



Artigo

Saiba como ter acesso a todos os conteúdos da DevMedia. ▶

Notificações :)

Certificados Digitais em .NET – Parte 1

Este tutorial vai demonstrar o desenvolvimento de uma Autoridade Certificadora simples, uma Autoridade de Registro, módulos de Assinatura, Validação e Criptografia, usando Certificados Digitais, colocando em prática os conceitos aprendidos.

Marcar como lido



Anotar



Desde 2001, existe legislação específica na constituição brasileira que permite a autenticação de comunicações e documentos eletrônicos por meio do uso de Certificados Digitais gerados sob a Autoridade Certificadora Brasileira. Do mesmo modo, o conceito pode ser aplicado para fortalecer as comunicações de qualquer aplicação a custo zero ou estar em acordo com a legislação vigente.

Esta série de artigos trata dos conceitos de Certificação Digital e PKI, e é um tutorial para o desenvolvimento de uma Autoridade Certificadora simples, uma Autoridade de Registro, módulos de Assinatura, Validação e Criptografia, usando Certificados Digitais, colocando em prática os conceitos aprendidos.

Dentro das empresas financeiras e demais setores bancários a Certificação



7



documentos eletrônicos e adotaram, junto à OAB, o uso de Certificados Digitais para autenticação de documentos e validação dos advogados e magistrados.

Alguns anos antes, a Receita Federal passou a exigir assinatura digital de empresas para envio da declaração de Imposto de Renda, mas isso não popularizou o uso desse mecanismo porque permitiu que apenas um representante da empresa, ou um contador, pudesse enviar as declarações para várias empresas.

Notificações :)

Saiba mais: [Certificados Digitais em .NET](#)

Estas foram as ações mais conhecidas que temos no uso de certificados digitais, mas o **custo de aquisição de um certificado digital** e leitor de cartão inteligente (Smartcard) ainda é a principal causa apontada para a impopularidade da tecnologia.

Alguns setores industriais também se utilizaram deste tipo de tecnologia e passaram a incluir, por exemplo, em alguns modelos de carros, certificados digitais incrustados nas chaves dos veículos e um módulo de validação no computador de bordo dos mesmos, permitindo autenticar com mais segurança se o mecanismo não estava sendo manipulado por chaves adulteradas ou outros processos.

Este tipo de verificação da chave, no entanto, apresentou problemas e até mesmo vulnerabilidades, fazendo com que a indústria automotiva fosse levada a procurar alternativas.





sua estrutura e como isso pode nos ajudar em alguma coisa? Nas próximas seções tentaremos responder a essa pergunta.

Curso de Certificado Digital: O que é Certificado Digital? - Aula Demons...



Notificações :)

Saiba mais: [O que é Certificado Digital?](#)

PKI – Public Key Infrastructure



7



determinado documento como válido, original e o registra em seus livros de notas.

A PKI é um conjunto de tecnologias que permite a um sistema autenticar e validar qualquer informação eletrônica.

A PKI moderna conta com uma estrutura que contém, ao menos:

- Uma Autoridade Certificadora;
- Uma Autoridade de Registro;
- Módulos de Assinatura e Verificação.

As tecnologias mais comumente empregadas para manter esta estrutura são:

- Certificados Digitais de acordo com a RFC-2459 (**BOX 1**) no padrão X.509;
- Algoritmos de criptografia diversos como MD5, SHA-1, RSA, etc.;
- Envelopes Digitais nos padrões RSA PKCS#12, PKCS#8, PKCS#7;
- OCSP (**BOX 2**) ou LCR (**BOX 3**) para verificação da validade dos certificados.

A Autoridade Certificadora é a responsável pela geração do Certificado Raiz.

BOX 1:

- **IETF** (Internet Engineering Task-Force) é o órgão internacional responsável pela padronização, controle e criação de uma série de protocolos, formatos e métodos utilizados amplamente na Internet. A RFC 2459 trata da estrutura, uso e evolução do formato X.509, proveniente da estrutura de dados X.500 usada em comunicações entre sistemas.

Este formato permite a criação de um arquivo estruturado em áreas que garantem o





informações de validade do mesmo, informações de como ele deve ser verificado (se por uma lista de certificados revogados disponível no site da Autoridade Certificadora que o gerou, ou por meio da consulta a um protocolo de verificação online – OCSP); quais as políticas que ele atende, ou seja, as regras nas quais aquele certificado pode ser utilizado como Assinatura de mensagens de E-mail, Criptografia de arquivos, etc.

Notificações :)

BOX 2:

- Online Certificate Status Protocol é um protocolo de comunicação via Internet que pode estar disponível pela **Autoridade Certificadora que assinou o Certificado Digital** (normalmente a Autoridade Certificadora Intermediária), no formato de um serviço que permite ao módulo de assinatura ou sistemas que vão utilizar o Certificado Digital verificarem a validade de um Certificado. Ele é um substituto à LCR – Lista de Certificados Revogados.

Quando um Certificado é assinado (chamamos a isto de criação do Certificado), ele ganha uma validade. Normalmente esta validade é de um ano, com renovações de mais um ano até um máximo de três anos de validade. Ao final deste período é necessário gerar um novo certificado com a necessidade de validação de documentos físicos. Veja o quadro de processo de certificação para maiores detalhes.

Este protocolo é a troca de mensagens em um formato de XML assinado entre o cliente que requisita a validação de um certificado e o servidor da Autoridade Certificadora emitente do mesmo. Este serviço é muito mais eficaz e seguro do que o uso de um arquivo contendo a lista de certificados revogados.





Esta lista é gerada em formato específico e fica disponível (24 horas por dia) no site do emitente do certificado, podendo ser baixada por qualquer um na internet. Entre outras informações, este arquivo contém o número de série do certificado, a data e hora (com precisão de milissegundos) em que o mesmo deixou de ser válido. O motivo de o mesmo deixar de ser válido fica armazenado no sistema da autoridade emitente.

O maior problema com o uso de uma LCR é a não obrigatoriedade da consulta da lista na hora de validar um certificado. Em um sistema customizado, onde se está implementando a regra de validação, devem ser seguidas determinadas premissas, como verificar se o ponto de acesso à LCR existe. Se existir, tentar baixar o arquivo. Se baixar, verificar sua estrutura, assinatura, etc.

Se tudo estiver OK com o arquivo, verificar se o ID do certificado se encontra na lista e desde quando. Caso isto não seja feito e você esteja confiando apenas na validade que está descrita dentro do certificado, você pode estar aceitando um certificado não expirado e que pode estar comprometido.

Muitas vezes há Autoridades Certificadoras que publicam atualizações da LCR de uma em uma hora. Você poderá legitimamente realizar a validação da LCR e verificar que o certificado que você está aceitando não se encontra na lista naquele momento. Porém, na próxima publicação, você poderia verificar que o mesmo já estava revogado no momento em que você o aceitou, mas não havia como verificar isto. É por este motivo que o OCSP é tão importante.

Como ele é um processo síncrono, você pode incluir esta verificação em seu processo de autenticação e receber a resposta imediatamente.

Tudo começa com o sorteio de um par de chaves (**BOX 4**). O par de chaves é o coração da estrutura. O método de geração deste par de chaves, o

armazenamento da parte privada do par de chaves e a maneira de consultá-la vão





processo de geração. Deste modo, é possível ter uma chave que é publicada e fica acessível a qualquer um (a chave pública) e que é usada para criptografar qualquer mensagem para a chave complementar desta chave pública (a chave privada), que fica em poder do proprietário das mesmas e cujo acesso é completamente restrito.

Desta forma, há uma e somente uma chave privada para uma determinada chave pública. Se o processo de geração das chaves foi seguro, ou seja, se as sementes randômicas usadas para gerar um número pseudoaleatório sem possibilidades de colisão (sem que seja possível gerar o mesmo número de qualquer outra maneira em qualquer outra parte do mundo) são seguras, pode-se garantir a força deste par de chaves e qualquer mensagem criptografada por elas estará protegida.

Como tudo, no mundo computacional, estes algoritmos também possuem suas fraquezas.

Um par de chaves, **ou um certificado digital**, está sujeito a ataques por “força bruta” (tentativa e erro). Por este motivo, usamos chaves de 2048 bits de tamanho que ajudam a “dar trabalho” ao invasor, de modo que o mesmo irá necessitar de um gigantesco e não trivial poder computacional para quebrar a chave e utilizá-la maliciosamente.

Por este motivo, os pares de chaves incrustados em um **certificado digital** têm validade limitada, de modo a não facilitar as coisas para quem deseja usar esta tecnologia de modo não autorizado.

