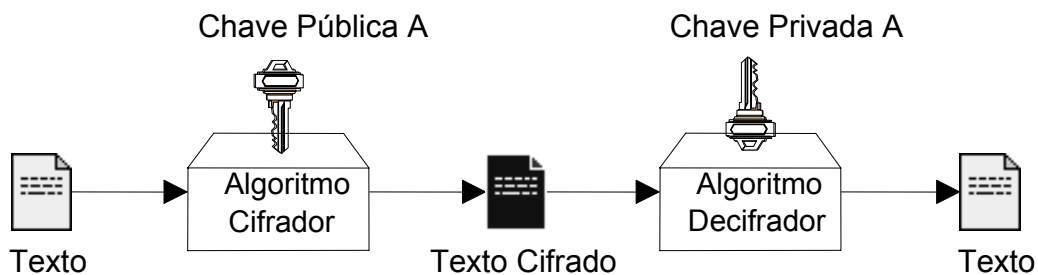


Figura 2-Criptografia Assimétrica

Como os próprios nomes dizem, a chave pública, pode ser distribuída a todos e divulgada por algum meio qualquer de comunicação, já a chave privada deve ser de conhecimento unicamente do próprio usuário.

Para transmitirmos informações seguras para uma pessoa por esta técnica, devemos ter o conhecimento da chave pública dela, para cifrarmos a informação necessária e ela fará uso da sua chave privada para a leitura da informação transmitida. Para ela nos enviar alguma informação de forma segura, ela deverá fazer uso da nossa chave pública e usaremos nossa chave privada para ler a informação.

De maneira inversa, se cifrarmos a informação com a chave privada de “A”, quem abrir esta mensagem sabe que somente “A” poderia ter produzido, garantindo assim autenticidade e assinatura.

## 2.5 Funções Resumo (*Hash*)

É uma função capaz de gerar um resumo (*digest*) de tamanho fixo de qualquer mensagem, independente de seu tamanho. Este resumo não permite que se consiga retornar ao texto original. Outra característica imprescindível desta função, é que dois textos diferentes nunca geram dois resumos iguais. Esta função também tem um custo computacional relativamente baixo.

Estas funções são usadas para a autenticação de mensagens, com a intenção de garantir, por exemplo, que a mensagem que o receptor venha a receber, não tenha sido modificada durante a transmissão, com a verificação da função resumo.

A função de resumo sendo aplicada com o uso da criptografia assimétrica é capaz de resolver grandes problemas nas transmissões de dados, como a autenticidade, o não repúdio e a integridade da informação. Um exemplo gráfico pode ser analisado abaixo:

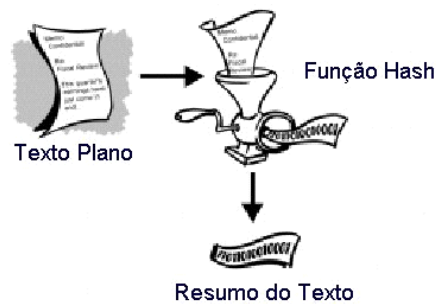


Figura 3-Como funciona a função Hash

## 2.6 Assinatura Digital

A assinatura digital, assim como a convencional, procura oferecer garantias de identificação da autoria do documento à qual é aposta, como também da integridade de seu conteúdo desde o ato de sua assinatura. Serve também para vincular vontade ou anuência do autor ao conteúdo do documento, em contratos. Por isso não se deve assinar papel em branco nem documento rasurado ou não lido, nem se dar credibilidade a documentos assinados que contenham rasura.

Mas esta comparação está ainda incompleta. Precisamos saber a quem, e como, tais garantias são oferecidas, antes de nos deixarmos levar pelas promessas virtuais. Nesse ponto imprecisões comprometedoras, e mesmo falácias, podem surgir da simplificação. Algumas chegam a mencionar riscos da assinatura convencional ser falsificada ou roubada, e que esses riscos não existiriam para a assinatura digital, quando o contrário seria, pretende-se mostrar, muito mais plausível.

Só teria sentido o "roubo" de assinatura convencional, à caneta e em papel, para reuso. Isto é, sua extração de um documento legítimo para autenticar um outro. O roubo literal produz rasura ou emenda no suporte físico da assinatura reusada -- o papel, que a vincula ao conteúdo pretensamente autenticado. Mas rasuras ou emendas são facilmente detectáveis por inspeção deste suporte. Entretanto, para a assinatura digital não há suporte material, pois o documento eletrônico é apenas uma seqüência binária que representa símbolos. Além de codificar seu conteúdo, esta seqüência terá que servir também como suporte para sua própria assinatura.

Para documentos eletrônicos, é ingênuo e perigoso pensar no meio magnético como suporte, já que cópias digitais são indistinguíveis de "originais". Sua assinatura digital deverá então ser calculada, a partir da seqüência binária que lhe dá suporte e de uma outra seqüência binária que servirá para identificar o assinante, denominada chave de assinatura. A seqüência de bits resultante deste cálculo é então aposta a seu suporte, isto é, concatenada a tal documento. Para eficácia do processo, tal chave precisa ser mantida em sigilo por seu titular, daí vem o nome chave privada, já mencionada anteriormente neste documento (ver criptografia assimétrica). O equivalente ao sigilo da chave privada na assinatura convencional é a exigência legal de que sua impressão seja cursiva, ou seja, de próprio punho. Por isso a reprografia é inválida.

A exigência da caneta e tinta serve, portanto para impedir falsificações não-cursivas. Impressões cursivas marcam o papel de modo rítmico, irregular, enquanto as reproduções fotográficas e carimbos não, sendo assim distinguíveis da escrita manual.

Ampliando-se o sentido literal de roubo tem-se a contrafação, que é a falsificação cursiva de uma assinatura de punho. A contrafação requer conhecimento e reprodução de padrões adquiridos pelo cerebelo do titular da assinatura, o que quase sempre revelará sua falta de autenticidade numa perícia gráfica. Se duas assinaturas são absolutamente

idênticas na forma, pelo menos uma delas terá sido produzida por impressão não-cursiva, já que ninguém produz à mão duas assinaturas exatamente iguais. E se duas assinaturas de punho, que pretendam a mesma titularidade, diferirem significativamente em ritmo e forma caligráfica, pelo menos uma será tida como falsa.

A verificação de assinaturas digitais não é, como a convencional, feita apenas por inspeção visual. Primeiro inverte-se o cálculo da assinatura, que deverá produzir a seqüência binária à qual foi aposta, representando o conteúdo por ela autenticado. Para isso o verificador precisa obter do assinante uma outra chave criptográfica capaz de sempre reverter à operação da chave privada que gera assinaturas. Estas duas chaves formam um par. A verificação se dá pela exatidão desta inversão, que assim atestará a integridade do suporte (o documento) desde o ato da assinatura, e vinculará a mesma titularidade às chaves usadas na assinatura e na verificação, dando suporte à identificação do assinante.

## 2.6.1 Descrição

Simploriamente, a assinatura digital é capaz de garantir a veracidade de uma informação e sua autoria. Ela é feita através uso de procedimentos matemáticos realizados com a utilização das técnicas de criptografia assimétrica e nos algoritmos de *Hash*. Com isto, pode-se garantir não só a transmissão de dados entre pessoas de forma segura e íntegra, pode-se garantir também a integridade de agente e sistemas computacionais.

Para tudo isso ser possível é necessário que a existência de um órgão íntegro que garanta a autenticidade da assinatura assim com suas validades. Tais órgãos são chamados de autoridade certificadora (*CA-Certificate Authority*), que tem como objetivo a autenticidade e inviolabilidade nas transações via Internet. Como grande exemplo temos a Verisign (<http://www.verisign.com>), considerada uma das maiores e bem conceituadas certificadoras do mundo.



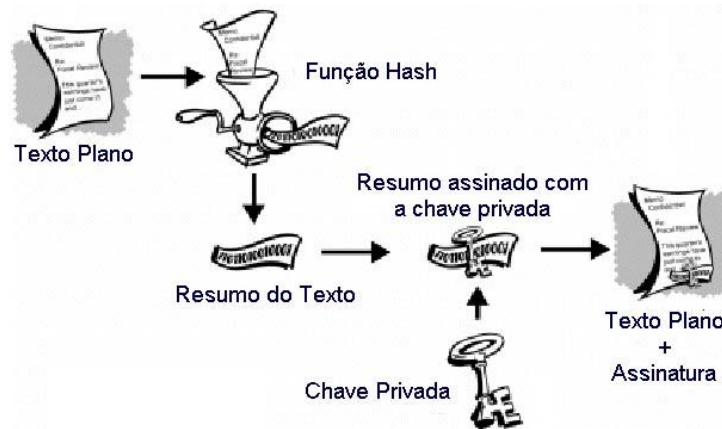


Figura 4-Função Hash com a assinatura digital

## 2.6.2 RSA (Rivest, Shamir e Adleman) aplicado a Assinatura Digital

RSA é um algoritmo, que leva o nome dos seus 3 criadores - Rivest, Shamir e Adleman – ,criado no inicio da década de 80. Foi o primeiro algoritmo de criptografia assimétrico desenvolvido.

O RSA é baseado na dificuldade de se fatorar dois números primos grandes. Seu grande inconveniente é a sua lentidão já que terá de ter como suporte, sistemas capazes de lidar com números muito grandes. Na maioria dos casos o RSA é usado para distribuir uma chave de criptografia simétrica, apenas no inicio de sessão, durante a sessão os dados são cifrados com essa chave, usando criptografia convencional, muito mais rápida.

O algoritmo RSA é usado na assinatura digital para cifrar o resumo(hash) gerado, a partir do texto plano, gerando a assinatura digital do documento. O RSA poderá também cifrar não só o resumo, mas a mensagem toda inclusive a assinatura, formando um apenas um bloco cifrado.

## 2.7 Padrões de Criptografia de Chave Pública

A empresa RSA Security definiu uma série de padrões para o uso da criptografia de chaves públicas. Esses padrões foram denominados de *Public Key Cryptography Standards – PKCS*.

PKCS descreve a sintaxe para mensagens de uma maneira abstrata, e oferece detalhes completos sobre os algoritmos. Porém esses padrões não descrevem a maneira correta de representar essas mensagens, embora BER seja a escolha mais lógica.

Atualmente, a grande maioria das aplicações que utilizam a criptografia de chaves públicas, utiliza esses padrões. Vamos analisar 2 destes padrões, que se enquadram em nosso projeto.

### **2.7.1 PKCS#7 v1.6- Padrão de Criptografia de Mensagem**

O padrão PKCS #7 descreve a sintaxe geral, para dados que podem sofrer a ação de criptografia, como a assinatura digital. Ele também permite o encapsulamento de uma mensagem, assinada ou cifrada, dentro de uma nova mensagem, com isto, uma mensagem pode ser cifrada e depois assinada.

O padrão é compatível com *Privacy-Enhanced Mail - PEM*, codificação utilizada na transferência de mensagens assinadas e cifradas por e-mail. Com essa compatibilidade, uma mensagem pode ser transferida através da Internet sem a necessidade de codificações adicionais.

O uso deste padrão não ficou limitado apenas a mensagens eletrônicas, e também está sendo utilizado em transações eletrônicas, como o pagamento com cartão de crédito (SET).

### **2.7.2 PKCS#10 v1.7- Padrão de Requisição de Certificado**

O PKCS #10 é o padrão que descreve a sintaxe de uma requisição de certificados. Uma requisição é formada pela identificação do requisitante ou nome distinto e uma chave pública, juntamente com outros atributos opcionais, faz parte desta requisição também um identificador do algoritmo da assinatura e a assinatura digital da informação da requisição do certificado.

Todo o conjunto de dados é assinado digitalmente pela entidade que esta requerendo a certificação. Requisições para certificados são enviadas para ACs, que as transformam

em certificados digitais. Depois de criado o certificado digital, a AC o envia para o requisitante.

## 2.8 ASN.1

Nesta seção será dada uma pequena introdução sobre a notação ASN.1, que é utilizada para definir certificados digitais e listas de certificados revogados. Posteriormente analisaremos as 2 formas de codificação para o formato ASN.1 : BER e DER.

*Abstract Syntax Notation One*, é a notação para descrever tipos e valores abstratos de dados.

Em ASN.1, um tipo é um conjunto de valores. Para alguns tipos, existe um número finito de valores, e para outros existem infinitos valores. ASN.1 possui quatro classes de tipo:

- **Simples:** são tipos primitivos, sem componentes adicionais.
- **Estruturado:** Possuem componentes, eles são *sets* ou *sequences* de outros tipos.
- **Rotulados:** São derivados de qualquer outro tipo.
- **Outros tipos:** São tipos especiais, que não se enquadram em nenhuma das 3 classes anteriores. São eles: *Choice* e *Any*.

Tipos podem receber nomes com ASN.1, através do operador ( $::=$ ) e esses nomes podem ser usados na definição de outros tipos.

Todos os tipos ASN.1 recebem um rótulo (TAG), exceto os tipos CHOICE e ANY, e este rótulo consiste de uma classe e um número não negativo. Como regra, temos que dois rótulos são iguais se, e somente se, seus números forem iguais. Deste modo, o significado abstrato de um tipo só é afetado pelo seu rótulo e não pelo seu nome.

Os rótulos são divididos em 4 classes:

1. **Universal:** Rótulos Universais são associados com os tipos cujo significado é o mesmo para todas as aplicações. Estes tipos são apenas definidos na especificação do ASN.1
2. **Aplicação:** Rótulos de Aplicação são associados com os tipos cujo significado é específico de uma determinada aplicação.
3. **Privado:** Rótulos privados são associados com os tipos cujos significados é específico para um determinado *enterprise*
4. **Contexto Específico:** Rótulos de Contexto Específico são associados com os tipos cujos significados é específico para um determinado tipo estruturado.

ASN.1 utiliza uma notação muito semelhante a uma linguagem de programação. Comentários começam com dois hífen (--). Identificadores devem começar com letra minúscula e referências a tipos devem começar com letras maiúsculas.

No documento X.208 temos definidos os tipos que recebem um rótulo da classe Universal e seus respectivos números de rótulo.

<b>Tipo</b>	<b>Número do Rótulo (decimal)</b>	<b>Número do Rótulo (hexadecimal)</b>
INTEGER	2	02
BIT STRING	3	03
OCTET STRING	4	04
NULL	5	05
OBJECT IDENTIFIER	6	06
SEQUENCE and SEQUENCE OF	16	10
SET and SET OF	17	11
PrintableString	19	13
T61String	20	14
IA5String	22	16
UTCTime	23	17

*Tabela 1. Tipos com rótulos da classe Universal.*

### ***Tipos Simples***

São aqueles que não possuem componentes, ditos atômicos. Estão divididos em duas classes:

- *String types:*  
Alguns tipos: *BIT STRING, OCTET STRING, PRINTABLE STRING e UTCTIME;*
- *Non-String types:*  
Alguns tipos: *INTEGER, NULL, OBJECT IDENTIFIER, BOOLEAN;*

### ***Tipos Estruturados***

São constituídos de componentes, sejam opcionais ou não. São eles:

- SEQUENCE  
Conjunto ordenado de um ou mais tipos.
- SEQUENCE OF  
Conjunto ordenado de zero ou mais ocorrências de um tipo.
- SET  
Conjunto de um ou mais tipos.
- SET OF  
Conjunto de zero ou mais ocorrências de um tipo.

### **Rótulos Implícitos e Explícitos**

Rotulamento é usado para distinguir tipos de componentes de tipos estruturados. Frequentemente, componentes opcionais dentro de um conjunto (set) ou seqüência (sequence) recebem distintos rótulos de contextos específicos para evitar ambigüidade. Existem dois caminhos para rotular um tipo: implicitamente e explicitamente.

#### **Outros Tipos**

O tipo *CHOICE* provê uma lista de alternativas de tipos. Somente uma dessas alternativas pode ser selecionada para uma particular instância. O tipo *ANY* significa um valor

arbitrário de tipo.

## 2.8.1 BER

*Basic Encoding Rules (BER)* é uma forma de codificação que descreve como representar ou codificar valores de cada tipo ASN.1 na forma de uma seqüência de octetos. Cada octeto é dividido desta forma:

7	6	5	4	3	2	1	0
Tipo da Classe		<i>Primitive or Constructed</i>	Valor do Rótulo				

BER possui 3(três) métodos para codificar um valor ASN.1, dependendo do tipo e se o tamanho do valor é conhecido. Strings do tipo simples podem usar qualquer um dos métodos, mas tipos estruturados exigem um dos tipos *Constructed*. Os três métodos são:

- **Primitive, definite-length encoding.** Utiliza-se este método para tipos simples e tipos rotulados implicitamente que são derivados de tipos simples. Este método requer que o tamanho do valor seja previamente conhecido
- **Constructed, definite-length encoding.** Utiliza-se este método para tipos *strings* simples, tipos estruturados, tipos rotulados implicitamente que são derivados de tipos simples e estruturados e a qualquer tipo explicitamente rotulado. Este método requer que o tamanho do valor seja previamente conhecido.
- **Constructed, indefinite-length encoding.** Utiliza-se este método para tipos *strings* simples, tipos estruturados, tipos rotulados implicitamente que são derivados de tipos simples e estruturados e a qualquer tipo explicitamente

rotulado. Este método não requer que o tamanho do valor seja previamente conhecido.

- Em cada método, a codificação BER possui 3 ou 4 partes:
- **Octetos de Identificação.** Estes octetos identificam a classe (universal, application, context-specific ou private), indicam se o tipo é *primitive* ou *constructed*, e incluem o número do rótulo do valor ASN.1. Se o valor é entre 0 a 30, então o identificador é um octeto simples.
- **Octetos de Tamanho.** Estes octetos contêm o número de octetos dentro dos Conteúdos. Se o tamanho estiver entre 0 e 127, então o tamanho é um simples octeto. Para os métodos *Constructed*, *indefinite-length encoding*, estes octetos contêm um *flag* ('80' em hexadecimal) que indica que o tamanho é indefinido.
- **Octetos de Conteúdo.** Para os métodos *primitive*, *definite-length encoding*, estes octetos contêm a representação de um valor. Para os métodos *constructed*, estes octetos contêm a junção dos componentes da codificação BER.
- **Octetos de fim de conteúdo.** Para os métodos *Constructed*, *indefinite-length encoding*, estes dois octetos denotam o fim do conteúdo. Os dois octetos contêm o valor de '00 00' hexadecimal. Para os outros métodos este octeto é ausente.

## 2.8.2 DER

O DER (*Distinguished Encoding Rules*) é um subconjunto do BER, provendo exatamente apenas uma maneira para representar um valor ASN.1 como um strings de octeto. DER é voltado para aplicações onde apenas uma única sequência de octetos é necessária. Por exemplo, no cálculo da assinatura digital como um valor ASN.1, em um certificado X.509.

DER requer que a codificação *definite-length* seja sempre usada. Quando o valor está entre 0 e 127, o tamanho deve ser codificado como um simples octeto. Quando o



tamanho é 128 ou maior, o tamanho deve ser codificado no menor número possível de octetos. Para tipos string simples e para tipos com rótulo implícito derivados de tipos string simples, a forma "PRIMITIVE DEFINITE-LENGTH" deve ser usada. Para tipos estruturados, tipos implicitamente rotulados derivados de tipos estruturados, e tipos com rotulamento explícito derivados de qualquer tipo, a forma "CONSTRUCTED DEFINITE-LENGTH", deve ser usada.

Algumas outras restrições são definidas para tipos específicos e podem ser encontradas na recomendação X.509.

# Capítulo 3

## Infra-estrutura de Chaves-Públicas

### 3.1 Introdução

ICP é um sistema que utiliza criptografia assimétrica e certificados digitais para conseguir serviços seguros na Internet. Sendo assim, a ICP define uma série de serviços para o uso das tecnologias baseadas em chaves públicas. Tais serviços se tornam imprescindíveis à medida que a Internet tornou-se um meio muito utilizado para realização de comunicações e transações.

Um sistema de ICP cumpre assim os quatro requisitos básicos da Internet segura:

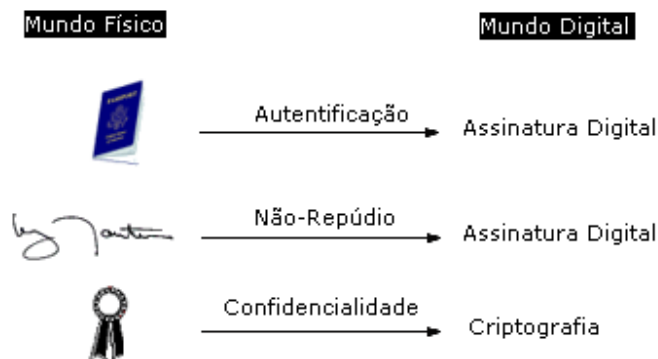


Figura 5-Requisitos para Internet segura

- **Autenticação – para identificar um usuário**  
Verificando que os usuários são realmente quem dizem ser
  
- **Não-Repúdio – para assegurar a origem de uma transação**  
Não-Repúdio significa que usuários são incapazes de negar que eles enviaram uma mensagem ou participaram de uma transação. Não-Repúdio, ou o ato de garantir a identidade do emissor é derivado dos benefícios da ICP.
  
- **Confidencialidade – para manter a informação cifrada e segura**  
Garante que apenas o receptor irá conseguir decifrar a mensagem.
  
- **Integridade**  
A integridade significa provar que a informação não foi alterada durante a transmissão.

As aplicações que utilizam a ICP conseguem assegurar estes quatro requisitos, fazendo uso de um sistema de certificados e chaves em conjunto com diversos algoritmos. O nível de segurança oferecida pela ICP depende da infra-estrutura e do comprimento das chaves utilizadas.

Neste capítulo falaremos sobre certificados digitais, autoridades certificadoras, caminhos de certificação, validação de um certificado digital.

## 3.2 Autoridades Certificadoras (ACs)

A Autoridade Certificadora (AC) é o bloco de construção básico do ICP. A AC é constituída por um conjunto de computadores, softwares e pelas pessoas que controlam tudo isso. Uma AC é caracterizada por 2 atributos básicos :

→ Nome

→ Chave pública.

As quatro funções ICP principais realizadas pela AC são:

1. Emitir certificados (ou seja, criar e assiná-los);
2. Emitir as LCRs (listas de certificados revogados);
3. Publicar os certificados e LCRs para que qualquer pessoa possa obtê-los.
4. Manter arquivos de informação sobre o *status* de certificados revogados que foi emitido.

Uma Autoridade Certificadora pode emitir certificados para usuários ou para outras ACs. Um certificado emitido é assinado com a chave privada da AC, e desta maneira pode ser comprovada sua autenticidade verificando-se a assinatura com a chave pública da AC.

Vale ressaltar que se alguém obtém a chave privada de uma AC, este poderá emitir certificados se passando por essa AC. Então destacamos que a principal responsabilidade de uma AC é *manter sua chave privada longe de qualquer ataque*.

