Universidade Lusófona de Humanidades e Tecnologias

"Aprovação Electrónica de Processos através do Cartão de Cidadão"

Relatório de Trabalho Final de Curso

do Curso de

Licenciatura em Engenharia Informática (LEI)

Ano Lectivo 2009 / 2010

Projecto realizado sob a orientação de:

Nuno Caneco – Create IT

André Silva – Create IT

Agradecimentos

Este trabalho não fica completo sem antes agradecer a todos aqueles que nos ajudaram a concretizá-lo.

Em primeiro lugar queremos agradecer aos nossos orientadores, Nuno Caneco e André Silva pela sua orientação, disponibilidade e paciência. Aprendemos bastante com eles e tivemos um acompanhamento excelente. Trabalhámos a um nível bastante superior ao exigido até agora, o que se traduz numa evolução clara da nossa forma de encarar os trabalhos futuros.

De seguida queremos agradecer à Create IT e a todas as pessoas que formam a empresa e que tornaram possível este projecto.

Por último, mas não menos importante, queremos agradecer à professora Inês Oliveira pelo seu apoio e disponibilidade.

Esperamos ter estado à altura do trabalho que nos foi solicitado.

Índice Geral

Índice de Figuras

Índice de Tabelas

Glossário

SDK – *Software Development Kit* (Kit de Desenvolvimento de Software)

Resumo

Este documento é um relatório do trabalho final de curso e tem por objectivo dar a conhecer o âmbito do trabalho, toda a pesquisa necessária, a sua implementação, as suas conclusões e alguns melhoramentos futuros que se traduzem numa evolução natural do projecto.

Como o título indica, **Aprovação Electrónica de Processos através do Cartão de Cidadão**, o trabalho tem como base duas áreas, primeiro a aprovação electrónica de processos, que nada mais é que a assinatura digital de documentos e a segunda o cartão de cidadão. O cartão de cidadão, em Portugal, é utilizado como cartão de identificação pessoal ao nível de várias entidades governamentais e possui tecnologia para uma identificação e assinatura digitais.

Neste trabalho o utilizador usa o seu cartão de cidadão para assinar digitalmente documentos. Nesse sentido é desenvolvida uma solução, que permite ao utilizador seleccionar um ou mais formulários InfoPath para assinar, permite também verificar a validade de assinaturas nesses formulários e por último, permite consultar os dados do detentor do cartão.

Para alcançar o mencionado anteriormente, foi feita em primeiro lugar uma pesquisa que abrangeu temas como, a assinatura digital, a assinatura XML, a criptografia e os seus vários modelos, certificados digitais e InfoPath. Na prática essa pesquisa traduz-se nas funcionalidades implementadas pelas aplicações desenvolvidas.

Abstract

This document is a final course project report and aims to create awareness of the scope of the work, all the necessary research, its implementation, its conclusions and some future improvements.

As the title indicates, **Electronic Approval Process through the Citizen Card**, the work is based on the document digital signing and the citizen card. In Portugal the citizen card is used as an identity card at various government entities, and possesses digital signature and identification technology. This technology grants the user the possibility to use he's card to digitally sign documents. In this sense a solution is developed, which allows the user to select one or more InfoPath forms to sign, also allows to verify the validity of signatures on these forms and finally, to consult data from the cardholder.

To achieve the previously mentioned, it was done in first place a survey which covered topics such as the digital signature, the XML signature, cryptography and its various models, digital certificates and InfoPath. Actually this research is reflected in the features implemented by the applications developed.

1 Introdução

A implementação generalizada do cartão de cidadão representa uma grande oportunidade na simplificação e uniformização da autenticação de cidadãos nacionais perante aplicações do Estado ou de terceiros. Este cartão, entre outras funcionalidades, possibilita a assinatura digital de documentos fazendo para tal uso de uma chave criptográfica de assinatura armazenada no próprio cartão e protegida por um PIN.

Neste trabalho são implementadas aplicações que fazem a leitura do cartão do cidadão, através de um leitor de cartões, e daí extraem o certificado de assinatura digital que depois é utilizado para assinar formulários InfoPath. É também implementada uma aplicação que faz a verificação de assinaturas. Ambas as aplicações também permitem que o utilizador faça a leitura do cartão e veja os dados nele contidos, tais como o nome, nacionalidade ou a fotografia do detentor do cartão.

1.1 Objectivo do trabalho

Este trabalho tem como objectivo o desenvolvimento de aplicações que permitem ao utilizador assinar, de forma compatível com o InfoPath, um conjunto de documentos em formato digital, para posterior integração num contexto de Microsoft SharePoint Server. Essas aplicações foram desenvolvidas utilizando a linguagem de programação C#.

1.2 Descrição do problema

Com o desenvolvimento deste projecto é criada uma solução que lê o cartão de cidadão, para obter os seus dados, mais especificamente o seu certificado de assinatura digital que depois é utilizado para assinar digitalmente ficheiros XML. Após a leitura do cartão, a solução permite listar os ficheiros XML contidos numa pasta seleccionada pelo utilizador, para que este marque um ou mais ficheiros a assinar.

Assim sendo o utilizador apenas tem que executar dois passos: o primeiro é colocar o seu cartão de cidadão no leitor e o segundo é escolher quais os documentos que deseja assinar.

É incluída ainda uma solução que faz verificação de assinaturas, que, conjuntamente com a solução anterior, responde ao problema deste trabalho final de curso.

1.3 Cartão de Cidadão

O Cartão de Cidadão é o novo cartão de identificação dos cidadãos nacionais que veio revolucionar a forma como o cidadão se pode relacionar com diversas Entidades (Públicas ou Privadas). Mais do que um documento de identificação físico, o Cartão apresenta-se como um documento electrónico, que possibilita a realização de várias operações sem necessidade de interacção presencial.

Como documento físico, permite ao cidadão identificar-se presencialmente de forma segura**.**

Uma das grandes vantagens deste novo cartão de identificação é a integração num só documento, do bilhete de identidade, do cartão de contribuinte, do cartão de eleitor, do cartão do serviço nacional de saúde e do cartão da segurança social.

Como documento digital, o cartão de cidadão permite ao respectivo titular provar a sua identidade perante terceiros através de autenticação electrónica e permite ainda ao respectivo titular autenticar de forma unívoca através de uma assinatura electrónica qualificada a sua qualidade de autor de um documento electrónico.

Do ponto de vista físico, o cartão de cidadão tem um formato de *smart card* e substitui todos os documentos mencionados anteriormente.

Do ponto de vista electrónico, possui um chip que contém:

- Certificado para autenticação segura
- Certificado qualificado para assinatura electrónica
- Dados biométricos do cidadão (impressões digitais)
- Morada
- Data de emissão
- Bloco de notas de leitura pública onde o titular, e apenas este, pode arquivar notas pessoais
- Aplicações informáticas necessárias ao desempenho das funcionalidades do Cartão de Cidadão e à sua gestão e segurança

De referir que a assinatura do cidadão não está contida no chip.

O Cartão de Cidadão, para além de ser um documento físico de identificação múltipla, é também um documento de identificação digital pois permite ao seu titular provar a sua identidade perante terceiros através de autenticação electrónica. Esta prova de identidade também é extensível à assinatura digital de documentos, que para isso faz uso de Certificados Digitais.

2 Fundamentos teóricos

2.1 Assinatura digital

A assinatura digital é uma forma de autenticar informação digital e é obtida através de um conjunto de procedimentos matemáticos realizados com a utilização de técnicas de criptografia. É equivalente à assinatura manual tradicional e mais difícil de forjar, e tem como objectivo autenticar a identidade do remetente de uma mensagem ou o signatário de um documento, garantindo também que o conteúdo original da mensagem ou documento que foi enviado mantém-se inalterado.

A capacidade de assegurar que a mensagem original assinada chegou ao destino, significa que o remetente não pode facilmente repudiá-la mais tarde.

As características mencionadas anteriormente fazem com que as assinaturas digitais sejam vulgarmente utilizadas em distribuições de software, transacções financeiras ou em outros cenários onde seja importante evitar falsificações e adulterações.

As assinaturas digitais fazem uso de criptografia simétrica ou criptografia assimétrica.

A figura seguinte ilustra a criação de uma assinatura digital e posterior verificação, baseados no processo criptográfico assimétrico.

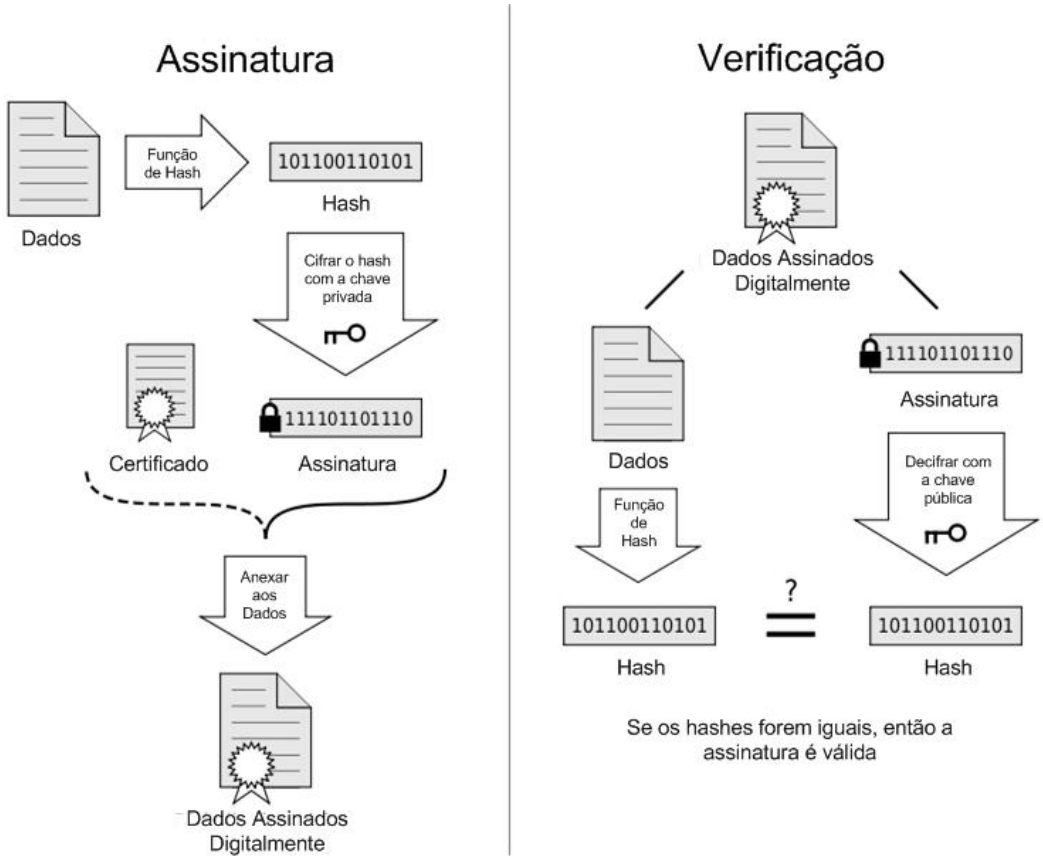


Figura 1: Assinatura Digital (Wikipédia, 2010)

O primeiro processo é o da assinatura e segue os seguintes passos:

- Cálculo do *hash* da mensagem
- Cifrar esse mesmo *hash* com a chave privada
- Por último é criado o "objecto" (Dados Assinados Digitalmente) que contém a mensagem original, o *hash* previamente calculado e a informação do certificado usado para cifrar o *hash*

No segundo processo, a verificação da assinatura digital, é utilizado o "objecto" (Dados Assinados Digitalmente) e são executados três passos:

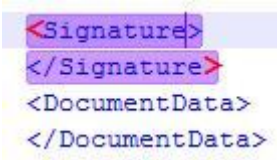
- Cálculo do *hash* da mensagem original
- Decifrar esse mesmo *hash* com a chave pública
- Comparação destes dois *hashes*. Se os *hashes* forem iguais, então a assinatura é válida.

2.1.1 Assinatura XML

A assinatura XML, que também pode ser chamada de XMLDSig, XML-DSig ou XML-Sig, especifica uma sintaxe XML para assinaturas digitais e é definida pela recomendação da W3C, em http://www.w3.org/TR/xmldsig-core. É usada por diversas tecnologias da Web como SOAP, SAML ou outros.

Existem três técnicas para assinaturas XML, detalhadas de seguida:

(i) Quando uma assinatura XML é utilizada para assinar quaisquer dados exteriores ao seu ficheiro XML, falamos numa *detached signature*, cuja estrutura se assemelha à seguinte:



Em assinaturas do tipo *enveloping* ou *enveloped* os dados a assinar fazem parte do mesmo documento XML que a assinatura.