Atualmente, uma Autoridade Certificadora precisa estar em acordo com padrões internacionais de segurança, de modo a poder gerar e manter um sistema de PKI confiável. Para ter uma ideia, o SERPRO, que mantém a Autoridade Certificadora Raiz da ICP-BRASIL, possui em seu Centro de Certificação Digital no Rio de Janeiro uma sala cofre com seis níveis de segurança, sendo que a sala onde fica a máquina que guarda o par de chaves é encravada numa rocha.





caso de uma invasão, guardas armados, entre outros itens de segurança física. A máquina onde é guardada a chave privada fica o tempo todo Off-line e possui diversos mecanismos de redundância, uma vez que a validade desta chave é longa, em termos computacionais.

Dura 10 anos, ao final dos quais um novo par de chaves deve ser gerado.

O mecanismo de geração é simples. Basta utilizar um determinado algoritmo de criptografia assimétrica para geração de chaves, como Diffie-Hellman, El Gamal, RSA, entre outros. Normalmente, gera-se um arquivo binário para cada chave. Em sistemas de PKI profissionais, geram-se os pares de chaves em dispositivos criptográficos conhecidos como HSM (do inglês Hardware Security Module – Módulo de Segurança Físico).

Estes dispositivos ficam conectados a uma máquina e respondem a protocolos específicos com controle de driver próprio, impedindo, por exemplo, acesso aos slots que contém as chaves; não permitem consultas não identificadas, nem ataques por invasão de memória, buffer overflows, etc.

São dispositivos caros, mas muitas vezes blindados fisicamente, o que impede roubo de módulos de memória por dump, etc.

Em existindo o par de chaves, basta armazenar a chave pública em formato X.509, e a chave privada de forma segura em um formato conhecido, como PKCS#8 (do inglês Public Key Cryptographic Standard Number Eighth – Padrão de Criptografia de Chave Pública Número Oito - **BOX 5**). Ele é um envelope digital que permite armazenar a chave privada de forma estruturada e protegida por uma senha de acesso.





está protegida por uma senha.

Com este par de chaves formatado, a Autoridade Certificadora é criada inserindo a chave pública num arquivo de formato PKCS#12, ou permitindo montar um certificado sob demanda (como os HSM fazem) quando há necessidade de assinar um certificado de Autoridades Intermediárias.

Notificações :)

BOX 5: Padrões PKCS – Public Key Cryptography Standard

Estes padrões foram especificados pela RSA em colaboração com diversas entidades do setor, de forma a acelerar o desenvolvimento das tecnologias de criptografia. Suas versões vêm sendo publicadas desde 1991. A seguir tem-se a lista dos padrões e um resumo do seu uso.

- **PKCS#1** - define as propriedades matemáticas para uso de chaves RSA, como o resultado de dois números primos longos baseados em módulo n .
- **PKCS#2** - definia a criptografia de algoritmos HASH, mas foi incorporado ao PKCS#1 em 2010.
- **PKCS#3** - acordo de uso do Algoritmo Diffie-Hellman. Um protocolo criptográfico que permite a duas partes distintas e sem qualquer correlação estabelecer entre si uma chave secreta em um canal de comunicação inseguro.
- **PKCS#4** - cobria a sintaxe RDA, mas foi incorporado ao PKCS#1.
- **PKCS#5** - padrão para criptografia baseada em senha. É descrito pela RFC 2898.
- **PKCS#6** - padrão para a Sintaxe de Extensões de Certificados. Encontra-se obsoleta desde a constituição da terceira versão da estrutura X.509.
- **PKCS#7** - padrão para a Sintaxe de Mensagens Criptografadas. É detalhada pela RFC 2315 e é usada para assinaturas ou criptografia de mensagens em uma PKI. É usada como **envelope digital de um certificado digital**, ou em resposta a uma requisição de um novo certificado. Formou a base para o padrão S/MIME e, atualmente, para o padrão CMS – Cryptographic Message Syntax), entre outros usos.





- **PKCS#9** - Tipos de Atributos Seleccionados. Define os tipos de atributos usados nos padrões PKCS#6, PKCS#7, PKCS#8 e PKCS#10.
- **PKCS#10** - padrão para requisição de Certificados Digitais. Formata as mensagens que serão enviadas à Autoridade Certificadora para a geração de um novo certificado.
- **PKCS#11** - Interface de Token Criptográfico. Uma API que define uma interface genérica para os tokens criptográficos, HSM, Criptografia de Discos e Smartcards.
- **PKCS#12** - padrão para Sintaxe de Troca de Informações Pessoais. Define o formato de arquivo usado para armazenar todo o **conjunto que forma um certificado digital**, contendo o par de chaves e outras informações, protegido por uma senha.
- **PKCS#13** - padrão para Criptografia de Curvas Elípticas.
- **PKCS#14** - padrão para a geração de números pseudorrandômicos.
- **PKCS#15** - padrão para a formatação de informação de tokens criptográficos.

Notificações :)

Neste ponto é importante saber que existe uma hierarquia dentro da PKI. Entenda a Autoridade Certificadora Raiz como o elemento ROOT de um XML. Qualquer autoridade criada sob sua autoridade é conhecida como Autoridade Certificadora Intermediária e ela, por si só, também tem toda uma estrutura abaixo.

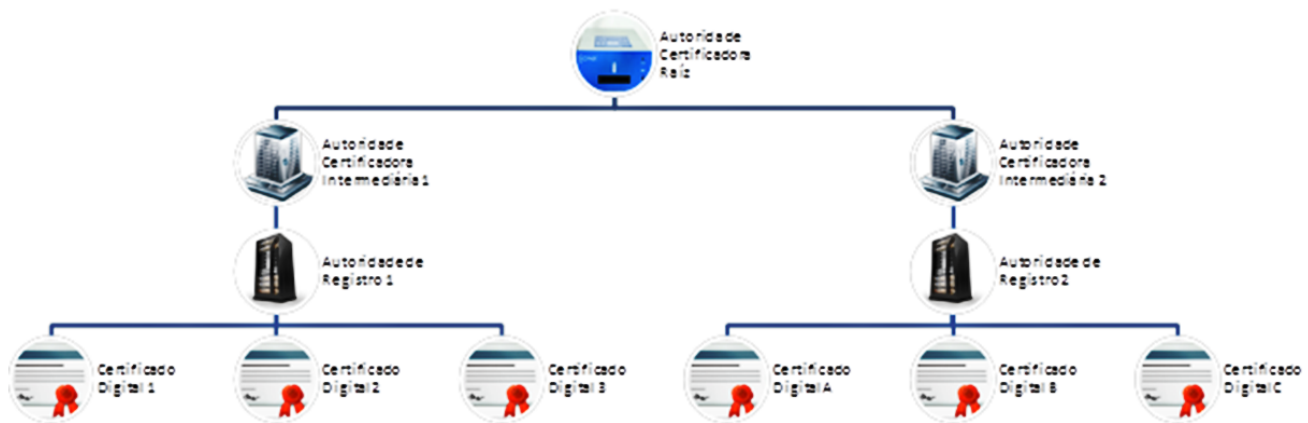
Um certificado digital é como um salvo conduto. É um documento que permite autenticar “ações” ou “informações” com respaldo legal. Significa que deve existir uma regra pela qual se limita o poder daquele documento. Imagine uma procuração com plenos poderes de forma ilimitada, ou seja, alguém que detivesse este documento poderia fazer em seu nome tudo o que você faz.

Para isto, foi criada uma Política de Certificação Digital, que prevê o que pode e o que não pode em determinada estrutura de PKI.





A **Figura 1** ilustra um quadro que explica visualmente esta ideia.



Notificações :)

Figura 1. Hierarquia da Política de Certificação Digital

O processo de certificação começa com a geração do par de chaves.

Normalmente, este processo se dá diretamente na máquina do usuário. Este é um ponto delicado, uma vez que ela pode ter diversos problemas de segurança. Para tentar minimizar este problema, a maioria dos sistemas operacionais e as linguagens de programação tem um mecanismo bastante seguro de realizar este sorteio. No Windows isto pode ser feito, por exemplo, pelo CAPICOM. O CAPICOM é um componente de 32 bits disponível para uso desde o Windows XP. Ao usar o objeto COM disponível, você pode embutir a Crypto API do sistema operacional diretamente em seus projetos.