Estrutura Geral de uma AC:

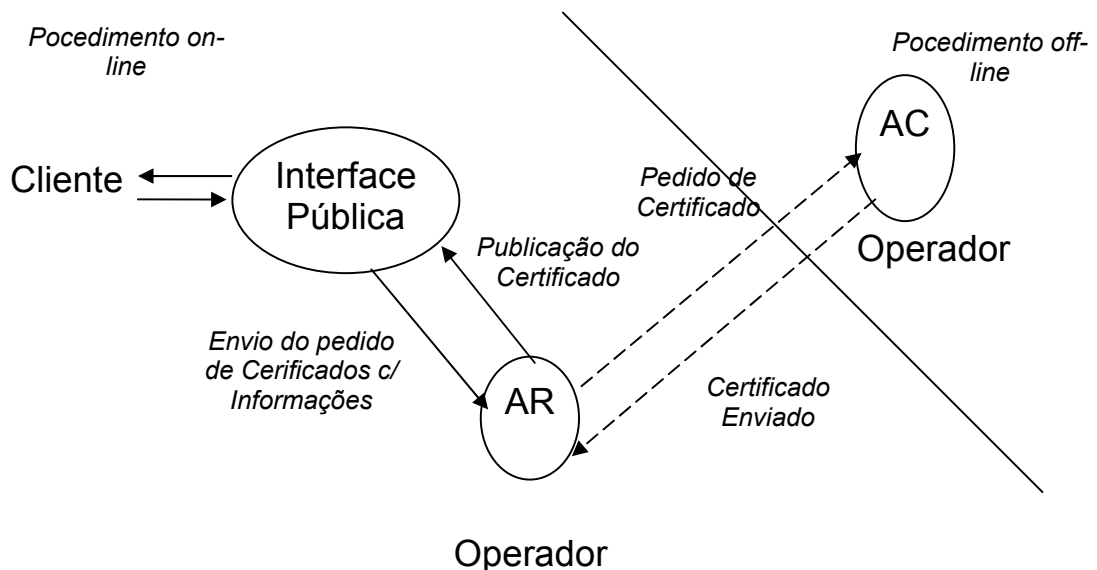


Figura 6-Arquitetura de uma AC

O cliente faz uma requisição de certificado através de uma Interface Pública, num procedimento em tempo real. A interface pública repassa essas informações para uma Autoridade de Registro (AR) que verifica a veracidade das informações contidas no pedido. Só depois da aprovação da AR, o pedido é enviado a AC, que geralmente é feito por uma mídia de dados (zip-driver, disquete...), por questões de segurança. Quando o certificado fica pronto, é enviado pelo caminho inverso até o cliente.

### 3.3 Certificados Digitais

Sem dúvida alguma, a segurança é a principal preocupação nas transações via Internet. Tendo em vista que a Internet é um sistema aberto e informações enviadas de um ponto ao outro podem ser lidas e/ou alteradas por várias pessoas, é necessário prover uma solução para garantir a Confidencialidade (privacidade), integridade, não repúdio e autenticação, conforme as especificações da ICP.

A solução foi usar a **criptografia** (para garantir a privacidade) e os **certificados digitais** (para garantir que a comunicação está ocorrendo entre os dois pontos desejados e para garantir a confidencialidade)

#### 3.3.1 Descrição

Podemos definir um certificado digital como sendo uma associação entre a chave pública de uma entidade e informações relacionadas a sua identidade.

Isto é feito de tal forma que qualquer pessoa, a qualquer momento, sem o conhecimento de informações especiais, possa verificar que a mensagem foi assinada por uma autoridade certificadora, e assim obter confiança na chave pública do usuário portador do certificado.

Por ser implementado de tal maneira, temos nos certificados digitais, um meio para verificação de identidade de pessoas no mundo eletrônico.

Um Certificado Digital une Identificação real do utilizador com a sua Identificação digital.

*Alice* (um usuário qualquer) pode processar o certificado com seu computador e não precisará mais digitar outra chave pública. O certificado contém campos com o nome e a chave pública de *Bob* (outro usuário). O certificado de *Bob* pode indicar sua companhia ou organização com o seu nome, e pode incluir informações para contato (geralmente seu endereço de e-mail). O certificado também contém 2 campos que especificam a data de ativação e expiração. Ele também contém um campo indicando a entidade confiável que criou o certificado.

Finalmente, todo conteúdo de um certificado é protegido pela assinatura digital da autoridade certificadora confiável que o enviou.

### **3.4 Lista de Certificados Revogados (LCR)**

Como podemos determinar se um certificado é válido? Considere que *Alice* trocou de trabalho logo após obter seu certificado digital. Sendo assim, as informações de contato mudaram, e ela obteve uma nova chave pública. *Alice* gostaria de ter certeza que todas as pessoas iriam se livrar de seu velho certificado. Infelizmente ele não irá se expirar por um longo tempo. O que *Alice* deverá fazer?

Certificados Digitais possuem uma importante característica. Uma vez que o certificado é emitido, é impossível determinar quem possui uma cópia, ou quantas cópias foram reproduzidas. Sendo assim, fica inviável a recuperação e destruição destes certificados.

A solução para este tipo de problema seria o envio de uma notificação, por parte de *Alice*, para a Autoridade Certificadora de que seu certificado não é mais válido. Sendo assim, quem desejasse obter a situação deste certificado deveria requisitá-la da AC.

A ferramenta ICP básica para distribuição da situação de um determinado certificado é a LCR – Lista de Certificados Revogados. A LCR contém uma lista de seriais dos certificados não confiáveis. A LCR é um objeto eletrônico, então o emissor pode distribuir e qualquer um pode processá-lo eletronicamente, assim como um certificado. Da mesma forma que um certificado digital, as LCR também são assinadas pela AC. Um exemplo da estrutura de uma LCR:

Emitido por: AT Certificados Co.		
Data de Emissão: 26 de agosto de 2002		
Data de Expiração: 26 de Setembro de 2002		
<table border="1"><tr><td>Lista dos Certificados Revogados:</td></tr><tr><td>39, 89, 32, 40, 109, 450, 10, 340, 206, 300, 17, 311, 44, 52</td></tr></table>	Lista dos Certificados Revogados:	39, 89, 32, 40, 109, 450, 10, 340, 206, 300, 17, 311, 44, 52
Lista dos Certificados Revogados:		
39, 89, 32, 40, 109, 450, 10, 340, 206, 300, 17, 311, 44, 52		
Assinatura Digital da AT Certificados		
def45434343d298309acb7d0972a23bcf 7eff23369399374ba74acf4307bg39aa0b 46bd45692001a1bc12bc546398acb389		

Figura 7-Estrutura básica de uma LCR

### 3.5 Componentes ICP

Uma ICP é formulada por 3 componentes básicos: servidor de certificação, servidor de diretórios e recuperação de chaves (ver figura 6). Estes componentes interagem com outros componentes necessários para uma transição digital.

A arquitetura de uma ICP (ou também pode ser chamada de PKI – Public key Infrastructure) pode ser descrita conforme a figura abaixo:

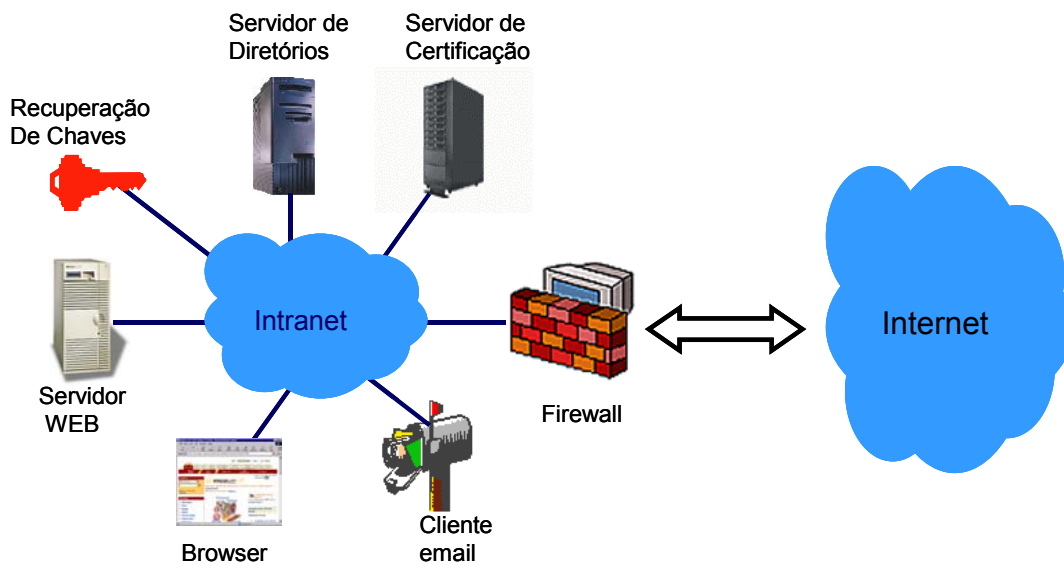


Figura 8-Arquitetura ICP

Os servidores PKI são a Autoridade Certificadora, os diretórios e sistemas de recuperação de chaves. Os clientes PKI são o servidor WEB, browser, email e aplicações de arquivos.



### **3.5.1 Servidor de Certificados**

Um servidor de certificados ou autoridade certificadora (AC) emite, administra e revoga certificados. O certificado da AC (a chave pública) tem credibilidade e é conhecido por todas as entidades participantes. A AC pode delegar a sua credibilidade para uma autoridade subordinada assinando o seu certificado e criando uma hierarquia de certificação. Isto pode ser feito para facilitar a administração (por exemplo: políticas diferenciadas de emissão), para melhorar a performance (evitar congestionamento de redes) ou por razões de segurança (evitar que a falha de um único ponto comprometa todo o sistema). A seqüência de certificados entre a última AC subordinada e a AC Root é chamada de cadeia de certificação. Cada certificado contém a identificação e a assinatura da AC emitente. A AC Root assina o seu certificado com a sua própria chave privada (ou seja, no X.509 v3 a entidade emitente e a entidade certificada são iguais).

### **3.5.2 Servidor de diretórios**

O servidor de diretórios fornece um único ponto de administração para as informações pessoais e corporativas. Os registros nos diretórios podem incluir os recursos da rede tais como servidores de arquivos, impressoras, URLs ou pessoas. As informações dos usuários como email, endereço, telefone, privilégios e certificados podem ficar disponíveis para múltiplas aplicações de acordo com uma política de segurança definida. Os clientes de diretórios podem localizar registros e atributos utilizando protocolos de acesso como, por exemplo, o LDAP.

O LDAP (Lightweight Directory Access Protocol) foi originalmente definido para possibilitar que aplicações rodando em plataformas distintas pudessem ter acesso a diretórios X.500. O LDAP é definido pelas RFCs 1777 e 1778 e é um protocolo orientado a bit, semelhante ao HTTP, que roda em ambientes TCP/IP. Ele cria um modo padronizado para as aplicações solicitarem e administrarem informações de diretórios. Os

registros nos diretórios são organizados em uma estrutura hierárquica que reflete as condições políticas e geográficas da corporação.

### **3.5.3 Servidor para recuperação de chaves**

O servidor para recuperação de chaves permite que clientes armazenem e recuperem chaves de criptografia. Esta função é necessária para ter acesso à arquivos criptografados caso a chave privada seja danificada. Este servidor também pode ser utilizado como um tipo de procurador do usuário para permitir, na ausência deste, o acesso à informações de propriedade da empresa. Todos os textos que transitam em um ambiente profissional são, em tese, propriedade da corporação, que deve ter acesso a sua chave de criptografia. Entretanto, para garantir o não repúdio da origem, os funcionários devem ter o controle exclusivo da sua chave privada. Para garantir tanto o acesso como a origem da informação, o funcionário usa uma chave exclusivamente sua para assinar digitalmente os documentos transmitidos e usa uma chave privada diferente para criptografia. Esta segunda chave privada é mantida armazenada em um servidor para garantir o acesso às informações caso o funcionário seja desligado.

### **3.5.4 Aplicações ICP**

Aplicação ICP é toda e qualquer aplicação que utilize a tecnologia de chave pública. Na maioria dos casos a aplicação também fornece funções criptográficas complementares como geração de chaves pública/privada, assinaturas digitais, criptografia e administração de certificados. As funções de administração de certificados incluem a geração de solicitações de certificados, revogação e armazenamento seguro de chave(s) privada(s). Veremos agora alguns exemplos de aplicações ICP.

### **3.5.4.1 S/Mime**

O S/MIME (*Secure/Multipurpose Internet Mail Extension*) é uma extensão do padrão de mensagens de correio eletrônicos MIME.

O MIME é uma extensão do RFC 822, que define o formato das mensagens de texto enviadas através do correio eletrônico. MIME tem como principal objetivo suprir com as limitações do protocolo SMTP.

Através do S/MIME é possível assinar e/ou cifrar uma mensagem de correio eletrônico. Do mesmo modo ele permite a verificação da assinatura e decifragem da mensagem.

### **3.5.4.2 SSL**

O SSL (*Secure Socket Layer*) é um protocolo que define um canal de comunicação seguro entre duas máquinas. Além de proteger os dados do canal de comunicação, ele identifica as máquinas do canal. Sendo assim, o SSL garante os serviços de autenticação, integridade e confidencialidade.

O SSL foi desenvolvido pela Netscape, e atualmente a IETF assumiu o projeto, alterando o seu nome para TLS.

### **3.5.4.3 IPSEC**

O IPsec (*IP Security*) fornece suporte a serviços na camada de rede – IP. Ele é utilizado para a criação de Redes Privadas Virtuais, que permitem que usuários estabeleçam uma conexão segura utilizando redes públicas, como a Internet.

O IPsec também fornece suporte ao uso de certificados digitais X.509v3, propiciando a troca de pacotes autenticados e/ou cifrados.

#### **3.5.4.4 SET**

SET é um conjunto de especificações técnicas na área de segurança. O SET garante que todo processo que envolva transação através de um cartão de crédito seja realizado de forma segura.

A segurança é garantida através da utilização de criptografia e certificados digitais. Através do uso dos certificados digitais, todos os componentes da transação (cliente, empresa, banco) podem ser autenticados pelo sistema.

# Capítulo 4

## Recomendação X.509

### 4.1 Introdução

Este capítulo descreve os certificados e a lista de certificados revogados de acordo com a especificação X.509. Posteriormente descrevendo o processo de validação de um certificado X.509.

### 4.2 Certificados X509

Foi desenvolvido pela *International Standards Organization* (ISO) e incorporado pela *American National Standards Institute*(ANSI) e *Internet Engineering Task Force*(IETF). Atualmente se encontra na sua terceira versão X509 v3.

A instrutura interna é baseada em ASN.1 e codificada em BER ou DER.

As principais informações contidas neste certificado são:

- Número de série: Todo certificado tem que ter um número de série, e esse número é único para cada entidade certificadora.

- Identificador do algoritmo de assinatura: Uma assinatura digital pode ser feita de diversas formas distintas, dependendo do algoritmo utilizado. Este campo identifica o algoritmo utilizado para a geração desta assinatura.
- Emissor: Nome da entidade certificadora responsável pela emissão do certificado em questão.
- Período de Validade: A validade do certificado. É importante salientar que toda AC (Autoridade Certificadora) possui uma lista de certificados revogados, para fazer a verificação se o certificado analisado é válido ou não.
- Titular: Nome ou identificador do titular do certificado.
- Identificador único do emissor do titular: Este campo juntamente com o campo *titular* foi criado na versão dois (v2) do X.509 com a intenção do reuso do nome do emissor junto com o nome do titular no decorrer dos anos. Atualmente isto não é mais usado. Recomenda-se a emissão de um novo certificado.
- Extensões: A V3 do X.509 permite que o certificado contenha campos fora dos acima especificados e que, muitas vezes, devem ser proprietários.
- Assinatura do emissor: É a assinatura digital da autoridade certificadora. A assinatura digital é feita para todos os campos descritos acima, inclusive as extensões. Assim sendo qualquer alteração no certificado é facilmente identificada. Por esta razão um certificado digital jamais pode ser modificado. A única maneira de adulterar um certificado é descobrir a chave privada da AC e gerar um novo certificado com ela.

### 4.2.1 Estrutura Interna

Apresentaremos agora a estrutura interna de um Certificado X.509 na notação ASN.1

```

Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm  AlgorithmIdentifier,
    signatureValue      BIT STRING }

TBSCertificate ::= SEQUENCE {
    version             [0] EXPLICIT Version DEFAULT v1,
    serialNumber        CertificateSerialNumber,
    signature           AlgorithmIdentifier,
    issuer              Name,
    validity            Validity,
    subject             Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID     [1] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- Se presente, versão deve ser 2 ou 3
    subjectUniqueID    [2] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- Se presente, versão deve ser 2 ou 3
    extensions         [3] EXPLICIT Extensions OPTIONAL
                        -- Se presente, versão deve ser 3
}

Version ::= INTEGER { v1(0), v2(1), v3(2) }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
    notBefore          Time,
    notAfter           Time }

Time ::= CHOICE {
    utcTime            UTCTime,
    generalTime        GeneralizedTime }

UniqueIdentifier ::= BIT STRING

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm           AlgorithmIdentifier,

```

```
subjectPublicKey    BIT STRING }
```

```
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
```

```
Extension ::= SEQUENCE {  
    extnID          OBJECT IDENTIFIER,  
    critical        BOOLEAN DEFAULT FALSE,  
    extnValue       OCTET STRING }
```

- ***tbsCertificate***. Este campo contém o certificado para ser assinado.
- ***signatureAlgorithm***. Este campo contém um identificador de um algoritmo, e define o algoritmo usado pela AC para a assinatura digital do certificado.
- ***signatureValue***. Este campo contém a assinatura digital. Esta assinatura é computada usando-se a codificação ASN.1 DER.
- ***version***. Este campo opcional contém a versão do certificado (v1,v2 ou v3)
- ***serialNumber***. Este campo contém um inteiro atribuído pela AC para cada certificado. Este número deve ser único para cada certificado gerado pela AC.
- ***signature***. Este campo contém o identificador do algoritmo. É uma cópia do campo *signatureAlgorithm*, porém, este campo é assinado digitalmente.
- ***issuer***. Este campo contém o nome X.500 da AC.
- ***validity***. Apresenta 2 datas, uma indica que o certificado não é válido antes dela, e a outra indicando o limite da validade do certificado.
- ***subject***. Este campo possui o nome do portador da chave privada correspondente a chave pública contida no certificado.
- ***subjectPublicKeyInfo***. Este campo possui detalhes sobre a chave pública e identificador do algoritmo.
- ***issuerUniqueID e subjectUniqueID***. Só aparecem na versão 2 ou 3. É usado com a intenção de reuso dos nomes. Esta solução é muito pouco usada, e não é recomendada.
- ***extensions***. Só aparece na versão 3. Se presente, este campo contém uma ou mais extensões de certificado. Cada extensão contém um identificador de extensão, um



indicador de criticidade e um valor de extensão. Na próxima seção detalharemos todas as possíveis extensões de um certificado X.509.

## 4.2.2 Extensões

Com o tempo, percebeu-se que os atributos de um certificado da versão 1 e da versão 2 eram insuficientes. Os certificados eram incapazes de determinar informações sobre quem o emitiu, para quem foi emitido ou sobre sua própria chave pública. Por exemplo: como definir se o certificado de Bob é uma AC ou é final? A chave pública de Bob pode ser usada para verificar assinaturas? Para responder a essas perguntas e a outras, a versão 3 do padrão X.509 adicionou o campo de extensões.

Apresentaremos agora as extensões definidas pela recomendação X.509 e que serão utilizadas no processo de validação:

- *Certificate Policies* (Políticas do Certificado) : Contém uma lista das políticas permitidas no certificado. Em certificados finais essa extensão contém as políticas sobre qual é submetido o certificado. Em ACs essa extensão contém um conjunto de políticas que podem ser incluídas nos certificados subordinados a ela.
- *Basic Constraints* (Restrições Básicas) : Permite identificar se o certificado é uma AC ou não. Permite também setar a quantidade máxima de certificados intermediários entre o sujeito e o certificado final. Pode ser crítica ou não.
- *Policy Constraints* (Restrições de Política) : Esta extensão é usada para impor limitações no caminho de certificação. Pode ser usado para proibir mapeamentos de política e para obrigar a declaração de alguma restrição de política.
- *Inhibit Any-Policy* : Da mesma maneira que as Restrições de Política, esta extensão impõe limitações no caminho de certificação. Ela indica que o OID (*object identifier*) “qualquer-política” não é permitido.

- ➔ *Key Usage* (Uso da Chave) : Indica os propósitos para que a chave pública do certificado pode ser utilizada. Esta extensão pode ser crítica ou não;
- ➔ *Key Usage Ext* (Uso extendido da Chave) : Extensão que complementa a extensão Uso da Chave. Pode ser crítica ou não;
- ➔ *Private Key Validity* (Validade da chave Privada) : Contém a data de início e fim da validade da chave privada do certificado.
- ➔ *Subject Key Identifier* (Identificador da chave do sujeito) : Contém uma “impressão digital” da chave pública do certificado
- ➔ *Authority Key Identifier* (Identificador da chave da AC) : Identifica a chave pública da AC que emitiu o certificado do sujeito..
- ➔ *CRL Distribution Points* (Pontos de Distribuição da LCR) : Contém um ou mais ponteiros para a LCR do certificado.
- ➔ *Authority Information Access* (Informação sobre acesso a AC) : Geralmente contém um ponteiro para a AC do certificado em questão.

### **4.3 Lista de Certificados Revogados**

A LCR definida pela especificação X.509 encontra-se na sua segunda versão.

Sua estrutura atual é:

<i>Versão</i>
<i>Algoritmo</i>
<i>Emissor</i>
<i>Data de Emissão</i>
<i>Data da Próxima</i>
<b><i>Lista de Certificados Revogados</i></b>
<i>Extensões</i>
<i>Algoritmo</i>
<i>Assinatura</i>

<< ***Certificados Revogados*** >>

<i>Número Serial</i>
<i>Data de Revogação</i>
<i>Extensões</i>

*Tabela 2. Estrutura de uma LCR*

#### **4.4 Construção e Validação do Caminho de Certificação**

Um caminho de certificação é uma corrente de certificados, onde o emissor do primeiro certificado é um ponto confiável, e o último certificado é o do usuário final, o que estará sendo validado. Este último certificado contém a chave pública que deve ser usada para verificar a assinatura.

Existem vários modelos (arquiteturas) para a validação de um certificado. Entre elas:

- Arquitetura hierárquico
- Arquitetura Mesh

- Arquitetura lista de confiança estendida
- Arquitetura AC Bridge

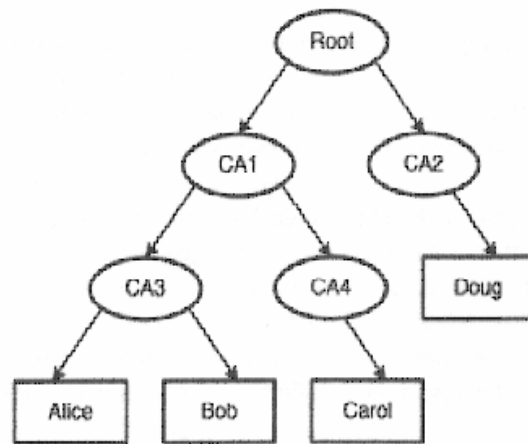
A proposta deste trabalho é trazer uma implementação para o modelo hierárquico.