(ii) Se a assinatura contém os dados assinados dentro de si mesma é chamada de *enveloping signature* e a sua estrutura é a seguinte:

```
<Signature>
    <DocumentData>
    </DocumentData>
</Signature>
```

(iii) Neste trabalho, é utilizado o modelo de assinatura *enveloped signature*, no qual a assinatura está contida dentro dos dados que estão a ser assinados e cuja estrutura pode ser observada na figura seguinte:

```
<DocumentData>
    <Signature>
    </Signature>
</DocumentData>
```

Na prática uma assinatura XML consiste num elemento **Signature** que está de acordo com o *namespace* http://www.w3.org/2000/09/xmldsig# e que tem a seguinte estrutura:

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference>
            <Transforms>
            </Transforms>
            <DigestMethod />
            <DigestValue />
        </Reference>
        <Reference>
            <DigestMethod />
            <DigestValue />
        </Reference>
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object>
    </Object>
</Signature>
```

Figura 2: Estrutura Assinatura XML

O elemento **SignedInfo** contém ou faz referência aos dados que são assinados e especifica quais os algoritmos que são utilizados neste processo.

O **CanonicalizationMethod** define qual o algoritmo que é usado para canonicalizar o elemento **SignedInfo**, ou seja, reorganizar as suas diversas partes utilizando um processo que não altera o seu significado.

O **SignatureMethod** define qual o algoritmo que é usado para converter o **SignedInfo** canonicalizado no elemento **SignatureValue** e para isso utiliza uma combinação de dois algoritmos, um algoritmo de *digest* e um algoritmo de chave pública (RSA-SHA1).

Um ou mais elementos **Reference** especificam qual o recurso que está a ser assinado e incluem o **DigestMethod**, o **DigestValue** e, por vezes, **Transforms**.

Os **Transforms** são uma lista opcional de transformações aplicadas ao conteúdo apontado pela **Reference**, antes de ser aplicado o algoritmo de *hash*.

O **DigestMethod** especifica o algoritmo de *hash* usado para calcular o **DigestValue**.

O **DigestValue** contém o resultado da aplicação do algoritmo de *hash* para os recursos transformados.

O elemento **SignatureValue** contém o valor real da assinatura digital, codificada em Base64.

O elemento **KeyInfo** inclui a chave que permite validar a assinatura, geralmente sob a forma de um ou mais certificados digitais X.509.

O elemento **Object** é um elemento opcional em assinaturas XML e é utilizado para incluir *data objects*, tais como **SignatureProperties,** dentro do elemento **Signature**. O elemento **SignatureProperties** é utilizado para colocar informação adicional respeitante à criação da assinatura. Como exemplo de informação adicional inclui-se um comentário, um selo data/hora da assinatura ou dados do hardware ou software usado.

2.2 Criptografia

Criptografia (do grego *kryptós*, "escondido", e *gráphein*, "escrita") é o estudo dos princípios e técnicas pelas quais a informação pode ser transformada da sua forma original para outra ilegível, de forma que possa ser conhecida apenas pelo seu destinatário (detentor da "chave"), o que a torna difícil de ser lida por alguém não autorizado. Assim sendo, só o receptor da mensagem pode ler a informação com facilidade.

Actualmente grande parte dos dados estão em formato digital, sendo representados por bits, assim sendo o processo de criptografia é assegurado por algoritmos que fazem a cifra dos bits desses dados a partir de uma determinada chave ou par de chaves, dependendo do sistema criptográfico escolhido. (Wikipédia, 2010)

Assim, criptografia refere-se ao processo de converter uma informação comum (texto claro ou aberto) numa sequência de bits ilegível, à qual se chama texto cifrado. A operação

inversa de converter uma dada informação ilegível em texto claro ou aberto tem o nome de decifra.

A criptografia tem quatro objectivos principais:

- ***Confidencialidade***: só o destinatário autorizado deve ser capaz de extrair o conteúdo da mensagem da sua forma cifrada.
- ***Integridade***: o destinatário deverá ser capaz de determinar se a mensagem foi alterada durante a transmissão.
- ***Autenticidade***: o destinatário deverá ser capaz de identificar o remetente e verificar que foi mesmo ele quem enviou a mensagem.
- ***Não-repúdio***: não deverá ser possível ao emissor negar a autoria da mensagem.

Na prática, juntamente com os algoritmos utilizam-se "chaves", mesmo que os algoritmos sejam conhecidos é necessária a "chave" correcta. Uma "chave" criptográfica nada mais é que um valor secreto que modifica um algoritmo de cifrar.

Em geral existem dois tipos de criptografia: (i) a criptografia simétrica utiliza uma única chave que serve para cifrar e para decifrar e; (ii) a criptografia assimétrica que utiliza uma chave para cifrar e outra chave para decifrar.

2.2.1 Criptografia simétrica

A criptografia simétrica é um método de criptografia que utiliza uma única chave partilhada por ambos os interlocutores, na premissa de que esta é conhecida apenas por eles. Esta partilha traduz-se na desvantagem dos algoritmos de chave simétrica, pois exige uma chave secreta partilhada, com uma cópia em cada extremidade. As chaves estão sujeitas à potencial descoberta por um adversário criptográfico, por isso necessitam ser mudadas frequentemente e mantidas seguras durante a sua distribuição e utilização.

Exemplos de algoritmos de criptografia simétrica:

- DES (*Data Encryption Standard*)
- IDEA (*International Data Encryption Algorithm*)
- AES (*Advanced Encryption Standard*)

No seguinte exemplo está representado o funcionamento de um algoritmo de criptografia simétrica, no qual a mensagem é cifrada e decifrada com a mesma chave.

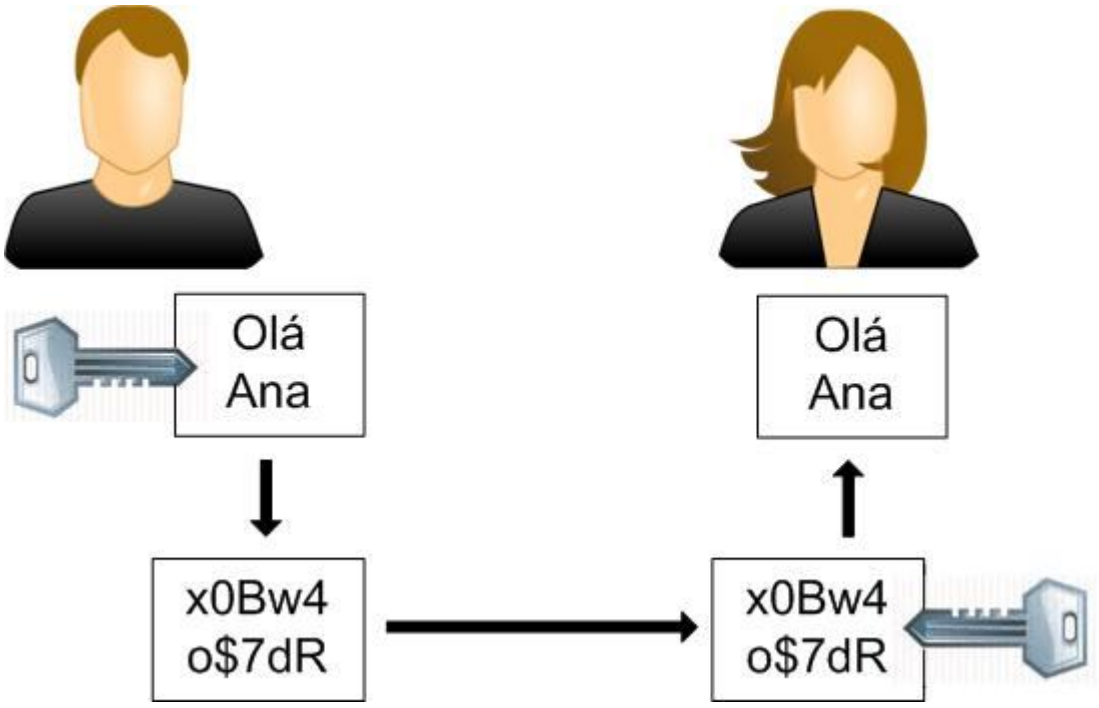


Figura 3: Exemplo de criptografia simétrica

2.2.2 Criptografia assimétrica

A criptografia assimétrica, ou criptografia de chave pública, é um método de criptografia que utiliza um par de chaves, uma chave pública e uma chave privada. A chave pública está acessível a todos enquanto a chave privada, como o nome indica, deve permanecer acessível apenas ao seu titular. Apesar de resolver definitivamente o problema da partilha da uma chave única presente no método de criptografia simétrico, a criptografia assimétrica cria um outro desafio, o da distribuição de chaves públicas, no entanto facilita significativamente a implementação de mecanismos de autenticação de mensagens e assinatura digital.

Neste tipo de criptografia, uma mensagem é cifrada com uma das chaves e para se decifrar essa mesma mensagem é necessário usar a chave complementar.

Com criptografia assimétrica é possível garantir **autenticidade** e **confidencialidade**. No primeiro caso, é utilizada a chave privada para assinar a mensagem enquanto para decifrar é usada a chave pública. Já no segundo caso, é usada a chave pública para cifrar e a chave privada para decifrar.

O exemplo seguinte mostra o funcionamento de um algoritmo de criptografia assimétrica que garante confidencialidade e está dividido em dois passos, no primeiro passo a Ana envia a sua chave pública ao João. No segundo passo o João cifra a mensagem com a chave pública da Ana e envia para a Ana, que a recebe e a decifra com a sua chave privada.

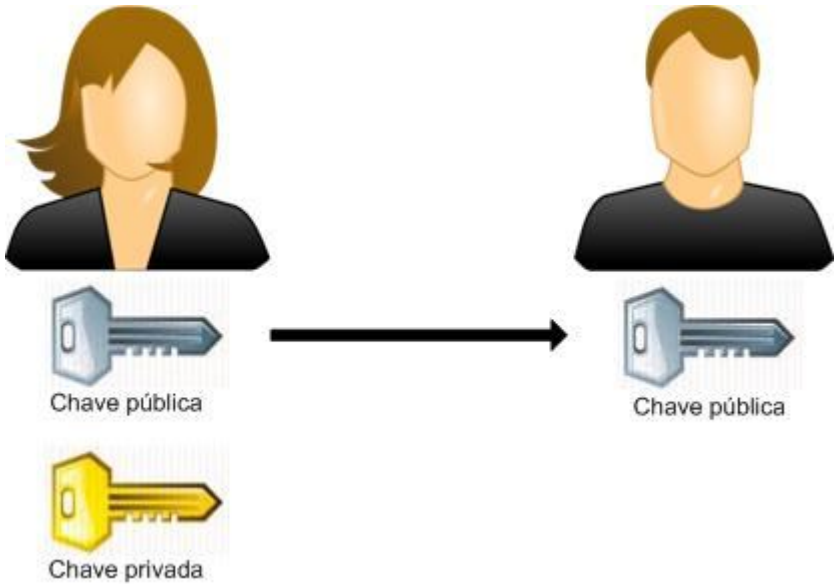


Figura 4: Passo 1 - Exemplo de criptografia assimétrica (Confidencialidade)

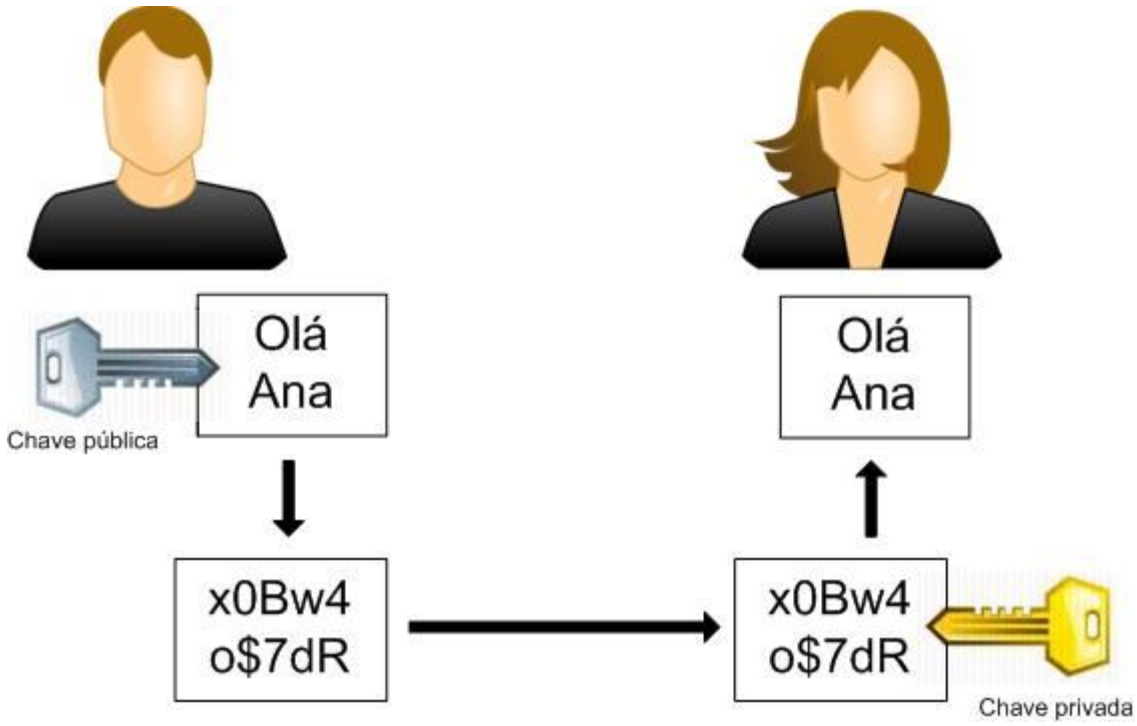


Figura 5: Passo 2 - Exemplo de criptografia assimétrica (Confidencialidade)

Este outro exemplo diverge do anterior ao garantir autenticidade ao invés de confidencialidade. Neste caso a Ana cifra a mensagem com a sua chave privada e de seguida envia a mensagem cifrada e a sua chave pública para o João, que as recebe e decifra com a chave pública da Ana.

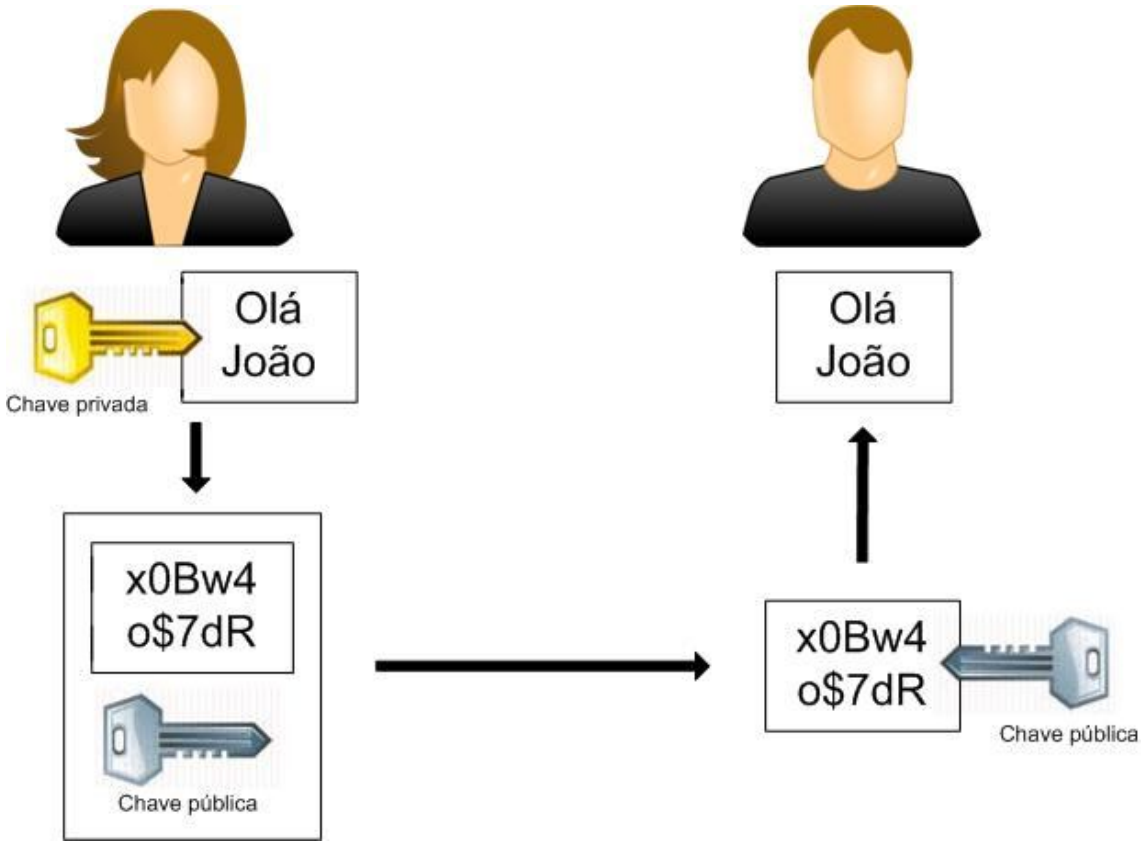


Figura 6: Exemplo de criptografia assimétrica (Autenticidade)

Exemplos de algoritmos assimétricos ou de chave pública:

- RSA (Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman)
- Diffie-Hellman

De referir ainda que a criptografia de chave pública ou assimétrica é a utilizada neste projecto.

2.2.3 *Hash* (*Digest*)

Um *hash* é uma sequência de bits gerados por um algoritmo de *hashing*. Essa sequência de bits é encontrada através de um método que transforma uma grande quantidade de informação numa pequena quantidade de informação de tal forma que esta seja (quase) exclusiva. Além disso é garantido, que não é possível a partir de um *hash* retornar à informação original e que dado um mesmo *input* é sempre retornado o mesmo *hash*, independentemente da data, hora ou mesmo computador em que este seja calculado.

Funções criptográficas de *hash* são o terceiro tipo de algoritmo de criptografia. Estes algoritmos recebem uma mensagem de qualquer comprimento como *input* e devolvem como