Como alternativa ao CAPICOM, existem os mecanismos de Certificação Digital e PKI embutidos no .NET Framework desde a versão 1.1. Esses mecanismos, entretanto, são muito simplórios e foram desenvolvidos pensando no uso de chamadas básicas da API com comandos do Windows Server 2008 e sua



comandos e métodos da API original do Windows, com boa chance de fazer algo errado).

Felizmente, há gente por aí preocupada com isso e diversas bibliotecas gratuitas e corretamente desenvolvidas (inclusive para uso em produção) estão à nossa disposição. Usaremos nesta série de artigos a Bouncy Castle, que desde 2002 está disponível para a linguagem JAVA e mais recentemente para C#. A Bouncy Castle foi concebida para ser um CSP – Crypto Service Provider (Provedor de Serviços de Criptografia), de forma que a API pode ser usada para qualquer fim que envolva criptografia, certificação digital e assinatura de arquivos e mensagens.

Notificações :)

De qualquer modo, via CAPICOM, Bouncy Castle ou outra API, o processo de sorteio do par de chaves deve acontecer.

Com o par de chaves gerado, é necessário realizar a geração de uma requisição de certificado e este, por sua vez, deve ser assinado pela autoridade certificadora emissora.

Deste modo, esta requisição é enviada à Autoridade Intermediária, que foi por sua vez assinada pela Autoridade Certificadora Raiz e irá, além de autenticar e assinar a requisição, incluir todas as informações de sua cadeia de certificação. Assim, quem **receber o certificado digital** terá todas as condições de validar sua estrutura e a da cadeia de certificação que o gerou.

Assim que a requisição é assinada, a Autoridade deve devolver o certificado completo.

Armazenando Certificados no Windows





formato .pfx (um PKCS#12 com formato um tanto antigo, de acordo com os padrões da RSA) é armazenado convenientemente num slot virtual correto (como se estivesse num HSM) e é protegido por uma senha.

As **Figuras 2 a 9** demonstram a instalação de um arquivo .pfx na Windows KeyStore. Primeiramente, execute o arquivo PFX para iniciar a instalação.

Notificações :)

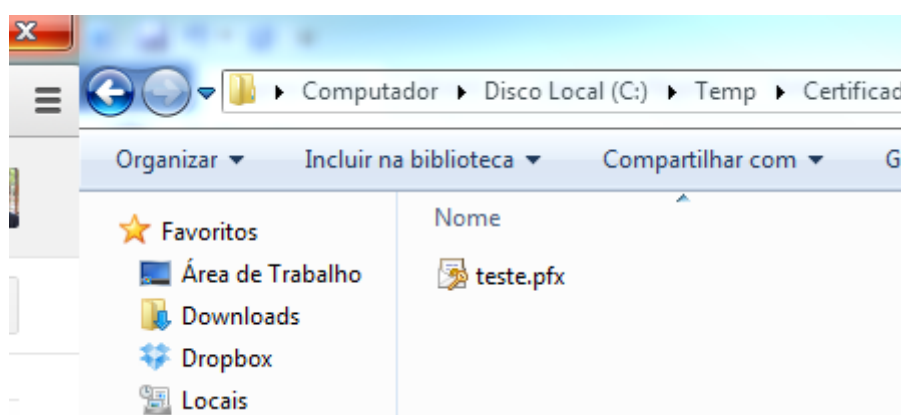
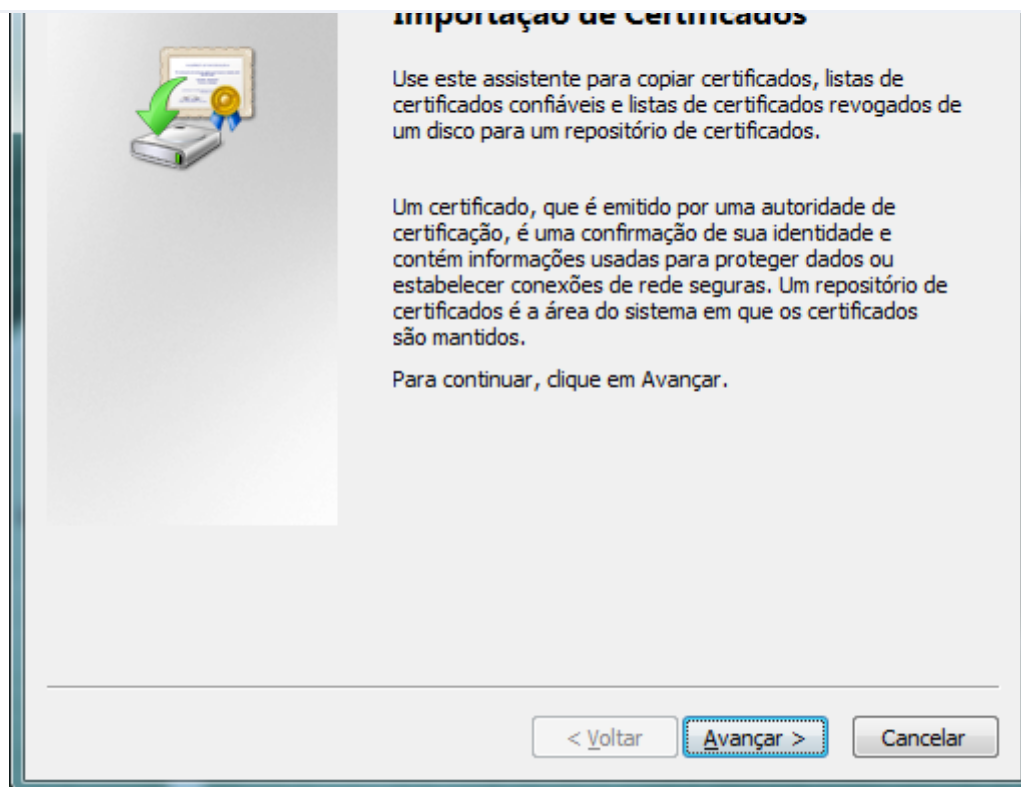


Figura 2. Arquivo .pfx visualizado no Windows Explorer

Feito isso, será iniciado o wizard de instalação do certificado, conforme mostra a **Figura 3**. Nessa etapa, basta clicar em Avançar.





Notificações :)

Figura 3. Wizard para instalação do Certificado Digital

Na tela seguinte (**Figura 4**) é exibido o caminho do arquivo .pfx a ser importado, continue o processo clicando em Avançar.



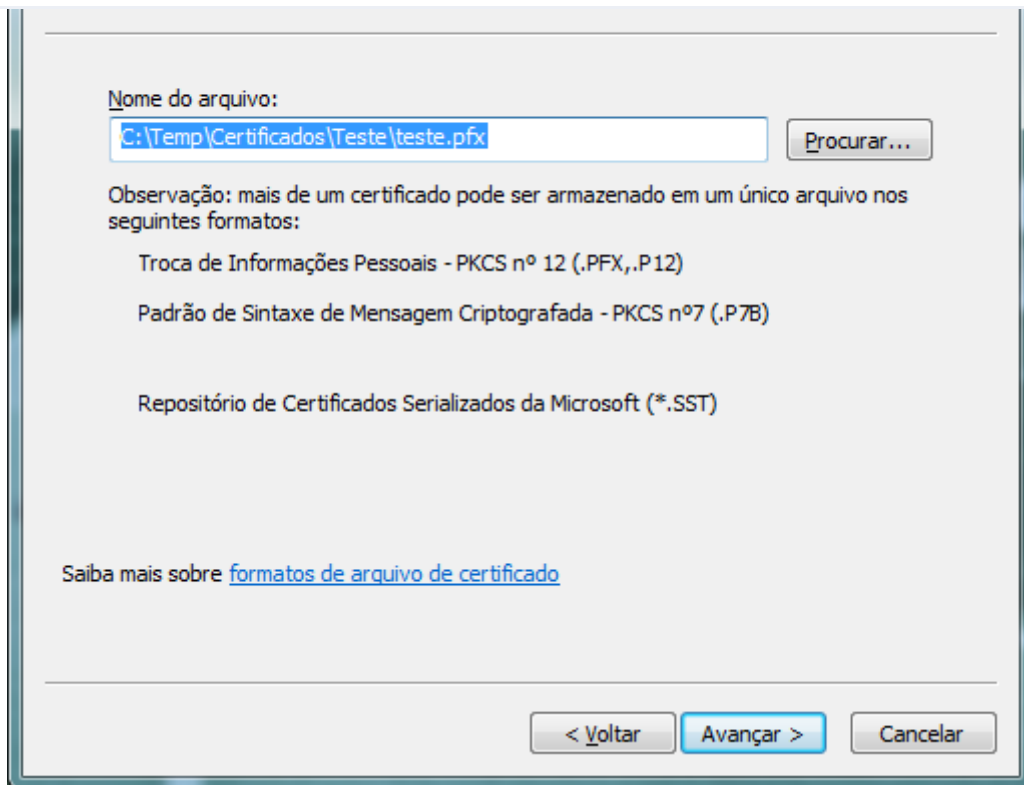


Figura 4. Especificação do arquivo o qual se quer instalar

A seguir é necessário digitar a senha de acesso ao Certificado (a mesma que foi gerada na requisição). Selecione a opção de proteção da chave privada (primeiro checkbox da tela, que aparece marcado na **Figura 5**). Mantenha a opção de chave exportável desmarcada, significando que ninguém vai conseguir reaver o certificado completamente.