#### **4.4.1 Modelo hierárquico**

Na Arquitetura Hierárquica, o caminho de certificação inicia-se na autoridade raiz, e termina no usuário final. Porém, eles são construídos na direção oposta. A construção inicia com o certificado do usuário final. Este certificado possui um emissor e uma extensão que identifica a chave da AC. Ao mesmo tempo, estes valores ajudam a localizar o certificado correto da AC. O nome do emissor é utilizado para localizar o certificado no repositório. O repositório pode conter diversos certificados que tem sido usado pela AC. Este processo é repetido até se encontrar um certificado emitido pela autoridade raiz na qual exista confiança.

A figura 7 mostra o caminho para Alice, Bob, Carol e Doug em uma ICP com estrutura hierárquica. Cada usuário final possui um, e somente um, caminho de certificação. Alguns são mais longos que outros, mas todos iniciam na raiz. A notação [( raiz -> AC2);(AC2 -> Doug)] indica que dois certificado compreendem o caminho do AC raiz até o certificado de Doug.

Sendo único este caminho é possível que este caminho de certificação seja transmitido com o certificado.



**Certification Paths:**

Alice: [(Root -> CA1); (CA1 -> CA3); (CA3 -> Alice)]

Bob: [(Root -> CA1); (CA1 -> CA3); (CA3 -> Bob)]

Carol: [(Root -> CA1); (CA1 -> CA4); (CA4 -> Carol)]

Doug: [(Root -> CA2); (CA2 -> Doug)]

Figura 9-Caminho de certificação em uma arquitetura hierárquica

#### 4.4.2 Validação de um certificado digital X.509

A validação de um certificado digital é feita mediante a construção de sua árvore de certificação. Uma árvore de certificação é uma corrente de certificados, onde a AC do primeiro certificado é o ponto confiável. Vamos analisar uma árvore de certificação do certificado de Alice:

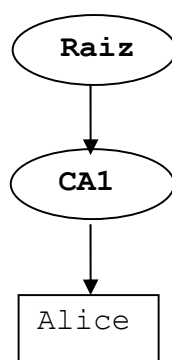


Figura 10- Exemplo de árvore de certificação

O caminho de certificação seria: [(Root->CA1);(CA1->Alice)].

O processo de validação então se divide em 2 etapas: a construção de árvore (caminho) de certificação a partir de um determinado certificado e a análise de cada certificado, iniciando-se do certificado confiável. Para que o certificado analisado seja válido, todos os certificados da árvore deverão ser válidos, o que implica na realização do processo de validação em todos os certificados.

O caminho de certificação deve satisfazer as seguintes condições para ser válido:

- Um ponto confiável emitiu o 1º certificado.
- Os campos emissor(*issuer*) e sujeito(*subject*) formam uma corrente. Para todos os certificados da sequência (exceto o 1º e o último). O *issuer name* deve ser igual ao *subject name* do certificado anterior e o *subject name* deve ser igual ao *issuer name* do certificado posterior.
- Os certificados não podem estar expirados.

Essas condições são necessárias, mas não são suficientes. *Basic Constraints*, *Name Constraints* e *Policy Constraints* devem ser considerados. Desta forma dividimos este processo em 4 etapas:

- 1) Inicialização
- 2) Checagem do Certificado
- 3) Preparação para o próximo certificado
- 4) Finalização

Estas etapas foram esquematizadas da seguinte forma:

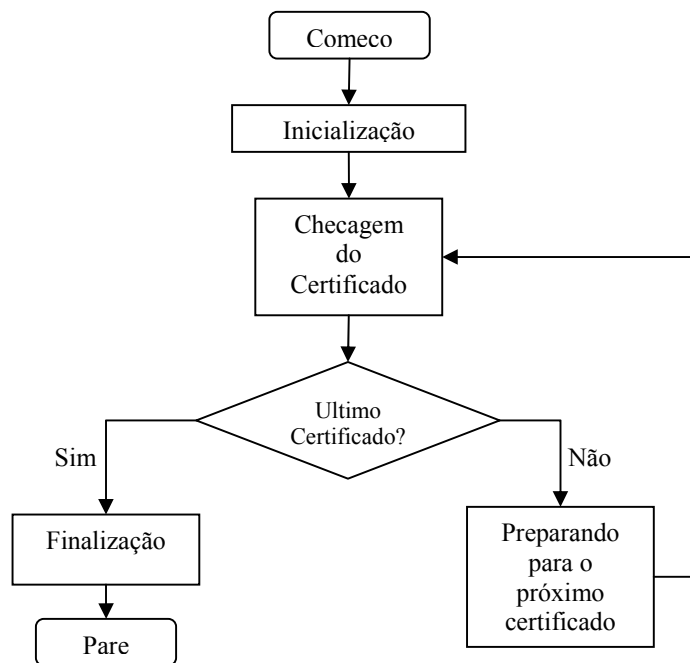


Figura 11-Etapas da validação de um certificado digital

### 1)-Inicialização

Essa etapa consiste na preparação dos valores e/ou entradas necessários para o processo de Checagem do Certificado. As entradas necessárias são:

- ➔ O caminho de certificação.
- ➔ Conjunto dos identificadores de política aceitáveis.

- Informação sobre o certificado confiável (geralmente o certificado auto-assinado –AC)
- Informação de se mapeamento de política são permitidos no caminho de certificação.
- Informação de se identificadores explícitos de política são requeridos nos certificados.

Baseado nessas entradas, definimos 3 grupos de variáveis necessárias para a validação. São eles:

a) Esse grupo de variáveis traz informações necessárias para a verificação da assinatura digital. Essas variáveis incluem a chave pública, parâmetros associados a chave pública e o algoritmo de assinatura digital que será usado para verificar a assinatura do próximo certificado da seqüência.

b) São as informações dos campos emissor(*issuer*) e sujeito(*subject*), e o tamanho do caminho de certificação. Estes campos são usados para verificar a correta relação entre o *emissor* e *sujeito* da corrente de certificados. O tamanho do caminho de certificação esta localizado na extensão *basic constraints* e é usado para definir o número máximo de certificados da corrente.

c) São informações da extensão *certificate policies*. Essas informações são necessárias para definir se uma *explicit policy identifier* é requerida e se *any-policy identifier* é permitida.

## 2)-Checagem do Certificado

Esta etapa é realizada em todos os certificados da seqüência. Esta checagem é realizada com o intuito de determinar se o certificado está expirado ou revogado. Essa etapa utiliza os três grupos de variáveis descritas acima para determinar se o certificado cumpre as restrições impostas pelas entradas. Os testes realizados são (se qualquer um destes falhar, o caminho de certificação é inválido):

- Checagem da validade do certificado: Este teste verifica o campo **not-before** e **not-after** no certificado, comparando-as com a data atual.



- ➔ Checagem na LCR (lista dos certificados revogados): Analisa a LCR, de modo a procurar uma possível revogação do certificado.
- ➔ Checagem da assinatura: Usando o 1º grupo de variáveis, verificamos se a assinatura do certificado pode ser validada com a chave pública do seu emissor.
- ➔ Checagem dos campos *emissor* e *sujeito*: Usando o 2º grupo de variáveis, verificamos se o campo *emissor* é igual ao campo *sujeito* do certificado anterior.
- ➔ Checagem das políticas de restrição: Usando o 3º grupo de variáveis, verificamos possíveis inconsistências nas políticas de restrições entre os certificados. Se a extensão *certificate policies* contém o identificador *any-policy*, verificamos o indicador que determina se *any-policy* é permitido. Se a extensão *certificate policies* não estiver presente, verificamos se uma *explicit policy* é requerida.

### 3)-Preparando para o próximo Certificado

Nessa etapa, os valores das variáveis são atualizados, de acordo com o certificado. Alguns procedimentos são realizados nessa etapa:

- ➔ Verificar que o certificado é um certificado AC: Essa informação está localizada na extensão *basic constraints*. Este certificado precisa conter uma chave pública para assinatura, e a extensão *KeyUsage* deve permitir que este seja usado para assinar certificados.
- ➔ Checagem do tamanho do caminho de certificação: Verificar se o tamanho máximo do caminho de certificação não foi excedido.
- ➔ Atualizar variáveis de verificação da assinatura digital: chave pública, parâmetros e algoritmo de assinatura digital, de acordo com os valores contidos no campo *subject public key*.
- ➔ Atualizar valores esperados no campo *emissor*: O valor esperado deve ser igual ao valor do campo *sujeito* do certificado atual.

→ Atualizar *policy constraints*: Setar variáveis para indicar se o certificado subsequente deve conter *explicit policy identifiers*, *any-policy* ou *policy mapping*.

→ Processar todas as extensões críticas: Se algumas dessas extensões não puderem ser processadas (reconhecidas), então o caminho de certificação deve ser inválido.

**4)-Finalização:** Nesta etapa, o processo de validação é finalizado. Resultando em alguns valores de saída:

→ As políticas do caminho de certificação válido.

→ A chave pública, parâmetros e algoritmo do certificado do usuário.

Ao término do processo, não encontrando nenhuma irregularidade, temos a certeza de que o Certificado analisado é válido.

# Capítulo 5

## Sistema de Validação e Visualização de um Certificado Digital

### 5.1 Introdução

A idéia do projeto é de se obter uma ferramenta para a validação e visualização de certificados digitais e listas de certificados revogados X.509. O programa será desenvolvido visando-se à performance, e por isso será escrito na linguagem C++, inicialmente para Windows.

O sistema possuirá um banco de dados, em que o usuário poderá armazenar suas ACs de confiança. Ao abrir um certificado, será apresentado ao usuário sua árvore de certificação, juntamente com seu tempo de validade. Caso o usuário clique em algum certificado da árvore, será lhe apresentado o conteúdo do mesmo, podendo ser visualizado em ASN.1 ou não. Da mesma forma, será possível visualizar a LCR de cada certificado com um simples clique.

Para nos apoiar na tarefa de criptografia (cálculo de funções hash, para validação), usaremos a biblioteca fornecida pelo Windows, CryptoAPI.

### 5.2 Detalhes do Projeto

O sistema desenvolvido funciona como um recipiente para certificados digitais. Inicialmente ele já vem com algumas ACs armazenados no banco de dados. O usuário terá a opção de adicionar certificados ao sistema, no formato X.509 (Codificação Binária-BER ou DER). Ao adicionar um certificado o sistema, um parser ASN.1 (**ASN1Parser**)

decodificará o arquivo, deixando-o no formato texto (em ASN.1). Desta forma varremos a estrutura ASN.1, através de um decodificador(**X509Decoder**). Então criamos uma estrutura **X509Cert**, que contém todos os dados pertencentes ao certificado

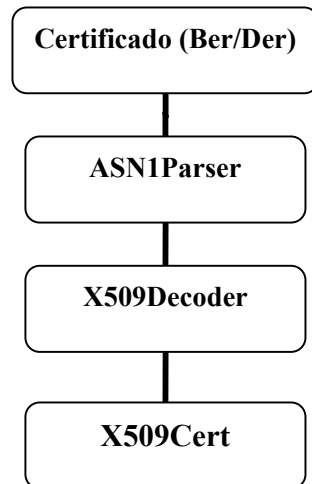


Figura 12- Estruturas(classes) utilizadas para criação de um certificado X.509

A classe ASN1Parser foi desenvolvida por Peter Gutmann, um importante nome no mundo da criptografia, que entre outras coisas desenvolveu a biblioteca *cryptlib*. Esta classe está disponível para uso público. Como dito anteriormente, ela realiza a decodificação BER/DER para ASN.1.

O certificado pode ser **Final**, **Intermediário** ou **Raiz**. Cada categoria possui um lugar específico. Tão logo o certificado tenha sido adicionado ao sistema, ele será submetido à **validação**. A organização dos certificados pode ser visualizada abaixo:

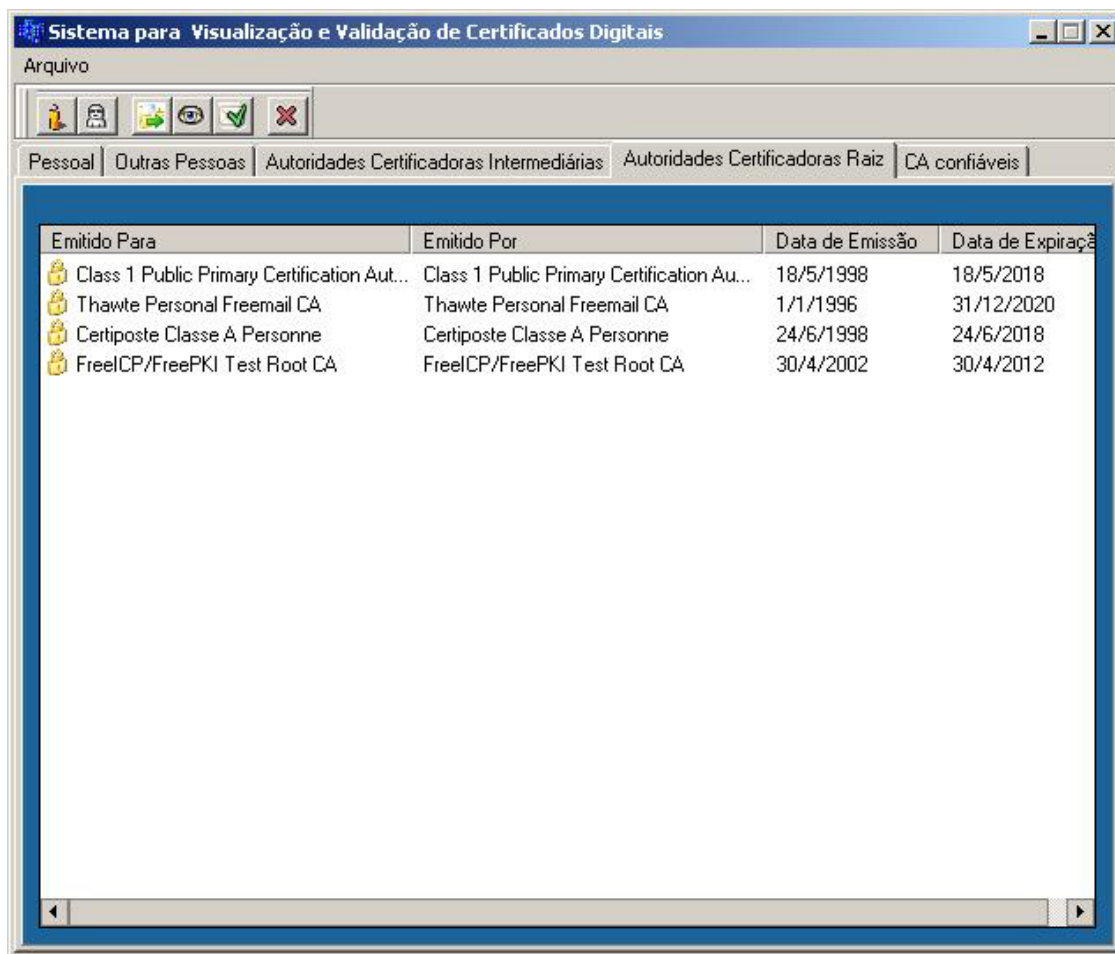


Figura 13-Tela Inicial

O processo de validação recebe um certificado (**X509Cert**), e a partir dele, tenta montar sua **árvore de certificação**, através da classe **X509Path**. Para a montagem da árvore, primeiramente consultamos o banco de dados, comparando os dados do campo Emissor do certificado com o Sujeito de todos os certificados cadastrados. Se algum Sujeito for igual ao Emissor, então encontramos a AC que o emitiu. Caso contrário, verificamos a existência do campo “Informação de Acesso a AC”. Caso exista, utilizamos o ponteiro para baixar o certificado da AC. Caso contrário não será possível construir o caminho de certificação.

Tendo o caminho de certificação, aplicaremos o algoritmo apresentado na seção 4.4.2, através da classe **X509Validation**.

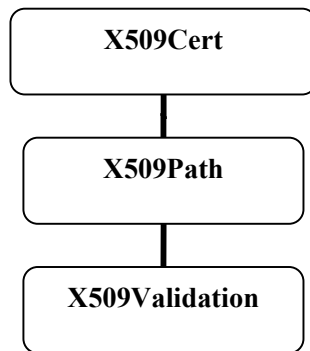


Figura 14- Classes utilizadas para a montagem e validação do caminho de certificação.

Com o término da validação será apresentados ao usuário o caminho de certificação e todos os possíveis erros e ou inconsistências. A figura a seguir mostra um exemplo de um caminho de certificação válido:

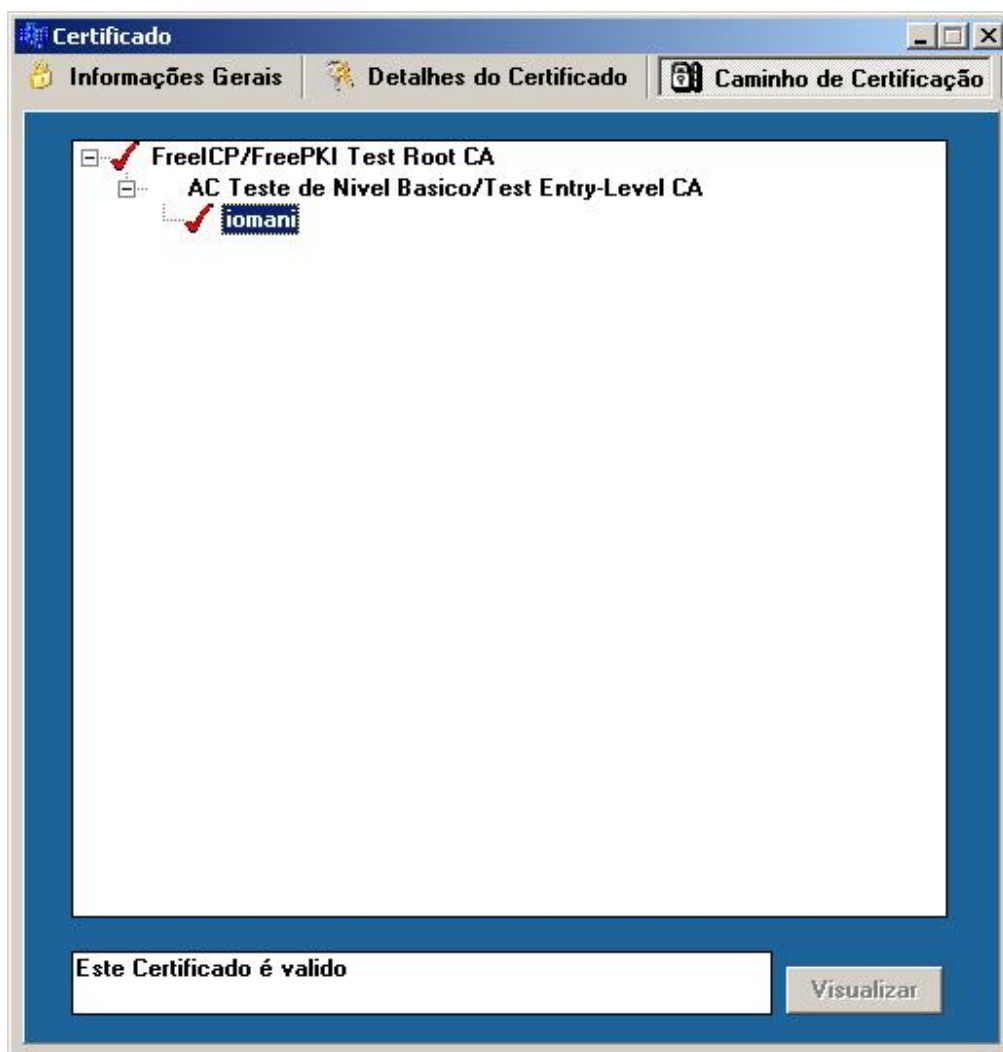


Figura 15-Caminho de Certificação

### 5.3 Persistência de dados

O sistema usa o banco de dados Interbase 6.0 na sua versão open source (<http://info.borland.com/devsupport/interbase/opensource>). A escolha deste banco foi por permitir uma boa performance em ambiente distribuído (via TCP/IP) e garantir boa interoperabilidade entre plataforma Windows e Linux. Além disso, permite a persistência de campos BLOB (binary large object) com grande garantia de integridade dos dados.

Esta arquitetura permite que exista um banco único, capaz de armazenar todos as ACs de um determinado grupo de usuários e que eles façam uso destes dados para validarem seus certificados.

## 5.4 Use Cases

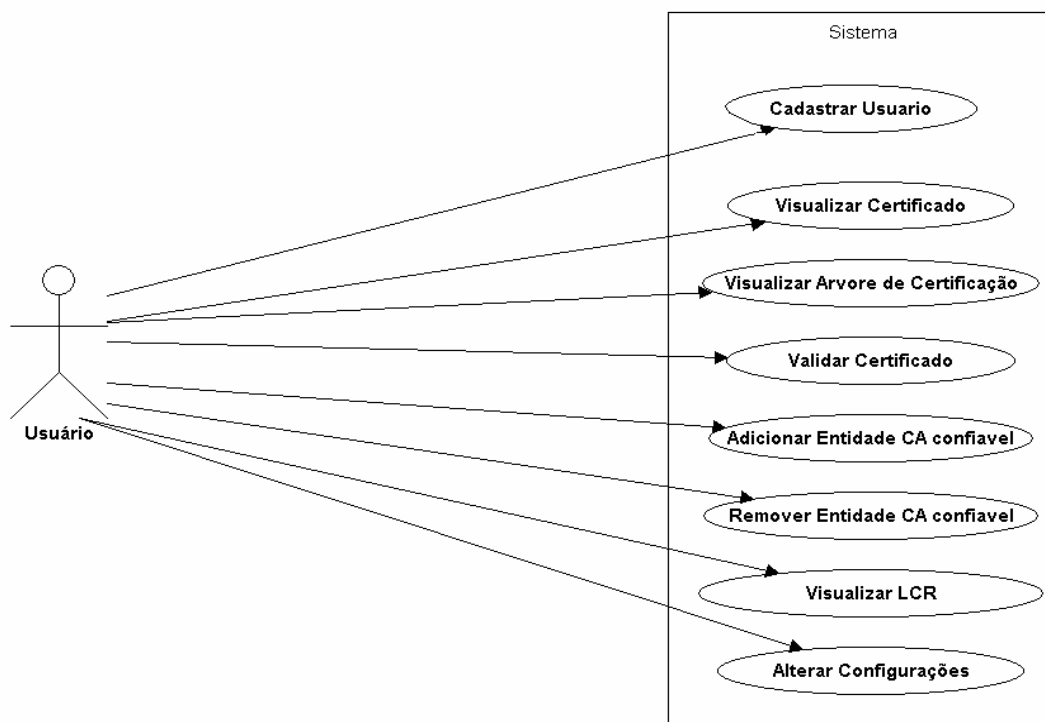


Figura 16 -Use Cases

- Cadastrar usuário: O usuário cadastrado tem uma senha de acesso. O usuário cadastrado tem algumas vantagens, por exemplo, ter uma lista de ACs que ele já considera como confiável, pode confiar em ACs adicionados por outros usuários. É importante salientar que o usuário não necessita estar cadastrado para validar um certificado qualquer.