output, uma pequena sequência de bits de tamanho fixo. Consideram-se boas funções de *hash* aquelas que não produzam o mesmo *hash* para duas mensagens diferentes.

Existem vários algoritmos de *hash*, no entanto destacam-se dois, o MD5 (***M**essage-**D**igest algorithm* **5**) desenvolvido pela RSA Data Security Inc., descrito na RFC 1321 e que calcula *digests* de 128 bits e o SHA-1 (***S**ecure **H**ash **A**lgorithm*) que faz parte de uma família desenvolvida pela National Security Agency (NSA), descrito na RFC 3174 e que calcula *digests* de 160 bits.

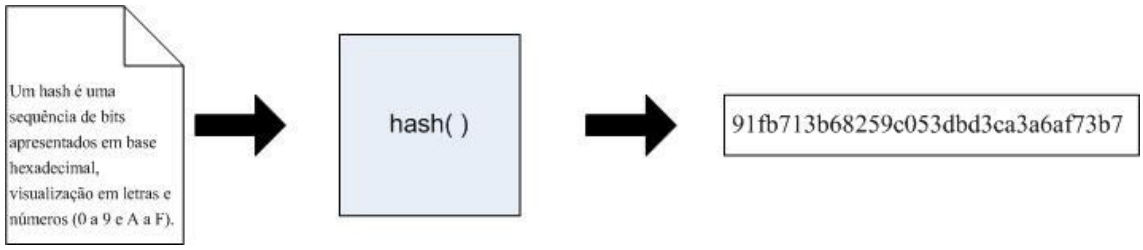


Figura 7: Exemplo de algoritmo de *hash*

2.3 *Middleware* do Cartão de Cidadão

O cartão de cidadão utiliza três APIs para o desenvolvimento de aplicações, que estão presentes no *middleware* disponibilizado:

- CryptoAPI/CSP (*Cryptographic Service Provider*) – API para operações criptográficas em ambientes Microsoft.
- PKCS#11 – API para operações criptográficas. Esta interface é usada por aplicações em ambientes não Microsoft.
- eID lib 'SDK'– API direccionada a funcionalidades não criptográficas do cartão de cidadão.

Existe também uma aplicação GUI de gestão de informação e PINs e um tray-applet na zona de notificações.

2.4 Certificados

Um certificado digital é um documento digital que comprova a identidade do seu titular, seja uma empresa, pessoa física ou computador, perante terceiros e é emitido por uma *Certification Authority* (CA) que é considerada independente, de confiança e reconhecida. Sem certificados digitais as entidades não teriam como provar quem são de forma legítima. Um certificado digital contém, entre outras informações, o nome da entidade, endereço, número de

série, chave pública, a data de validade e a assinatura digital da CA de forma a relacionar a chave pública com a identidade.

A Certificação Digital permite que informações transitem pela Internet com maior segurança assim como garante, com certeza, quem foi o autor de uma transacção ou de uma mensagem, ou, ainda, manter dados confidenciais protegidos contra a leitura por pessoas não autorizadas.

Exemplos do que pode fazer com um certificado digital:

- Assinar digitalmente qualquer ficheiro quer seja um documento, e-mail ou programa. No caso de um documento, tendo a mesma validade jurídica de um documento assinado à mão.
- Autenticação de um servidor HTTPS (Apache, IIS) por parte de um cliente.

Actualmente, o *standard* de certificados é o X.509 que especifica, entre outras coisas, formatos padrão para a chave pública de certificados, listas de revogação de certificados, os atributos dos certificados e um algoritmo de validação do caminho de certificação. Este *standard* foi definido em 1988 no âmbito da norma ITU/CCITT X.500, como refere a RFC 5280.

O Cartão de Cidadão contém dois tipos de certificados que podem ser utilizados pelo cidadão:

- **Certificado digital de Autenticação;** Permite identificar univocamente um cidadão e permite o acesso a serviços electrónicos de forma segura.
- **Certificado digital para assinatura digital qualificada;** Permite assinar documentos e ficheiros, assegurando a sua integridade e comprovando a identidade do autor.

Cada vez que é feita a leitura de um cartão de cidadão estes dois certificados são automaticamente copiados para o repositório de certificados do Windows (*Windows Certificate Store*). Noutros ambientes, onde não haja uma entidade equivalente à *Certificate Store*, os certificados não são registados e portanto, sempre que necessário, são lidos do cartão.

2.5 InfoPath

O Microsoft Office InfoPath é uma aplicação usada para o desenvolvimento de formulários baseados em XML. A forma mais comum de usar o InfoPath, é integrá-lo com a tecnologia Microsoft SharePoint através do uso de InfoPath Form Services.

A infra-estrutura das assinaturas digitais para assinar dados em formulários InfoPath segue a recomendação XML-Signature Syntax and Processing da Worldwide Web Consortium (W3C).

Uma característica importante do InfoPath é a possibilidade de assinar secções específicas do formulário e nesse caso estamos perante *co-signatures* ou *counter-signatures*. Além disso, é possível restringir o número de assinaturas a apenas uma e, no caso de o utilizador não especificar, o InfoPath por omissão assina o formulário inteiro com uma *counter-signature*.

Uma *co-signature* permite duas ou mais assinaturas paralelas e independentes sobre o mesmo conjunto de dados. A validade de qualquer *co-signature* não afecta a validade de outras *co-signatures* ou impede a adição de novas assinaturas para o mesmo conjunto de dados. Junto com cada assinatura, o InfoPath armazena alguns dados adicionais, tais como comentários e dados não-repúdio, no elemento **Signature** do documento. Se os dados assinados do formulário forem alterados, todos as *co-signatures* associadas a este conjunto de dados tornam-se inválidas. Se os dados do elemento **Signature**, tais como os dados não-repúdio, ou os comentários associados, forem adulterados, só a assinatura em causa se torna inválida. Ao utilizar *co-signatures* para um conjunto de secções específicas, qualquer assinatura pode ser apagada a qualquer momento sem afectar as outras assinaturas. Além disso, o InfoPath permite que sejam adicionadas novas assinaturas mesmo que o estado da assinatura anterior não seja válido.

Falamos de uma *counter-signature* quando os dados a assinar incluem outra assinatura previamente existente no documento. Ao adicionar a primeira assinatura a um conjunto de dados, assina-se esse mesmo conjunto de dados. Ao adicionar a segunda assinatura, não estamos a assinar os dados do formulário, mas sim a primeira assinatura. Cada *counter-signature* adicional assina a anterior. Se os dados assinados forem alterados, apenas a primeira assinatura torna-se inválida enquanto o resto das *counter-signatures* permanecem válidas. Se os dados (dentro do elemento **Signature**) de uma assinatura específica forem adulterados, tais como comentários ou informação não-repúdio, então essa assinatura e a seguinte tornam-se inválidas. Para cada conjunto específico de dados assinados, apenas a última assinatura pode ser removida sem afectar as outras e não podem ser adicionadas outras assinaturas se a última assinatura não for válida. Nesse caso essa assinatura deverá ser removida para que outras possam ser adicionadas.

De salientar que existem, como mencionado na secção 2.1.1, fundamentalmente dois tipos de assinatura em ficheiros XML, o tipo *enveloped* e o tipo *enveloping*. O InfoPath utiliza assinaturas do tipo *enveloped*.

3 Método

3.1 Solução conceptual

3.1.1 Diagrama *Sign Tool Application*

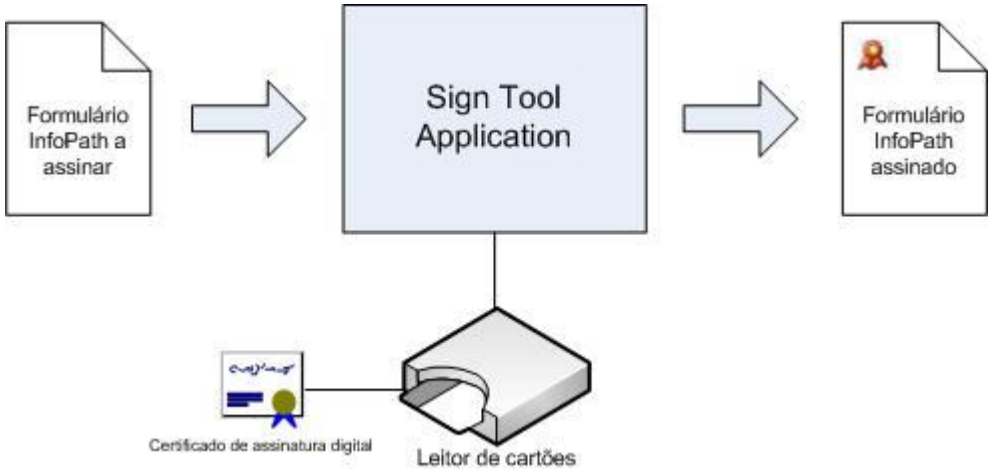


Figura 8: Diagrama *Sign Tool Application*

3.1.1.1 Descrição conceptual Sign Tool Application

A *Sign Tool Application* permite que o utilizador seleccione um ou mais formulários InfoPath para que sejam assinados digitalmente. Quando se procede à assinatura, a aplicação lê o cartão de cidadão para extrair o certificado de assinatura digital e assina os formulários InfoPath previamente seleccionados. Para cada formulário seleccionado a aplicação cria um novo ficheiro que inclui o conteúdo original do documento e a sua assinatura.

3.1.2 Diagrama *Verify Tool Application*

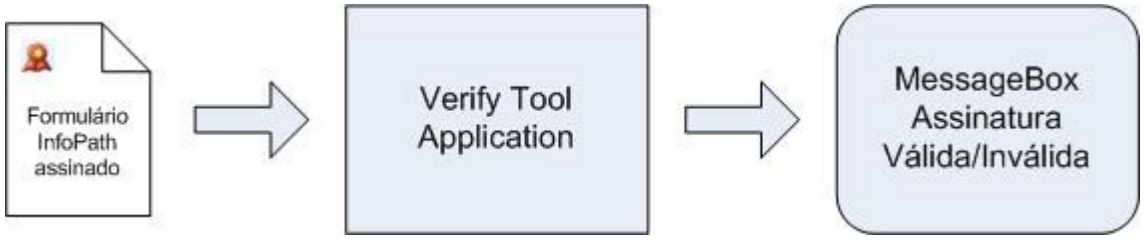


Figura 9: Diagrama *Verify Tool Application*

3.1.2.1 Descrição conceptual Verify Tool Application

A *Verify Tool Application* permite que o utilizador seleccione um ou mais formulários InfoPath assinados e faz a verificação das respectivas assinaturas. Quando se procede à

verificação, a aplicação lê o formulário assinado extraindo a informação do certificado e verifica as assinaturas dos formulários InfoPath previamente seleccionados. Após essa verificação a aplicação apresenta para cada assinatura de cada formulário, uma *message box* com o resultado de assinatura válida ou inválida.

3.2 Implementação

3.2.1 Signature Template

A solução encontrada para implementar assinaturas XML em formulários, foi criar um ficheiro XML (*Signature_Template.xml*) com a estrutura completa de uma assinatura XML e que actua como um template. Este ficheiro XML tem todos os elementos necessários a uma assinatura XML do tipo *enveloped*.

Durante a execução da aplicação, o *signature template* é carregado, para depois os seus elementos necessários serem actualizados. De realçar que existe um *template* para cada tipo de assinatura do InfoPath (*co-signature* e *counter-signature* referidos na secção *2.5*).

Quando o *signature template* se encontra totalmente preenchido, a sua estrutura - mais especificamente o elemento **Signature** - é utilizada juntamente com os dados do formulário InfoPath a assinar para criar um ficheiro XML assinado.

Para efeitos de compatibilidade com a assinatura do InfoPath, o *template* inclui ainda um elemento **Object** com a seguinte estrutura:

```
<Object>
    <SignatureProperties>
        <SignatureProperty Id="MyComment_999" Target="#MySignature_999">
            <Comment xmlns="http://schemas.microsoft.com/office/infopath/2003/SignatureProperties">Comment</Comment>
            <sp:NonRepudiation xmlns:sp="http://schemas.microsoft.com/office/infopath/2003/SignatureProperties">
                <sp:UntrustedSystemDateTime>9999-99-99T99:99:99Z</sp:UntrustedSystemDateTime>
                <sp:SystemInformation>
                    <sp:OperatingSystem>99</sp:OperatingSystem>
                    <sp:Office>99</sp:Office>
                    <sp:InfoPath>99</sp:InfoPath>
                </sp:SystemInformation>
                <sp:ScreenInformation>
                    <sp:NrOfMonitors>99</sp:NrOfMonitors>
                    <sp:PrimaryMonitor>
                        <sp:Width Unit="px">99</sp:Width>
                        <sp:Height Unit="px">99</sp:Height>
                        <sp:ColorDepth Unit="bpp">99</sp:ColorDepth>
                    </sp:PrimaryMonitor>
                </sp:ScreenInformation>
                <sp:SolutionInformation>
                    <sp:SolutionFingerprint></sp:SolutionFingerprint>
                    <sp:CurrentView>View 1</sp:CurrentView>
                </sp:SolutionInformation>
                <sp:ScreenDumpPNG></sp:ScreenDumpPNG>
            </sp:NonRepudiation>
        </SignatureProperty>
    </SignatureProperties>
</Object>
```

Figura 10: Elemento *Object* estrutura assinatura XML

3.2.2 DLLs utilizadas

- System.Security.dll

A System.Security.dll garante acesso a diversos *namespaces* que por sua vez incluem várias classes, entre as quais se encontram as classes relacionadas com criptografia XML. Um dos *namespaces* existentes na System.Security.dll é o System.Security.Cryptography que permite gerir detalhes da criptografia em geral, sendo alguns desses detalhes referentes à Microsoft Cryptography API (CryptoAPI). Os *namespaces* utilizados na solução são:

- O System.Security.Cryptography.X509Certificates fornece suporte para certificados digitais X509 v3, que são assinados com uma chave privada única e que, assim, identificam o seu titular.
- O System.Security.Cryptography.Xml contém classes para suportar a criação e validação de assinaturas digitais XML. Essas classes implementam a recomendação XML-Signature Syntax and Processing da Worldwide Web Consortium, descrita em http://www.w3.org/TR/xmldsig-core.

- System.Xml.dll

A System.Xml.dll, cujo *namespace* mais importante é o System.Xml que fornece suporte baseado em padrões para processamento XML. As suas classes permitem fundamentalmente leitura e escrita de XML.

- EIDPT.dll

A dll **eIDPT**, é a implementação da API eID do cartão de cidadão português, e conjugada com a aplicação *middleware* do cartão de cidadão, permite comunicar com o cartão e assim consultar os seus dados. É possível aceder aos dados do titular do cartão, tais como o nome e apelido, data de nascimento, número de identificação fiscal ou número de segurança social.

É importante referir que quando se introduz o cartão de cidadão pela primeira vez no leitor, os certificados presentes no cartão são copiados para a *Windows Certificate Store*, que é um repositório de certificados do Windows.

Não menos importante, referir também que não é possível, através de código, aceder directamente ao cartão e extrair a chave privada do certificado. Para ler e extrair o certificado de assinatura digital presente no cartão, procede-se da seguinte forma:

1. É feita a leitura do cartão e acede-se aos seus certificados
2. Escolhe-se o certificado desejado
3. De seguida é utilizado o nome do certificado escolhido, para fazer uma pesquisa na *Windows Certificate Store*
4. Quando é encontrado o certificado, é possível fazer uso dele para assinar digitalmente formulários InfoPath

- CSJ2K.dll

Da mesma forma que é possível aceder aos dados do titular do cartão, quando se procede à leitura do mesmo, também é possível obter a fotografia do titular do cartão. Para isso é necessário usar uma outra dll, a **CSJ2K**, que contém uma API simplificada para carregar e guardar imagens JPEG a partir de ficheiros e *streams*. Durante a execução da aplicação, é efectuada a leitura do cartão e é extraída a fotografia do seu titular para um byte[] que depois é convertido numa imagem para ser apresentada.

3.3 Arquitectura da solução

3.3.1 Esquema técnico *Sign Tool Application*

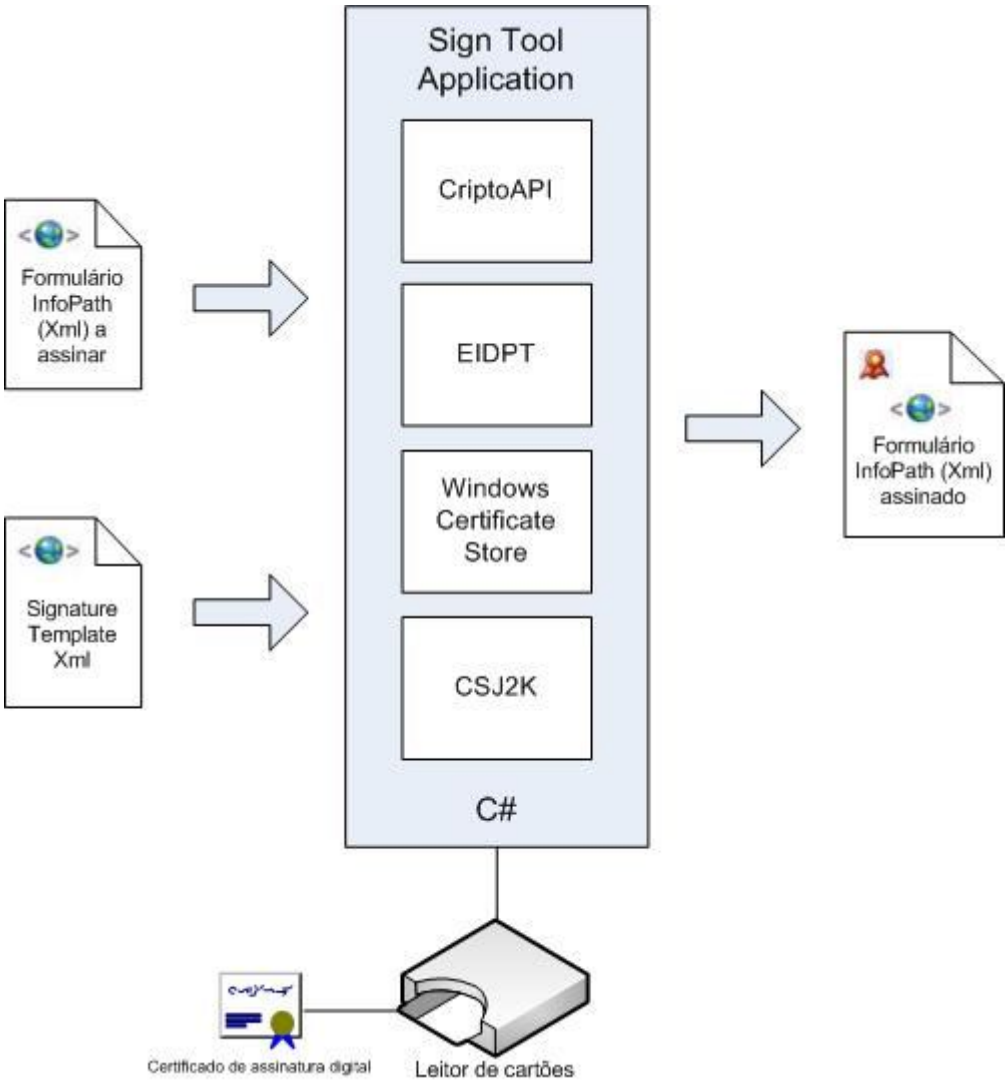


Figura 11: Esquema técnico *Sign Tool Application*

3.3.2 Esquema técnico *Verify Tool Application*

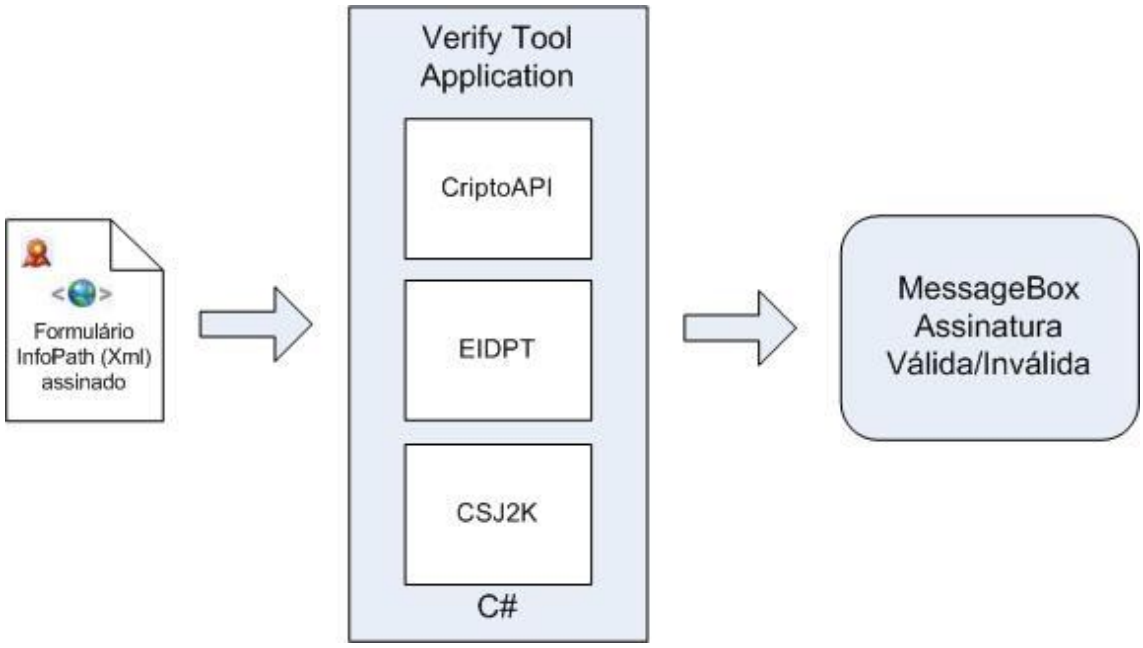


Figura 12: Esquema técnico *Verify Tool Application*

4 Resultados

4.1 Grelha de testes

Durante o desenvolvimento das diversas aplicações, surgiu a necessidade de realizar vários testes no sentido de melhor mapear a evolução do projecto e o comportamento das aplicações. Assim as seguintes tabelas foram sendo preenchidas de acordo com o sucesso ou insucesso da aplicação em cumprir determinada funcionalidade.

De realçar que de todos os testes realizados, apenas um não foi bem sucedido, o adicionar múltiplas *counter-signatures* num só formulário com a *Sign Tool*.

	Uma assinatura	Múltiplas assinaturas	Verificação com InfoPath	Verificação com Tool	Formulário InfoPath em qualquer linguagem
Sign Tool Co-Signature	✓	✓	✓	✓	✓
Sign Tool Counter-Signature	✓	✗	✓	✓	✓

Tabela 1: Testes *Sign Tools*

	Verificação de vários formulários	Verificação de várias assinaturas	Verificação de formulários em qualquer linguagem	Verificação de vários formulários com várias assinaturas
Verify Tool	✓	✓	✓	✓

Tabela 2: Testes *Verify Tool*

5 Conclusões

5.1 Outros trabalhos realizados

No decorrer deste projecto, foram criadas outras aplicações que mostram os passos intermédios que foram dados no sentido de alcançar o objectivo final deste trabalho.

Nesse sentido, o primeiro conjunto de aplicações que foi criado tinha como finalidade assinar digitalmente qualquer ficheiro existente no computador. Essa aplicação utiliza a função de *hash* **MD5** para calcular os *hashes* e a classe **RSACryptoServiceProvider** para gerar o conjunto de chaves assimétricas utilizadas para assinar digitalmente os ficheiros. Junto com essa aplicação de assinaturas foi criada a respectiva aplicação de verificação.

Foi também criado um segundo conjunto de aplicações que tinha como finalidade assinar digitalmente apenas ficheiros XML, ainda sem qualquer requisito de compatibilidade com o InfoPath. Este conjunto de aplicações funciona com múltiplos ficheiros, ou seja, no caso da aplicação de assinar, podiam ser assinados vários ficheiros em simultâneo e no caso da aplicação de verificar, podiam ser verificados vários ficheiros em simultâneo.

5.2 Eficiência da abordagem ao problema

A solução implementada foi o *signature template* mencionado na secção 3.2.1. É uma solução que facilita a manutenção futura pois no caso de a assinatura InfoPath ser alterada é mais fácil fazer a alteração no *template*, mas em contra partida é uma solução que levanta potenciais problemas de segurança pois o *template* é um ficheiro aberto que pode ser corrompido pondo assim em causa o funcionamento da aplicação. Para solucionar este problema de integridade do *template*, este podia ser assinado e assim garantir que não se está a usar um ficheiro corrompido.

5.3 Trabalhos futuros

Com o trabalho desenvolvido até este ponto, seria interessante estender este projecto de modo que tenha mais utilidade e mais funcionalidades.

A funcionalidade mais relevante a implementar de futuro seria a integração com Microsoft SharePoint, visto que iria permitir a partilha de ficheiros, gestão de documentos e publicação de relatórios para ajudar todos a tomar melhores decisões.

Outro trabalho futuro a considerar, seria a implementação de mecanismos de gestão de certificados revogados. No caso da *Sign Tool Application* essa implementação passaria por

verificar no Cartão de Cidadão se o certificado está ou não revogado, enquanto no caso da *Verify Tool Application* seria necessário verificar na *Windows Certificate Store* se o certificado está ou não válido.

Por último, seria uma mais-valia para o projecto a criação de "installers" de forma a facilitar a distribuição das aplicações.

6 Bibliografia

6.1 URLs

6.1.1 Suporte teórico

http://pt.wikipedia.org

http://en.wikipedia.org

http://www.cartaodecidadao.pt/

http://msdn.microsoft.com/en-us/library/dd963876(v=office.12).aspx

http://msdn.microsoft.com/en-us/library/cc313058(v=office.12).aspx

http://cartaodecidadao.codeplex.com/

http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/Overview.html

http://www.w3.org/TR/2001/REC-xml-c14n-20010315

http://www.w3.org/TR/1999/REC-xslt-19991116

http://www.w3.org/1999/XSL/Transform

http://www.w3.org/TR/xpath20/

http://msdn.microsoft.com/en-us/library/bb250992%28office.12%29.aspx

http://msdn.microsoft.com/en-us/library/yxw286t2.aspx

http://www.w3.org/TR/xmldsig-core/

http://www.kitcc.pt/ccidadao/kits

6.1.2 Suporte técnico

http://msdn.microsoft.com/en-us/library/bb608325%28office.11%29.aspx

http://blogs.msdn.com/b/shawnfa/archive/2005/11/03/488807.aspx

http://blogs.msdn.com/b/shawnfa/archive/2004/03/31/105137.aspx

http://msdn.microsoft.com/en-us/library/ms229745.aspx

http://msdn.microsoft.com/en-us/library/ms229950.aspx

http://msdn.microsoft.com/en-us/library/system.xml.aspx

http://msdn.microsoft.com/en-us/library/system.security.cryptography.x509certificates.aspx

http://msdn.microsoft.com/en-us/library/system.security.cryptography.xml.aspx

http://msdn.microsoft.com/en-us/library/as0w18af.aspx

http://www.infopathdev.com/

http://stackoverflow.com

6.2 RFCs

http://www.ietf.org/rfc/rfc3275.txt

http://www.ietf.org/rfc/rfc2807.txt

http://www.rfc-editor.org/rfc/rfc3076.txt

http://www.ietf.org/rfc/rfc1321.txt

http://www.faqs.org/rfcs/rfc3174.html

http://www.ietf.org/rfc/rfc5280.txt

6.3 Referências

Wikipédia. (4 de Setembro de 2010). *Criptografia*. Obtido de Wikipédia: http://pt.wikipedia.org/wiki/Criptografia

Wikipédia. (13 de Setembro de 2010). *Digital signature*. Obtido de Wikipédia: http://en.wikipedia.org/wiki/Digital_signature

7 Anexos

7.1 Anexo A - *Source code* das aplicações desenvolvidas

7.1.1 InfoPath Sign Tool Co-Signature

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using DevScope.CartaoDeCidadao;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Xml;
using System.Security.Cryptography.Xml;
using System.Configuration;
using Microsoft.Win32;



namespace InfoPath_Sign_Tool_Co_Signature
{
   public partial class Form1 : Form
   {
      public String[] filePaths;


      public Form1()
      {
         InitializeComponent();
      }


      private void ButtonReadCard_Click(object sender, EventArgs e)
      {
         try
         {
            // Initialize the EIDPT and check if there is one card in card reader
            EIDPT.Init("");

            // Verify if card is activated
            if (EIDPT.IsActivated() != CardActivationStatus.ACTIVE)
            {
               MessageBox.Show("Cartão não ativo");
            }

            // SODChecking = to verify the validity of the card data
            EIDPT.SetSODChecking(false);

            // Read data from card
```

```
            Id citizen = EIDPT.GetID();
            Picture photo = EIDPT.GetPicture();

            // Convert photo byte array in memory stream
            System.IO.MemoryStream ms = new System.IO.MemoryStream(photo.Bytes, 0,
photo.BytesLength, false);
            Image tempImage = CSJ2K.J2kImage.FromStream(ms);
            ms.Close();

            pictureBoxPhoto.Image = tempImage;
            textBoxFirstName.Text = citizen.FirstName;
            textBoxName.Text = citizen.Name;
            textBoxNationality.Text = citizen.Nationality;
            textBoxExpiryDate.Text = citizen.ExpiryDate;
         }
         catch (Exception em)
         {
            MessageBox.Show(em.ToString());
         }
         finally
         {
            // Close the EIDPT and clean temporary data
            EIDPT.Exit(ExitMode.UNPOWER);
         }

      }


      private void ButtonBrowseXmlFiles_Click(object sender, EventArgs e)
      {
         // Clear the treeview
         treeView.Nodes.Clear();

         try
         {
            // Get folderBrowserDialog1 result
            DialogResult result = this.folderBrowserDialog1.ShowDialog();

            if (result == DialogResult.OK)
               // If result == OK, set textBoxXmlFiles.Text
               textBoxXmlFiles.Text = this.folderBrowserDialog1.SelectedPath;

            // String array with all xml file paths
            filePaths = Directory.GetFiles(textBoxXmlFiles.Text, "*.xml",
SearchOption.AllDirectories);

            int size = filePaths.Length;

            // Add nodes to treeView
            for (int i = 0; i < size; ++i)
            {
               // Add a root node
               treeView.Nodes.Add(filePaths[i]);
            }
         }
```

```
        catch (Exception)
        {}
    }

    private X509Certificate2 GetCertificate()
    {
        try
        {
            // Initialize the EIDPT and check if there is one card in card reader
            EIDPT.Init("");

            // Verify if card is activated
            if (EIDPT.IsActivated() != CardActivationStatus.ACTIVE)
            {
                MessageBox.Show("Cartão não ativo");
                return null;
            }

            // Get certificates from citizen card
            DevScope.CartaoDeCidadao.Certificate[] certificates = EIDPT.GetCertificates();

            // Get one certificate from certificates, position 1
            // Position 0 - Authentication Certificate
            // Position 1 - Signature Certificate
            DevScope.CartaoDeCidadao.Certificate certificate = certificates[1];

            // Create a X509Certificate2 object from certificate
            X509Certificate2 xCertificate = new X509Certificate2(certificate.Bytes);

            // Get the xCertificate name
            string xName = xCertificate.Subject.ToString();

            // Get all certificates from windows certificate store
            X509Store store = new X509Store("MY", StoreLocation.CurrentUser);

            // Set store flags
            store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);

            // Create a collection with store certificates
            X509Certificate2Collection storeCollection =
(X509Certificate2Collection)store.Certificates;

            // Create a collection with the needed certificate from storeCollection
            X509Certificate2Collection findStoreCollection =
(X509Certificate2Collection)storeCollection.Find(X509FindType.FindBySubjectDistinguished
Name, xName, false);

            if (findStoreCollection.Count == 0)
            {
                MessageBox.Show("Erro. Não foi encontrado nenhum certificado na Windows
Certificate Store");
                return null;
            }
```

```
            // Get the certificate from findStoreCollection position 0
            xCertificate = findStoreCollection[0];

            return xCertificate;
        }
        catch (Exception em)
        {
            MessageBox.Show(em.ToString());
        }
        finally
        {
            // Close the EIDPT and clean temporary data
            EIDPT.Exit(ExitMode.UNPOWER);
        }
        return null;
    }


    private bool SetAttribute(XmlDocument sigTemplate, string elementName, int
elementIndex, int attributeIndex, string value)
    {
        // Get all nodes with the name "elementName"
        XmlNodeList xmlNodeListTemplate = sigTemplate.SelectNodes("//*[local-name(.) = '"
+ elementName + "']");

        // Create a XmlNode and set it to the elementIndex position of the
xmlNodeListTemplate
        XmlNode xmlNodeSigTemplate = xmlNodeListTemplate[elementIndex];