Por fim, marque a opção para incluir todas as propriedades estendidas (terceiro checkbox), de modo, a saber, a cadeia de certificação, usos específicos do certificado, etc, e clique em Avançar.





Digite a senha da chave privada.

Senha:

☒ Ativar proteção de chaves privadas fortes. Se ativar esta opção, você será avisado todas as vezes que uma chave privada for usada por um aplicativo.

☐ Marcar esta chave como exportável. Isso possibilitará o backup e o transporte das chaves posteriormente.

☒ Incluir todas as propriedades estendidas.

Saiba mais sobre [como proteger chaves privadas](#)

< Voltar Avançar > Cancelar

Figura 5. Definição da senha e informações sobre a chave

Na próxima tela (**Figura 6**) deve-se escolher o local onde será armazenado o certificado. É possível deixar o assistente escolher um repositório automaticamente, ou você mesmo pode especificar um slot. Pressione novamente o botão Avançar.



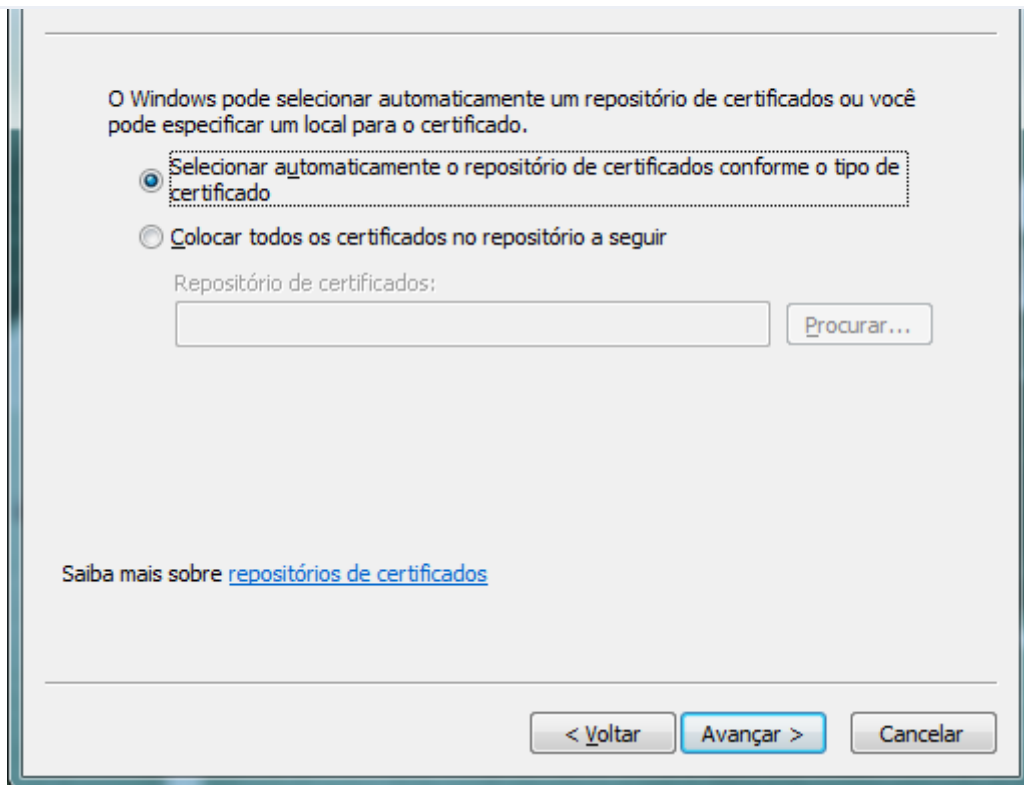
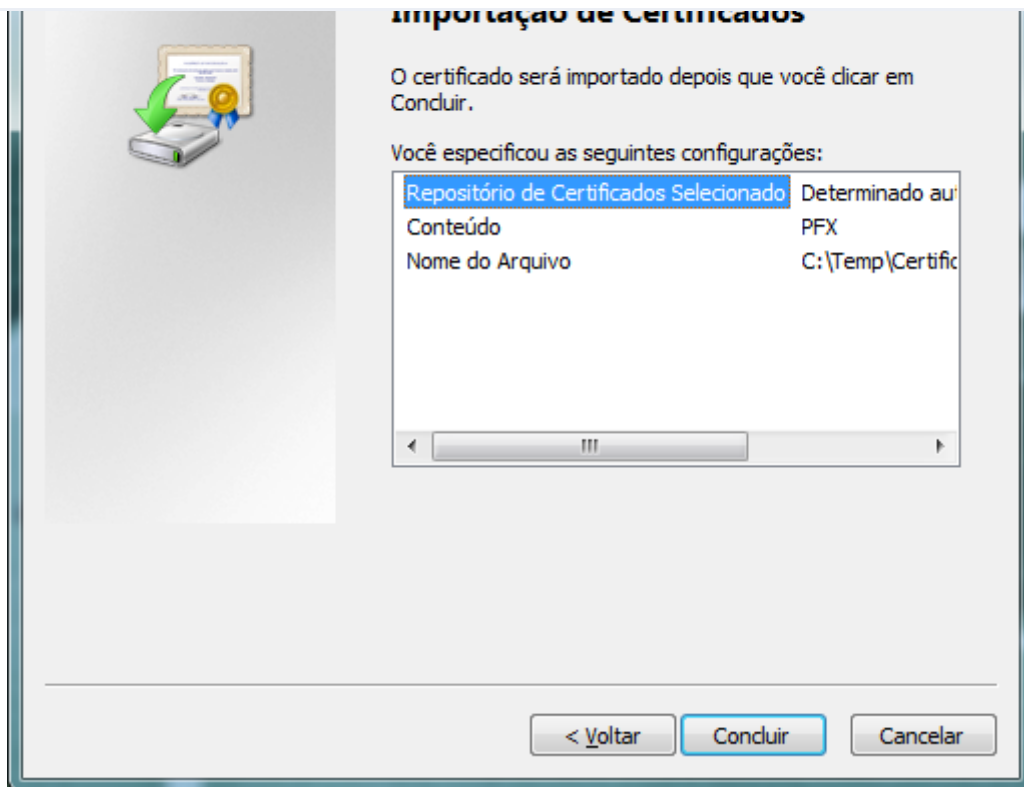


Figura 6. Definição do local onde será armazenado o certificado

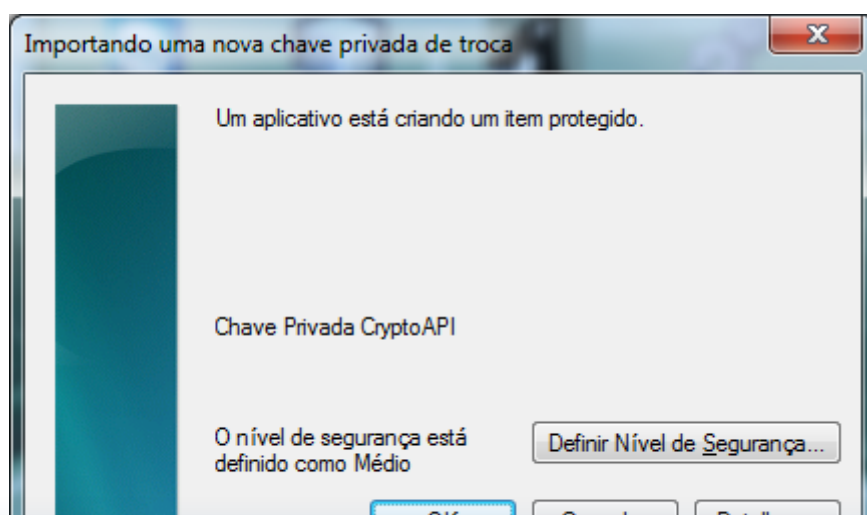
A tela da próxima etapa (**Figura 7**) serve apenas para apresentar alguns dados sobre a importação que será feita, basta clicar em Concluir para iniciar a importação propriamente dita.



Notificações :)

Figura 7. Previsão de resultado do Wizard aguardando confirmação do usuário

Na sequência, o mecanismo de armazenamento do Windows exibirá uma pequena tela demonstrando que a chave privada será guardada e protegida. Clique em OK para confirmar.



7



Por fim, será exibida uma mensagem indicando o sucesso do procedimento, conforme a **Figura 9**.

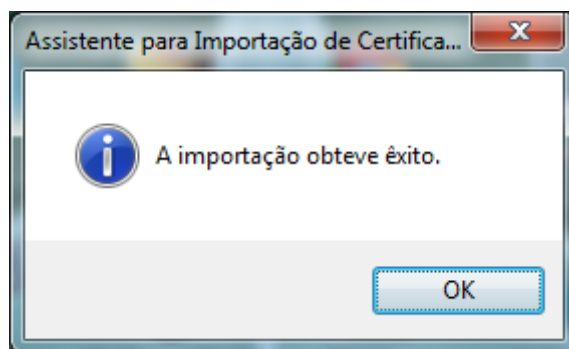


Figura 9. Mensagem de confirmação da importação do arquivo

Você pode **visualizar este Certificado Digital** a qualquer momento usando o atalho do Internet Explorer, por exemplo, ou chamando o gerenciador de certificados digitais através do comando **certmgr.msc**. Clique no botão Iniciar (até o Windows 7), ou na tecla Windows (no Windows 8), e digite o comando. O programa será listado, então o execute. Será aberto gerenciador de certificados digitais como o da **Figura 10**.



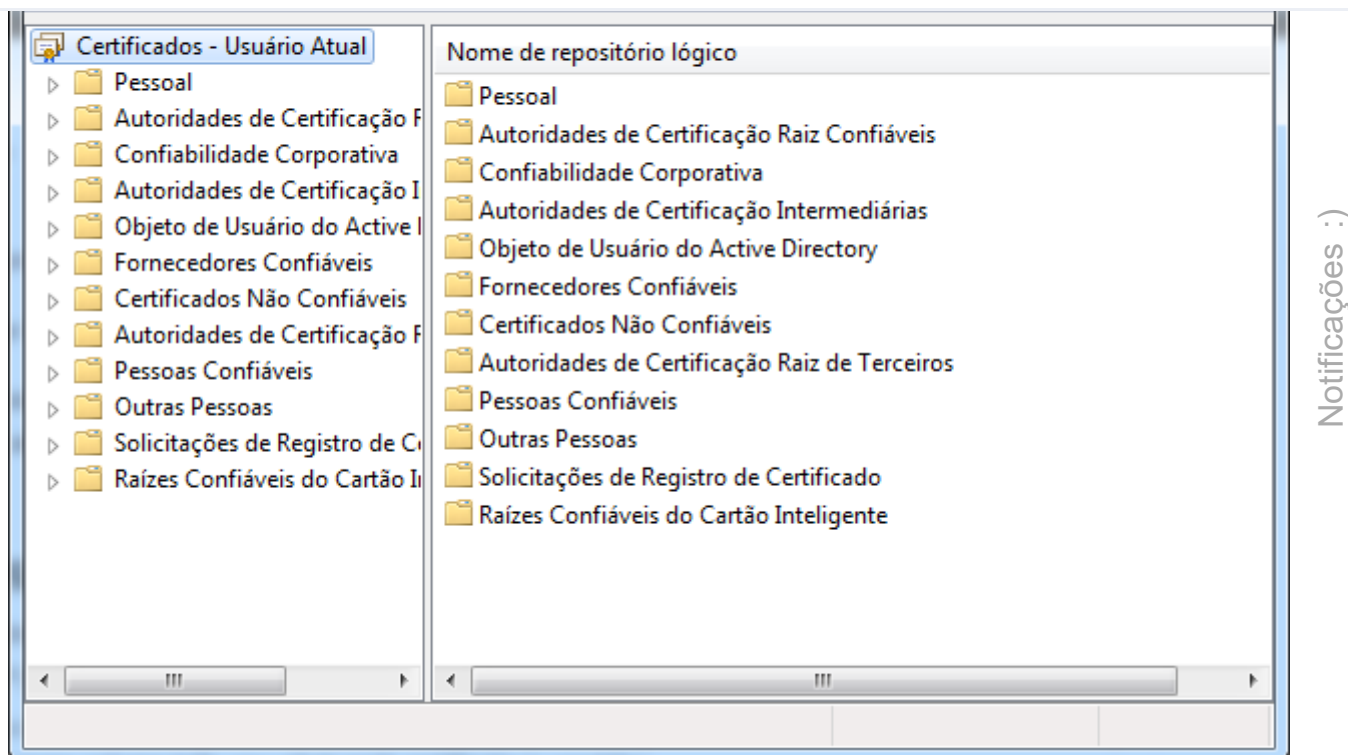
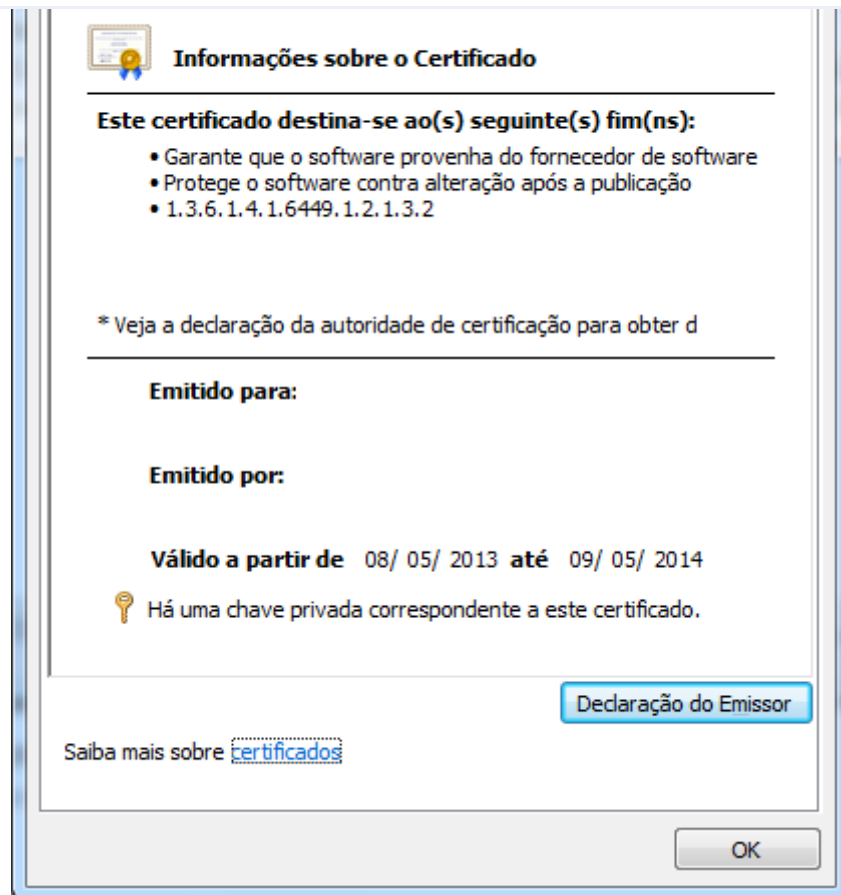


Figura 10. Gerenciador de Certificados Digitais

Abra a aba **Pessoal > Certificados** e você **verá seu certificado digital na lista**.
Clique duas vezes e veja os detalhes do Certificado, semelhante aos que são mostrados na **Figura 11**.





Notificações :)

Figura 11. Detalhes do Certificado Digital

Até este ponto foi explicado o que é PKI, o que é uma Autoridade Certificadora e suas hierarquias, bem como o **que é um Certificado Digital**. Agora vamos começar a praticar a Certificação Digital. Nas próximas seções, vamos, passo a passo, **gerar um certificado digital**, simular uma Autoridade Certificadora para assinar o certificado e vamos utilizá-lo para assinar um documento.

A Instalação do Certificado Via Código

Um certificado digital pode ser instalado por um programa que você pode escrever. Observe no código da **Listagem 1** que o procedimento é bastante simples





```
04     dialog.Filter = "Certificado Digital (*.p12)|*.p12|Certificado Digital (*.p12)|*.p12";
05     dialog.FilterIndex = 2;
06     if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
07     {
08         string senha = Microsoft.VisualBasic.Interaction.InputBox("Digite sua
                                \"ATENÇÃO\", \"Default\", 0, 0);

09         X509Certificates.X509Certificate2 certificado =
                new X509Certificate2(dialog.FileName, senha, X509KeyStorageFlags.Exportable |
                X509KeyStorageFlags.MachineKeySet | X509KeyStorageFlags.PersistKeySet);
10         X509Certificates.X509Store storePessoal = new
                X509Certificates.X509Store(StoreName.My, StoreLocation.CurrentUser);
11         storePessoal.Open(OpenFlags.ReadWrite);
12         storePessoal.Add(certificado);
13         MessageBox.Show("Certificado Importado com Sucesso", "ATENÇÃO",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
14     }
15 }
16 catch (Exception ex)
17 {
18     MessageBox.Show("Erro ao importar certificado", "ATENÇÃO", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
19 }
```

Notificações :)

Listagem 1. Importação de Certificado Digital via C#

O código inicia invocando uma caixa de diálogo para abertura de arquivo. Isto vai permitir que o usuário busque o **arquivo do certificado digital** em qualquer diretório de seu disco. Em seguida, o programa vai utilizar um recurso da linguagem Visual Basic chamado `InputBox`, que exibe uma caixa de diálogo pronta para que o usuário possa informar qualquer string.

Deste modo, pediremos que ele digite uma senha para proteger o certificado. Essa não é a forma mais segura de receber esta senha, mas como trata-se de um exemplo didático, é o bastante para esse caso.





System.Security.Cryptography, que deve ser referenciado através do comando using). Este objeto representa um **Certificado Digital X509 Versão 3** com todas as suas propriedades, atributos e funções. Adicionamos, junto ao construtor do objeto, as flags, permitindo que o mesmo seja exportável, persistível e localizável na máquina do usuário.

Isto fará com que este certificado possa ser armazenado na KeyStore do Windows, possa ser visualizado na lista de Certificados pessoais e possa ser Exportado, incluindo sua chave privada, a qualquer momento que o usuário queira.

Não é recomendável, no entanto, armazenar um certificado com a chave exportável, porque se alguém tiver acesso e privilégios, poderá reaver o certificado e utilizar o mesmo sem maiores problemas.

Por fim, abrimos a KeyStore Pessoal (StoreName.My) para o usuário corrente e adicionamos o Certificado recém importado.

Até este momento já descrevemos os principais conceitos de Certificação Digital, seus objetivos, sua estrutura e mecanismos e demonstramos superficialmente como lidar com um Certificado.

Agora precisamos começar a estruturar um processo que será usado ao longo desta série para demonstrar em detalhes a geração dos pares de chaves de toda a cadeia de certificação, os processos de geração de certificados desde a requisição até a assinatura e o uso de um certificado de cliente, de modo que um dia estes mecanismos possam compor um dos módulos de autenticação e criptografia dos sistemas que você vier a desenvolver em sua vida profissional.





conhecida como CSR (Certification Signing Request, do inglês, Requisição de Assinatura de Certificado).

Todo Certificado digital gerado tem uma formatação e codificação específica chamada de ASN.1, do inglês, Abstract Syntax Notation versão 1. A ASN.1 é um padrão de estrutura de dados que contém regras para a representação, codificação, transmissão e decodificação de dados em telecomunicações e redes de computadores.

Este padrão possui diversos métodos de codificação e é importante porque todos os certificados digitais gerados sob o padrão X.509 usam essa estrutura para sua constituição. Os métodos de codificação mais amplamente utilizados para manter pares de chaves e certificados é o BER (Basic Encoding Rules) e o DER (Canonical Encoding Rules).

Você não precisa ser um expert nesta notação, porque justamente por isso utilizaremos uma biblioteca para nos ajudar a ler, gerar e usar Certificados e Pares de Chaves sem ter que descer a um nível tão baixo de implementação de código e poderemos nos concentrar em nossas regras de negócio, nas quais utilizaremos o poder da Criptografia e Certificação Digital para nos ajudar.

A Biblioteca BouncyCastle

Na próxima seção deste artigo iniciaremos a implementação de um projeto usando a biblioteca, portanto, vamos definir o que ela é e como adquirir e instalar.

A BouncyCastle é uma iniciativa OpenSource e Gratuita. Foi concebida





necessidade de implementar o pacote JCE (Java Cryptographic Extensions) da Oracle, que é muito complexo para quem é leigo no assunto e que demanda um certo tempo para que você o utilize com cuidado em suas aplicações.

Além disto, a JCE demanda que você tenha adicionado determinadas configurações que te permitam utilizar tamanhos de chaves maiores que 1024 bits. Esta configuração demanda que você modifique sua Java Virtual Machine, de modo a permitir esse novo tamanho de chaves. Além disso, distribuir essa configuração entre clientes, principalmente via web, é penoso.

A Oracle (na verdade isto vem desde a época da Sun) teve de realizar isso por uma imposição do governo Norte Americano de não permitir o uso de tecnologias de proteção com criptografia forte para não facilitar a vida de seus inimigos, como Afeganistão, Iraque, entre outros. Recentemente vimos o resultado desta decisão em processos de quebra de privacidade.

Usando a BouncyCastle você não precisa se preocupar com isso, porque a biblioteca reimplementa os métodos de geração de chaves e permite a geração de pares de chaves com tamanhos superiores a 4096 bits.

Em .NET não há certas imposições e nem a necessidade de realizar qualquer configuração para a geração de chaves, mas as bibliotecas existentes estão em objetos ActiveX ou COM e não têm implementação em código gerenciado, fazendo-se necessário mesclar código não gerenciado em seus programas. Além disso, as implementações são fracamente documentadas e de difícil entendimento.

Com a BouncyCastle, temos uma API de alto nível e amplamente documentada. Também é muito fácil encontrar em fóruns diversas referências e exemplos de





Para usar essa biblioteca, é bem simples. No site oficial (ver seção **Links**), basta seguir o link no menu à esquerda, onde diz “C# home”. Lá existirão todos os recursos da implementação para .NET (em linguagem C#). Ali estarão disponíveis os releases da biblioteca já compilados, os fontes, extensões para uso de outros algoritmos, etc.

Para os exemplos deste artigo estaremos usando a versão 1.7 da biblioteca. Basta realizar o download do zip, extrair e referenciar a DLL BouncyCastle.Crypto.dll no seu projeto.

A Geração da Requisição

Para desenvolver o exemplo desta seção, crie um novo projeto do tipo Windows Forms Application no Visual Studio. No novo formulário Form1 que será gerado automaticamente pelo IDE, inclua da paleta de componentes um componente MenuStrip para que tenhamos um menu suspenso em nossa aplicação, e crie a seguinte estrutura de menu:

- Autoridade Certificadora Raiz
- Gerar Par de Chaves
- Gerar Certificado Raiz
- Instalar Certificado Raiz
- Assinar Requisição de Autoridade Intermediária
- Autoridade Certificadora Intermediária
- Gerar Par de Chaves
- Gerar Requisição de Certificado Intermediário
- Instalar Certificado Intermediário





- Certificado Digital Cliente
- Gerar Par de Chave
- Gerar Requisição de Certificado Cliente
- Importar Certificado Cliente
- Assinar Documento
- Visualizar Assinatura

Notificações :)

Esta estrutura de Menu demonstra as funcionalidades que iremos cobrir ao longo desta série de artigos.

Depois de criada a estrutura de menus, clique na opção Gerar Par de Chaves do Menu Autoridade Certificadora Raiz, e no seu evento Click crie uma chamada para o método GerarParChaves (). Em seguida, implemente esse método, como mostra a **Listagem 2**.

```
01 private void GerarParChaves()  
02 {  
03     try  
04     {  
05         var geradorAleatoriedade = new Org.BouncyCastle.Crypto.Prng.  
CryptoApiRandomGenerator();  
06  
07         var valorAleatorio = new Org.BouncyCastle.Security.  
SecureRandom(geradorAleatoriedade);  
08  
09         var parametrosGeracaoChave = new Org.BouncyCastle.Crypto.  
KeyGenerationParameters(valorAleatorio, 2048);  
10  
11         var geradorChaves = new Org.BouncyCastle.Crypto.Generators.  
RsaKeyPairGenerator();  
12         geradorChaves.Init(parametrosGeracaoChave);  
13  
14         var parChaves = geradorChaves.GenerateKeyPair();  
15
```





```
22
23     tw = new StringWriter();
24     pw = new PemWriter(tw);
25     pw.WriteObject(parChaves.Public);
26     pw.Writer.Flush();
27
28     string stringPublic = tw.ToString();
29
30     SaveFileDialog dialog = new SaveFileDialog();
31     dialog.Filter = "Chave Pública (*.cer)|*.cer";
32     dialog.FilterIndex = 1;
33     dialog.RestoreDirectory = true;
34
35     if (dialog.ShowDialog() == DialogResult.OK)
36     {
37         File.WriteAllText(dialog.FileName, stringPublic, Encoding.ASCII);
38     }
39
40     MessageBox.Show("Chave Pública salva com sucesso", "ATENÇÃO",
41     MessageBoxButtons.OK, MessageBoxIcon.Information);
42
43     dialog = new SaveFileDialog();
44     dialog.Filter = "Chave Privada (*.pvk)|*.pvk";
45     dialog.FilterIndex = 1;
46     dialog.RestoreDirectory = true;
47
48     if (dialog.ShowDialog() == DialogResult.OK)
49     {
50         File.WriteAllText(dialog.FileName, stringPrivate, Encoding.ASCII);
51     }
52
53     MessageBox.Show("Chave Privada salva com sucesso", "ATENÇÃO",
54     MessageBoxButtons.OK, MessageBoxIcon.Information);
55 }
56 catch (Exception ex)
57 {
58     MessageBox.Show(ex.Message, "ATENÇÃO", MessageBoxButtons.OK,
59     MessageBoxIcon.Exclamation);
60 }
```

Notificações :)

Listagem 2. Geração do Par de Chaves





CryptoApiRandomGenerator. Esta classe pertence à API da BouncyCastle e serve para criar números pseudorrandômicos de uma forma relativamente segura, em conjunto com a classe SecureRandom.

Este objeto será usado sempre que precisemos de números seguros e o mais aleatórios possível.

O objeto parametrosGeracaoChave é utilizado para conter os parâmetros estruturais das chaves que vamos criar. Deste modo, estamos dizendo que iremos precisar de uma chave com determinada sequência aleatória e um tamanho de 2048 bytes para as chaves. O tamanho máximo da chave pode ser de 4096 bytes, mas quanto maior a chave, maior o poder de processamento para geração e uso das chaves.

Obviamente, alguns segundos a mais de espera para gerar uma chave não é problema, mas em um uso massivo, uma chave muito grande pode gerar certa lentidão no uso do sistema.

Enfim, chegamos à instância do objeto RsaKeyPairGenerator, responsável por realmente criar o par de chaves, necessitando apenas que passemos nossos parâmetros estruturais como informação de inicialização.

Ao usar o método GenerateKeyPair de nosso gerador de chaves, estaremos criando um objeto do tipo AsymmetricCipherKeyPair, que contém as chaves pública e privada, que podem ser obtidas usando as propriedades Public e Private do objeto.

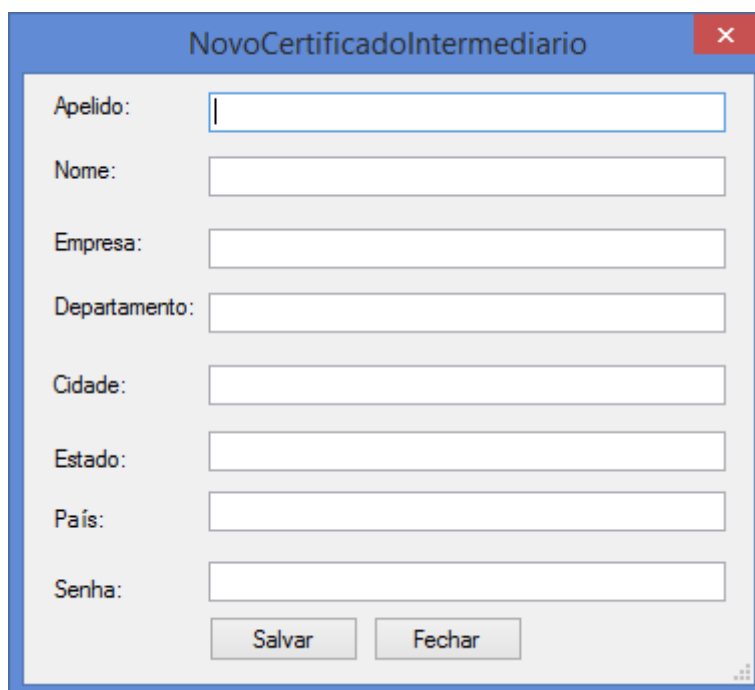
Cada uma dessas propriedades nos devolve um objeto do tipo AsymmetricKeyParameter contendo as informações próprias de cada pedaço da



requisições de certificado a qual veremos mais adiante.

Para gerar a requisição de um certificado, que posteriormente vai ser assinado por uma autoridade certificadora, crie um novo formulário na sua aplicação chamado de NovoCertificadoIntermediario e coloque os controles como o exemplo da **Figura 12**.

Notificações :)



The image shows a Windows-style dialog box titled "NovoCertificadoIntermediario" with a red close button in the top right corner. The dialog contains a vertical stack of text input fields, each preceded by a label: "Apelido:", "Nome:", "Empresa:", "Departamento:", "Cidade:", "Estado:", "País:", and "Senha:". At the bottom of the dialog, there are two buttons: "Salvar" and "Fechar". The dialog has a blue border and a light gray background.

Figura 12. Novo Formulário de Geração de Requisição de Certificado

Nomeie as caixas de texto de acordo com a **Tabela 1**.



Nome	CN
Empresa	O
Departamento	OU
Cidade	L
Estado	ST
País	C
Senha	Senha

Notificações :)

Tabela 1. Nomes dos textboxes do form NovoCertificadoIntermediario

Você deve estar se perguntando o porquê dos nomes tão diferentes para os textboxes empresa, departamento, cidade, estado e país. Um certificado digital no padrão X.509 segue a estrutura de organização de dados comumente encontrado em formulários do LDAP e outros mecanismos de autenticação.

Deste modo, estamos usando a mesma nomenclatura utilizada nestes padrões, onde o CN é o Common Name (Nome Ordinário), O para Organization (Organização, Empresa), OU para Organization Unit (Departamento, Setor), L para Location (Cidade, Localização), ST para State (Estado Federativo ou Unidade Federativa) e C para Country (País).

Com os textboxes configurados, crie o event handler para o evento Click do botão salvar (duplo clique sobre o botão) e inclua o código da **Listagem 3**.

```
01 if (apelido.Text.Equals(""))
02 {
03     MessageBox.Show("Digite o Apelido do Certificado", "ATENÇÃO",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
```





```
09 else if (O.Text.Equals(""))
10 {
11     MessageBox.Show("Digite a Empresa do Certificado", "ATENÇÃO",
12         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
13 }
14 else if (OU.Text.Equals(""))
15 {
16     MessageBox.Show("Digite o Departamento do Certificado", "ATENÇÃO",
17         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
18 }
19 else if (L.Text.Equals(""))
20 {
21     MessageBox.Show("Digite a Cidade do Certificado", "ATENÇÃO",
22         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
23 }
24 else if (ST.Text.Equals(""))
25 {
26     MessageBox.Show("Digite o Estado do Certificado", "ATENÇÃO",
27         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
28 }
29 else if (C.Text.Equals(""))
30 {
31     MessageBox.Show("Digite o País do Certificado", "ATENÇÃO",
32         MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
33 }
34 else
35 {
36     string DN = "CN=" + CN.Text + ",O=" + O.Text + ",OU=" + OU.Text + ",L="
37         + L.Text + ",ST=" + ST.Text + ",C=" + C.Text;
38     X509Name subject = new X509Name(DN);
39     Pkcs10CertificationRequest csr = new Pkcs10CertificationRequest
40         ("SHA1WithRSA", subject, GetChavePublica(), null, GetChavePrivada());
41     StringBuilder CSRPem = new StringBuilder();
42     PemWriter CSRPemWriter = new PemWriter(new StringWriter(CSRPem));
43     CSRPemWriter.WriteObject(csr);
44     CSRPemWriter.Writer.Flush();
45
46     SaveFileDialog dialog = new SaveFileDialog();
47     dialog.Filter = "Requisição de Certificado (*.csr)|*.csr";
48     dialog.FilterIndex = 1;
49     if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
```





Listagem 3. Geração da Requisição

Notificações :)

Este código demanda que todos os campos sejam preenchidos. Caso não tenham sido preenchidos, uma mensagem de erro específica é lançada.

Se todos os campos tiverem sido preenchidos, o código cria uma nova instância do objeto X509Name, que recebe o DN (Distinguished Name) para a requisição, que na verdade é a concatenação de todos os campos que foram preenchidos, separados por uma vírgula.

A BouncyCastle tem uma classe Pkcs10CertificationRequest que é, por si só, uma estrutura completa de requisição de certificado (CSR = Certificate Signing Request). Basta que informemos um conjunto de algoritmos para assinatura e criptografia, a estrutura X509Name gerada (que contém todos os parâmetros de dados do futuro certificado), a chave pública e a chave privada que podemos gerar com o método já explicado anteriormente.

Para resgatar estas chaves que foram geradas e salvas em disco, use os métodos GetChavePublica e GetChavePrivada que estão, respectivamente, nas **Listagens 4** e **5**.

Podemos salvar a requisição feita para que a mesma possa ser enviada para a autoridade certificadora ou para que, no próximo artigo desta série, possamos usar a requisição e assinar o certificado como se fôssemos uma autoridade certificadora.





```
05     dialog.Filter = "Chave Pública (*.cer)|*.cer";
06     dialog.FilterIndex = 1;
07     if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
08     {
09         string conteudo = File.ReadAllText(dialog.FileName).Replace
10             ("-----BEGIN PUBLIC KEY-----\r\n", "").Replace
11             ("-----END PUBLIC KEY-----", "");
12         byte[] bytesConteudo = Convert.FromBase64String(conteudo);
13         parametro = PublicKeyFactory.CreateKey(bytesConteudo);
14     }
15     return parametro;
16 }
```

Notificações :)

Listagem 4. Obtenção da Chave Pública

Na **Listagem 4** utilizamos a classe `AsymmetricKeyParameter`, do namespace `Org.BouncyCastle.Crypto`, como retorno do método `GetChavePublica`. Este método de auxílio vai permitir buscar o arquivo de chave pública gerado nas funcionalidades de geração de pares de chaves, já descritas, permitindo que o usuário selecione o arquivo através de uma caixa de diálogo.

Observe o uso da classe `OpenFileDialog`, nativa do .NET e encontrada no namespace `System.Windows.Forms`. Assim que o usuário seleciona o arquivo, o código lê todo seu conteúdo (texto plano), substitui os cabeçalhos (marcas) de início e fim do bloco de chave, converte o conteúdo que está em Base64 para uma matriz de bytes e gera, através da classe `PublicKeyFactory`, do namespace `Org.BouncyCastle.Security`, um objeto do tipo `AsymmetricKeyParameter` que contém os dados da chave pública.

O mesmo mecanismo pode ser observado no método para obtenção da chave privada que foi salva em arquivo e que pode ser verificado na **Listagem 5**.





```
04
05 OpenFileDialog dialog = new OpenFileDialog();
06 dialog.Filter = "Chave Privada (*.pvk)|*.pvk";
07 dialog.FilterIndex = 1;
08 if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
09 {
10     var bytesChave = Convert.FromBase64String
    (File.ReadAllText(dialog.FileName).Replace
    ("-----BEGIN RSA PRIVATE KEY-----\r\n", "").Replace
    ("-----END RSA PRIVATE KEY-----", ""));
11     AsymmetricCipherKeyPair pair;
12     using (var reader = File.OpenText(dialog.FileName))
13     {
14         pair = (AsymmetricCipherKeyPair)
            new Org.BouncyCastle.OpenSsl.PemReader(reader).ReadObject();
15         parametro = pair.Private;
16     }
17 }
18
19 return parametro;
20 }
```

Notificações :)

Listagem 5. Obtenção da Chave Privada

A diferença na **Listagem 5** é que após a carga do arquivo e a remoção das marcas de bloco da chave, ao invés de usar uma Factory para gerar um objeto do tipo `AsymmetricCipherKeyPair`, usamos a classe `PemReader`, do namespace `Org.BouncyCastle.OpenSsl` para ler o conteúdo do arquivo.

Esse Reader converte o conteúdo em Base64 para uma matriz de bytes e devolve um objeto, que por meio de um cast convertemos no objeto `AsymmetricCipherKeyPair`. Este objeto contém o par de chaves, incluindo a chave privada, que pode ser acessada usando a propriedade `Private` da classe. E esse é o parâmetro de retorno do método de obtenção da chave privada.

Com este conjunto de códigos você já pode criar um par de chaves para seu projeto.





certificados de cliente.

Com o mesmo conjunto, podemos criar a requisição de certificado da autoridade intermediária, que depois iremos assinar com a chave raiz e podemos criar a requisição do certificado de cliente, que iremos assinar com as chaves da autoridade intermediária. Tudo isto será desenvolvido no próximo artigo desta série.

Inclua como evento dos itens de menu “Autoridade Certificadora > Gerar Par de Chaves”, “Autoridade Intermediária > Gerar Par de Chaves” e “Certificado Digital Cliente > Gerar Par de Chaves” o método GerarParChaves da **Listagem 2** e você terá completado estas opções do menu.

Crie três novos formulários para permitir a geração de requisições de certificado, conforme a **Figura 13**, e inclua no evento do botão salvar o código da **Listagem 3**. Deste modo, concatenamos os exemplos de geração de chaves e criação dos certificados raiz, intermediário e de cliente.

Atente para o fato de que, quando você for gerar a requisição de certificado, você deve apontar, quando pedido pelo programa, para a chave pública e privada corretas. Ou seja, se está gerando a requisição de certificado da Autoridade Certificadora Intermediária, não deve gerar a requisição com as chaves da Autoridade Certificadora Raiz.

Isto só será feito futuramente nesta série de artigos quando a Autoridade Certificadora for Assinar o Certificado do próximo nível da cadeia e assim por diante. Continuaremos no próximo artigo a demonstrar como implementar esse processo.

Notificações :)





Públicas, do inglês PKI (Public Key Infrastructure). Demonstramos a hierarquia que pode existir numa estrutura comum de PKI, os diversos componentes, módulos, hardware e software e alguns usos para esta tecnologia.

Em nossa parte prática, demonstramos o início de nosso projeto de PKI simples, aprendemos a gerar pares de chave RSA usando a biblioteca BouncyCastle. Aprendemos a gerar requisições de certificados para posterior assinatura.

Além disso, aprendemos a instalar um certificado no Windows tanto pelo wizard do sistema operacional quanto por código. Falamos dos padrões de criptografia de chaves públicas e listamos praticamente todos os padrões (PKCS).

Nos próximos artigos, iremos gerar certificados completos e, ao final, criaremos meios de utilizar estes certificados para proteção e autenticação. Além disto, falaremos sobre como controlar o ciclo de vida de certificados.

O objetivo final desta série de artigos é permitir a criação de um programa em Windows Forms que permita funcionar como Autoridade Certificadora Raiz, Intermediária e Aplicação Cliente, expondo todos os meios de gerar, usar e validar certificados digitais. Ao final, estará habilitando o leitor a utilizar qualquer detalhe da tecnologia em qualquer projeto, seja web ou cliente/servidor.

Saiu na DevMedia!

- [JavaScript Substring - Manipulando strings](#): Aprenda neste artigo a utilizar o método `substring()` para manipulação de strings em JavaScript.
- [O que é React?](#): Neste curso aprenderemos o que é o React, um dos frameworks JavaScript que mais está em alta no mercado atualmente.





PostgreSQL.

Links

- [BouncyCastle](#)
- [RSA Labs](#)
- [ICP-BRASIL](#)
- [Certificação Digital e a Receita Federal Brasileira](#)

Notificações :)

Tecnologias:

.NET

Certificado Digital

Marcar como lido



Anotar



Por Paulo

Em 2014

RECEBA NOSSAS NOVIDADES

Informe o seu e-mail





Suporte ao aluno - Deixe a sua dúvida.

Notificações :)

ASSINATURA DEV MEDIA

Faça parte dessa comunidade 100% focada em programação e tenha acesso ilimitado. Nosso compromisso é tornar a sua experiência de estudo cada vez mais dinâmica e eficiente. Portanto, se você quer programar de verdade seu lugar é aqui. Junte-se a mais de...

+ 800 MIL
PROGRAMADORES

69,90*
/ MÊS

Séries

Projetos completos

Cursos

Guias de carreiras

DevCasts



7



Assine

A assinatura é cobrada através do seu cartão de crédito. *Tempo mínimo de assinatura: 12 meses.

Notificações :)



Plataforma para Programadores

Revistas

Baixe o App

Fale conosco

Trabalhe conosco

Assinatura Empresarial



Hospedagem web por Porta 80 Web Hosting



7