- Visualizar Certificado: Traz as informações, em um modo gráfico, do certificado selecionado.
- Visualizar árvore de Certificação: Caso tenha sido possível construir a árvore de certificação do certificado aberto, é mostrada de maneira gráfica a árvore de certificação ao usuário.
- Adicionar entidade AC confiável: A partir deste momento, todo o certificado que tem a AC adicionada pelo usuário, dentro do caminho de certificação, será dado como válido, sem percorrer as ACs superiores a este.
- Remove entidade AC confiável: Remove a AC da lista de ACs confiáveis do usuário.
- Visualizar LCR: Faz a visualização da lista de certificados revogados de uma determinada AC.
- Alterar Configurações: O usuário pode alterar as configurações dele assim como sua senha de acesso.

## **5.5 Modelo conceitual**

A análise do Use Case nos levou a construção de um modelo conceitual para o sistema. Ele dá idéia das classes que o sistema possui os relacionamentos entre classes.

A figura do modelo conceitual pode ser visualizada a seguir:

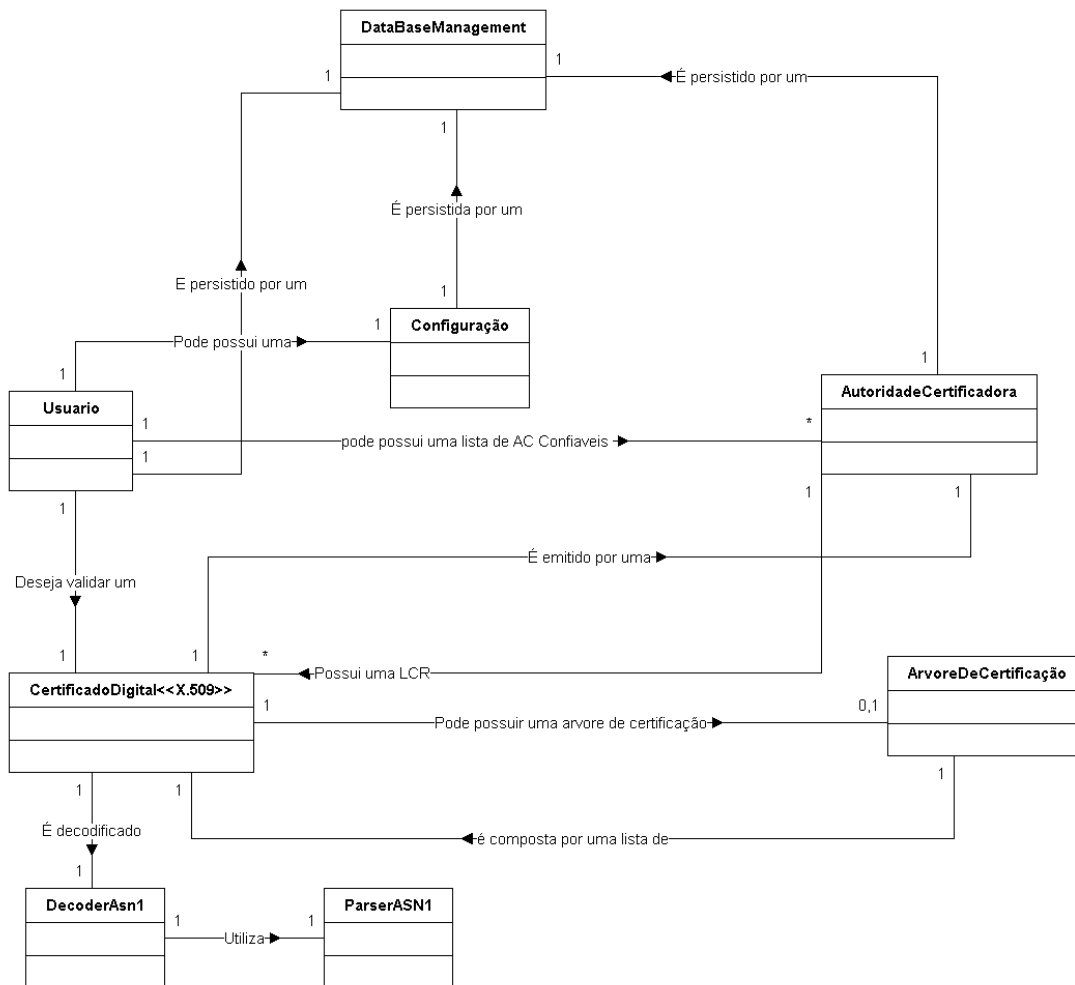


Figura 17 –Modelo Conceitual

## 5.6 Visualização dos campos do Certificado

É possível verificar os campos de um certificado selecionado. Para isto, basta selecioná-lo na seção desejada, para isto selecione o certificado na seção desejada e apertar o botão “Visualizar” (quinto botão na barra de tarefa) ou ainda simplesmente aperte duas vezes com o mouse em cima do mesmo. Após isso selecione a coluna “Detalhes do Certificado” como é visto na figura abaixo:

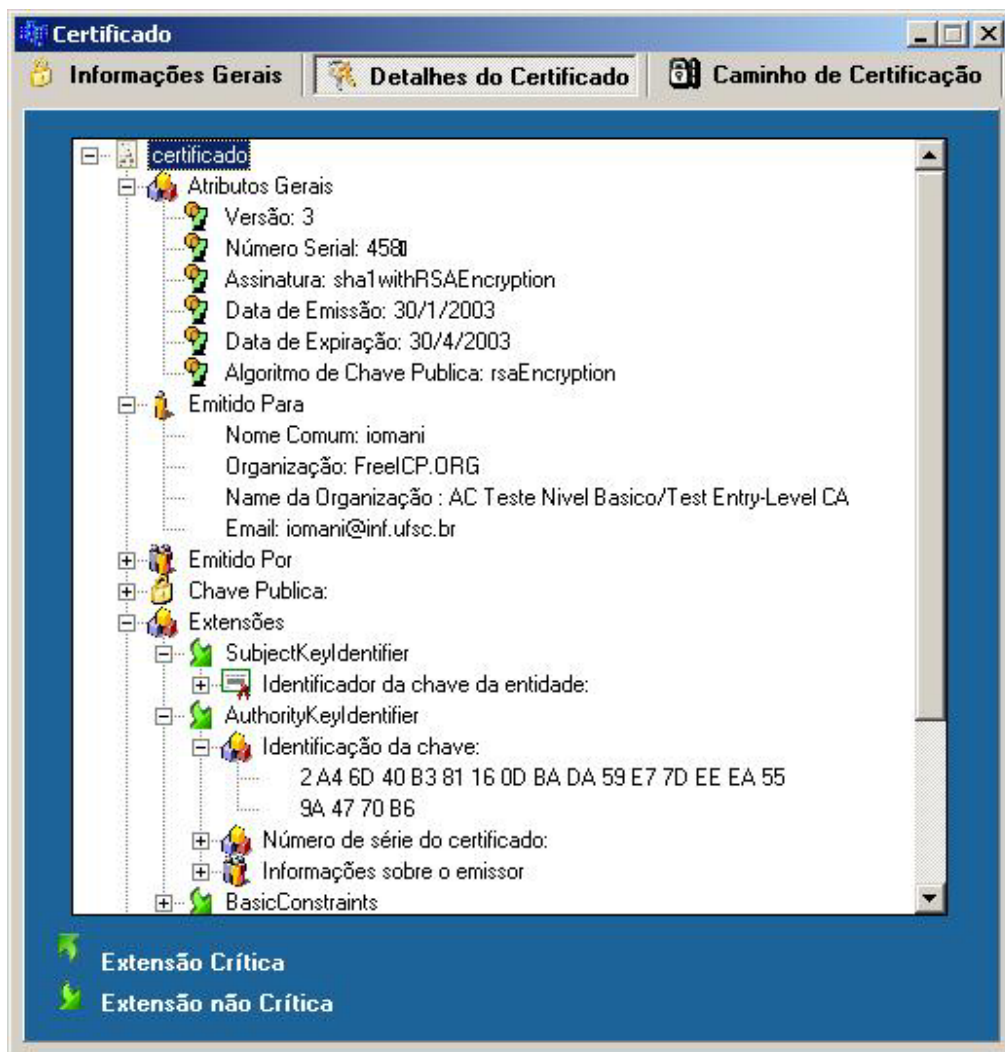


Figura 18 –Visualização dos atributos do certificado

Depois de aberto a visualização do certificado, é possível navegar dentro da árvore montada com os atributos do certificado selecionado.

## 5.7 Exemplos

Apresentaremos dois exemplos do sistema:

### 5.7.1 Caminho de Certificação Válido

O Usuário adiciona um certificado ao sistema. É construída sua árvore de certificação e ela é validada com sucesso.

1. O usuário seleciona a opção de adicionar um certificado. Uma tela para seleção é mostrada. O usuário seleciona o certificado desejado, a figura abaixo mostra este passo:

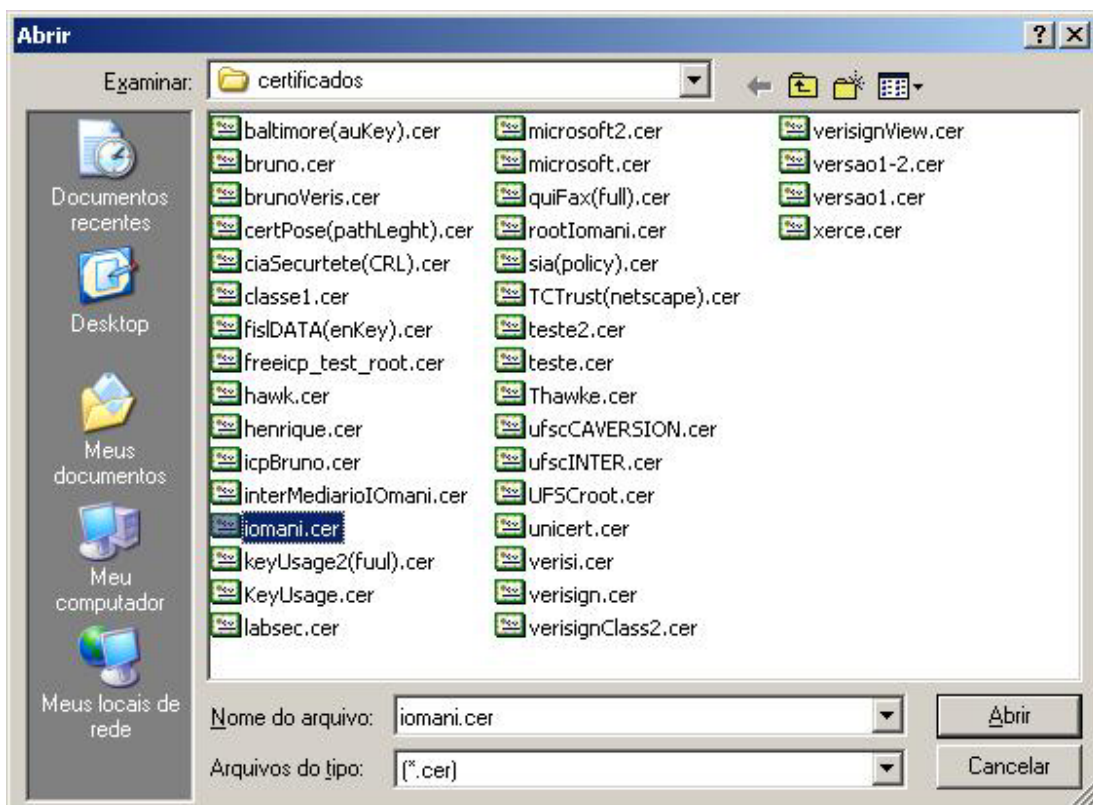
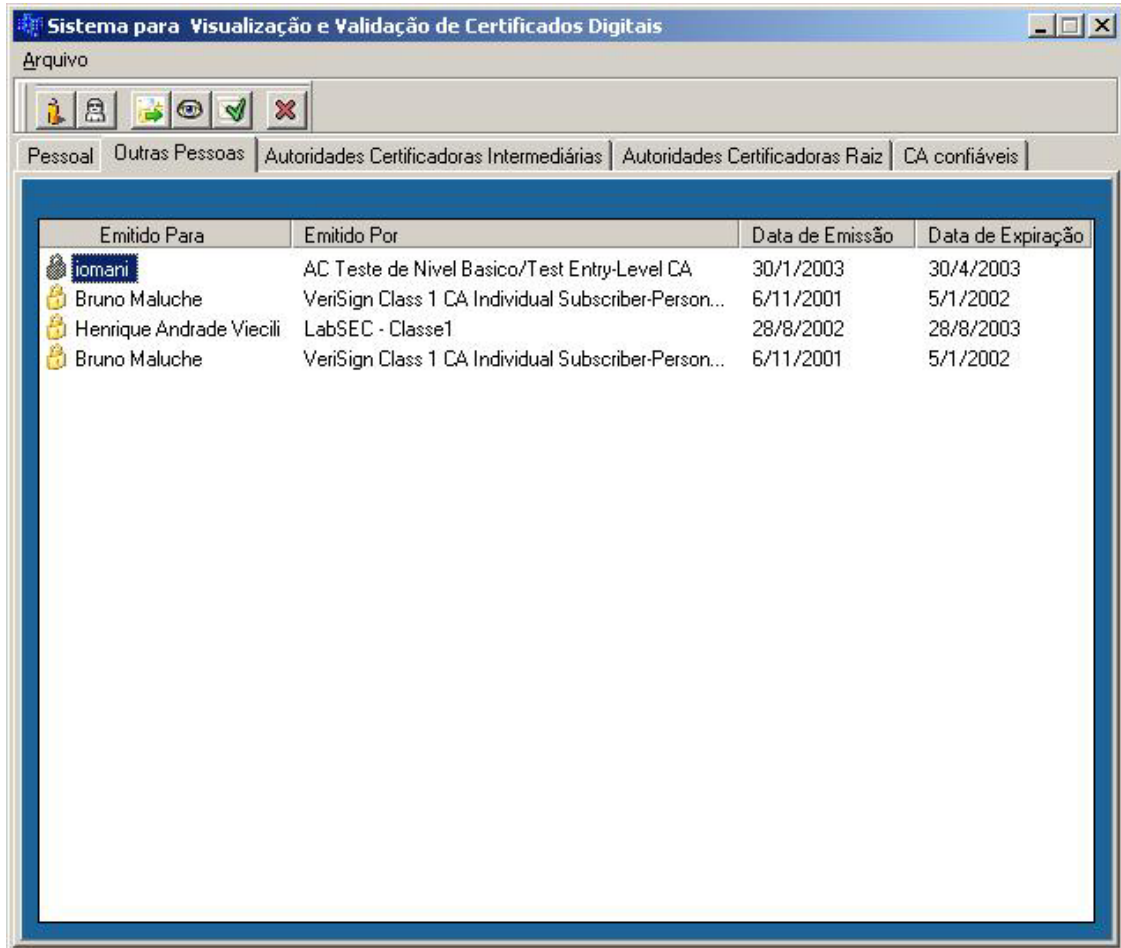


Figura 19–Inserindo um certificado

2. O usuário pode ver o certificado na seção “Outras pessoas”, se o certificado for de usuário final, ou em “Autoridades certificados Intermediárias” se for uma AC intermediária ou ainda em “Autoridades certificados Raiz” se for uma AC raiz.

No exemplo, o certificado foi adicionado na seção “Outras pessoas”. É possível visualizar algumas informações do certificado neste momento,

como a data de emissão do certificado, a data de expiração, o nome da AC que o emitiu e para quem foi emitido o certificado. A figura com esta visualização esta abaixo:



Emitido Para	Emitido Por	Data de Emissão	Data de Expiração
lomani	AC Teste de Nivel Basico/Test Entry-Level CA	30/1/2003	30/4/2003
Bruno Maluche	VeriSign Class 1 CA Individual Subscriber-Person...	6/11/2001	5/1/2002
Henrique Andrade Viecili	LabSEC - Classe1	28/8/2002	28/8/2003
Bruno Maluche	VeriSign Class 1 CA Individual Subscriber-Person...	6/11/2001	5/1/2002

Figura 20 –Informações básica do certificado

3. Para fazer a visualização do caminho de certificação do certificado desejado, basta selecioná-lo na lista e apertar o botão “Visualizar” (quinto botão na barra de tarefa) ou ainda simplesmente aperte duas vezes com o mouse em cima do mesmo. A tela de informações do certificado será mostrada. Após isso selecione a coluna “Caminho de Certificação” como é visto na figura abaixo:

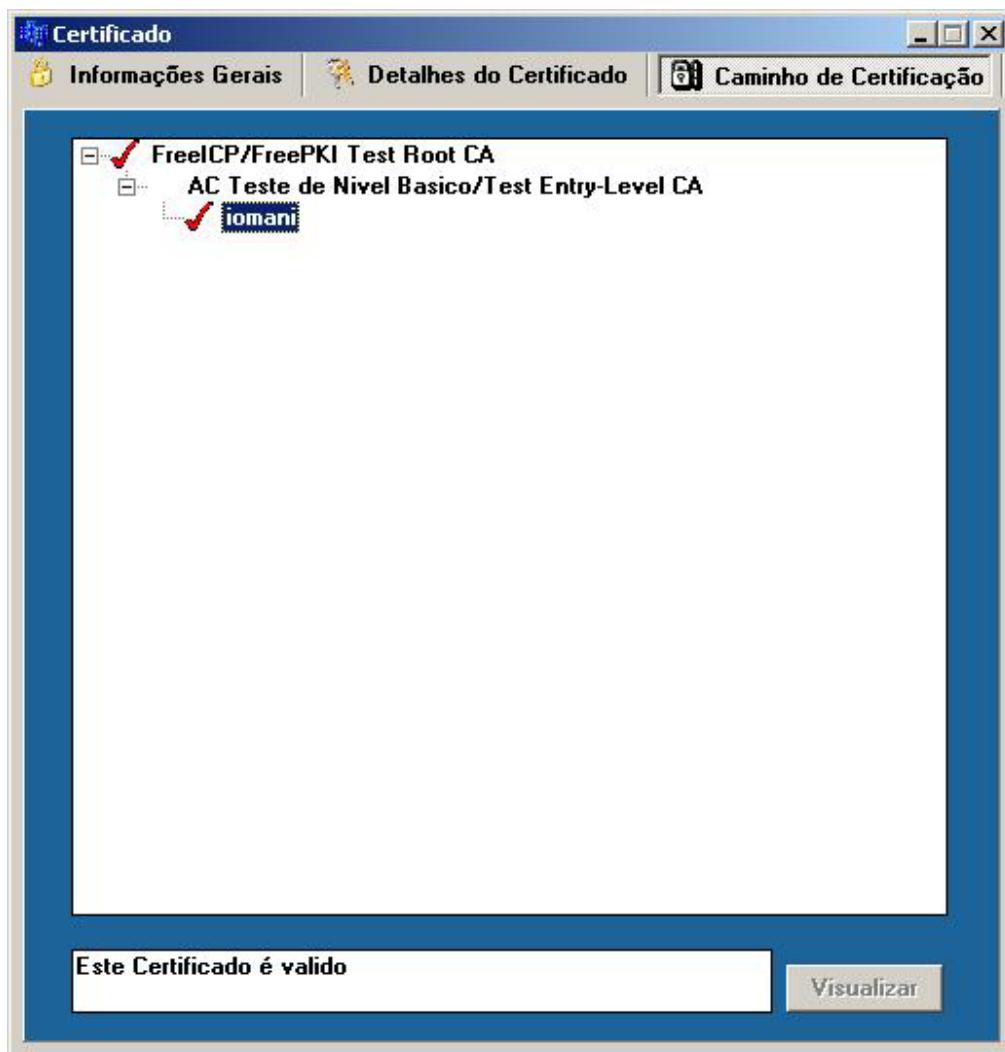


Figura 21 –Visualização de um caminho de certificação válido

4. A figura no canto superior direito da tela (um cadeado fechado) indica que o certificado teve sua árvore de certificação construída com sucesso.

### 5.7.2 Caminho de Certificação Inválido

O Usuário adiciona um certificado ao sistema. É construída sua árvore de certificação e ela é inválida.

O usuário pode repetir os passos 1,2 3 da seção anterior, se o certificado for tiver realmente um caminho de certificação inválido, a figura mostrada será esta:

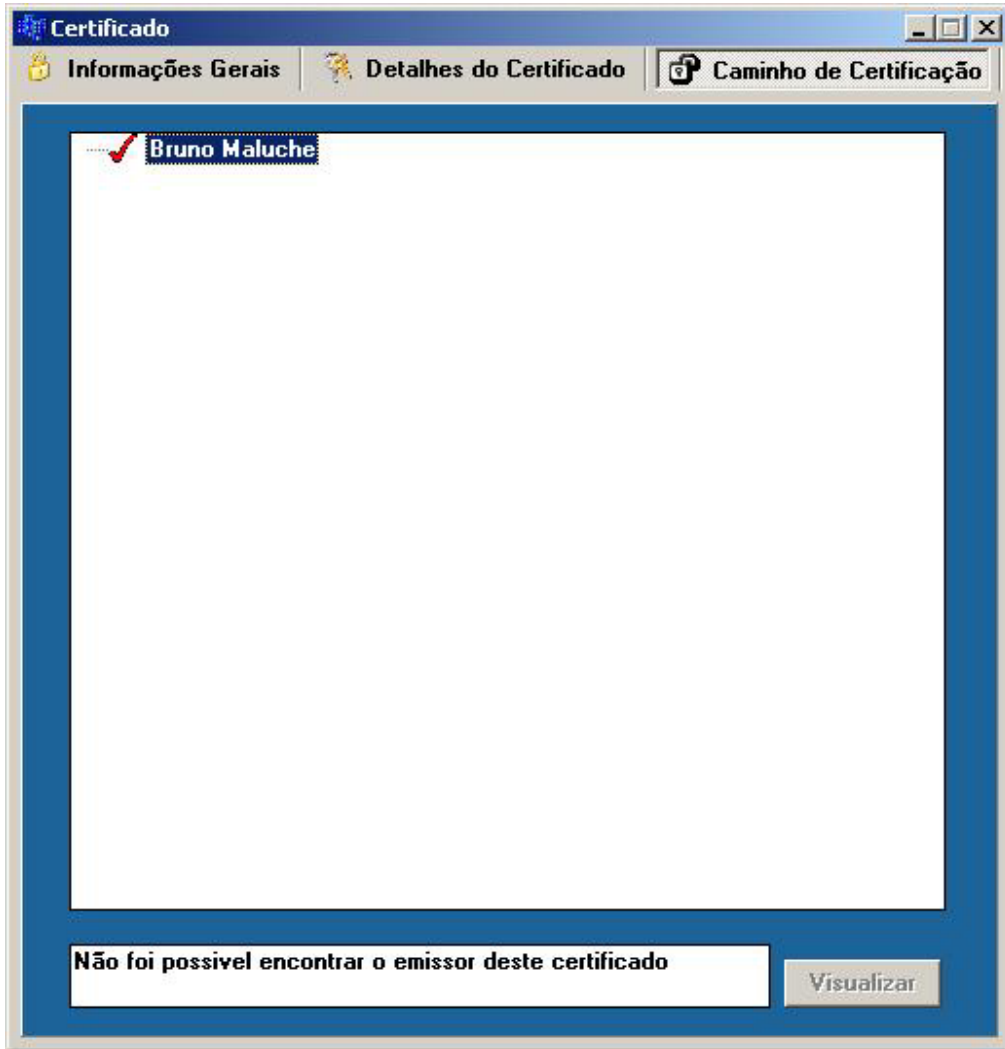


Figura 22 –Visualização de um caminho de certificação inválido

1. A figura no canto superior direito da tela (um cadeado aberto) indica que o caminho de certificação não pode ser construído corretamente.