        if (xmlNodeListTemplate != null)
        {
            // Set the value for xmlNodeSigTemplate attribute number attributeIndex
            xmlNodeSigTemplate.Attributes[attributeIndex].Value = value;

            return true;
        }
        return false;
    }


    private bool SetValue(XmlDocument sigTemplate, string elementName, int elementIndex,
string value)
    {
        // Get all nodes with the name "elementName"
        XmlNodeList xmlNodeListTemplate = sigTemplate.SelectNodes("//*[local-name(.) = '"
+ elementName + "']");

        // Create a XmlNode and set it to the elementIndex position of the
xmlNodeListTemplate
        XmlNode xmlNodeSigTemplate = xmlNodeListTemplate[elementIndex];

        // Set the value for the first child node in xmlNodeSigTemplate
        xmlNodeSigTemplate.FirstChild.Value = value;

        if (xmlNodeListTemplate != null)
```

```
        {
            // Set the value for the first child node in xmlNodeSigTemplate
            xmlNodeSigTemplate.FirstChild.Value = value;

            return true;
        }
        return false;
    }


    private void ButtonSign_Click(object sender, EventArgs e)
    {
        if (textBoxXmlFiles.Text == "")
        {
            MessageBox.Show("Erro. Primeiro tem que escolher uma pasta com o botão
browse");
            return;
        }

        int count = 0;
        String[] checkedXML = new String[filePaths.Length];
        TreeNode node;

        for (int i = 0, j = 0; i < filePaths.Length; ++i)
        {
            node = treeView.Nodes[i];

            if (node.Checked == true)
            {
                // String array with checked items names
                checkedXML[j] = node.Text;
                j++;
                // Count the checked items
                count++;
            }
        }

        if (checkedXML[0] == null)
        {
            MessageBox.Show("Erro. Tem que selecionar pelo menos um ficheiro para assinar");
            return;
        }

        // Create a X509Certificate2 object and get certificate from citizen card
        X509Certificate2 xCertificate = GetCertificate();

        // Create a XmlDocument object
        XmlDocument xmlDocument = new XmlDocument();

        for (int x = 0; x < count; x++)
        {
            // Set PreserveWhitespace true
            xmlDocument.PreserveWhitespace = true;

            // Load an XML file into the XmlDocument object
```

```
            xmlDocument.Load(checkedXML[x]);

            // Create a XmlNodeList with all "Signature" nodes in Xml file to sign
            XmlNodeList xmlNodeListDocument =
xmlDocument.GetElementsByTagName("Signature");

            // Call function CoSign()
            // If Count == 0, no Signatures were found, so this is the first Signature
            CoSign(xmlNodeListDocument.Count, xmlDocument, checkedXML[x],
xCertificate);
        }
    }


    private void CoSign(int signatureID, XmlDocument xmlDocument, string checkedXML,
X509Certificate2 xCertificate)
    {
        try
        {
            string signatures1NodeName = null;

            // If Xml file to sign already has one or more signatures
            if (signatureID > 0)
            {
                // Create a XmlNode and set it to "Signature" node of Xml file to sign
                XmlNode xmlNodeTemp = xmlDocument.SelectSingleNode("//*[local-name(.) =
'Signature']");

                // Get "my:signatures1" node name from Signature grandfather node
                signatures1NodeName = xmlNodeTemp.ParentNode.ParentNode.Name;
            }

            // Create a XmlNodeList with all Xml nodes in Xml file to sign
            XmlNodeList xmlNodeListDocument = xmlDocument.GetElementsByTagName("*",
"*");

            string myFields = null;

            // Cicle through all fields in Xml file to sign, except (Count - 2) for my:myFields,
my:signatures1 and my:signatures2
            for (int i = 0; i < xmlNodeListDocument.Count - 2; i++)
            {
                // If field name equals "my:signatures1" exit cicle
                if (xmlNodeListDocument[i].Name.Equals(signatures1NodeName))
                    break;
                // Else append "/" + field name to string
                myFields += "/" + xmlNodeListDocument[i].Name;
            }

            // Get the "my:signatures1" node index
            // Count is the number of nodes in Xml file to sign and -1 is the node before the last
one
            int mySignatures1NodeIndex = xmlNodeListDocument.Count - 1;

            // Create a XmlNode and set it to the root node of Xml file to sign
```

```
            XmlNode xmlNodeDocument = xmlNodeListDocument[0];

            string rootName = "/";

            // Append "my:myFields" to string
            rootName += xmlNodeDocument.Name;

            // If Xml file to sign already has one or more signatures
            if (signatureID > 0)
               // Append "/" + "my:signatures1" to string
               rootName += "/" + signatures1NodeName;

            // Create a XmlAttributeCollection and get all attributes from the root node of Xml
file to sign
            XmlAttributeCollection xmlNodeAttributeCollection =
xmlNodeDocument.Attributes;

            // Create a XmlAttribute and set it to first attribute (xmlns:my) of the root node of
Xml file to sign
            XmlAttribute xmlNodeAttributeDocument = xmlNodeAttributeCollection[0];

            // Create string and get the "xmlns:my=" value from root node of Xml file to sign
            string rootNodeXmlnsMy = xmlDocument.DocumentElement.NamespaceURI;

            // If Xml file to sign already has one or more signatures
            if (signatureID > 0)
            {
               // Set xmlNodeDocument to "Signature" node of Xml file to sign
               xmlNodeDocument = xmlDocument.SelectSingleNode("//*[local-name(.) =
'Signature']");

               // Set xmlNodeDocument to "my:signatures2" node of Xml file to sign
               xmlNodeDocument = xmlNodeDocument.ParentNode;
            }
            else
               // Set xmlNodeDocument to "my:signatures1" node of Xml file to sign
               xmlNodeDocument = xmlNodeListDocument[mySignatures1NodeIndex];

            // Create a string with "my:signatures2" node name (signatures2)
            string signatures2NodeName = xmlNodeDocument.Name.Substring(3,
xmlNodeDocument.Name.Length - 3);

            // If Xml file to sign has no signatures
            if (signatureID == 0)
               // Append "/" + "my:signatures1"
               rootName += "/" + xmlNodeDocument.ParentNode.Name;

            // Append "/" + "my:signatures2" + "/node()" to string
            rootName += "/" + xmlNodeDocument.Name + "/node()";

            // Create a XmlDocument for the signature template
            XmlDocument sigTemplate = new XmlDocument();

            // Load the signature template
            sigTemplate.Load("template/Signature_Template.xml");
```

```
            // Set my:myFields/my:field1/... in attribute 0 (match), element position 2, element
name "template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 2, 0, myFields))
               return;

            // Set http://schemas.microsoft.com... in attribute 3 (xmlns:my), element position 2,
element name "template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 2, 3, rootNodeXmlnsMy))
               return;

            // Set "/my:myFields/my:signatures1/my:signatures2/node()" in attribute 0 (match),
element position 4, element name "template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 4, 0, rootName))
               return;

            // Set http://schemas.microsoft.com... in attribute 3 (xmlns:my), element position 4,
element name "template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 4, 3, rootNodeXmlnsMy))
               return;

            // Set "#MyComment_" + textBoxSectionToSign.Text + "_00" + signatureID in
attribute 0 (URI), element position 1, element name "Reference" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "Reference", 1, 0, "#MyComment_" +
textBoxSectionToSign.Text + "_00" + signatureID))
               return;

            // Set "MyComment_" + textBoxSectionToSign.Text + "_00" + signatureID in
attribute 0 (Id), element position 0, element name "SignatureProperty" in XmlDocument
sigTemplate
            if (!SetAttribute(sigTemplate, "SignatureProperty", 0, 0, "MyComment_" +
textBoxSectionToSign.Text + "_00" + signatureID))
               return;

            // Set "#MySignature_" + textBoxSectionToSign.Text + "_00" + signatureID in
attribute 1 (Target), element position 0, element name "SignatureProperty" in XmlDocument
sigTemplate
            if (!SetAttribute(sigTemplate, "SignatureProperty", 0, 1, "#MySignature_" +
textBoxSectionToSign.Text + "_00" + signatureID))
               return;

            // Set textBoxComment.Text in FirstChild.Value, element position 0, element name
"Comment" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Comment", 0, textBoxComment.Text))
               return;

            // Create the Untrusted System DateTime stamp
            string timeStamp = System.DateTime.UtcNow.ToString("yyyy-MM-
ddTHH:mm:ssZ");

            // Set timeStamp in FirstChild.Value, element position 0, element name
"UntrustedSystemDateTime" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "UntrustedSystemDateTime", 0, timeStamp))
               return;
```

```
            // Set System.Environment.OSVersion.Version.ToString() in FirstChild.Value,
element position 0, element name "OperatingSystem" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "OperatingSystem", 0,
System.Environment.OSVersion.Version.ToString()))
                return;

            // Create a RegistryKey and open the HKEY_CLASSES_ROOT key in the registry of
this machine
            RegistryKey rkClassesRoot = Registry.ClassesRoot, rkCurVer;

            // Open HKEY_CLASSES_ROOT\Word.Application\CurVer key
            rkCurVer = rkClassesRoot.OpenSubKey(@"Word.Application\CurVer");

            // Get the HKEY_CLASSES_ROOT\Word.Application\CurVer key value
            string rkValue = (string)rkCurVer.GetValue("");

            // Discard all characters in string except the two last ones
            rkValue = rkValue.Substring(17, 2);

            // Set rkValue in FirstChild.Value, element position 0, element name "Office" in
XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Office", 0, rkValue))
                return;

            // Set rkValue in FirstChild.Value, element position 0, element name "InfoPath" in
XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "InfoPath", 0, rkValue))
                return;

            // Set SystemInformation.MonitorCount.ToString() in FirstChild.Value, element
position 0, element name "NrOfMonitors" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "NrOfMonitors", 0,
SystemInformation.MonitorCount.ToString()))
                return;

            // Set SystemInformation.PrimaryMonitorSize.Width.ToString() in FirstChild.Value,
element position 0, element name "Width" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Width", 0,
SystemInformation.PrimaryMonitorSize.Width.ToString()))
                return;

            // Set SystemInformation.PrimaryMonitorSize.Height.ToString() in FirstChild.Value,
element position 0, element name "Height" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Height", 0,
SystemInformation.PrimaryMonitorSize.Height.ToString()))
                return;

            // Set Screen.PrimaryScreen.BitsPerPixel.ToString() in FirstChild.Value, element
position 0, element name "ColorDepth" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "ColorDepth", 0,
Screen.PrimaryScreen.BitsPerPixel.ToString()))
                return;

            // Create a XmlNode and set it to "my:signatures2" node of Xml file to sign
```

```
            XmlNode signatureNotComputed = xmlDocument.SelectSingleNode("//*[local-
name(.) = '" + signatures2NodeName + "']");

            // Append Signature node from signature template. True means the signature node will
be
            //appended with all its descendants
            // At this point the signature node does not have any digest values, Signature value or
KeyInfo

signatureNotComputed.AppendChild(xmlDocument.ImportNode(sigTemplate.DocumentEleme
nt, true));

            // Create a SignedXml from Xml file to sign
            SignedXml signedXml = new SignedXml(xmlDocument);

            // Load the signature template into signedXml
            signedXml.LoadXml(sigTemplate.DocumentElement);

            // Set the signing key to citizens card certificate
            signedXml.SigningKey = xCertificate.PrivateKey;

            // Create KeyInfo element
            KeyInfo keyInfo = new KeyInfo();

            // Add information from citizens card certificate to KeyInfo
            keyInfo.AddClause(new KeyInfoX509Data(xCertificate));

            // Add KeyInfo to signedXml
            signedXml.KeyInfo = keyInfo;

            // Set the signature Id
            signedXml.Signature.Id = "MySignature_" + textBoxSectionToSign.Text + "_00" +
signatureID;

            // Compute the signature
            signedXml.ComputeSignature();

            // Create a XmlElement to get the XML representation of the signature
            XmlElement xmlSignature = signedXml.GetXml();

            // Create a XmlNode and append Signature node from xmlSignature. True means the
signature node will be
            //appended with all its descendants
            // At this point the signature node already have its digest values, Signature value and
KeyInfo
            XmlNode signatureComputed = xmlDocument.ImportNode(xmlSignature, true);

            // Create a XmlNode and set it to "my:signatures2" node of Xml file to sign
            XmlNode signatures2 = xmlDocument.SelectSingleNode("//*[local-name(.) = '" +
signatures2NodeName + "']");

            // If Xml file to sign has no signatures
            if (signatureID == 0)
               // Replace signatures2 first child with the computed signature node
               signatures2.ReplaceChild(signatureComputed, signatures2.FirstChild);
```

```
            else
               // Replace signatures2 last child with the computed signature node
               signatures2.ReplaceChild(signatureComputed, signatures2.LastChild);

            // Display message box asking for another signature
            DialogResult result = MessageBox.Show("Deseja adicionar outra assinatura?",
"Adicionar assinatura", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            // If user wants to add another signature to file
            if (result == DialogResult.Yes)
            {
               // Close the EIDPT and clean temporary data
               EIDPT.Exit(ExitMode.UNPOWER);

               // Insert the next citizen card message box
               MessageBox.Show("Insira o próximo cartão", "Adicionar assinatura");

               //Get certificate from citizen card
               xCertificate = GetCertificate();

               // Run CoSign method to add the next signature
               CoSign(signatureID + 1, xmlDocument, checkedXML, xCertificate);
            }

            // Create the name for the signed XML file
            string nameXML = checkedXML;
            nameXML = nameXML.Insert(checkedXML.Length - 4, "_Assinado");

            // Save the signed XML file
            xmlDocument.Save(nameXML);

            // If user does not want to add another signature to file
            if (result == DialogResult.No)
            {
               MessageBox.Show("Ficheiro XML assinado com sucesso");
            }
         }
         catch (Exception em)
         {
            MessageBox.Show(em.ToString());
         }
      }
   }
}
```

7.1.2 InfoPath Sign Tool Counter-Signature

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Windows.Forms;
using DevScope.CartaoDeCidadao;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Xml;
using System.Security.Cryptography.Xml;
using System.Configuration;
using Microsoft.Win32;


namespace InfoPath_Sign_Tool_Counter_Signature
{
   public partial class Form1 : Form
   {
      public String[] filePaths;

      public Form1()
      {
         InitializeComponent();
      }


      private void ButtonReadCard_Click(object sender, EventArgs e)
      {
         try
         {
            // Initialize the EIDPT and check if there is one card in card reader
            EIDPT.Init("");

            // Verify if card is activated
            if (EIDPT.IsActivated() != CardActivationStatus.ACTIVE)
            {
               MessageBox.Show("Cartão não ativo");
            }

            // SODChecking = to verify the validity of the cards data
            EIDPT.SetSODChecking(false);

            // Read data from card
            Id citizen = EIDPT.GetID();
            Picture photo = EIDPT.GetPicture();

            // Convert photo byte array in memory stream
            System.IO.MemoryStream ms = new System.IO.MemoryStream(photo.Bytes, 0,
photo.BytesLength, false);
            Image tempImage = CSJ2K.J2kImage.FromStream(ms);
            ms.Close();

            pictureBoxPhoto.Image = tempImage;
            textBoxFirstName.Text = citizen.FirstName;
            textBoxName.Text = citizen.Name;
            textBoxNationality.Text = citizen.Nationality;
            textBoxExpiryDate.Text = citizen.ExpiryDate;
         }
         catch (Exception em)
```

```
        {
            MessageBox.Show(em.ToString());
        }
        finally
        {
            // Close the EIDPT and clean temporary data
            EIDPT.Exit(ExitMode.UNPOWER);
        }

    }


    private void ButtonBrowseXmlFiles_Click(object sender, EventArgs e)
    {
        // Clear the treeview
        treeView.Nodes.Clear();

        try
        {
            // Get folderBrowserDialog1 result
            DialogResult result = this.folderBrowserDialog1.ShowDialog();

            if (result == DialogResult.OK)
                // If result == OK, set textBoxXmlFiles.Text
                textBoxXmlFiles.Text = this.folderBrowserDialog1.SelectedPath;

            // String array with all Xml file paths in tree view
            filePaths = Directory.GetFiles(textBoxXmlFiles.Text, "*.xml",
SearchOption.AllDirectories);

            // Add nodes to treeView
            for (int i = 0; i < filePaths.Length; ++i)
            {
                // Add a root node
                treeView.Nodes.Add(filePaths[i]);
            }
        }
        catch (Exception)
        {}
    }


    private X509Certificate2 GetCertificate()
    {
        try
        {
            // Initialize the EIDPT and check if there is one card in card reader
            EIDPT.Init("");

            // Verify if card is activated
            if (EIDPT.IsActivated() != CardActivationStatus.ACTIVE)
            {
                MessageBox.Show("Cartão não ativo");
                return null;
            }
```

```
            // Get certificates from citizen card
            DevScope.CartaoDeCidadao.Certificate[] certificates = EIDPT.GetCertificates();

            // Get one certificate from certificates, position 1
            // Position 0 - Authentication Certificate
            // Position 1 - Signature Certificate
            DevScope.CartaoDeCidadao.Certificate certificate = certificates[1];

            // Create a X509Certificate2 object from certificate
            X509Certificate2 xCertificate = new X509Certificate2(certificate.Bytes);

            // Get the xCertificate name
            string xName = xCertificate.Subject.ToString();

            // Get all certificates from windows certificate store
            X509Store store = new X509Store("MY", StoreLocation.CurrentUser);

            // Set store flags
            store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);

            // Create a collection with store certificates
            X509Certificate2Collection storeCollection =
(X509Certificate2Collection)store.Certificates;

            // Create a collection with the needed certificate from storeCollection
            X509Certificate2Collection findStoreCollection =
(X509Certificate2Collection)storeCollection.Find(X509FindType.FindBySubjectDistinguished
Name, xName, false);

            if (findStoreCollection.Count == 0)
            {
               MessageBox.Show("Erro. Não foi encontrado nenhum certificado na Windows
Certificate Store");
               return null;
            }

            // Get the certificate from findStoreCollection position 0
            xCertificate = findStoreCollection[0];

            return xCertificate;
         }
         catch (Exception em)
         {
            MessageBox.Show(em.ToString());
         }
         finally
         {
            EIDPT.Exit(ExitMode.UNPOWER);
         }
         return null;
      }
```

```
    private bool SetAttribute(XmlDocument sigTemplate, string elementName, int
elementIndex, int attributeIndex, string value)
    {
      // Get all nodes with the name "elementName"
      XmlNodeList xmlNodeListTemplate = sigTemplate.SelectNodes("//*[local-name(.) = '"
+ elementName + "']");

      // Create a XmlNode and set it to the elementIndex position of the
xmlNodeListTemplate
      XmlNode xmlNodeSigTemplate = xmlNodeListTemplate[elementIndex];

      if (xmlNodeListTemplate != null)
      {
        // Set the value for xmlNodeSigTemplate attribute number attributeIndex
        xmlNodeSigTemplate.Attributes[attributeIndex].Value = value;

        return true;
      }
      return false;
  }


    private bool SetValue(XmlDocument sigTemplate, string elementName, int elementIndex,
string value)
    {
        // Get all nodes with the name "elementName"
        XmlNodeList xmlNodeListTemplate = sigTemplate.SelectNodes("//*[local-name(.) =
'" + elementName + "']");

        // Create a XmlNode and set it to the elementIndex position of the
xmlNodeListTemplate
        XmlNode xmlNodeSigTemplate = xmlNodeListTemplate[elementIndex];

        if (xmlNodeListTemplate != null)
        {
          // Set the value for the first child node in xmlNodeSigTemplate
          xmlNodeSigTemplate.FirstChild.Value = value;

          return true;
        }
        return false;
    }


    private void ButtonSign_Click(object sender, EventArgs e)
    {
      if (textBoxXmlFiles.Text == "")
      {
        MessageBox.Show("Erro. Primeiro tem que escolher uma pasta com o botão
browse");
        return;
      }

      int count = 0;
      String[] checkedXML = new String[filePaths.Length];
```

```
        TreeNode node;

        for (int i = 0, j = 0; i < filePaths.Length; ++i)
        {
           node = treeView.Nodes[i];

           if (node.Checked == true)
           {
              // String array with checked items names
              checkedXML[j] = node.Text;
              j++;
              // Count the checked items
              count++;
           }
        }

        if (checkedXML[0] == null)
        {
           MessageBox.Show("Erro. Tem que selecionar pelo menos um ficheiro para assinar");
           return;
        }

        // Create a X509Certificate2 object and get certificate from citizen card
        X509Certificate2 xCertificate = GetCertificate();

        // Create a XmlDocument object
        XmlDocument xmlDocument = new XmlDocument();

        for (int x = 0; x < count; x++)
        {
           // Set PreserveWhitespace true
           xmlDocument.PreserveWhitespace = true;

           // Load an XML file into the XmlDocument object
           xmlDocument.Load(checkedXML[x]);

           // Call function CounterSign()
           CounterSign(xmlDocument, checkedXML[x], xCertificate);
        }
     }


     private void CounterSign(XmlDocument xmlDocument, string checkedXML,
X509Certificate2 xCertificate)
     {

        try
        {
           // Create a XmlNodeList with all nodes in Xml file to sign
           XmlNodeList xmlNodeListDocument = xmlDocument.GetElementsByTagName("*",
"*");

           // Get the "my:signatures1" node index
           // Count is the number of nodes in Xml file to sign and -1 is the node before the last
one
```

```
            int mySignatures1NodeIndex = xmlNodeListDocument.Count - 1;

            // Create a XmlNode and set it to the root node of Xml file to sign
            XmlNode xmlNodeDocument = xmlNodeListDocument[0];

            // Create a string to get the "/my:myFields/my:signatures1/node()" value
            string myFieldsSignatures1 = "/";

            // Append "my:myFields" to string
            myFieldsSignatures1 += xmlNodeDocument.Name;

            // Create a XmlAttributeCollection and get all attributes from the root node of Xml
file to sign
            XmlAttributeCollection xmlNodeAttributeCollection =
xmlNodeDocument.Attributes;

            // Create a XmlAttribute and set it to first attribute (xmlns:my) of the root node of
Xml file to sign
            XmlAttribute xmlNodeAttributeDocument = xmlNodeAttributeCollection[0];

            // Get number of Attributes in the root node of Xml file to sign
            int elementCount = xmlNodeAttributeCollection.Count;