# Capítulo 6

## 6.1 Conclusão

O propósito do trabalho foi atingido, foi possível construir a árvore de certificação de um certificado em geral, assim como validá-lo, porém, no decorrer da implementação, encontramos certificados que são codificados diferentemente dos suportados por nosso sistema, para estes certificados seria necessário uma extensão deste trabalho.

Os métodos de construção da árvore de certificação ainda não utilizam a estrutura de diretórios LDAP, que é um importante repositório para certificados.

Certificação e assinatura digital são soluções que ajudam garantir que a evolução tecnológica na área de TI (tecnologia da informação) proporcione benefícios a todos nas grandezas de suas descobertas. Porém, na construção deste trabalho percebemos que as normas para o uso destes artifícios ainda não estão totalmente padronizadas e definidas. Com isso, a dificuldade de garantir segurança, integridade e confidencialidade ainda não é trivial nestes sistemas.

Notamos também que muitos sistemas disponibilizam as informações do certificados de maneira incorreta aos seus usuários e isto pode trazer grande confusão aos mesmos.

O processo de certificação digital dentro do Brasil é muito recente (o projeto de lei foi aprovado somente em agosto de 2001) e existem muitos trabalhos a serem feitos. Trabalhos futuros a este poderiam completar as outras estruturas de validação de um certificado digital e não somente o hierárquico, tratado neste trabalho. Também podemos



sugerir a implementação de um sistema de validação de certificados utilizando LDAP para montar a árvore de certificação.

Esperamos que este trabalho impulse a motivação para trabalhos futuros, para garantir um conjunto de sistemas capaz de beneficiar todos que queiram efetuar transições seguras dentro de um sistema computacional, assim como se torne um pequeno guia de como funciona a certificação digital de maneira clara e objetiva.

## Bibliografia

- [1] Housley, Russ; Polk, Tim. Planning for PKI, New York: Wiley, 2001
- [2] KINGDON, K. W.; JR., B. S. K. Extensions and revisions to PKCS #7. An RSA Laboratories Technical Note, [S.l.], 1997.
- [3] LABORATORIES, R. PKCS #7: Cryptographic Message Syntax Standard. Version 1.5.
- [4] LABORATORIES, R. Pkcs #10 v1.7 : Certification request syntax standard. RSA Laboratories, Maio, 2000. Relatório técnico.
- [5] Larmouth, John. ASN.1 Complete: Open System Solutions, 1999.
- [6] IETF, editor. SPKI Certificate Theory, [www.ietf.org/rfc/rfc2693.txt](http://www.ietf.org/rfc/rfc2693.txt), 1999.
- [7] ITU-T. Recommendation X.509. Draft.
- [8] FEGHHI, J.; FEGHHI, J.; WILLIAMS, P. **Digital Certificates**. 1. ed. AddisonWesley, Setembro, 1999.
- [9] ARAÚJO, R. S. D. S. **Protocolos Criptográficos Para Votação Digital**. Florianópolis: Universidade Federal de Santa Catarina, 2002. Dissertação de Mestrado.
- [10] ADAMS, C.; LLOYD, S. **Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations**. 1. ed. Macmillan Technical Publishing, 1999.



## **Anexos**

## **ANEXO 1: ARTIGO DO TRABALHO**

## Sistema para Validação e Visualização de Certificados Digitais

Com a atual “explosão” da Internet, surgiram uma infinidade de facilidades para o acesso a informação. Entre essas facilidades podemos citar o comércio eletrônico e as transações bancárias através dos *home-bankings*. Contudo, essa facilidade de acesso a informação trouxe um grande problema, pois a Internet não possuía meios eficientes para garantir a segurança eletrônica.

Nesse ambiente, tornou-se indispensável o desenvolvimento de novas tecnologias que garantam segurança nas transações *on-line*. A tecnologia que surgiu com maior êxito foi a criptografia.

Suponha que alguém deseje enviar uma mensagem para um amigo, e deseje ter a certeza que ninguém terá acesso a ela. Entretanto existe a possibilidade de alguém interceptar a mensagem e abri-la. A mensagem enviada é chamada de **texto plano**. Um modo de esconder o conteúdo da mensagem é denominado de **cifragem**, e o resultado da cifragem é o **texto cifrado**. O processo inverso, para se obter o texto plano novamente, é denominado **decifragem**. Criptografia então, pode ser definida como a arte ou ciências de esconder informações.

É interessante notar que muitas aplicações tecnológicas não são efetuadas hoje no mundo digital por falta de segurança e privacidade. Por exemplo, tecnologicamente hoje não seria difícil ou mesmo custoso fazer a implementação de um sistema de laudos médicos via Internet. O problema está, em como garantir que o laudo será efetuado pelo médico responsável pelo paciente. Problemas como este nos levam a criar expiração em prover uma tecnologia segura para prover maior uso e conforto das ferramentas hoje já disponível para grande parte da população.

A segurança em sistemas computacionais está baseada na privacidade das informações e no restrito acesso às pessoas ou mesmo outros agentes computacionais,

mantendo o sigilo e a autenticidade da informação. Para que isto seja possível, é necessário o uso da criptografia entre as transmissões de dados mencionadas.

Com o avanço da tecnologia da informação (TI), esta necessidade cresceu imensamente, e em vários pontos é uma barreira a ser vencida para que sistemas contribuam de melhor forma para a sociedade como um todo. Vários exemplos ainda hoje podem ser mencionados, que não são implantados pela falta de segurança e auditoria dos mesmos, apesar de que tecnologicamente os sistemas estariam aptos para fazer o que se propõe. Um exemplo simples: imagine um sistema onde um médico pudesse verificar on-line tomografias computadorizadas de seus pacientes, e submeter os laudos ao hospital, sem o deslocamento físico. Apesar de tecnologicamente possível, sem a garantia deste sistema, nada pode provar que o laudo passado ao hospital é mesmo do devido médico, somado a isso, provar que as informações transmitidas dos pacientes passarão pela rede sem que ninguém, por algum meio indevido tivesse acesso a elas.

## Criptografia simétrica

Existem dois grandes grupos de técnicas para o uso da criptografia. A convencional, que é baseada em uma chave privada (secreta), onde todos que devem ter acesso à informação requerida devem conhecer a chave. Adicionado isso, outra grande característica desta técnica é que a mesma chave é usada para a cifragem (ou codificação) da informação e para decodificação. O outro grupo seria a criptografia assimétrica.

## Criptografia assimétrica

Esta técnica de criptografia também é conhecida como criptografia por chaves públicas. Ela se baseia no uso de uma par de chaves, uma pública e outra privada, para a cifragem das informações. Diferentemente da técnica convencional, nesta técnica a informação cifrada com a chave pública, só poderá ser decodificada com o uso da chave privada, e de maneira contrária também, ou seja, uma informação cifrada ou codificada com a chave privada, só poderá ser lida com o uso da chave pública. Outra característica

importante desta técnica de criptografia, é que o conhecimento da chave pública não permite a descoberta da chave privada.

## Assinatura Digital

A assinatura digital, assim como a convencional, procura oferecer garantias de identificação da autoria do documento à qual é aposta, como também da integridade de seu conteúdo desde o ato de sua assinatura. Serve também para vincular vontade ou anuência do autor ao conteúdo do documento, em contratos. Por isso não se deve assinar papel em branco nem documento rasurado ou não lido, nem se dar credibilidade a documentos assinados que contenham rasura.

## PKCS

A empresa RSA Security definiu uma série de padrões para o uso da criptografia de chaves públicas. Esses padrões foram denominados de *Public Key Cryptography Standards – PKCS*.

PKCS descreve a sintaxe para mensagens de uma maneira abstrata, e oferece detalhes completos sobre os algoritmos. Porém esses padrões não descrevem a maneira correta de representar essas mensagens, embora BER seja a escolha mais lógica.

Atualmente, a grande maioria das aplicações que utilizam a criptografia de chaves públicas, utiliza esses padrões.

## ASN.1

*Abstract Syntax Notation One*, é a notação para descrever tipos e valores abstratos de dados.

Em ASN.1, um tipo é um conjunto de valores. Para alguns tipos, existe um número finito de valores, e para outros existem infinitos valores. ASN.1 possui quatro classes de tipo:



- **Simple:** são tipos primitivos, sem componentes adicionais.
- **Estruturado:** Possuem componentes, eles são *sets* ou *sequences* de outros tipos.
- **Rotulados:** São derivados de qualquer outro tipo.
- **Outros tipos:** São tipos especiais, que não se enquadram em nenhuma das 3 classes anteriores. São eles: *Choice* e *Any*.

Tipos podem receber nomes com ASN.1, através do operador ( $::=$ ) e esses nomes podem ser usados na definição de outros tipos.

Todos os tipos ASN.1 recebem um rótulo (TAG), exceto os tipos CHOICE e ANY, e este rótulo consiste de uma classe e um número não negativo. Como regra, temos que dois rótulos são iguais se, e somente se, seus números forem iguais. Deste modo, o significado abstrato de um tipo só é afetado pelo seu rótulo e não pelo seu nome.

## ICP

ICP é um sistema que utiliza criptografia assimétrica e certificados digitais para conseguir serviços seguros na Internet. Sendo assim, a ICP define uma série de serviços para o uso das tecnologias baseadas em chaves públicas. Tais serviços se tornam imprescindíveis à medida que a Internet tornou-se um meio muito utilizado para realização de comunicações e transações.

## Autoridades Certificadoras (ACs)

A Autoridade Certificadora (AC) é o bloco de construção básico do ICP. A AC é constituída por um conjunto de computadores, softwares e pelas pessoas que controlam tudo isso. Uma AC é caracterizada por 2 atributos básicos :

- Nome
- Chave pública.

As quatro funções ICP principais realizadas pela AC são:

5. Emitir certificados (ou seja, criar e assiná-los);
6. Emitir as LCRs (listas de certificados revogados);

7. Publicar os certificados e LCRs para que qualquer pessoa possa obtê-los.
8. Manter arquivos de informação sobre o *status* de certificados revogados que foi emitido.

Uma Autoridade Certificadora pode emitir certificados para usuários ou para outras ACs. Um certificado emitido é assinado com a chave privada da AC, e desta maneira pode ser comprovada sua autenticidade verificando-se a assinatura com a chave pública da AC.

## Certificados Digitais

Sem duvida alguma, a segurança é a principal preocupação nas transações via Internet. Tendo em vista que a Internet é um sistema aberto e informações enviadas de um ponto ao outro podem ser lidas e/ou alteradas por várias pessoas, é necessário prover uma solução para garantir a Confidencialidade (privacidade), integridade, não repúdio e autenticação, conforme as especificações da ICP.

A solução foi usar a **criptografia** (para garantir a privacidade) e os **certificados digitais** (para garantir que a comunicação está ocorrendo entre os dois pontos desejados e para garantir a confidencialidade)

## Certificados X509

Foi desenvolvido pela *International Standards Organization* (ISO) e incorporado pela *American National Standards Institute*(ANSI) e Internet Engineering Task Force(IETF). Atualmente se encontra na sua terceira versão X509 v3.

A instrutura interna é baseada em ASN.1 e codificada em BER ou DER.

As principais informações contidas neste certificado são:

- Número de série: Todo certificado tem que ter um número de série, e esse número é único para cada entidade certificadora.
- Identificador do algoritmo de assinatura: Uma assinatura digital pode ser feita de diversas formas distintas, dependendo do algoritmo utilizado. Este campo identifica o algoritmo utilizado para a geração desta assinatura.
- Emissor: Nome da entidade certificadora responsável pela emissão do certificado em questão.
- Período de Validade: A validade do certificado. É importante salientar que toda AC (Autoridade Certificadora) possui uma lista de certificados revogados, para fazer a verificação se o certificado analisado é válido ou não.
- Titular: Nome ou identificador do titular do certificado.
- Identificador único do emissor do titular: Este campo juntamente com o campo *titular* foi criado na versão dois (v2) do X.509 com a intenção do reuso do nome do emissor junto com o nome do titular no decorrer dos anos. Atualmente isto não é mais usado. Recomenda-se a emissão de um novo certificado.
- Extensões: A V3 do X.509 permite que o certificado contenha campos fora dos acima especificados e que, muitas vezes, devem ser proprietários.
- Assinatura do emissor: É a assinatura digital da autoridade certificadora. A assinatura digital é feita para todos os campos descritos acima, inclusive as extensões. Assim sendo qualquer alteração no certificado é facilmente identificada. Por esta razão um certificado digital jamais pode ser modificado. A única maneira de adulterar um certificado é descobrir a chave privada da AC e gerar um novo certificado com ela.

## Estrutura Interna X509

Apresentaremos agora a estrutura interna de um Certificado X.509 na notação ASN.1

```
Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
```

```

signatureAlgorithm  AlgorithmIdentifier,
signatureValue      BIT STRING  }

TBSCertificate ::= SEQUENCE {
    version          [0]  EXPLICIT Version DEFAULT v1,
    serialNumber     CertificateSerialNumber,
    signature        AlgorithmIdentifier,
    issuer           Name,
    validity         Validity,
    subject          Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                    -- Se presente, versão deve ser 2 ou 3
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                    -- Se presente, versão deve ser 2 ou 3
    extensions      [3]  EXPLICIT Extensions OPTIONAL
                    -- Se presente, versão deve ser 3
}

Version ::= INTEGER { v1(0), v2(1), v3(2)  }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
    notBefore      Time,
    notAfter       Time  }

Time ::= CHOICE {
    utcTime        UTCTime,
    generalTime    GeneralizedTime  }

UniqueIdentifier ::= BIT STRING

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm       AlgorithmIdentifier,
    subjectPublicKey BIT STRING  }

Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

```

```
Extension ::= SEQUENCE {
    extnID      OBJECT IDENTIFIER,
    critical    BOOLEAN DEFAULT FALSE,
    extnValue   OCTET STRING }
```

## Validação de um certificado digital X.509

A validação de um certificado digital é feita mediante a construção de sua árvore de certificação. Uma árvore de certificação é uma corrente de certificados, onde a AC do primeiro certificado é o ponto confiável

## Implementação

O sistema desenvolvido funciona como um recipiente para certificados digitais. Inicialmente ele já vem com algumas ACs armazenados no banco de dados. O usuário terá a opção de adicionar certificados ao sistema, no formato X.509 (Codificação Binária-BER ou DER). Ao adicionar um certificado o sistema, um parser ASN.1 (**ASN1Parser**) decodificará o arquivo, deixando-o no formato texto (em ASN.1). Desta forma varremos a estrutura ASN.1, através de um decodificador(**X509Decoder**). Então criamos uma estrutura **X509Cert**, que contém todos os dados pertencentes ao certificado

## **ANEXO 2: SENTENÇAS SQL PARA CRIAÇÃO DO BANCO DE DADOS**

```

CREATE TABLE "CA"
(
  "ID"          INTEGER NOT NULL,
  "NOME"       CHAR(255) NOT NULL,
  "SERIAL"     CHAR(255) NOT NULL,
  "CERTIFICADO" BLOB SUB_TYPE 0 SEGMENT SIZE 80 NOT NULL,
  "TYPECA"     INTEGER,
  "COUNTRYNAMEISSUER" CHAR(255),
  "ORGANIZATIONUNITNAMEISSUER1" CHAR(255),
  "ORGANIZATIONUNITNAMEISSUER2" CHAR(255),
  "COMMONNAMEISSUER" CHAR(255),
  "LOCALITYNAMEISSUER" CHAR(255),
  "STATEORPROVINCENAMEISSUER" CHAR(255),
  "EMAILADDRESSISSUER" CHAR(255),
  "ORGANIZATIONISSUER" CHAR(255),
  CONSTRAINT "PK_CA" PRIMARY KEY ("ID")
);

```

```

CREATE TABLE "USUARIO"
(
  "ID"          INTEGER NOT NULL,
  "NOME"       CHAR(255) NOT NULL,
  "SENHA"     CHAR(10) NOT NULL,
  CONSTRAINT "PK_USUARIO" PRIMARY KEY ("ID")
);

```

```

CREATE TABLE "USUARIO_CA"
(
  "ID"          INTEGER NOT NULL,
  "ID_USUARIO"  INTEGER NOT NULL,
  "ID_CA"       INTEGER NOT NULL,
  CONSTRAINT "PK_USUARIO_CA" PRIMARY KEY ("ID")
);

```

## **ANEXO 3: CÓDIGO FONTE**



ASN1Decoder

```
//-----  
-----  
#ifndef ASN1DecoderH  
#define ASN1DecoderH  
//-----  
-----  
#include <string>  
#include <sstream.h>  
#include <fstream.h>  
#include <vcl.h>  
//-----  
-----  
#include <X509Cert.h>  
#include <KeyUsage.h>  
#include <KeyUsageExt.h>  
#include <SubjectKeyIdentifier.h>  
#include <AuthorityKeyIdentifier.h>  
#include <BasicConstraints.h>  
#include "CRLDistributionPoints.h"  
#include "PvtKeyUsagePeriod.h"  
#include "CertificatePolicies.h"  
#include "NetscapeCertType.h"  
#include "AuthorityInfoAccess.h"  
#include "PolicyConstraints.h"  
#include "InhibitAnyPolicy.h"  
#include "X509Functions.h"  
//-----  
-----  
enum extensions {  
    AUTHORITY_KEY_IDENTIFIER = 1, SUBJECT_KEY_IDENTIFIER  
, KEY_USAGE,  
    BASIC_CONSTRAINTS, POLICY_CONSTRAINTS, CERTIFICATE_POLICIES,  
SUBJECT_ALT_NAME, ISSUER_ALT_NAME,  
  
POLICY_MAPPINGS, SUBJECT_DIRECTORY_ATTRIBUTES, PRIVATE_KEY_USAGE_PERIOD,  
    NETSCAPE_CERT_TYPE, CRL_DISTRIBUTION_POINTS, EXT_KEY_USAGE,  
  
INHIBIT_ANY_POLICY, FRESHEST_CRL, AUTHORITY_INFO_ACCESS, SUBJECT_INFO_ACCE  
SS,  
    NETSCAPE_CA_POLICY_UL  
};  
using namespace std;  
//-----  
-----  
  
class ASN1Decoder  
{  
    protected:  
        fstream file;  
        X509Cert *certificate;  
        int extension;  
        int controlOpenExt;  
        int controlCloseExt;
```

```

public:
    ASN1Decoder(string);
    ASN1Decoder(string*);
    ~ASN1Decoder();
    bool decode();
    bool getVersion();
    void getSerialNumber();
    void skipWhiteSpaces();
    void getSignature();
    void getIssuerFields();
    string getString();
    string getStringNormal();
    void getValidity();
    void getSubjectFields();
    int countClose();
    void getPublicKeyAlgorithm();
    void getPublickey();
    void getRSAExpoente();
    void identifyExtension();
    void getExtension();
    void extensionProcess();
    bool getCritical();
    void getSignatureValue();
    void find(string,int);
    bool verifyCrl();
    void getCRLFields(CRLDistributionPoints& __crl);
    char findFirstChar();
    X509Cert* getCert(){return certificate;};
    void closeFile() { file.close();};
    void setType();

};
//-----
-----
#endif

#include "ASN1Decoder.h"

////////////////////////////////////
// Construction/Destruction
////////////////////////////////////

ASN1Decoder::ASN1Decoder(string __filename)
{
    file.open(__filename.c_str(),ios::in);
    certificate = new X509Cert();
    certificate->setCertificateStream(&file);
}

ASN1Decoder::ASN1Decoder(string* __stream)
{
    file.open(__stream->c_str() ,ios::in);
}

ASN1Decoder::~ASN1Decoder()

```

```

{}

/*****
*****
Nome: decode

Ação: decodifica um arquivo ASN.1 retirando todas as informações para a
visualização e posterior
      validação.

Parametros: sem parametros.

Returns: bool; true se a decodificação ocorreu sem erros, false caso
ocorra algum erro
*****
*****/
bool ASN1Decoder::decode()
{
    getVersion();
    getSerialNumber();
    getSignature();
    getIssuerFields();
    getValidity();
    getSubjectFields();
    getPublicKeyAlgorithm();
    getPublicKey();
    getRSAExpoente();
    if (certificate->getVersion() == 3)
        extensionProcess();
    getSignatureValue();
    setType();
    return true;
}

/*****
*****
Nome: getVersion

Ação: Percorre o certificado em busca da versão do mesmo, que pode ser
1,2 ou 3.

Parametros: sem parametros.

Returns: bool; true se a versão foi encontrada, false caso contrário.

Commentário: A versão apresenta-se no formato n-1, por isso é
necessário incrementar o valor
             encontrado.
*****
*****/
bool ASN1Decoder::getVersion()
{
    string buffer,dummy;
    int version;

    file >> dummy >> dummy >> dummy >> dummy >> buffer >> dummy;;
    if(buffer == "[0]")

```

```

        file >> dummy >> version;
    else
    {
        /* Versão não presente (Versão 0) */
        version = -1; // Somente para testes futuros
        file.unget(); file.unget(); file.unget();
    }
    ++version; // v1(0),v2(1),v3(2)
    certificate->setVersion(version);

    // ShowMessage(version);

    return false;
}

/*****
*****
Nome: getSerialNumber

Ação: Percorre o certificado em busca da versão do mesmo, que pode ser
1,2 ou 3.

Parametros: sem parametros.

Returns: void.

Commentário: A versão apresenta-se no formato n-1, por isso é
necessário incrementar o valor
encontrado.
*****/
void ASN1Decoder::getSerialNumber()
{
    string serialNumber;
    char* dummy = new char[30],ch;
    if(certificate->getVersion() != 0)
    {
        file >> dummy >> dummy; // Somente para chegar a informação
desejada
        file.get(); // Retirar '\n'
    }

    skipWhiteSpaces();
    ch = file.get();
    do
    {
        if(ch == '\n')
            skipWhiteSpaces();
        serialNumber.push_back(ch);
        ch = file.get();
    }while(ch != 'S');

    if(serialNumber[serialNumber.size()-1] == '\n')
        serialNumber.pop_back();

    certificate->setSerialNumber(serialNumber);
    //ShowMessage(serialNumber.c_str()); // only test
}

```

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

Nome: skipWhiteSpaces

Ação: Elimina os espaços Brancos do trecho desejado.

Parametros: sem parametros.

Returns: void.

```
*****  
*****  
*****
```

```
void ASN1Decoder::skipWhiteSpaces()  
{
```

```
    char ch;  
    do  
    {  
        ch = file.get();  
    }while(ch == ' ');  
    file.unget();  
}
```

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

Nome: getSignature

Ação: Percorre o certificado em busca da Asssinatura.

Parametros: sem parametros.

Returns: void.

```
*****  
*****  
*****
```

```
void ASN1Decoder::getSignature()  
{
```

```
    string signature;  
    string dummy;  
    //file >> dummy >> dummy >> dummy >> dummy >> signature;  
    find("IDENTIFIER",1);  
    file >> signature;  
    certificate->setSignature(signature);  
    //ShowMessage(signature.c_str());    // only test  
}
```

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

Nome: getIssuerFields

Ação: Percorre o certificado em busca dos campos relacionados ao 'Issuer'.

Parametros: sem parametros.

Returns: void.

```
*****  
*****  
*****
```

```
void ASN1Decoder::getIssuerFields()
```

```

{
    string buffer,temp;
    int cont = 1;
    X509DN *_issuer;
    _issuer = certificate->getIssuer();
    while(!file.eof())
    {
        file >> buffer;
        if (buffer == "IDENTIFIER")
        {
            file >> temp;
            if(temp == "emailAddress")
                _issuer->setEmailAddress(getString());
            else if(temp=="countryName")
                _issuer->setCountryName(getString());
            else if(temp=="stateOrProvinceName")
                _issuer->setStateOrProvinceName(getString());
            else if(temp=="localityName")
                _issuer->setLocalityName(getString());
            else if(temp=="organizationName")
                _issuer->setOrganization(getString());
            else if(temp=="organizationalUnitName")
            {
                if(cont ==1)
                {
                    _issuer->setOrganizationUnitName1(getString());
                    cont++;
                }
                else
                    _issuer->setOrganizationUnitName2(getString());
            }
            else if(temp=="commonName")
                _issuer->setCommonName(getString());
            else if(temp=="rfc822Mailbox")
                _issuer->setMailBox(getString());
            else if(temp=="telephoneNumber")
                _issuer->setTelephoneNumber(getString());
            if(countClose() >= 3) return; //Testa o fim dos campos
Issuers
        }
    }
}

/*****
*****
Nome: find

Ação: Localiza o string ou o caracter desejado no certificado.

Parametros: string, int; É o string que será procurado no
certificado.__type define se a procura é por um string(1) ou
char (2).

Returns: void.
*****/

```

```

void ASN1Decoder::find(string __target, int __type)
{
    string _temp;
    switch (__type)
    {
        case 1:
            file >> _temp;
            while (_temp != __target)
                file >> _temp;
            break;
        case 2:
            _temp = file.get();
            while (_temp != __target)
                _temp = file.get();
            break;
    }
}

```

```

/*****
*****

```

Nome: findFirstChar

Ação: Encontra o primeiro char.

Parametros: void;

Returns: char; o caracter encontrado.

```

*****
*****/

```

```

char ASN1Decoder::findFirstChar()
{
    char _ch;
    _ch = file.get();
    while ((_ch == '\n') || (_ch == ' '))
        _ch = file.get();
    return _ch;
}

```

```

/*****
*****

```

Nome: countClose

Ação: Conta a quantidade de '}', para verificar se os campos do Issuer ou Subject chegou ao fim.

Parametros: sem parametros.

Returns: int; Indicando o numero de '}' consecutivos encontrado.

Comentário: Um retorno 3 indica que os campos de Issuer ou Subject terminou.

```

*****
*****/

```

```

int ASN1Decoder::countClose()
{

```

```

string text;
int cont =0;      // Quantidade de '}'
while(text != "}") // Encontrar o 1º '}'
{
    file >> text;
    if (text == "}")
        cont++;
}
file >> text;
while(text == "}")
{
    cont++;
    file >> text;
    // if(text == "}")
    //     cont++;
}
return cont;
}

```

```

/*****
*****

```

Nome: getString

Ação: Percorre o certificado em busca do próximo string, verificando a possibilidade de um ' no meio do string.

Parametros: sem parametros.

Returns: string; retorna o String solicitado

```

*****
*****/

```

```

string ASN1Decoder::getString()
{

```

```

    char ch;
    ch= '9'; //dummy
    string texto;

```

```

    while(ch != '\\') // Procura o início do String, representado por '
        ch = file.get();

```

```

    ch = file.get();
    while(ch != '\\') //Procura o final do String.
    {

```

```

        texto.push_back(ch);
        ch = file.get();
        if (ch == '\\') // Verifica se o caracter ' pertence ao
string ou eh delimitador final

```

```

        {
            ch = file.get();
            if (ch == '\\n')
            {
                file.unget();
                ch = '\\';
            }
            else
                texto.push_back('\\');

```



```

    }

    }
    return texto;
}

/*****
*****
Nome: getStringNormal

Ação: Percorre o certificado em busca do próximo string.

Parametros: sem parametros.

Returns: string; retorna o String solicitado
*****/
string ASN1Decoder::getStringNormal()
{
    char ch;
    ch= '9'; //dummy
    string texto;

    while(ch != '\\') // Procura o início do String, representado por '
        ch = file.get();

    ch = file.get();
    while(ch != '\\') //Procura o final do String.
    {
        texto.push_back(ch);
        ch = file.get();
    }
    return texto;
}

/*****
*****
Nome: getValidity

Ação: Percorre o certificado em busca dos campos relacionados ao
'Subject'.

Parametros: sem parametros.

Returns: void.
*****/
void ASN1Decoder::getValidity()
{
    DateTime *notBefore;
    DateTime *notAfter;
    notBefore = new DateTime();
    notAfter = new DateTime();
    int year;

```

```

string before = getString();
string after = getString();

notBefore->decodeDate(before);

certificate->setNotBefore(notBefore);
notAfter->decodeDate(after);
certificate->setNotAfter(notAfter);
}

/*****
*****
Nome: getSubjectFields

Ação: Percorre o certificado em busca das datas de validade e
decodifica elas.

Parametros: sem parametros.

Returns: void.
*****/
void ASN1Decoder::getSubjectFields()
{
    string buffer,temp;
    X509DN *_subject;
    _subject = certificate->getSubject();
    int cont = 1;
    while(!file.eof())
    {
        file >> buffer;
        if (buffer == "IDENTIFIER")
        {
            file >> temp;
            if(temp == "emailAddress")
                _subject->setEmailAddress(getString());
            else if(temp=="countryName")
                _subject->setCountryName(getString());
            else if(temp=="stateOrProvinceName")
                _subject->setStateOrProvinceName(getString());
            else if(temp=="localityName")
                _subject->setLocalityName(getString());
            else if(temp=="organizationName")
                _subject->setOrganization(getString());
            else if(temp=="organizationalUnitName")
            {
                if(cont ==1)
                {
                    _subject->setOrganizationUnitName1(getString());
                    cont++;
                }
                else
                    _subject->setOrganizationUnitName2(getString());
            }
            else if(temp=="commonName")
                _subject->setCommonName(getString());
            else if(temp=="rfc822Mailbox")

```

```

        _subject->setMailBox(getString());
    else if(temp=="telephoneNumber")
        _subject->setTelephoneNumber(getString());

    if(countClose() >= 3) return; //Testa o fim dos campos
Subject
    }
}

/*****
*****
Nome: getPublicKeyAlgorithm

Ação: Percorre o certificado em busca do 'Public Key Algorithm'.

Parametros: sem parametros.

Returns: void.
*****/
void ASN1Decoder::getPublicKeyAlgorithm()
{
    string buffer,publicKeyAlgorithm;
    while(!file.eof())
    {
        file >> buffer;
        if (buffer == "IDENTIFIER")
        {
            file >> publicKeyAlgorithm;
            certificate->setPublicKeyAlgorithm(publicKeyAlgorithm);
            //ShowMessage(publicKeyAlgorithm.c_str()); // only test
            return;
        }
    }
}

/*****
*****
Nome: getPublickey

Ação: Percorre o certificado em busca da chave pública.

Parametros: sem parametros.

Returns: void.
*****/
void ASN1Decoder::getPublickey()
{
    string buffer,publicKey;
    char ch;
    while(!file.eof())
    {
        file >> buffer;
        if(buffer == "INTEGER") // Começo da chave Pública

```

```

    {
        skipWhiteSpaces();
        ch = file.get();
        do
        {
            if(ch == '\n')
                skipWhiteSpaces();
            publicKey.push_back(ch);
            ch = file.get();
        }while(ch != 'I');
        certificate->setPublicKey(publicKey);
        //ShowMessage(publicKey.c_str()); // only test
        return;
    }
}

/*****
*****
Nome: getRSAExpoente

Ação: Percorre o certificado em busca do expoente RSA.

Parametros: sem parametros.

Returns: void.
*****/

void ASN1Decoder::getRSAExpoente()
{
    string dummy, expoente;
    char ch;

    file >> dummy;

    skipWhiteSpaces();
    ch = file.get();
    do
    {
        if(ch == '\n')
            skipWhiteSpaces();
        expoente.push_back(ch);
        ch = file.get();
    }while(ch != '}');

    if(expoente[expoente.size()-1] == '\n')
        expoente.pop_back();
    if(expoente[0] == '\n')
        expoente = expoente.erase(0,1);

    certificate->setRSAExpoente(expoente);
}

/*****
*****/

```

Nome: extensionProcess

Ação: Encontra o começo dos campos de Extensão, identificando e armazenado todas elas.

Parametros: sem parametros.

Returns: void.

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

```
void ASN1Decoder::extensionProcess()  
{  
    string temp;  
    while(temp!="[3]") //Encontra o começo das Extensões  
        file >> temp;  
  
    while (true)  
    {  
        identifyExtension();  
        if(extension != -1)  
        {  
            getExtension();  
            extension = 0;  
        }  
        else  
            return;  
    }  
}
```

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

Nome: identifyExtension

Ação: Identifica o OID da próxima extensão.

Parametros: sem parametros.

Returns: void.

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

```
void ASN1Decoder::identifyExtension()  
{  
    string buffer,temp;  
  
    while(!file.eof())  
    {  
        file >> buffer;  
        if (buffer == "IDENTIFIER")  
        {  
            file >> temp;  
            if(temp == "keyUsage")  
                extension = KEY_USAGE;  
            else if(temp=="extKeyUsage")  
                extension = EXT_KEY_USAGE;  
        }  
    }  
}
```

```

else if(temp=="subjectKeyIdentifier")
    extension = SUBJECT_KEY_IDENTIFIER;
else if(temp=="authorityKeyIdentifier")
    extension = AUTHORITY_KEY_IDENTIFIER;
else if(temp=="cRLDistributionPoints")
    extension = CRL_DISTRIBUTION_POINTS;
else if(temp=="authorityInfoAccess")
    extension = AUTHORITY_INFO_ACCESS;
else if(temp=="netscape-cert-type")
    extension = NETSCAPE_CERT_TYPE;
else if(temp=="privateKeyUsagePeriod")
    extension = PRIVATE_KEY_USAGE_PERIOD;
else if(temp=="basicConstraints")
    extension = BASIC_CONSTRAINTS;
else if(temp=="policyConstraints")
    extension = POLICY_CONSTRAINTS;
else if(temp=="certificatePolicies")
    extension = CERTIFICATE_POLICIES;
else if(temp=="privateKeyUsagePeriod")
    extension = PRIVATE_KEY_USAGE_PERIOD;
else if (temp==certificate->getSignature())
    extension = -1;

return;
}
}
}

```

```

/*****
*****

```

Nome: getExtension

Ação: A partir do OID da extensão, decodifica todos os campos referentes a cada uma.

Parametros: sem parametros.

Returns: void.

```

*****
*****/

```

```

void ASN1Decoder::getExtension()
{
    KeyUsage *key;
    KeyUsageExt *keyExt;
    SubjectKeyIdentifier *subjKey;
    AuthorityKeyIdentifier *authKey;
    BasicConstraints *basicCons;
    CRLDistributionPoints *crlPoints;
    PvtKeyUsagePeriod *pvtPeriod;
    CertificatePolicies *policies;
    NetscapeCertType *netscapeCertType;
    AuthorityInfoAccess *authorityInfoAccess;
    PolicyConstraints *policyConstraints;
    InhibitAnyPolicy *inhibitAny;
    string _temp,_before,_after;

```

```

DateTime notBefore,notAfter;

char _ch;
int _num;

switch (extension)
{
    case KEY_USAGE:
        key = new KeyUsage();
        key->setIsCritical(getCritical());
        key->setBits(getStringNormal());
        key->decodeBits();
        key->setType(KEY_USAGE);
        certificate->addExtension(key);
        break;

    case EXT_KEY_USAGE:
        keyExt = new KeyUsageExt();
        keyExt->setIsCritical(getCritical());
        keyExt->setType(EXT_KEY_USAGE);
        find("OBJECT",1);
        keyExt->processKeyExt(file);
        certificate->addExtension(keyExt);

        break;

    case SUBJECT_KEY_IDENTIFIER:
        subjKey = new SubjectKeyIdentifier;
        subjKey->setType(SUBJECT_KEY_IDENTIFIER);
        subjKey->setIsCritical(getCritical());
        find("STRING",1);
        file.get();
        skipWhiteSpaces();
        subjKey->extractKey(file);
        certificate->addExtension(subjKey);
        // ShowMessage(subjKey->getKey().c_str()); // only test
        break;

    case AUTHORITY_KEY_IDENTIFIER:
        authKey = new AuthorityKeyIdentifier;
        authKey->setIsCritical(getCritical());
        authKey->setType(AUTHORITY_KEY_IDENTIFIER);
        find("[",2);
        authKey->setFields(file);
        certificate->addExtension(authKey);
        // ShowMessage(authKey->getKey().c_str()); // only test
        break;

    case BASIC_CONSTRAINTS:
        basicCons = new BasicConstraints();
        basicCons->setIsCritical(getCritical());
        basicCons->setType(BASIC_CONSTRAINTS);
        find("SEQUENCE",1);
        file >> _temp >> _temp;
        if(_temp == "BOOLEAN")
        {
            file >> _temp;

```

```

        if(_temp == "true" || _temp == "TRUE")
            basicCons->setCa(true);
        else
            basicCons->setCa(false);
        file >> _temp;
        if(_temp == "INTEGER")
        {
            file >> _num;
            basicCons->setPathLen(_num);
        }
    }
    certificate->addExtension(basicCons);
    break;

case NETSCAPE_CERT_TYPE:
    netscapeCertType = new NetscapeCertType();
    netscapeCertType->setIsCritical(getCritical());
    netscapeCertType->setType(NETSCAPE_CERT_TYPE);
    netscapeCertType->setBits(getStringNormal());
    netscapeCertType->decodeBits();
    certificate->addExtension(netscapeCertType);

    break;

case CRL_DISTRIBUTION_POINTS:
    crlPoints = new CRLDistributionPoints();
    crlPoints->setIsCritical(getCritical());
    crlPoints->setType(CRL_DISTRIBUTION_POINTS);
    if (verifyCrl())
    {
        _ch = file.get();
        switch (_ch)
        {
            case '4': //directoryName
                getCRLFields(*crlPoints);
                break;
            case '6':
                crlPoints->putUrl(getString());
                //uniformResourceIdentifier
                if(countClose() <= 3)
                {
                    if (verifyCrl())
                    {
                        _ch = file.get();
                        if(_ch == '6')
                            crlPoints->putUrl(getString());
                        else
                            ShowMessage("BEHHHHHHH CRL CRL");
                    }
                }
            }
        }
    }
    certificate->addExtension(crlPoints);
    break;

```



```

After
    case PRIVATE_KEY_USAGE_PERIOD: // ARRUMAR [0] Before / [1]
        pvtPeriod = new PvtKeyUsagePeriod();
        pvtPeriod->setIsCritical(getCritical());
        pvtPeriod->setType(PRIVATE_KEY_USAGE_PERIOD);

        _before = getString();
        _after = getString();

        notBefore.decodeDate(_before);
        pvtPeriod->setNotBefore(notBefore);

        notAfter.decodeDate(_after);
        pvtPeriod->setNotBefore(notAfter);
        certificate->addExtension(pvtPeriod);
        break;

    case CERTIFICATE_POLICIES:
        policies = new CertificatePolicies();
        policies->setIsCritical(getCritical());
        policies->setType(CERTIFICATE_POLICIES);
        do
        {
            find("IDENTIFIER",1);
            _ch = findFirstChar();
            if(_ch == '\\')
            {
                file.unget();
                policies->setIdentifier(getString());
            }
            else
            {
                file.unget();
                _temp.clear();
                _ch = file.get();
                while(_ch != '(')
                {
                    _temp.push_back(_ch);
                    _ch = file.get();
                }
                policies->setIdentifier(_temp);
            }
            file >> _temp;
            if(_temp == "}") // Testar fim da extensão
            {
                file >> _temp;
                if(_temp == "}")
                {
                    certificate->addExtension(policies);
                    return;
                }
                else
                    continue;
            }
        }
        else
            break;

```

```

}while(!file.eof());

do
{
    find("IDENTIFIER",1);
    file >> _temp;
    if(_temp == "cps")
    {
        policies->setQualifierID(_temp);
        policies->setQualifier(getString());
    }
    else
    {
        policies->setQualifierID2(_temp);
        policies->setNoticeRef(getString());
        find("INTEGER",1);
        file >> _num;
        policies->setNoticeNumber(_num);
        policies->setQualifier(getString());
    }
}while(countClose() == 1);

certificate->addExtension(policies);
break;

case AUTHORITY_INFO_ACCESS:
    authorityInfoAccess = new AuthorityInfoAccess();
    authorityInfoAccess->setIsCritical(getCritical());
    authorityInfoAccess->
>setType(AUTHORITY_INFO_ACCESS);
    find("IDENTIFIER",1);
    file >> _temp;
    authorityInfoAccess->setAccessMethod(_temp);
    do
    {
        authorityInfoAccess->addCAIssuers(getString());
    }while(countClose() == 1);
    certificate->addExtension(authorityInfoAccess);
    break;

case POLICY_CONSTRAINTS: // // FIX IT
    policyConstraints = new PolicyConstraints();
    policyConstraints->setIsCritical(getCritical());
    policyConstraints->setType(POLICY_CONSTRAINTS);

    find("[",2);
    _ch = file.get();file.get();
    file >> _num;
    switch (_ch)
    {
        case '0':
            policyConstraints->
>setRequiredExplicit(_num);
            break;
        case '1':
            policyConstraints->setInhibitPolicy(_num);

```

```

        break;
    }
    certificate->addExtension(policyConstraints);
    break;
case INHIBIT_ANY_POLICY:
    inhibitAny = new InhibitAnyPolicy();
    inhibitAny->setIsCritical(getCritical());
    inhibitAny->setType(INHIBIT_ANY_POLICY);

    find("[",2);
    file.get();file.get();
    file >> _num;
    inhibitAny->setInhibitAnyPolicy(_num);
    certificate->addExtension(inhibitAny);

    break;
}
}

/*****
*****
Nome: getCritical

Ação: Verifica se a extensão é critica ou não.

Parametros: sem parametros.

Returns: bool; true se for critico, false caso contrário.
*****/

bool ASN1Decoder::getCritical()
{
    string test;
    char ch;

    while(!file.eof())
    {
        ch = file.get();
        if(ch == 'B')
        {
            file >> test;
            if (test == "BOOLEAN")
                return true;
            else
                return false;
        }
    }
}

/*****/
bool ASN1Decoder::verifyCrl()
{
    char _ch;
    find("[",2);

```

```

    _ch = file.get();
    if(_ch == '0') //distributionPoint
    {
        find("[",2);
        _ch = file.get();
        if(_ch == '0') //GeneralNames
        {
            find("[",2);
            return true;
        }
    }
    return false;
}

void ASN1Decoder::getCRLFields(CRLDistributionPoints& __crl)
{
    string buffer,temp;
    int cont = 1;
    while(!file.eof())
    {
        file >> buffer;
        if (buffer == "IDENTIFIER")
        {
            file >> temp;
            if(temp == "emailAddress")
                __crl.getDN()->setEmailAddress(getString());
            else if(temp=="countryName")
                __crl.getDN()->setCountryName(getString());
            else if(temp=="stateOrProvinceName")
                __crl.getDN()->setStateOrProvinceName(getString());
            else if(temp=="localityName")
                __crl.getDN()->setLocalityName(getString());
            else if(temp=="organizationName")
                __crl.getDN()->setOrganization(getString());
            else if(temp=="organizationalUnitName")
            {
                if(cont ==1)
                {
                    __crl.getDN()-
>setOrganizationUnitName1(getString());
                    cont++;
                }
                else
                    __crl.getDN()-
>setOrganizationUnitName2(getString());
            }
            else if(temp=="commonName")
                __crl.getDN()->setCommonName(getString());

            if(countClose() >= 3) return; //Testa o fim dos campos
Subject
        }
    }
}

```

```

void ASN1Decoder::getSignatureValue()
{
    char _ch;
    string signatureValue;

    find("bits",1);
    skipWhiteSpaces();
    _ch = file.get();
    do
    {
        if(_ch == '\n')
            skipWhiteSpaces();
        signatureValue.push_back(_ch);
        _ch = file.get();
    }while(_ch != '}');
    certificate->setSignatureValue(signatureValue);
    //ShowMessage(signatureValue.c_str()); // only test
    return;
}

void ASN1Decoder::setType()
{
    int _index;
    vector<X509Extensions*>* _lExtensions;
    _lExtensions = certificate->getExtensionList();

    _index =
X509Functions::getIndexExtension(*certificate,"BasicConstraints",_lExte
nsions);
    if (_index != -1)
    {
        BasicConstraints *basic; // = new AuthorityInfoAccess();
        basic = static_cast<BasicConstraints*> (_lExtensions-
>at(_index));
        if(basic->getCa() == true)
        { // Eh CA, precisamos verificar se é Raiz ou Intermediário
            if
(X509Functions::compareIssuerSubject(*certificate,*certificate,0))
                certificate->setType(2); // Raiz
            else
                certificate->setType(1); // Intermediário
        }
        else
            certificate->setType(0); //UserCert
    }
    else
    {
        if
(X509Functions::compareIssuerSubject(*certificate,*certificate,0))
            certificate->setType(2); // Raiz
        else
            certificate->setType(0); // User ou Intermediário ???
    }
}
}

```

```
#pragma package (smart_init)
```

```
X509Cert
//-----
-----
#ifndef X509CertH
#define X509CertH
//-----
-----
#include <string>
#include <vector>
//-----
-----
#include "DateTime.h"
#include "X509Extensions.h"
#include "X509DN.h"
#include "CRL.h"
//-----
-----
class X509Cert
{
    protected:
        int version;
        int type;
        int ID;
        string auxFile;

        fstream* certificateStream;

        string serialNumber;
        string signature;
        string publicKeyAlgorithm;
        string publicKey;
        string rsaExpoente;
        string signatureValue;

        DateTime *notBefore;
        DateTime *notAfter;

        X509DN *issuer;
        X509DN *subject;

        CRL* crl;

        vector<X509Extensions*> extensionsList;

    public:
        X509Cert ();
```

```

~X509Cert();
inline void setVersion(int __version){ version = __version; };
inline void setSerialNumber(string __serialNumber){
serialNumber = __serialNumber; };
inline void setSignature(string __signature){ signature =
__signature; };
inline void setNotBefore(DateTime* __notBefore){ notBefore =
__notBefore; };
inline void setNotAfter(DateTime* __notAfter){ notAfter =
__notAfter; };
inline void setCertificateStream(fstream* __stream)
{certificateStream = __stream;};
inline void setID(int __ID) {ID = __ID; };
inline void setAuxFile(string __auxFile) {auxFile =
__auxFile;};

inline void setPublicKeyAlgorithm(string __publicKeyAlgorithm){
publicKeyAlgorithm = __publicKeyAlgorithm; };
inline void setPublicKey(string __publicKey){ publicKey =
__publicKey; };
inline void setRSAExpoente(string __rsaExpoente){ rsaExpoente =
__rsaExpoente; };
inline void setSignatureValue(string __signatureValue){
signatureValue = __signatureValue; };

inline int getVersion() const {return version;};
inline string getSerialNumber() const {return serialNumber;};

inline string getSignature() const { return signature; };
inline DateTime* getNotBefore() { return notBefore; };
inline DateTime* getNotAfter() { return notAfter; };

inline string getPublicKeyAlgorithm() const { return
publicKeyAlgorithm; };
inline string getPublicKey() const { return publicKey; };
inline string getRSAExpoente()const { return rsaExpoente; };
inline string getSignatureValue()const { return signatureValue;
};

inline fstream* getCertificateStream() {return
certificateStream;};
inline string getAuxFile() {return auxFile; };
inline int getID() {return ID; };

inline void setType(int __type) {type = __type;};
inline int getType() const {return type;};

vector<X509Extensions*> *getExtensionList(){return
&extensionsList;};
void addExtension(X509Extensions*
__extension){extensionsList.push_back(__extension);};
X509DN *getIssuer(){return issuer;};
X509DN *getSubject(){return subject;};

void setIssuer(X509DN* __issuer) { issuer = __issuer;};
void setSubject(X509DN* __subject) { subject = __subject;};

```

```

        // X509Extensions getExtensionAt(int __index){return
extensionsList->at(__index)};};
        //inline string getKeyUsage(){ return keyUsage; };

        void setCrl(CRL* __crl) {crl = __crl};};
        CRL* getCrl() {return crl};};

};
//-----
-----
#endif

X509DN
#ifndef X509DNH
#define X509DNH
//-----
-----
#include <string>
//-----
-----
using namespace std;
class X509DN
{
    protected:
        string countryName;
        string organizationUnitName[2];
        string organization;
        string commonName;
        string localityName;
        string stateOrProvinceName;
        string emailAddress;
        string mailBox;
        string telephoneNumber;

    public:
        X509DN();
        ~X509DN() {};
        inline void setCountryName(string __countryName){ countryName =
__countryName; };
        inline void setOrganizationUnitName1(string
__organizationUnitName){ organizationUnitName[0] =
__organizationUnitName; };
        inline void setOrganizationUnitName2(string
__organizationUnitName){ organizationUnitName[1] =
__organizationUnitName; };
        inline void setOrganization(string __organization){
organization = __organization; };
        inline void setCommonName(string __commonName){ commonName =
__commonName; };
        inline void setLocalityName(string __localityName){
localityName = __localityName; };
        inline void setStateOrProvinceName(string
__stateOrProvinceName){ stateOrProvinceName = __stateOrProvinceName; };
        inline void setEmailAddress(string __emailAddress){
emailAddress = __emailAddress; };

```



```

        inline void setTelephoneNumber(string __telephoneNumber){
telephoneNumber = __telephoneNumber; };
        inline void setMailBox(string __mailBox){ mailBox = __mailBox;
};

        inline string getCountryName() const { return countryName; };
        inline string getOrganizationUnitName1() const { return
organizationUnitName[0]; };
        inline string getOrganizationUnitName2() const { return
organizationUnitName[1]; };
        inline string getOrganization() const { return organization;
};

        inline string getCommonName() const { return commonName ; };
        inline string getLocalityName() const { return localityName ;
};

        inline string getStateOrProvinceName() const { return
stateOrProvinceName; };
        inline string getEmailAddress() const { return emailAddress;
};

        inline string getTelephoneNumber() const {return
telephoneNumber;};
        inline string getMailBox() const {return mailBox;};

};
//-----
-----
#endif

//-----
-----
#ifndef X509ExtensionsH
#define X509ExtensionsH
//-----
-----
#include <string>
//-----
-----
using namespace std;
//-----
-----
class X509Extensions
{
    protected:
        string oid;
        bool isCritical;
        int type;
        string name;

    public:
        X509Extensions() {};
        ~X509Extensions() {};
        inline void setIsCritical(bool __isCritical) {isCritical =
__isCritical;};
        inline void setOid(string __oid) {oid = __oid;};

        inline string getOid() const {return oid;};

```

```

inline bool getIsCritical() const {return isCritical;};

inline int getType() const {return type;};
inline void setType(int __type) {type = __type;};
inline string getName() {return name;};
inline void setName(string __name) {name = __name;};

};
//-----
#endif
//-----
#include "X509Functions.h"
//-----
int X509Functions::getIndexExtension(X509Cert& __cert, string
__extension, vector<X509Extensions*>* __lExtensions)
{
    for(unsigned int i=0;i < __lExtensions->size(); i++)
        if (__lExtensions->at(i)->getOid() == __extension)
            return i;
    return -1;
}

//-----
bool X509Functions::compareIssuerSubject(X509Cert& __certA, X509Cert&
__certB, int __type)
{
    X509DN *_issuerA = __certA.getIssuer();
    X509DN *_issuerB = __certB.getIssuer();
    X509DN *_subject = __certA.getSubject();

    if(__type == 0)
    {
        if (_issuerA->getCountryName() == _subject->getCountryName())
            if(_issuerA->getEmailAddress() == _subject-
>getEmailAddress())
                if(_issuerA->getLocalityName() == _subject-
>getLocalityName())
                    if(_issuerA->getOrganization() == _subject-
>getOrganization())
                        if(_issuerA->getMailBox() == _subject-
>getMailBox())
                            if(_issuerA->getTelephoneNumber() ==
__subject->getTelephoneNumber())
                                if(_issuerA->getOrganizationUnitName1()
== _subject->getOrganizationUnitName1())
                                    if(_issuerA-
>getOrganizationUnitName2() == _subject->getOrganizationUnitName2())
                                        return true;
                    }
                }
            }
        }
    }
    else
    {
        if (_issuerB->getCountryName() == _subject->getCountryName())

```

```

        if(_issuerB->getEmailAddress() == _subject-
>getEmailAddress())
            if(_issuerB->getLocalityName() == _subject-
>getLocalityName())
                if(_issuerB->getOrganization() == _subject-
>getOrganization())
                    if(_issuerB->getMailBox() == _subject-
>getMailBox())
                        if(_issuerB->getTelephoneNumber() ==
_subject->getTelephoneNumber())
                            if(_issuerB->getOrganizationUnitName1()
== _subject->getOrganizationUnitName1())
                                if(_issuerB-
>getOrganizationUnitName2() == _subject->getOrganizationUnitName2())
                                    return true;
                            }
                        }
                    }
                }
            }
        }
    }
}
//-----
X509Cert* X509Functions::findIssuerByCertificate(X509Cert* __cert) {
    X509Finder* _finder = new X509Finder();
    X509Cert* _cert;
    vector<X509Cert*> _vect = _finder->selectAllRootMidle();
    delete _finder;
    for (unsigned int i = 0;i<_vect.size();i++) {
        _cert = _vect.at(i);
        if (compareIssuerSubject(*_cert,*__cert,1)) {
            return _cert;
        }
    }
    return NULL;
}
//-----
string X509Functions::getInformationDN(X509DN* _dn) {
    string _sgeneric;
    if (_dn->getCommonName()!="") {
        _sgeneric = _dn->getCommonName().c_str();
    } else {
        if (_dn->getOrganizationUnitName1()!="") {
            _sgeneric = _dn->getOrganizationUnitName1().c_str();
        } else {
            if (_dn->getOrganizationUnitName2()!="") {
                _sgeneric = _dn->getOrganizationUnitName2().c_str();
            } else {
                _sgeneric = _dn->getOrganization().c_str();
            }
        }
    }
    return _sgeneric;
}
//-----
X509Cert* X509Functions::findCertificate(vector<X509Cert*>
__vectorCert,int __idCert) {

```

```

    for (vector<X509Cert*>::reverse_iterator i =
__vectorCert.rbegin();i!=__vectorCert.rend();i++) {
        X509Cert* _cert = *i;
        if (_cert->getID()==__idCert) {
            return _cert;
        }
    }
    return NULL;
}
//-----
-----

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

#pragma hdrstop

#include "X509Helper.h"

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

vector<X509Cert*> X509Helper::addObject(TQuery* __query) {
    X509Cert* _certificate;
    vector<X509Cert*> _vector;
    unsigned char* _data;
    while (!__query->Eof) {
        string _path = PathFile::getPath()+"\\temp.dat";
        fstream* _in = new fstream(_path.c_str(),ios::out|ios::in);
        std::auto_ptr<TMemoryStream> stream (new TMemoryStream);
        TBlobField* field =dynamic_cast<TBlobField *>(__query-
>FieldByName("certificado"));
        if(field) {
            field->SaveToStream(stream.get());
            _data = new unsigned char[stream->Size];
            stream->Position = 0;
            stream->Read(_data,stream->Size);
            _in->seekg(0);
            _in->write(_data, stream->Size);
            _in->seekg(0, ios::end);
            int _size = _in->tellg();
            _in->close();
        }
        ASN1Decoder decoder(_path.c_str()); //--> se quiser o
certificado todo decodificado
        if (decoder.decode()) {
            decoder.closeFile();
            _certificate = decoder.getCert();

            //bool _isdel = DeleteFile("temp.dat");
            _certificate->setID(__query->FieldByName("id")->AsInteger);
            X509DN* _issuer = new X509DN();

            _issuer->setCountryName(__query-
>FieldByName("countryNameIssuer")->AsString.c_str());
            _issuer->setOrganizationUnitName1(__query-
>FieldByName("organizationUnitNameIssuer1")->AsString.c_str());

```

```

        _issuer->setOrganizationUnitName2(__query-
>FieldByName("organizationUnitNameIssuer2")->AsString.c_str());
        _issuer->setOrganization(__query-
>FieldByName("organizationIssuer")->AsString.c_str());
        _issuer->setLocalityName(__query-
>FieldByName("localityNameIssuer")->AsString.c_str());
        _issuer->setStateOrProvinceName(__query-
>FieldByName("stateOrProvinceNameIssuer")->AsString.c_str());
        _issuer->setEmailAddress(__query-
>FieldByName("emailAddressIssuer")->AsString.c_str());
        _issuer->setCommonName(__query-
>FieldByName("commonNameIssuer")->AsString.c_str());
        _certificate->setIssuer(_issuer);
        _vector.push_back(_certificate);
    }
    __query->Next();
}
return _vector;
}
//-----
-----

//-----
-----
#include "X509Path.h"
//-----
-----

bool X509Path::build(X509Cert* __cert,X509Process* __processCert)
{
    int _idIssuer;
    clearPath();
    bool hasMoreCert = true;
    X509Cert* _cert;
    vector<X509Cert*> _vect = X509finder->selectByID(__cert->getID());
//--> cópia
    X509Cert* _cert2 = _vect.at(0);
    while(hasMoreCert) {
        if (_cert2->getType() != 2) { //Não é Raiz
            _cert = verifyIssuer(_cert2); // Verifica se o Issuer
está no BD
            if(_cert != NULL) {
                _cert2 = _cert;
            } else {
                //Tentar AuthorityInfoAccess ou LDAP. Senão, impossível
montar árvore
                _cert =
tryFindIssuerByAuthorityInfo(_cert2,__processCert);
                if (_cert!=NULL) {
                    X509controller->setCertificate(_cert);
                    X509controller->insert();
                    _cert2 = _cert;
                } else {
                    codeError = EMISSOR_NOT_FOUND;
                    path.push(_cert2);
                    return false;
                }
            }
        }
    }
}

```

```

        }
    } else {
        hasMoreCert = false;
        path.push(_cert2);
        return true;
    }
}
return false;
}
}

//-----
void X509Path::addUserCertStore(X509Cert& __userCert)
{
    // Verificar se __userCert já está armazenado no BD
    // Se não estiver, adicionar ele.

}
//-----
X509Cert* X509Path::verifyIssuer(X509Cert* __userCert)
{
    X509Cert* _cert =
X509Functions::findIssuerByCertificate(__userCert);
    X509Cert* _cert2;
    if (_cert!=NULL) {
        _cert2 = X509finder->selectByID(__userCert->getID()).at(0);
//--> copita de __userCert
        path.push(_cert2);
        return _cert;
    } else {
        return NULL;
    }
}
//-----
X509Cert* X509Path::tryFindIssuerByAuthorityInfo(X509Cert*
__cert,X509Process* __processCert)
{
    X509Cert* _cert = NULL;
    X509Cert* _cert2 = NULL;
    int _index;
    vector<X509Extensions*>* _lExtensions;
    vector<string> *_lIssuers;
    string _url;
    _lExtensions = __cert->getExtensionList();
    for (unsigned int i=0;i<_lExtensions->size();i++) {
        _url = _lExtensions->at(i)->getOid();
    }
    _index =
X509Functions::getIndexExtension(*__cert,"AuthorityInfoAccess",_lExtens
ions);
    if (_index!= -1) {
        AuthorityInfoAccess *info = static_cast<AuthorityInfoAccess*>
(_lExtensions->at(_index));

```

```

        _lIssuers = info->getIssuersList();
        for(unsigned int j=0; j < _lIssuers->size(); j++) {
            _url = _lIssuers->at(j); //--> verificar se esta conectado
            //_fstream* in = netTCC->getUrl(_url);
            netTCC->getUrl(_url);
            if(netTCC->getSavePath()!="") {
                _cert = X509Process::createCert(netTCC->getSavePath());
                _cert2 = X509finder->selectByID(__cert-
>getID()).at(0); //--> copita de __userCert
                path.push(_cert2);
                return _cert;
            }
        }
    }
    return _cert;
}
//-----
void X509Path::clearPath() {
    while(path.size()) {
        path.pop();
    }
}
//-----

//-----
//-----
#include "X509Process.h"
//-----
string X509Process::cgfFile =PathFile::getPath()+"\\dumppasnl.cfg";
//-----
void X509Process::process()
{
    parser->analisaCert(certFile, (char*)cgfFile.c_str(), output);
    fclose(output);
    decoder = new ASN1Decoder("auxfile.dat");
    //ASN1Decoder as1("auxfile.dat");
    decoder->decode();
    cert = decoder->getCert();
}
//-----
void X509Process::process2(char* __stream)
{
    FILE *output = fopen("auxfile.dat", "w+");
    certFile = "C:\\Documents and Settings\\Bruno Maluche Neto\\My
Documents\\UFSC\\TCC\\certificados\\teste.cer";

    parser->analisaCert2(certFile,__stream , (char*)cgfFile.c_str(),
output);

    fclose(output);
    decoder = new ASN1Decoder(__stream);
}

```

```

        //ASN1Decoder as1("auxfile.dat");
        decoder->decode();
        cert = decoder->getCert();
    }
//-----
-----
X509Cert* X509Process::createCert(string __pathFile) {
    X509Cert* _cert;
    FILE *outFile = fopen("auxfile.dat", "w+");
    ParserASN1 _parser;
    _parser.analisaCert((char*)__pathFile.c_str(),
(char*)cgfFile.c_str(), outFile);
    fclose(outFile);
    ASN1Decoder _decoder("auxfile.dat");
    _decoder.decode();
    _cert = _decoder.getCert();
    return _cert;
}
//-----
-----
X509Cert* X509Process::createCert(fstream* __inFile) {
    return NULL;
}
//-----
-----
CRL* X509Process::createCRL(string __pathFile) {
    CRL* _crl;
    FILE *outFile = fopen("auxfile.dat", "w+");
    ParserASN1 _parser;
    _parser.analisaCert((char*)__pathFile.c_str(),
(char*)cgfFile.c_str(), outFile);
    fclose(outFile);
    CRLDecoder _decoder("auxfile.dat");
    _decoder.decode();
    _crl = _decoder.getCrl();
    return _crl;
}
//-----
-----

//-----
-----
#include "X509Validation.h"
#include <wincrypt.h> // CryptoAPI definitions
//-----
-----

X509Validation::X509Validation(CERT_STACK __path)
{
    certPath = __path;
}

void X509Validation::validate()
{
    X509Cert* _cert = certPath.top();
    initialization(_cert);
}

```



```

certPath.pop();
pathCont++;
basicChecking(_cert);
preparationNext(_cert);

if(certPath.size() > 0)
{
    _cert = certPath.top();
    certPath.pop();
    while(certPath.size() > 0)
    {
        basicChecking(_cert);
        preparationNext(_cert);
        _cert = certPath.top();
        certPath.pop();

    }
    basicChecking(_cert);
    preparationNext(_cert);
    wrapUp();
}
}

/*****
*****
Nome: initialization

Ação: Fase de inicialização, onde todas as entradas e valores
necessários para validação serão
        processados.

Parametros: sem parametros.

Returns: void.

*****/
void X509Validation::initialization(X509Cert* __cert)
{
    workerIssuer = *__cert;
    pathCont = 0;
    pathLenght = 0;
}

void X509Validation::basicChecking(X509Cert* __cert)
{
    checkCertificateValidity(__cert->getNotBefore(), __cert->
getNotAfter());
    if(__cert->getType() != 2) // Não é raiz
    {
        checkCertificateRevocation(__cert);
        ckeckChain(__cert);
        if(__cert->getType() == 1)
        {
            checkBasicConstraints(__cert);

```

```

        checkKeyUsage(__cert);
    }
}

}

void X509Validation::preparationNext(X509Cert* __cert)
{
    int _index;
    workerIssuer = *__cert;
    if(pathLenght > pathCont)
        lErros.push(new X509ValidationError(" tamanho máximo de
certificação foi ultrapassado.",certPath.size()));

    vector<X509Extensions*>* _lExtensions;
    _lExtensions = __cert->getExtensionList();

    _index =
X509Functions::getIndexExtension(*__cert,"PolicyConstraints",_lExtensio
ns);
    if (_index != -1)
    {
        PolicyConstraints* _pol = static_cast<PolicyConstraints*>
(_lExtensions->at(_index));
        if(_pol->getRequireExplicitPolicy() != -1)
            workerRequiredExplicit = _pol->getRequireExplicitPolicy();
    }
    pathCont++;
}

void X509Validation::wrapUp()
{}
//-----
void X509Validation::ckeckChain(X509Cert* __cert)
{
    if(!X509Functions::compareIssuerSubject(workerIssuer,*__cert,1))
        lErros.push(new X509ValidationError(" corrente Emissor/Assunto
Inválida.",certPath.size()));
}

void X509Validation::checkCertificateRevocation(X509Cert* __cert)
{
    vector<Revocation*> *_lRevocation;
    _lRevocation = __cert->getCrl()->getRevocationList();

    if (__cert->getCrl() == NULL)
    {
        lErros.push(new X509ValidationError(" a lista de Revogados não
foi encontrada.",certPath.size()));
    }
    else
    {
        for(int i=0;i<_lRevocation->size();i++)
        {

```

```

        Revocation* _revocation;
        _revocation = _lRevocation->at(i);
        string _serial = _revocation->getSerial();
        if(_serial == __cert->getSerialNumber())
        {
            lErros.push(new X509ValidationError(" está
revogado.", certPath.size()));
        }
    }
}

void X509Validation::checkBasicConstraints(X509Cert* __cert)
{
    int _index;
    vector<X509Extensions*>* _lExtensions;
    _lExtensions = __cert->getExtensionList();

    _index =
X509Functions::getIndexExtension(*__cert, "BasicConstraints", _lExtension
s);
    if (_index != -1)
    {
        BasicConstraints *basic; // = new AuthorityInfoAccess();
        basic = static_cast<BasicConstraints*> (_lExtensions-
>at(_index));
        if(basic->getCa() == false)
        {
            lErros.push(new X509ValidationError(" contém
BasicConstraints setado com falso.", certPath.size()));
        }
        else
        {
            if(basic->getPathLen() != -1)
            {
                pathLength = basic->getPathLen();
                pathCont = 0;
            }
        }
    }
    else
    {
        lErros.push(new X509ValidationError(" não contém a extensão
BasicConstraints.", certPath.size()));
    }
}

void X509Validation::checkKeyUsage(X509Cert* __cert)
{
    int _index;
    string _string;
    vector<string*>* _strings;
    vector<X509Extensions*>* _lExtensions;

```

```

    _lExtensions = __cert->getExtensionList();
    bool _keyCertSign = false;
    bool _crlSign = false;

    _index =
X509Functions::getIndexExtension(*__cert,"KeyUsage",_lExtensions);
    if (_index != -1)
    {
        KeyUsage* _key = static_cast<KeyUsage*> (_lExtensions-
>at(_index));
        _strings = _key->getKeyUsageList();
        for (unsigned j = 0;j<_strings->size();j++)
        {
            _string = _strings->at(j);
            if(_string == "Key CertSign")
                _keyCertSign = true;
            if(_string == "CRL Sign")
                _crlSign = true;
        }
        if(!_keyCertSign || !_crlSign)
            lErros.push(new X509ValidationError(" bits de assinatura
            nao setados na extensao KeyUsage.",certPath.size()));
    }
}

bool X509Validation::checkCertificateValidity(DateTime*
__notBefore,DateTime* __notAfter)
{
    TDateTime time;
    time = time.CurrentDateTime();

    if(!checkNotBefore(__notBefore, time))
    {
        lErros.push(new X509ValidationError(" ainda não é
válido.",certPath.size()));
        return false;
    }
    if(!checkNotAfter(__notAfter , time))
    {
        lErros.push(new X509ValidationError(" está
expirado.",certPath.size()));
        return false;
    }
    return true;
}

bool X509Validation::checkNotBefore(DateTime* __notBefore, TDateTime
__time)
{
    unsigned short cYear, cMonth, cDay, cSec, cHour, cMin, cMil; //c =
Current
    DecodeDate(__time, cYear, cMonth, cDay);
    DecodeTime(__time, cHour, cMin, cSec, cMil);
}

```

```

// Verifica se data notBefore está Ok
if(cYear < __notBefore->getYear() )
{
    return false;
}
else
{
    if(__notBefore->getYear() == cYear)
    {
        if(cMonth < __notBefore->getMonth() )
        {
            return false;
        }
        else
        {
            if(__notBefore->getMonth() == cMonth)
            {
                if(cDay < __notBefore->getDay() )
                {
                    return false;
                }
                else
                {
                    if (__notBefore->getDay() == cDay)    //year,
month e day = current
                    {
                        if(cHour < __notBefore->getHour())
                        {
                            return false;
                        }
                        else
                        {
                            if(__notBefore->getHour() == cHour)
                            {
                                if(cMin < __notBefore->getMin() )
                                {
                                    return false;
                                }
                                else
                                {
                                    if(__notBefore->getMin() ==
cMin)
                                    {
                                        if(cSec < __notBefore-
>getSec())
                                        {
                                            return false;
                                        }
                                        else
                                        {
                                            return true; // Sex >=
cSec
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        return true; // Min > cMin
    }
}
else
{
    return true; //Hour > cHour
}
}
else
{
    return true; //Day > cDay
}
}
else
{
    return true; // Month > cMonth
}
}
else
{
    return true; // Year > cYear
}
}
}

```

```

bool X509Validation::checkNotAfter(DateTime* __notAfter, TDateTime
__time)
{
    unsigned short cYear, cMonth, cDay, cSec, cHour, cMin, cMil; //c =
Current

    DecodeDate(__time, cYear, cMonth, cDay);
    DecodeTime(__time, cHour, cMin, cSec, cMil);

    // Verifica se data notAfter está Ok
    if(cYear > __notAfter->getYear() )
    {
        return false;
    }
    else
    {
        if(__notAfter->getYear() == cYear)
        {
            if(cMonth > __notAfter->getMonth())
            {
                return false;
            }
            else
            {
                if(__notAfter->getMonth() == cMonth)
                {
                    if(cMonth > __notAfter->getDay())
                    {

```

```

        return false;
    }
    else
    {
        if ( __notAfter->getDay() == cDay)    //year,
month e day = current
        {
            if(cHour > __notAfter->getHour())
            {
                return false;
            }
            else
            {
                if(__notAfter->getHour() == cHour)
                {
                    if(cMin > __notAfter->getMin())
                    {
                        return false;
                    }
                    else
                    {
                        if(__notAfter->getMin() ==
cMin)
                        {
                            if(cSec > __notAfter-
>getSec() )
                            {
                                return false;
                            }
                            else
                            {
                                return true; // Sec <=
cSec
                            }
                        }
                    }
                }
            }
        }
    }
    else
    {
        return true; //Day < cDay
    }
}
}
else
{
    return true; // Month < cMonth
}
}

```

```

        }
    }
    else
    {
        return true; // Year < cYear
    }
}

#pragma package(smart_init)

//-----
#include "CRLDecoder.h"
//-----
CRLDecoder::CRLDecoder(char* __filename)
{
    stream.open(__filename);
    crl = new CRL();
}
//-----
void CRLDecoder::decode()
{
    getVersion();
    getSignatureAlgorithm();
    getIssuerFields();
    getDates();
    getRevocationList(); //--> fix it
    if(crl->getVersion() >=2)
        getExtensions();
}
//-----
void CRLDecoder::getVersion()
{
    int version;
    string _dummy,_temp;

    stream >> _dummy >> _dummy >> _dummy >> _dummy >> _temp;
    if(_temp == "INTEGER")
        stream >> version;
    else
        version = 0;

    crl->setVersion(version++);
}

void CRLDecoder::getSignatureAlgorithm()
{
    string _temp;
    find("IDENTIFIER",1);
    stream >> _temp;
}

```



```

        crl->setSignatureValue(_temp);
    }

/*****
*****
Nome: getIssuerFields

Ação: Percorre o certificado em busca dos campos relacionados ao
'Issuer'.

Parametros: sem parametros.

Returns: void.
*****/
void CRLDecoder::getIssuerFields()
{
    string buffer,temp;
    X509DN *issuer = crl->getIssuer();
    int cont = 1;
    while(!stream.eof())
    {
        stream >> buffer;
        if (buffer == "IDENTIFIER")
        {
            stream >> temp;
            if(temp == "emailAddress")
                issuer->setEmailAddress(getString());
            else if(temp=="countryName")
                issuer->setCountryName(getString());
            else if(temp=="stateOrProvinceName")
                issuer->setStateOrProvinceName(getString());
            else if(temp=="localityName")
                issuer->setLocalityName(getString());
            else if(temp=="organizationName")
                issuer->setOrganization(getString());
            else if(temp=="organizationalUnitName")
            {
                if(cont ==1)
                {
                    issuer->setOrganizationUnitName1(getString());
                    cont++;
                }
                else
                    issuer->setOrganizationUnitName2(getString());
            }
            else if(temp=="commonName")
                issuer->setCommonName(getString());
            if(countClose() >= 3) return; //Testa o fim dos campos
        }
    }
}

void CRLDecoder::getDates()
{
    string _temp;

```

```

    DateTime _effectiveDate, _nextUpdate;

    _temp = getString();
    _effectiveDate.decodeDate(_temp);
    crl->setEffectiveDate(_effectiveDate);

    _temp = getString();
    _nextUpdate.decodeDate(_temp);
    crl->setNextUpdate(_nextUpdate);
}

void CRLDecoder::getRevocationList()
{
    string _temp;
    char _ch;
    bool hasReason=false;
    stream >> _temp;
    if(_temp == "[0]")
        return; // n tem revogados.

    Revocation *revocation;
    while(!stream.eof())
    {
        revocation = new Revocation();
        _temp.clear();
        find("INTEGER",1);
        _ch = findFirstChar();
        do
        {
            _temp.push_back(_ch);
            _ch = stream.get();
        }while(_ch != '\n');
        revocation->setSerial(_temp);
        _temp.clear();

        _temp = getString();
        DateTime* date = new DateTime();
        date->decodeDate(_temp);
        revocation->setDate(date);

        stream >> _temp;
        if(_temp == "SEQUENCE")
        {
            hasReason = true;
            find("ENUMERATED",1);
            stream >> _temp;
            revocation->setReason(_temp);
        }
        else
            stream.unget();

        crl->addRevocation(revocation);
        if(hasReason)
        {
            if(countClose() > 4)
                return;
        }
    }
}

```

```

        else
            if(countClose() == 3)
                return;
    }
}

void CRLDecoder::getExtensions()
{
}

/*****
*****
Nome: find

Ação: Localiza o string ou o caracter desejado no certificado.

Parametros: string, int; É o string que será procurado no
certificado.__type define se a procura é por um string(1) ou
char (2).

Returns: void.
*****/

void CRLDecoder::find(string __target, int __type)
{
    string _temp;
    switch (__type)
    {
        case 1:
            stream >> _temp;
            while (_temp != __target)
                stream >> _temp;
            break;
        case 2:
            _temp = stream.get();
            while (_temp != __target)
                _temp = stream.get();
            break;
    }
}

/*****
*****
Nome: countClose

Ação: Conta a quantidade de '}', para verificar se os campos do Issuer
ou Subject chegou ao fim.

Parametros: sem parametros.

Returns: int; Indicando o numero de '}' consecutivos encontrado.

Comentário: Um retorno 3 indica que os campos de Issuer ou Subject
terminou.

```

```

*****
*****/
int CRLDecoder::countClose()
{
    string text;
    int cont =0;          // Quantidade de '}'
    while(text != "") // Encontrar o 1º '}'
    {
        stream >> text;
        if (text == "")
            cont++;
    }
    stream >> text;
    while(text == "")
    {
        cont++;
        stream >> text;
        // if(text == "")
        //     cont++;
    }
    return cont;
}

```

```

/*****
*****

```

Nome: getString

Ação: Percorre o certificado em busca do próximo string, verificando a possibilidade de um ' no meio do string.

Parametros: sem parametros.

Returns: string; retorna o String solicitado

```

*****
*****/

```

```

string CRLDecoder::getString()
{
    char ch;
    ch= '9'; //dummy
    string texto;

    while(ch != '\\') // Procura o início do String, representado por '
        ch = stream.get();

    ch = stream.get();
    while(ch != '\\') //Procura o final do String.
    {
        texto.push_back(ch);
        ch = stream.get();
        if (ch == '\\') // Verifica se o caracter ' pertence ao
string ou eh delimitador final
        {
            ch = stream.get();
            if (ch == '\\n')
            {

```

```

        stream.unget();
        ch = '\\';
    }
    else
        texto.push_back('\\');
}

}
return texto;
}

/*****
*****
Nome: findFirstChar

Ação: Encontra o primeiro char.

Parametros: void;

Returns: char; o caracter encontrado.
*****/

char CRLDecoder::findFirstChar()
{
    char _ch;
    _ch = stream.get();
    while ((_ch == '\\n') || (_ch == ' '))
        _ch = stream.get();
    return _ch;
}

#pragma package(smart_init)

//-----
//-----
#include "KeyUsage.h"
//-----
//-----

/*****
*****
Nome: decodeBits

Ação: Análisa quais dos bits estão setados (=1), adicionando a lista de
keyUsage, os valores
    apropriados.

Parametros: sem parametros.

Returns: void.
*****/
void KeyUsage::decodeBits()

```

```

{
    string temp;
    temp = bits;
    int _lastPos = bits.size()-1;

    int _pos = _lastPos;
    for(int i=0;i<=_lastPos;i++)
    {
        if (bits[_pos] == '1')
        {
            switch(i)
            {
                case 0:
                    keyUsageList.push_back("Digital Signature");
                    break;
                case 1:
                    keyUsageList.push_back("Non Repudiation");
                    break;
                case 2:
                    keyUsageList.push_back("Key Encipherment");
                    break;
                case 3:
                    keyUsageList.push_back("Data Encipherment");
                    break;
                case 4:
                    keyUsageList.push_back("key Agreement");
                    break;
                case 5:
                    keyUsageList.push_back("Key CertSign");
                    break;
                case 6:
                    keyUsageList.push_back("CRL Sign");
                    break;
                case 7:
                    keyUsageList.push_back("Encipher Only");
                    break;
                case 8:
                    keyUsageList.push_back("Decipher Only");
                    break;
            }
        }
        _pos--;
    }
}
//-----
-----
#pragma package(smart_init)

//-----
-----
#include "KeyUsageExt.h"
//-----
-----

```

```

/*****
*****
Nome: processKeyExt

Ação: Percorre o certificado em busca das Extended Key Usages,
adicionando a lista.

Parametros: ifstream; É o stream que contém o certificado.

Returns: void.
*****
*****/
void KeyUsageExt::processKeyExt(fstream& _file)
{
    string _temp, _key;
    char _ch;

    while (_key != "")
    {
        do{
            _file >> _key;
        }while(_key != "IDENTIFIER");

        _file >> _key;

        if(_key[0] == '\\')
        {
            do
            {
                _key.push_back(_ch = _file.get());
            }while(_ch != '\\');
        }

        if(_key=="serverAuth")
            addExtKeyUsage("serverAuth");
        else if(_key=="clientAuth")
            addExtKeyUsage("clientAuth");
        else if(_key=="codeSigning")
            addExtKeyUsage("codeSigning");
        else if(_key=="emailProtection")
            addExtKeyUsage("emailProtection");
        else if(_key=="timeStamping")
            addExtKeyUsage("timeStamping");
        else if(_key=="OCSPSigning")
            addExtKeyUsage("OCSPSigning");
        else if(_key=="windowsVer")
            addExtKeyUsage("windowsVer");
        else
            addExtKeyUsage(_key);

        if(_key[0] != '\\')
            do{
                _ch = _file.get();
            }while(_ch != ' ');

        _file >> _key;
    }
}

```

```

    }
    _file.unget();

}
//-----
-----

#pragma package(smart_init)
//-----
-----
#include "X509Functions.h"
//-----
-----
int X509Functions::getIndexExtension(X509Cert& __cert, string
__extension, vector<X509Extensions*>* __lExtensions)
{
    for(unsigned int i=0;i < __lExtensions->size(); i++)
        if (__lExtensions->at(i)->getOid() == __extension)
            return i;
    return -1;
}

//-----
-----
bool X509Functions::compareIssuerSubject(X509Cert& __certA, X509Cert&
__certB,int __type)
{
    X509DN *_issuerA = __certA.getIssuer();
    X509DN *_issuerB = __certB.getIssuer();
    X509DN *_subject = __certA.getSubject();

    if(__type == 0)
    {
        if (_issuerA->getCountryName() == _subject->getCountryName())
            if(_issuerA->getEmailAddress() == _subject-
>getEmailAddress())
                if(_issuerA->getLocalityName() == _subject-
>getLocalityName())
                    if(_issuerA->getOrganization() == _subject-
>getOrganization())
                        if(_issuerA->getMailBox() == _subject-
>getMailBox())
                            if(_issuerA->getTelephoneNumber() ==
__subject->getTelephoneNumber())
                                if(_issuerA->getOrganizationUnitName1()
== _subject->getOrganizationUnitName1())
                                    if(_issuerA-
>getOrganizationUnitName2() == _subject->getOrganizationUnitName2())
                                        return true;
                    }
            }
        else
        {
            if (_issuerB->getCountryName() == _subject->getCountryName())

```



```

        if(_issuerB->getEmailAddress() == _subject-
>getEmailAddress())
            if(_issuerB->getLocalityName() == _subject-
>getLocalityName())
                if(_issuerB->getOrganization() == _subject-
>getOrganization())
                    if(_issuerB->getMailBox() == _subject-
>getMailBox())
                        if(_issuerB->getTelephoneNumber() ==
_subject->getTelephoneNumber())
                            if(_issuerB->getOrganizationUnitName1()
== _subject->getOrganizationUnitName1())
                                if(_issuerB-
>getOrganizationUnitName2() == _subject->getOrganizationUnitName2())
                                    return true;
                            }
                        }
                    }
                }
            }
        }
    }
}
//-----
X509Cert* X509Functions::findIssuerByCertificate(X509Cert* __cert) {
    X509Finder* _finder = new X509Finder();
    X509Cert* _cert;
    vector<X509Cert*> _vect = _finder->selectAllRootMidle();
    delete _finder;
    for (unsigned int i = 0;i<_vect.size();i++) {
        _cert = _vect.at(i);
        if (compareIssuerSubject(*_cert,*__cert,1)) {
            return _cert;
        }
    }
    return NULL;
}
//-----
string X509Functions::getInformationDN(X509DN* _dn) {
    string _sgeneric;
    if (_dn->getCommonName()!="") {
        _sgeneric = _dn->getCommonName().c_str();
    } else {
        if (_dn->getOrganizationUnitName1()!="") {
            _sgeneric = _dn->getOrganizationUnitName1().c_str();
        } else {
            if (_dn->getOrganizationUnitName2()!="") {
                _sgeneric = _dn->getOrganizationUnitName2().c_str();
            } else {
                _sgeneric = _dn->getOrganization().c_str();
            }
        }
    }
    return _sgeneric;
}
//-----
X509Cert* X509Functions::findCertificate(vector<X509Cert*>
__vectorCert,int __idCert) {

```

```

    for (vector<X509Cert*>::reverse_iterator i =
__vectorCert.rbegin();i!=__vectorCert.rend();i++) {
        X509Cert* _cert = *i;
        if (_cert->getID()==__idCert) {
            return _cert;
        }
    }
    return NULL;
}
//-----
-----

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

#include <vcl.h>
#pragma hdrstop

#include "Main.h"
#include "InterfaceManager.h"
//-----
-----

#pragma package(smart_init)
#pragma resource "*.dfm"
TFormMain *FormMain;
//-----
-----

__fastcall TFormMain::TFormMain(TComponent* Owner)
: TForm(Owner) {
    finderCert = new X509Finder();
    state = stNotLogin;
    InhibitAnyPolicy* __test;
}
//-----
-----

void TFormMain::listCertificates(TListView*
__listView,vector<X509Cert*> __certs) {
    TListItem* _listItem;
    X509Cert* _cert;
    for (unsigned int i = 0 ;i <__certs.size();i++) {
        _cert = __certs.at(i);
        _listItem = __listView->Items->Add();
        if (_cert->getSubject()->getCommonName()!="") {
            _listItem->Caption =_cert->getSubject()->getCommonName().c_str();
        } else {
            if (_cert->getSubject()->getOrganizationUnitName1()!="") {
                _listItem->Caption =_cert->getSubject()-
>getOrganizationUnitName1().c_str();
            } else {
                if (_cert->getSubject()->getOrganizationUnitName2()!="") {
                    _listItem->Caption =_cert->getSubject()-
>getOrganizationUnitName2().c_str();
                } else {
                    _listItem->Caption =_cert->getSubject()-
>getOrganization().c_str();
                }
            }
        }
    }
}

```

```

    }
}
if (_cert->getIssuer()->getCommonName()!="") {
    _listItem->SubItems->Add(_cert->getIssuer()-
>getCommonName().c_str());
} else {
    if (_cert->getIssuer()->getOrganizationUnitName1()!="") {
        _listItem->SubItems->Add(_cert->getIssuer()-
>getOrganizationUnitName1().c_str());
    } else {
        if (_cert->getIssuer()->getOrganizationUnitName2()!="") {
            _listItem->SubItems->Add(_cert->getIssuer()-
>getOrganizationUnitName2().c_str());
        } else {
            _listItem->SubItems->Add(_cert->getIssuer()-
>getOrganization().c_str());
        }
    }
}
_listItem->SubItems->Add(_cert->getNotBefore()-
>toString().c_str());
_listItem->SubItems->Add(_cert->getNotAfter()->toString().c_str());
_listItem->SubItems->Add(IntToStr(_cert->getID()));

}
}
//-----
-----
void TFormMain::clearAllListViews() {
    ltvRootCA->Items->Clear();
    ltvMidleCA->Items->Clear();
    ltvOtherPeople->Items->Clear();
    ltvPrivate->Items->Clear();
    ltvTrustCA->Clear();
}
//-----
-----
void __fastcall TFormMain::NewLogin1Click(TObject *Sender) {
    interfaceManager->showNewUser();
}
//-----
-----

void __fastcall TFormMain::InserirCertificados1Click(TObject *Sender)
{
    vector<string> _files;
    if(odlInsertCER->Execute()) {
        for (int i = 0;i<odlInsertCER->Files->Count;i++) {
            string _file =odlInsertCER->Files->Strings[i].c_str();
            _files.push_back(_file);
        }
        if (interfaceManager->insertCertificates(_files)) {
            clearAllListViews();
            buildTreeCertificates();
        }
    }
}
}
}

```

```

//-----
-----
void TFormMain::buildSharedTreeCertificates() {
    listCertificates(ltvRootCA,interfaceManager->getRootsCA());
    listCertificates(ltvMidleCA,interfaceManager->getMidleCA());
}
//-----
-----
void TFormMain::buildPersonalTreeCertificates() {
    ltvOtherPeople->Items->Clear();
    ltvPrivate->Items->Clear();
    ltvTrustCA->Clear();
    listCertificates(ltvOtherPeople,interfaceManager->getOtherCer());
    listCertificates(ltvPrivate,interfaceManager->getPrivateCer());
    listCertificates(ltvTrustCA,interfaceManager->getTrustCA());
}
//-----
-----
void TFormMain::buildTreeCertificates() {
    buildSharedTreeCertificates();
    buildPersonalTreeCertificates();
}
//-----
-----
void __fastcall TFormMain::FormShow(TObject *Sender) {
    clearAllListViews();
    buildTreeCertificates();
    pgcCertChange(Sender);
}
//-----
-----

void __fastcall TFormMain::Sair1Click(TObject *Sender) {
    Close();
}
//-----
-----

void __fastcall TFormMain::ltvPrivateDblClick(TObject *Sender) {
    TListView* _listView =(TListView*)Sender;
    if(_listView->ItemIndex > -1) {
        int _idCert = StrToInt(_listView->Items->Item[_listView-
>ItemIndex]->SubItems->Strings[3]);
        X509Cert* _cert =
X509Functions::findCertificate(currentVector,_idCert);
        if (_cert!=NULL) {
            interfaceManager->showX509Cert(_cert);
        } else {
            //--> isto não deveria acontecer
        }
    }
}
//-----
-----
void TFormMain::loadInformationCert(int __idCert) {

```

```

}
//-----
-----
void __fastcall TFormMain::ToolButton8Click(TObject *Sender) {

    if (state == stNotLogin) {
        TCCUtils::getInstance()->showMessage("Você deve estar logado para
adicionar um certificado");
    } else {
        if (currentListView->ItemIndex>=0) {
            int _idCert = StrToInt(currentListView->Items-
>Item[currentListView->ItemIndex]->SubItems->Strings[3]);
            X509Cert* _cert= NULL;
            switch (pgcCert->TabIndex) {
                case tbOther:
                case tbMidle:
                case tbRoot:
                    _cert =
X509Functions::findCertificate(currentVector,_idCert);
                    break;
                case tbPrivate:
                    TCCUtils::getInstance()->showMessage("Este Certificado já
esta na sua lista de certificados Pessoais");
                    break;
                case tbTrust:
                    TCCUtils::getInstance()->showMessage("Este Certificado já
esta na sua lista de certificados Confiáveis");
                    break;
            }
            if (_cert!=NULL) {
                if (interfaceManager->insertCertificateToCurrentUser(_cert))
{
                    TCCUtils::getInstance()->showMessage("Certificado
adicionado com sucesso");
                    buildPersonalTreeCertificates();
                } else {
                    TCCUtils::getInstance()->showMessage(interfaceManager-
>getMessageError());
                }
            } else {
                }
            } else {
                TCCUtils::getInstance()->showMessage("Selecione o certificado
que você deseja adicionar");
            }
        }
    }
}
//-----
-----
void __fastcall TFormMain::ToolButton1Click(TObject *Sender) {
    if (interfaceManager->showLogin()) {
        buildPersonalTreeCertificates();
        pgcCertChange(Sender);
    }
}

```

```

}
//-----
-----
/*
X509Cert* TFormMain::findCertificate(vectorcer __vectorCert,int
__idCert) {
    for (vectorcer::reverse_iterator i =
__vectorCert.rbegin();i!=__vectorCert.rend();i++) {
        X509Cert* _cert = *i;
        if (_cert->getID()==__idCert) {
            return _cert;
        }
    }
    return NULL;
} */
//-----
-----

void __fastcall TFormMain::pgcCertChange(TObject *Sender)
{
    switch (pgcCert->TabIndex) {
        case tbPrivate: {
            currentListView = ltvPrivate;
            currentVector = interfaceManager->getPrivateCer();
            break;
        }
        case tbOther: {
            currentListView = ltvOtherPeople;
            currentVector = interfaceManager->getOtherCer();
            break;
        }
        case tbMidle: {
            currentListView = ltvMidleCA;
            currentVector = interfaceManager->getMidleCA();
            break;
        }
        case tbRoot: {
            currentListView = ltvRootCA;
            currentVector = interfaceManager->getRootsCA();
            break;
        }
        case tbTrust: {
            currentListView = ltvTrustCA;
            currentVector = interfaceManager->getTrustCA();
            break;
        }
    }
}

}
//-----
-----

void __fastcall TFormMain::ToolButton5Click(TObject *Sender) {
    ltvPrivateDb1Click(currentListView);
}
//-----
-----

```

```
void __fastcall TFormMain::Sobre1Click(TObject *Sender)
{
    interfaceManager->showAbout();
}
//-----
-----
```