            // Create string and get the "xmlns:my=" value from root node of Xml file to sign
            string rootNodeXmlnsMy = xmlDocument.DocumentElement.NamespaceURI;

            // Set xmlNodeDocument to "my:signatures1" node of the Xml file to sign
            xmlNodeDocument = xmlNodeListDocument[mySignatures1NodeIndex];

            // Create a string with "my:signatures1" node name (signatures1)
            string signatures1NodeName = xmlNodeDocument.Name.Substring(3,
xmlNodeDocument.Name.Length - 3);

            // Append "/" + "my:signatures1" + "/node()" to string
            myFieldsSignatures1 += "/" + xmlNodeDocument.Name + "/node()";

            // Create a XmlDocument for the signature template
            XmlDocument sigTemplate = new XmlDocument();

            // Load the signature template
            sigTemplate.Load("template/Signature_Template.xml");

            // Set myFieldsSignatures1 in attribute 0 (match), element position 1, element name
"template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 1, 0, myFieldsSignatures1))
               return;

            // Set rootNodeXmlnsMy in attribute 1 (xmlns:my), element position 1, element name
"template" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "template", 1, 1, rootNodeXmlnsMy))
               return;

            // Set "#MyComment_000" in attribute 0 (URI), element position 1, element name
"Reference" in XmlDocument sigTemplate
            if (!SetAttribute(sigTemplate, "Reference", 1, 0, "#MyComment_000"))
```

```
            return;

        // Set "MyComment_000" in attribute 0 (Id), element position 0, element name
"SignatureProperty" in XmlDocument sigTemplate
        if (!SetAttribute(sigTemplate, "SignatureProperty", 0, 0, "MyComment_000"))
            return;

        // Set "#MySignature_000" in attribute 0 (Target), element position 1, element name
"SignatureProperty" in XmlDocument sigTemplate
        if (!SetAttribute(sigTemplate, "SignatureProperty", 0, 1, "#MySignature_000"))
            return;

        // Set textBoxComment.Text in FirstChild.Value, element position 0, element name
"Comment" in XmlDocument sigTemplate
        if (!SetValue(sigTemplate, "Comment", 0, textBoxComment.Text))
            return;

        // Create the Untrusted System DateTime stamp
        string timeStamp = System.DateTime.UtcNow.ToString("yyyy-MM-
ddTHH:mm:ssZ");

        // Set timeStamp in FirstChild.Value, element position 0, element name
"UntrustedSystemDateTime" in XmlDocument sigTemplate
        if (!SetValue(sigTemplate, "UntrustedSystemDateTime", 0, timeStamp))
            return;

        // Set System.Environment.OSVersion.Version.ToString() in FirstChild.Value,
element position 0, element name "OperatingSystem" in XmlDocument sigTemplate
        if (!SetValue(sigTemplate, "OperatingSystem", 0,
System.Environment.OSVersion.Version.ToString()))
            return;

        // Create a RegistryKey and open the HKEY_CLASSES_ROOT key in the registry of
this machine
        RegistryKey rkClassesRoot = Registry.ClassesRoot, rkCurVer;

        // Open HKEY_CLASSES_ROOT\Word.Application\CurVer key
        rkCurVer = rkClassesRoot.OpenSubKey(@"Word.Application\CurVer");

        // Get the HKEY_CLASSES_ROOT\Word.Application\CurVer key value
        string rkValue = (string)rkCurVer.GetValue("");

        // Discard all characters in string except the two last ones
        rkValue = rkValue.Substring(17, 2);

        // Set rkValue in FirstChild.Value, element position 0, element name "Office" in
XmlDocument sigTemplate
        if (!SetValue(sigTemplate, "Office", 0, rkValue))
            return;

        // Set rkValue in FirstChild.Value, element position 0, element name "InfoPath" in
XmlDocument sigTemplate
        if (!SetValue(sigTemplate, "InfoPath", 0, rkValue))
            return;
```

```
            // Set SystemInformation.MonitorCount.ToString() in FirstChild.Value, element
position 0, element name "NrOfMonitors" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "NrOfMonitors", 0,
SystemInformation.MonitorCount.ToString()))
                return;

            // Set SystemInformation.PrimaryMonitorSize.Width.ToString() in FirstChild.Value,
element position 0, element name "Width" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Width", 0,
SystemInformation.PrimaryMonitorSize.Width.ToString()))
                return;

            // Set SystemInformation.PrimaryMonitorSize.Height.ToString() in FirstChild.Value,
element position 0, element name "Height" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "Height", 0,
SystemInformation.PrimaryMonitorSize.Height.ToString()))
                return;

            // Set Screen.PrimaryScreen.BitsPerPixel.ToString() in FirstChild.Value, element
position 0, element name "ColorDepth" in XmlDocument sigTemplate
            if (!SetValue(sigTemplate, "ColorDepth", 0,
Screen.PrimaryScreen.BitsPerPixel.ToString()))
                return;

            // Create a XmlNode and set it to "my:signatures1" node of Xml file to sign
            XmlNode signatureNotComputed = xmlDocument.SelectSingleNode("//*[local-
name(.) = '" + signatures1NodeName + "']");

            // Append Signature node from signature template. True means the signature node will
be
            //appended with all its descendants
            // At this point the signature node does not have any digest values, Signature value or
KeyInfo

signatureNotComputed.AppendChild(xmlDocument.ImportNode(sigTemplate.DocumentEleme
nt, true));

            // Create a SignedXml from Xml file to sign
            SignedXml signedXml = new SignedXml(xmlDocument);

            // Load the signature template into signedXml
            signedXml.LoadXml(sigTemplate.DocumentElement);

            // Set the signing key to citizens card certificate
            signedXml.SigningKey = xCertificate.PrivateKey;

            // Create KeyInfo element
            KeyInfo keyInfo = new KeyInfo();

            // Add information from citizens card certificate to KeyInfo
            keyInfo.AddClause(new KeyInfoX509Data(xCertificate));

            // Add KeyInfo to signedXml
            signedXml.KeyInfo = keyInfo;
```

```
            // Set the signature Id
            signedXml.Signature.Id = "MySignature_000";

            // Compute the signature
            signedXml.ComputeSignature();

            // Create a XmlElement to get the XML representation of the signature
            XmlElement xmlSignature = signedXml.GetXml();

            // Create a XmlNode and append Signature node from xmlSignature. True means the
signature node will be
            //appended with all its descendants
            // At this point the signature node already have its digest values, Signature value and
KeyInfo
            XmlNode signatureComputed = xmlDocument.ImportNode(xmlSignature, true);

            // Create a XmlNode and set it to "my:signatures1" node of Xml file to sign
            XmlNode signatures1 = xmlDocument.SelectSingleNode("//*[local-name(.) = '" +
signatures1NodeName + "']");

            // Replace signatures1 first child with the computed signature node
            signatures1.ReplaceChild(signatureComputed, signatures1.FirstChild);

            // Create the name for the signed XML file
            string nameXML = checkedXML;
            nameXML = nameXML.Insert(checkedXML.Length - 4, "_Assinado");

            // Save the signed XML file
            xmlDocument.Save(nameXML);
            MessageBox.Show("Ficheiro XML assinado com sucesso");
        }
        catch (Exception em)
        {
            MessageBox.Show(em.ToString());
        }
    }
  }
}
```

7.1.3 InfoPath Verify Tool

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using DevScope.CartaoDeCidadao;
using System.IO;
using System.Security.Cryptography.X509Certificates;
using System.Xml;
using System.Security.Cryptography.Xml;
```

```
using System.Security.Cryptography;
using System.Configuration;


namespace Infopath_Verify_Tool
{
   public partial class Form1 : Form
   {
      public String[] filePaths;

      public Form1()
      {
         InitializeComponent();
      }

      private void ButtonReadCard_Click(object sender, EventArgs e)
      {
         try
         {
            // Initialize the EIDPT and check if there is one card in card reader
            EIDPT.Init("");

            // Verify if card is activated
            if (EIDPT.IsActivated() != CardActivationStatus.ACTIVE)
            {
               MessageBox.Show("Cartão não ativo");
            }

            // SODChecking = to verify the validity of the cards data
            EIDPT.SetSODChecking(false);

            // Read data from card
            Id citizen = EIDPT.GetID();
            Picture photo = EIDPT.GetPicture();

            // Convert photo byte array in memory stream
            System.IO.MemoryStream ms = new System.IO.MemoryStream(photo.Bytes, 0,
photo.BytesLength, false);
            Image tempImage = CSJ2K.J2kImage.FromStream(ms);
            ms.Close();

            pictureBoxPhoto.Image = tempImage;
            textBoxFirstName.Text = citizen.FirstName;
            textBoxName.Text = citizen.Name;
            textBoxNationality.Text = citizen.Nationality;
            textBoxExpiryDate.Text = citizen.ExpiryDate;
         }
         catch (Exception em)
         {
            MessageBox.Show(em.ToString());
         }
         finally
         {
            // Close the EIDPT and clean temporary data
            EIDPT.Exit(ExitMode.UNPOWER);
```

```
        }
    }

    private void ButtonBrowseXmlSign_Click(object sender, EventArgs e)
    {
        // Clear the treeview
        treeView.Nodes.Clear();

        try
        {
            // Get folderBrowserDialog1 result
            DialogResult result = this.folderBrowserDialog1.ShowDialog();

            if (result == DialogResult.OK)
                // If result  == OK, set textBoxXmlFiles.Text
                textBoxXmlSign.Text = this.folderBrowserDialog1.SelectedPath;


            // String array with all xml file paths in tree view
            filePaths = Directory.GetFiles(textBoxXmlSign.Text, "*.xml",
SearchOption.AllDirectories);

            // Add nodes to treeView
            for (int i = 0; i < filePaths.Length; ++i)
            {
                // Add a root node
                treeView.Nodes.Add(filePaths[i]);
            }
        }
        catch (Exception)
        {}
    }

    private void ButtonVerify_Click(object sender, EventArgs e)
    {
        if (textBoxXmlSign.Text == "")
        {
            MessageBox.Show("Erro. Primeiro tem que escolher uma pasta com o botão
browse");
            return;
        }

        int count = 0;
        String[] checkedXML = new String[filePaths.Length];
        TreeNode node;

        for (int i = 0, j = 0; i < filePaths.Length; ++i)
        {
            node = treeView.Nodes[i];

            if (node.Checked == true)
            {
                // String array with checked items names
                checkedXML[j] = node.Text;
                j++;
```

```
            // Count the checked items
            count++;
        }
    }

    if (checkedXML[0] == null)
    {
        MessageBox.Show("Erro. Tem que selecionar pelo menos um ficheiro para assinar");
        return;
    }

    for (int x = 0; x < count; x++)
    {
        // Call function VerifyXml
        VerifyXml(checkedXML[x]);
    }
}


public static void VerifyXml(String path)
{

    // Create a XmlDocument object
    XmlDocument xmlDoc = new XmlDocument();

    // Set PreserveWhitespace true
    xmlDoc.PreserveWhitespace = true;

    // Load an XML file into the XmlDocument object
    xmlDoc.Load(path);

    // Create a SignedXml from signed Xml file
    SignedXml signedXml = new SignedXml(xmlDoc);

    // Create a XmlNodeList and get all nodes with the name "Signature"
    XmlNodeList nodeListSignature = xmlDoc.GetElementsByTagName("Signature");

    // Get all nodes with the name "X509Certificate"
    XmlNodeList nodeListCert = xmlDoc.GetElementsByTagName("X509Certificate");

    // Show a error message box if no signature was found
    if (nodeListSignature.Count <= 0)
    {
        MessageBox.Show("Erro na verificação: Não foi encontrada nenhuma assinatura no
documento");
        return;
    }

    for (int i = 0; i < nodeListSignature.Count; i++)
    {
        // Copy X509Certificate node text to a string
        String certificate = nodeListCert[i].InnerText;

        // Convert certificate string to byte array
        byte[] byteCertificate = ASCIIEncoding.UTF8.GetBytes(certificate);
```

```
            // Create a X509Certificate2 object from byteCertificate byte[]
            X509Certificate2 xCertificate = new X509Certificate2(byteCertificate);

            // Create string with the public key from xCertificate
            String publicKey = xCertificate.PublicKey.Key.ToXmlString(false);

            // Generate a RSACryptoServiceProvider object
            RSACryptoServiceProvider rsakey = new RSACryptoServiceProvider();

            // Load public key from string
            rsakey.FromXmlString(publicKey);

            // Show a error message box if no public key was found
            if (rsakey == null)
            {
               MessageBox.Show("Erro ao carregar chave pública");
               return;
            }

            // Load the number i Signature node into signedXml
            signedXml.LoadXml((XmlElement)nodeListSignature[i]);

            // Display the signature verification results in a MessageBox
            if (signedXml.CheckSignature(rsakey))
            {
               MessageBox.Show("A assinatura ( número " + i + " ) de " +
xCertificate.GetNameInfo(X509NameType.SimpleName, false) + " no ficheiro " + path + " é
válida.");
            }
            else
            {
               MessageBox.Show("A assinatura ( número " + i + " ) de " +
xCertificate.GetNameInfo(X509NameType.SimpleName, false) + " no ficheiro " + path + " é
inválida.");
            }
         }
      }
   }
}
```

7.2 Anexo B - Maquetes desenvolvidas durante o projecto

7.2.1 InfoPath *Sign Tool*

Esta aplicação está dividida em duas áreas distintas, uma de leitura do cartão de cidadão e apresentação dos respectivos dados e uma segunda área que utiliza o cartão de cidadão, e respectivo certificado de assinatura digital, para assinar digitalmente ficheiros XML.

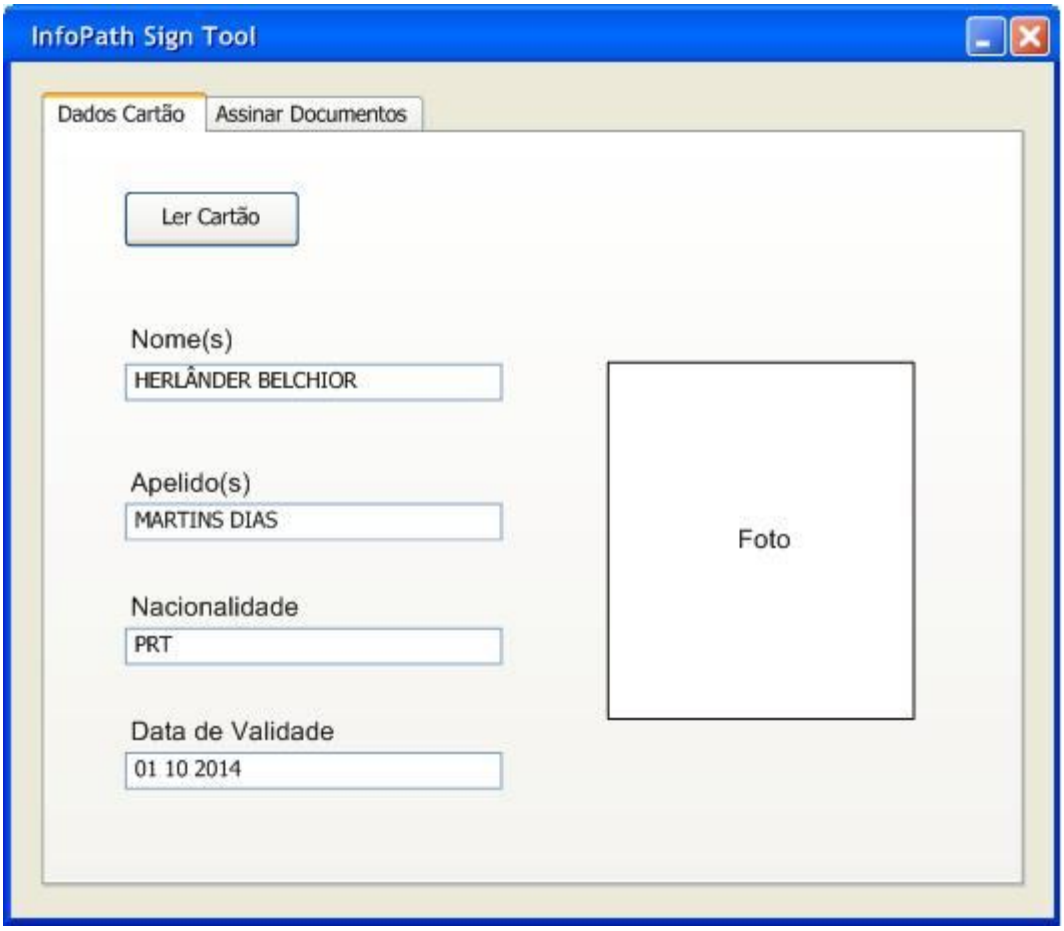


Figura 13: Maquete *Sign Tool* (área destinada à leitura do cartão de cidadão)

Para proceder à leitura e apresentação dos dados do cartão de cidadão, basta clicar no botão "**Ler Cartão**", isto após ter ligado o leitor de cartões ao computador e de ter inserido o cartão no leitor. Após alguns segundos os dados surgem no ecrã, como mostra a Figura 13.

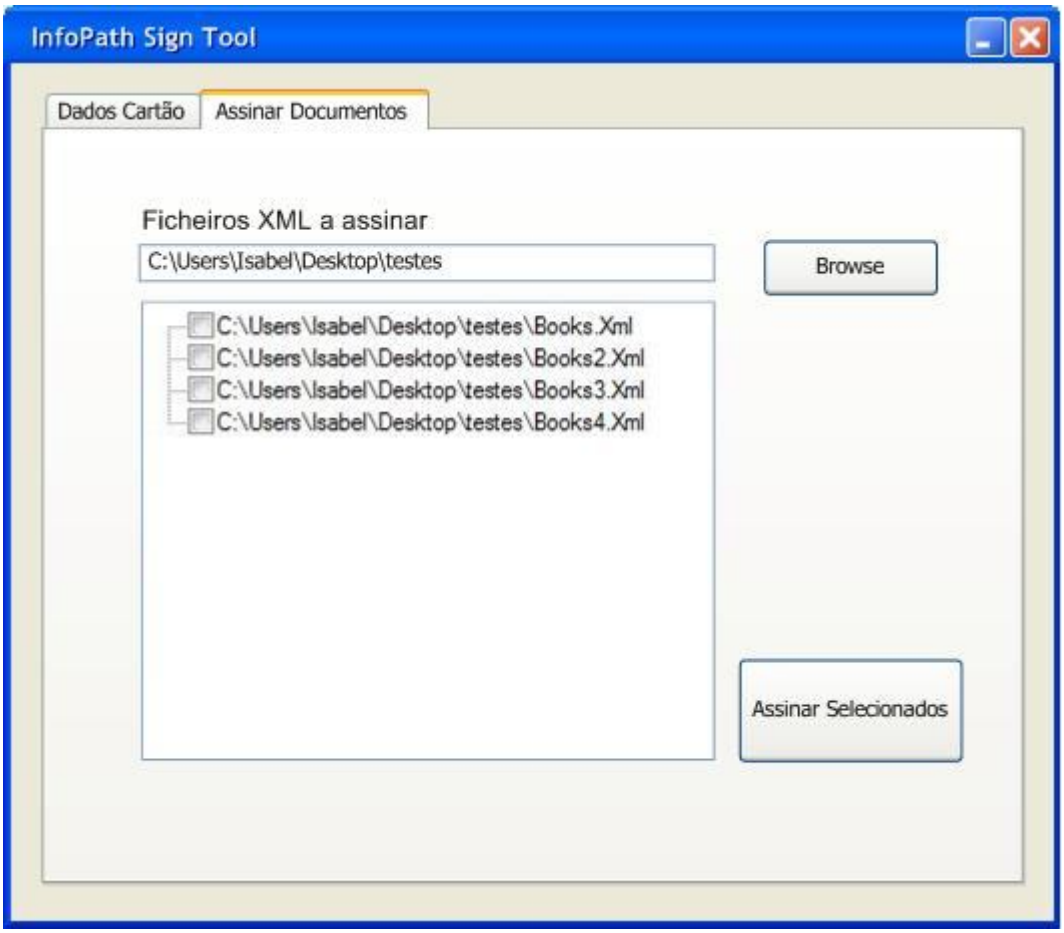
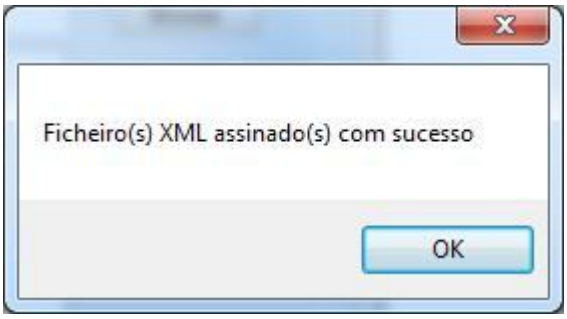


Figura 14: Maquete *Sign Tool* (área destinada à assinatura de ficheiros XML)

Nesta segunda secção é possível assinar digitalmente ficheiros XML e para isso é necessário que o leitor e o cartão de cidadão estejam ligados ao computador, para depois se clicar no botão "**Browse**" e seleccionar o directório que contenha os ficheiros XML a assinar. O caminho deste directório surge numa caixa de texto à esquerda do botão *browse* e os ficheiros XML que estejam dentro do directório seleccionado ou sub directórios, surgem na *treeview* abaixo da caixa de texto. De seguida o utilizador pode marcar os ficheiros que deseja assinar e clicar no botão "**Assinar Seleccionados**". A aplicação lê o cartão de cidadão para extrair o certificado de assinatura digital, e assina os ficheiros seleccionados. Se as assinaturas forem efectuadas com sucesso, são criados no mesmo directório os ficheiros XML assinados e surge a seguinte mensagem:



7.2.2 InfoPath *Verify Tool*

Esta aplicação está dividida em duas áreas distintas, uma de leitura do cartão de cidadão e apresentação dos respectivos dados e uma segunda área que utiliza o cartão de cidadão, e respectivo certificado de assinatura digital, para verificar a validade de assinaturas em ficheiros XML.

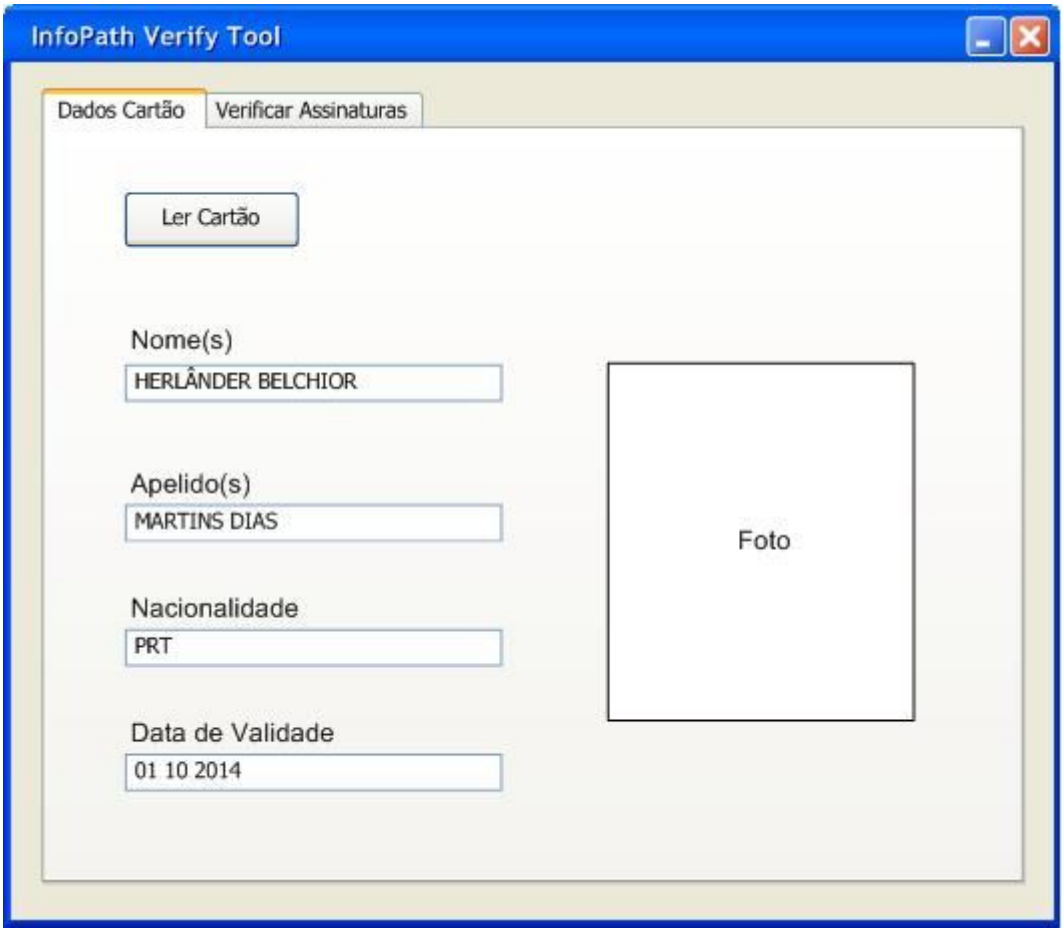


Figura 15: Maquete *Verify Tool* (área destinada à leitura do cartão de cidadão)

Para proceder à leitura e apresentação dos dados do cartão de cidadão, basta clicar no botão "**Ler Cartão**", isto após ter ligado o leitor de cartões ao computador e de ter inserido o cartão no leitor. Após alguns segundos os dados surgem no ecrã, como mostra a Figura 15.

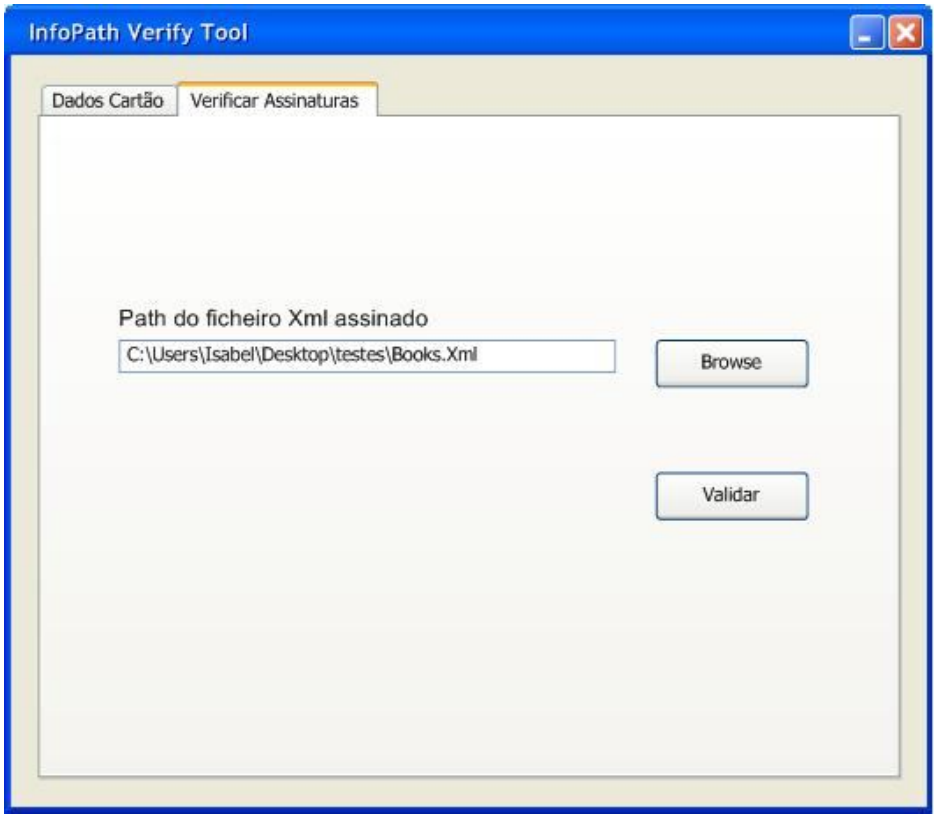


Figura 16: Maquete Verify Tool (área destinada à verificação de assinaturas em ficheiros XML)

Nesta segunda secção é possível verificar a validade de assinaturas em ficheiros XML e para isso é necessário que o leitor e o cartão de cidadão estejam ligados ao computador, para depois se clicar no botão "**Browse**" e seleccionar o ficheiro XML assinado digitalmente. O caminho deste ficheiro surge numa caixa de texto à esquerda do botão *browse* e de seguida é necessário clicar no botão "**Validar**". A aplicação lê o cartão de cidadão para extrair a chave pública contida no certificado de assinatura digital, e verifica se a assinatura é válida ou não, mostrando o resultado através de uma mensagem como a seguinte: