Securing XML Web Services with WSE 2.0 and cryptographic hardware
Preface

More than one year ago I decided to go on an adventure: Erasmus. This student exchange project gave me the chance to come in contact with Germany, in my opinion one of the most technologically evolved countries of Europe.

It became clear to me by the very beginning; this is exactly what I expected. I would like to thank my German mentor Prof. Dr. Uwe Heuert for his incredible support, for sharing his knowledge and for the great time we had together. Next, I would like to thank my Belgian mentor Lic. Hans Gruyaert for his periodically emails. Thanks to KaHo Sint-Lieven and Hochschule Merseburg (FH) for giving me the opportunity to go on an Erasmus project.

Special thanks to nCipher and ReinerSCT for providing an ‘nCipher nShield HSM’ and the ‘ReinerSCT cyberJack pinpad’ Smart Card reader together with Smart Cards. Thanks to Chris Allen from nCipher for the email support.

Of course not only work has been done. Almost every weekend our Erasmus group went to visit a well known city in Germany. Among others, Leipzig, Magdeburg, Eisenach, Erfurt, Postdam, Weimar, Dresden and last but not least Berlin were visited. I would like to take this opportunity to thank the whole group of Erasmus students for the wonderful time we spent together.

Filip Van Lerberge
1 Introduction

XML Web Services use the Internet as transport medium. The Internet is in fact very insecure: there are lots of persons trying to ‘sniff’ IP packets or trying to crack servers and databases, especially when the data in the database is valuable.

SSL provides a partial solution to this problem: the connection between client and server can be secured using authenticity and encryption.

The biggest problem using this approach is that once the data is arrived at the server, the protection is removed. Programs or persons can never check again whether the data is really sent by the person it pretends to be, or that the data is not altered, because the authenticity information doesn’t exist no more.

XML Digital Signatures solve these shortcomings. Digital signatures can be checked any time since they are not stripped off once the data arrives at the server. Programs or persons can check the validity of the data any time.

This project will use those XML Digital Signatures together with XML Encryption using WSE: Web Service Enhancements. This optional package for .NET is provided by Microsoft and gives the possibility to do cryptographic operations such as RSA using a CSP: a Cryptographic Service Provider.

Of course, both server and client need to be sure that their signing certificate cannot be compromised. Therefore, the private key needs to be protected. Software solutions do a good job protecting private keys, but hardware can protect them even more securely.

A Hardware Security Module from nCipher will be used to protect the private keys at the server side. At the client side, certificates and its private keys will be protected using Smart Cards and a PIN using a ReinerSCT PIN pad reader.


This document comes together with a CD containing the programs (including the source) and the referenced PDF documents.
2 Concepts

2.1 What are XML Web Services?

XML (eXtensible Markup Language) Web Services are the fundamental building blocks in the move to distributed computing on the Internet. Open standards and the focus on communication and collaboration among people and applications have created an environment where XML Web services are becoming the platform for application integration. Applications are constructed using multiple XML Web services from various sources that work together regardless of where they reside or how they were implemented.

XML Web Services use standard web protocols: TCP/IP, HTTP and XML. Binary formats are not used which makes it even more transparent. Because it uses standard web protocols, it can bypass firewalls very easily and it can make use of standard HTTP proxies.

XML Web Services are platform and programming language independent: besides .NET languages, Web Services can be developed in e.g. JAVA, TCL and even PHP.

In the earlier years, distributed computing was often done by CORBA (Common Object Request Broker Architecture), RPC, RMI or DCOM, which had all some disadvantages: some technologies had no possibilities to throw exceptions; some technologies had problems bypassing firewalls as they don’t work over HTTP; some are simply too complex; etc.

There are lots of definitions of XML Web Service, but almost all definitions have these things in common:

- XML Web Services expose useful functionality to Web users through a standard Web protocol for communication. Mostly, the protocol used is the Simple Object Access Protocol (SOAP).
- XML Web Services provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them. This description is provided in an XML document called a Web Services Description Language (WSDL) document which is generated automatically together with proxy classes in the corresponding programming language.
- XML Web services can be registered so that potential users can find them easily. This is done with Universal Discovery Description and Integration (UDDI).

This project will handle the possibilities to secure those XML Web Services through software and hardware.

2.2 Why not using SSL (only)?

SSL (Secure Sockets Layer) provides the secure interchange of sensitive data between a browser and Web server, but once received, the data is left unprotected on the server. In other words, SSL protects the data while it is in transport.

Imagine a hacker trying to find out credit card numbers. If this hacker had the choice between sniffing IP packets in transit in order to obtain a single user's credit card number and breaking into a back-end database containing thousands of credit card numbers, he would certainly choose the last option. If the data itself was encrypted, as opposed to just its transport, it would help reduce the incidents of unencrypted data left vulnerable on public servers.
In the picture above, users send encrypted data to the server, but once the data is arrived, it is decrypted and stored in a database. Optionally, users have a client certificate in their browsers.

Next to protecting the confidentiality of business messages (private communication accomplished by using encryption), there are other needs for XML Web Services and in addition, they have to be long-term:

- **Authenticity**: The Web service knows the identity of the client and vice versa.
- **Integrity**: The communications between the client and the Web Service are protected from tampering while in transit.
- **Non-repudiation**: The client cannot disown the requests sent to the Web Service and the Web Service cannot disown the replies sent to the client.

The globally-recognized method for satisfying these requirements for secure business transactions is to use digital certificates to enable the encryption and digital signing of the exchanged data. Chapter 2.6: ‘X.509 certificates’ gives more information about this topic.

The term ‘Public Key Infrastructure’ (PKI) is used to describe the processes, policies and standards that govern the issuance, maintenance and revocation of the certificates (and public and private keys) that the encryption and signing operations require. For an example using PKI, please refer to chapter 2.9: ‘XML Digital Signatures in depth’ and appendix 8.6: ‘Email and Digital Signatures’.

To ensure long-term authenticity, integrity and non-repudiation, XML Web Services can be used. In addition, XML Encryption has the possibility to ensure long-term confidentiality.

### 2.3 What are XML Digital Signatures?

XML Digital Signatures (or XML DSIG for short) are developed by a joint Working Group of the IETF and W3C. The goal of this working group is to develop an XML compliant syntax used for representing the signature of Web resources and portions of protocol messages (anything able to be referenced by a URI) and procedures for computing and verifying such signatures.

XML Encryption is solely a W3C effort.
Figure 2: Using XML Digital Signatures to assure long-term authenticity, integrity and non-repudiation

XML Digital Signatures are described in detail in chapter 2.9: ‘XML Digital Signatures in depth’.

Of course, XML Digital Signatures and XML Encryption can be used together with SSL to provide even more invisibility and security.

2.4 What is WSE 2.0?

WSE 2.0 stands for Web Service Enhancements, second version. WSE 2.0 is made by Microsoft and it is a supported add-on to Microsoft Visual Studio .NET and the Microsoft .NET Framework providing developers the latest advanced Web services capabilities to keep pace with the evolving Web Services protocol specifications. It gives XML Web Services possibilities such as using XML Digital Signatures and XML Encryption. WSE 2.0 uses high level cryptographic functions and can use certificates very easily.

At the moment of writing, the current version of WSE 2.0 is WSE 2.0 Service Pack 3. Microsoft is working on a new collection of programming frameworks using WSE 2.0, codenamed Indigo. It will take part of the new Windows version codenamed Longhorn. Indigo marries the best features of .NET Enterprise Services and Web Services into a unified programming and administration model to make the implementation and deployment of Web Services very easily. Indigo is based on the Web Service Enhancements specifications. Any WSE 2.0 code developed today can be used in Indigo without any big changes.

The current WSE 2.0 can be seen as a playground for the upcoming Indigo, but it can already be used in production environments.
2.5 Interfaces to cryptographic hardware

Cryptographic hardware is a piece of hardware that helps the computer and/or the user to use cryptography. There are different sorts of cryptographic hardware. In this project a HSM and Smart Cards will be used.

2.5.1 What is a HSM?

HSM stands for Hardware Security Module (HSM). There are several reasons to use a HSM:

- **Safety:** Private keys exist in the HSM itself and not outside. This way, the private keys are hardware protected and far more difficult to hack.
- **Speed:** The HSM has a processor which is able to perform more cryptographic operations (RSA) per second than a normal CPU\(^1\). In addition, the CPU is freed because cryptographic operations are offloaded to the HSM.
- **Randomness:** Key generation (and random number generator where key generation relies on) are performed by the HSM. Random number generation done by a HSM is far more random than done by software, approved by the FIPS (Federal Information Processing Standards) 140-2 level 2.
- **Load balancing:** There are possibilities to link multiple HSMs together to spread the load between them.

2.5.2 What is a Smart Card?

A smart card, or integrated circuits card (ICC), is defined as any integrated circuitry embedded into a flat, plastic body. Mostly Smart Cards look like normal credit cards in shape and size, except cell phone SIM cards. Although there are a diverse range of applications, there are two broad categories: Memory cards and Microprocessor cards.

- **Memory cards** contain only non-volatile memory storage components and perhaps some specific security logic. As an example health system cards can be mentioned.
- **Microprocessor cards** contain memory and a microprocessor (often 8 bit running at 5MHz).

The difference between the two cards cannot be seen from the outside (unless the card’s label gives more information).

\(^1\) This is not always true. Please refer to chapter 4.3: ‘Signing Benchmark Tool’
To be able to access the data on a Smart Card, a PIN (Personal Identification Number) is often needed.

Certificates can be stored on a Smart Card. Operations with the private key are performed using the built-in processor, so the private key does not leave the Smart Card. This way, together with on-board key generation, a higher level of security is achieved.

2.5.3 Interfaces to Cryptographic Hardware

There are different approaches how to use cryptographic hardware. The two most common are by using PKCS#11 and CSPs.

Because different vendors were asking for a standard, Public-Key Cryptography Standard number 11 (PKCS#11) has been developed by RSA Laboratories. PKCS#11 is an API (Application Programming Interface), also called Cryptoki (pronounced Crypto-Key), to devices which hold cryptographic information and perform cryptographic functions. Cryptoki follows an approach addressing the goals of technology independence (any kind of device) and resource sharing (multiple applications accessing multiple devices), presenting applications a common, logical view of the device called a cryptographic token. PKCS#11 is implemented in a .DLL on Windows or in a .SO shared library on Linux and other operating systems.

For an example how to use PKCS#11, please refer to 8.6.2: ‘Mozilla Thunderbird’.

Cryptographic Service Provider (CSP) is developed by Microsoft and is a direct competitor of PKCS#11. It also contains implementations of cryptographic standards and algorithms. At a minimum, a CSP consists of a dynamic-link library (DLL) that implements the functions in CryptoSPI, which is the system program interface used with a CSP. Microsoft Windows XP has 12 CSPs installed by default, of which ‘Microsoft Enhanced Cryptographic Provider’ is the most important for this project. Others include the weak ‘Microsoft Base Cryptographic Provider’ and ‘Gemplus GemSAFE Card CSP’.

The HSM used in this project can both handle CSP and PKCS#11. This project will only focus on CSP because the use of it in Windows is better supported and more transparent than PKCS#11, and WSE 2.0 is built in top of CSPs.
2.6 X.509 certificates

This part of the document will describe how X.509 certificates are built and where they are stored.

A certificate is issued by a CA (Certification Authority). When the certificate for the CA is a self-signed certificate, the CA is considered a Root CA.

Typically, an offline Root CA (thus, far more difficult to compromise) will issue certificates to one or more online CA’s. Those CA’s will issue certificates to users and devices. This way, a certificate chain is created.

![Figure 4: An example of a Certificate Chain using an offline Root CA](image)

In X.509 certificates, all fields are built using OIDs (Object Identifiers). Structurally an OID is a node in a hierarchically assigned namespace, formally defined using the ITU-T’s (ITU Telecommunication Standardization Sector) ASN.1 (Abstract Syntax Notation one) standard. Each node in the tree is identified by the numbers of the nodes starting at the root of the tree. New nodes are created by registering them under the node’s registration authority.

This project only uses X.509 version 3 certificates, which is the current format of certificates used in all recent operating systems and applications. X.509v3 certificates are based on X.509v1 certificates but allow adding any number of custom fields (extensions).

![Figure 5: X.509 version 3 certificate structure](image)

First the X.509 mandatory version 1 fields will be described, which are marked with a white background in the figure above.
Version:
The version number of the certificate format

Certificate Serial Number:
A unique identifier for each certificate that a certain CA issues

Certificate Algorithm Identifier:
Signature algorithm used by the issuer (e.g. sha1RSA)

Issuer:
A distinguished name (DN) of the CA that issued the certificate. This issuer name is commonly represented using an X.500 or Lightweight Directory Access Protocol (LDAP) format.

Validity Period:
The date when the certificate becomes valid and the date when the certificate is no longer considered valid.

Subject:
A distinguished name (DN) of the user that the CA issues the certificate to. This name is often represented using an X.500 or Lightweight Directory Access Protocol (LDAP) format.

Public Key:
The public key of the key pair that is associated with the certificate. The key type identifier is also defined (e.g. RSA).

Certification Authority’s Digital Signature:
The digital signature of the authority that issued the certificate.

The following fields are defined in X.509 version 3 and are optional. Only the most important ones for this project will be described:

CRL distribution points (CDP):
Provides one or more URLs where the application or service can retrieve a Certificate Revocation List (CRL) from.

Enhanced Key Usage (EKU):
Defines which applications can be used together with a certificate. Examples of EKU are ‘Secure Email’, ‘Client Authentication’, ‘Server Authentication’ and ‘All Application Policies’. These policies are the same as ‘Intended Purposes’.

![Certificate Extension Table](image)

When the EKU extension is not set, all certificate purposes are added by default. Purposes can be added or removed using the properties of the certificate. The sample WSE 2.0 certificate named ‘WSE2QuickStartClient’ has no extensions set.

![Certificate Settings](image)

Besides the certificate, there exists a corresponding ‘Private Key’ which should be, as the name implies, kept private. When a user has access to a corresponding ‘Private Key’, digital signing and decrypting data is possible.

Certificates can be stored in different locations. First of all, a certificate can be stored on any place in the file system using a file containing a certificate (such as .CER, .DER, .PEM, .P12 or .PFX), but in this project certificates will be stored in Certificate Stores. This way the management is easier and the usage of certificates is more transparent to applications. A certificate issued by a ‘Microsoft Certificate Authority’ will automatically be installed in the correct store when Internet Explorer is used.
On a Windows system, every user has its own private ‘Current User’ store. The containing certificates cannot be seen, used or altered by other users. Within each store, there are several folders. Here is a short description of the most important ones for this project:

- The ‘Personal/Certificates’ folder contains the certificates issued to the user. This folder is empty by default.
- The ‘Trusted Root Certification Authorities/Certificates’ folder contains the Root CA’s that the user wants to trust. By default, Verisign, Thawte and other Root CA’s exist in this folder.

Besides a ‘Current User’ store, there is also a ‘Local Computer’ store. This store has, in general, the same folders as the ‘Current User’ store (the differences are not important for this project), but is accessible for all administrators. Typically server certificates will be stored in the ‘Local Computer’ store.

The content of the stores can be viewed and altered using the Microsoft Management Console (‘mmc.exe’) using the corresponding certificates snap-in or via ‘Control Panel’, ‘Internet Options’, ‘Content’, ‘Certificates’.

Figure 9: Different certificate store locations in MMC with their corresponding folders
2.7 Requesting and issuing certificates

There are several ways to request a certificate. The easiest way could be opening the Certificates store (via ‘mmc.exe’ as described before for all the stores or via ‘certmgr.msc’ for the Personal store only) and request a certificate from the menu. Unfortunately, this only works when an Active Directory has been installed and correctly configured on the server.

![Requesting a certificate using the certificates snap-in](image1)

![An Active Directory must be installed](image2)

A certificate without extensions can be made using the ‘makecert.exe’ tool, provided by Microsoft in the optional ‘Authenticode for Internet Explorer 5.0 & Authenticode for DEC Alpha - Internet Explorer 5.0’ package, downloadable on the Microsoft web site. Private keys can be made using the ‘PVK Digital Certificate Files Importer’ which is also an optional package from Microsoft.

Another option is to use OpenSSL to make a certificate request. This option is described in appendix 8.5: ‘Using OpenSSL’.

In this project, a third option will be used to request certificates: using Microsoft Internet Explorer. This approach has the major advantage that it is very easy and the issued certificates will be installed automatically in the correct store.

Other browsers that are not built upon Internet Explorer cannot be used to request a certificate when a Microsoft Certification Authority is used, because Mozilla Firefox, Opera and other Netscape-alike browsers have a different request format, named SPKAC: ‘Signed Public Key And Challenge Protocol’.

Internet Explorer uses PKCS#10 and CMC: Certificate Management Messages over CMS (using PKCS#7 and PKCS#10).

The Microsoft Certification Authority doesn’t understand the SPKAC protocol by default. A possible solution to this problem is to use another CA, e.g. ‘The OpenCA Project’ (http://www.openca.info), which supports both Internet Explorer and Mozilla Firefox.

To request a certificate from a CA, the corresponding web site must be visited. On the main page, users have the choice between the following tasks:

- ‘Request a certificate’
- ‘View the status of a pending certificate request’
- ‘Download a CA certificate, certificate chain, or CRL’
After choosing to request a certificate, the certificate type must be selected. Users have the choice between a ‘Web Browser Certificate’, an ‘E-mail Protection Certificate’ or an ‘Advanced Certificate Request’.

In this example, an ‘E-mail Protection Certificate’ will be requested. Only Name and E-Mail are required fields, the rest is optional. In this example, the default ‘Microsoft Enhanced Cryptographic Provider v1.0’ is chosen and the ‘Request Format’ is not altered.

![Figure 12: Requesting an E-Mail Protection Certificate](image)

Because Windows XP is being used, there is a choice by default between 12 CSPs. If there are other CSPs installed on the machine, they will be added to the list and therefore they can be used just like the default CSPs. The SafeSign CSP is described in chapter 3.1.2: ‘SafeSign’.
When a special OID is needed for Enhanced Key Usage, it can be specified using the ‘Advanced Certificate Request’.

After requesting the certificate from the CA, the corresponding CA must be opened and the certificate must be issued by the administrator.

When the certificate has been issued, the user must reopen the web site from the CA and choose to view the status of a pending certificate request.

A ‘Potential Scripting Violation’ will be shown because the certificate store could be filled with certificates by a malicious script and could cause the certificate store to be full. After agreeing with the possible security risk, the certificate will be installed correctly.
Figure 16: Installing certificates involves a potential security risk
2.8 Certificate revocation

Certificates are issued for a certain period: they are valid for e.g. 20 years. Imagine that a hacker gets control of the private key of a certificate issued to person X? The certificate becomes worthless: the hacker can e.g. sign data and pretend he is person X. Another example: imagine a certificate issued to person Y working in organisation A. Person Y changes work to organisation B and therefore a new certificate is issued. The old certificate, telling that person Y works in organisation A is still there and is still valid, although the information is not correct.

There is need for a way to revoke a certificate. Certificate Revocation Lists, or for short CRLs, can be used to do this. A Certificate Authority has a list of revoked certificates. This digitally signed list is published to clients at a regular interval, e.g. every week.

CRLs are stored in a ‘Certificate Revocation List’ folder in an existing CA folder in a certificate store.

![Figure 17: Certificate Revocation List store locations](image)

The content of a ‘Certificate Revocation List’ folder are one or more CRLs, containing a list of certificates that have been revoked.

![Figure 18: The content of a CRL shows different revoked certificates](image)

But there are several problems concerning the usage of CRLs. For example:
- CRLs are not issued frequently enough to be effective against an attacker: prompt revocations are not possible
• At the expiry date of the CRL, all relying parties connect to the CA to fetch the new CRL, so massive peak loads can occur. Imagine 10,000 clients downloading a 5MB CRL. This means a peak of 50 gigabyte traffic at the expiry date.

There is a need for real time checking the validity of a certificate. The solution to this problem is to use OCSP: the ‘Online Certificate Status Protocol’, which can be relayed across TCP/IP networks. OCSP specifies a ‘request-response’ message syntax between a client application that requires certificate revocation status information and a server application that has knowledge of the revocation status. OCSP can be either a replacement or a supplement to check against a CRL.

There are still several problems with the usage of OCSP:
• Real-time checking slows down the applications. (Caching is a possible solution.)
• Vulnerable for Denial of Service (DoS) attacks. If the OCSP responder is unavailable, the requesting application reverts to CRL processing.
• Possible HTTP error messages sent by the OCSP responder are not digitally signed, so an attacker could return false messages back to the requester. (Note: The standard highly recommends that all revocation status information is signed.)

Besides these problems, problems with revocation checking in general exist such as:
• Revocation checking is expensive: CRLs require processor time, redundant servers, network bandwidth, so it requires money, but it should be free to use it frequently.

In practice, revocation checking is often turned off in user software. CRLs are useful in certain special-case situations e.g. where there exists a contractual obligation to use them.

In this project, CRL checking is not used, although some research and experiments were done. These can be found the appendix 8.3: ‘Implementation of certificate revocation lists’.
2.9 XML Digital Signatures in depth

The X509Signing sample code from the WSE 2.0 package will be used to explain XML Digital Signatures in depth. This sample performs following items:

- Generating a digital signature at the client
- Checking the digital signature for validity at the server

The message sent back to the client is not signed.

First the used certificates need to be installed in the proper locations according to the readme.htm file.

After the installation of the Web Service using the batch script included, the project can be run.

![Figure 19: Running the sample X509Signing program](image)

There are two possibilities to obtain more information about what happened when the project is run:

- Using a ‘Packet Sniffer’ such as Ethereal if the project is not running on localhost
- Using the included ‘Message Tracing’in WSE 2.0

The second option will be used. Therefore, the ‘WSE Settings 2.0’ configuration needs to be opened on the project of the client. The Diagnostics tab contains an option to enable message tracing.

![Figure 20: Enabling ‘Message Trace’ in the WSE 2.0 options dialog](image)
When running the project again, two log files containing the XML files with the SOAP requests are created in the 'bin/' folder of the client:

- **OutputTrace.webinfo**: Lists the request from the client to the server, which contains the digital signature. This file is the most important one for this chapter.
- **InputTrace.webinfo**: Lists the messages the client receives from the server.

To improve the readability of the code, certain URI's are shortened by three dots (…).

First request will be investigated.

```xml
<?xml version="1.0" encoding="utf-8"?>
<log>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
                 xmlns:wsse="urn:oasis-200401-wss-wssecurity-secext-1.0.xsd"
                 xmlns:wsu="urn:oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <soap:Header>
      <wsa:Action wsu:Id="Id-588f681e-b663-4a00-a889-df71844f62b2">
        ...StockQuoteRequest
      </wsa:Action>
      <wsa:MessageID wsu:Id="Id-4a97f6d-132d-4221-9881-8541217cc86f">
        uuid:8bd590b-6556-4e99-bf72-5f9e8021c133
      </wsa:MessageID>
      <wsa:ReplyTo wsu:Id="Id-55d1902d-69cd-4746-8138-bd427d824949">
        .../anonymous
      </wsa:ReplyTo>
      <wsa:To wsu:Id="Id-6f2e6615-19cc-4db8-8e9e-2512ad3f9615">
        ...X509SigningService.asmx
      </wsa:To>
    </soap:Header>
    <wsse:Security soap:mustUnderstand="1">
      <wsu:Timestamp wsu:Id="Timestamp-b2ee456-5efa-4997-adbe-c79410303c0">
        <wsu:Created>2005-03-07T10:41:21Z</wsu:Created>
      </wsu:Timestamp>
      <wsse:BinarySecurityToken
        ValueType="#X509v3"
        EncodingType="#Base64Binary"
        xmlns:wsu="urn:oasis-wss-wssecurity-utility-1.0.xsd">
        MIIBxDCCAW6gAwIBAgIQxUSXFzWJYYtOZnmmuOMKkJANBgkqhkiG9w0BAQFAQADAWMRQwEgYDVQQ
        DBwtsB291IEFlZ0VQYzI6I4wZCeEzIyMzMzNjA5N2RiMDArMC4wMDA0MDc2MDAxN2YzZGMxOTkX
        MTFFdTRJrWj1a1N0YXZ0Q2xZW5ICGEMA0CSgSIb3QEBQAAo4GNDACB1QKbGQCiL6a9
        x9286oY4+0Q85xXnkEQE4quJ17c3PUPvdV7k9A02hRG481XIfWhrDY5170EB7KGW7qFJo0tLLeMec
        /UkKUWcvY3Vjrs2nE9xO3S5WdNaDukYh+Cxt+FUU6tUkDegq?dqwioX0H0RyO13HgbWbtu
        maLdc8jufzL2HaQIDAQBo0sWSTBHBqNVQnEoQfjgQa+gBASSAktBhs0dTwCNYS8ePmRj0j0rgwFjEUMB
        IGA1UEAxMkLU9v9dCBB22Yw7s3mCEAY3baCgAGSKeC+41KpcFvWQDYJKoZIhvcNAQEEOBQADQAsI
        bnMPVYkNNfX1tG1F+qF1HwJdfeDSuUp4RPucWF5qkh6sDsdWVBY5s7/txNhVgJziy08DPYdu2fPM
        ER8ajJf1
      </wsse:BinarySecurityToken>
    </wsse:Security>
  </soap:Envelope>
</log>
```
<Reference URI="#Id-4a97fd6c-132d-4221-9881-8541217cc86f">
  <Transforms>
    <Transform Algorithm="...xml-exc-c14n#" />
  </Transforms>
  <DigestMethod Algorithm="...xmlmdsig#sha1" />
  <DigestValue>kabHA/8CU1UjyousOd7gU7YKIJ4=</DigestValue>
</Reference>

<Reference URI="#Id-55d1902d-69cd-4746-8138-bd427d824949">
  <Transforms>
    <Transform Algorithm="...xml-exc-c14n#" />
  </Transforms>
  <DigestMethod Algorithm="...xmlmdsig#sha1" />
  <DigestValue>h0JsuKHaI8Unc6sSUcVnFGSvtdY=</DigestValue>
</Reference>

<Reference URI="#Id-6f2e6615-19cc-4db8-8e9e-2512ad3f9615">
  <Transforms>
    <Transform Algorithm="...xml-exc-c14n#" />
  </Transforms>
  <DigestMethod Algorithm="...xmlmdsig#sha1" />
  <DigestValue>MlvteS8XqZ9c+jY15cCmOfVnS8U=</DigestValue>
</Reference>

<Reference URI="#Id-6f2e6615-19cc-4db8-8e9e-2512ad3f9615">
  <Transforms>
    <Transform Algorithm="...xml-exc-c14n#" />
  </Transforms>
  <DigestMethod Algorithm="...xmlmdsig#sha1" />
  <DigestValue>MlvteS8XqZ9c+jY15cCmOfVnS8U=</DigestValue>
</Reference>

<Reference URI="#Id-26e51ec7-6776-484e-b6c6-b98ebeae8123">
  <Transforms>
    <Transform Algorithm="...xml-exc-c14n#" />
  </Transforms>
  <DigestMethod Algorithm="...xmlmdsig#sha1" />
  <DigestValue>Ykn83BwnExGDbDdMaP4wlge/0BI=</DigestValue>
</Reference>

<SignatureValue>
  JFfwrYMis9umeOn9Rq+zOLKK32ThNeleIEdKbHFKMB7WugbTNzA7UgLUGNnCu3r3BnRGk4fnjWJ
V8O3PozS5etC6TmOM8Rxurhato/UrxfWpNZfM1Q1W4GypUj3hX+DzpJ0bkiIKUeKsybUyhMcAJml
37RwAwpd8SzOo2sP961sk=
</SignatureValue>

<KeyInfo>
  <wsse:SecurityTokenReference>
    <wsse:Reference
      URI="#SecurityToken-3db27ed5-25bd-47fa-b698-ef084e60de38"
      ValueType="...#X509v3" />
  </wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
</wsse:Security>
</soap:Header>
<soap:Body wsu:Id="Id-26e51ec7-6776-484e-b6c6-b98ebeae8123">
  <StockQuoteRequest xmlns=".../wse/samples/2003/06">
    <symbols>
      <!--_symbols_content-->
The body (the message) has an ID: `Id-26e51ec7-6776-484e-b6c6-b98ebeae8123`. This ID can also be found in a child element of the `SignedInfo` tag: the `Reference` tag. This `Reference` tag has a child element too, named `DigestValue`. The content of this `DigestValue` is the BASE64 encoded SHA-1 hash representation of the canonical form of the actual body.

\[
\text{DigestValue} = \text{BASE64}(\text{SHA1(c14n(data)))};
\]

**Code 2: Calculation of DigestValue**

The other child elements of the `SignedInfo` tag are `Reference` tags with an ID from `wsa:Action`, `wsa:MessageID`, `wsa:ReplyTo`, `wsu:Timestamp` etc. Every `Reference` tag has its own `DigestValue`, calculated as described above.

At last, the `SignatureValue` is made out of a BASE64 encoded SHA-1 hash of all elements in the `SignedInfo` tag in canonical form, including the tag itself, by using RSA public key encryption.

\[
\text{SignatureValue} = \text{BASE64}(\text{RSA}(\text{SHA1(c14n(SignedInfo))}));
\]

**Code 3: Calculation of SignatureValue**

This `SignatureValue` contains the actual digital signature.

The conversion between the normal and the BASE64 representation can be done using for example 'Total Commander' or an online tool. The SHA-1 representation can be calculated using for example 'HashCalc' from 'SlavaSoft' or OpenSSL.

The XML file also contains the certificate used for checking the signature. It resides between the `BinarySecurityToken` tags. This string is a BASE64 representation of an ASN.1 encoded object (i.e. a certificate). To have a deeper look into this certificate, it must be converted from BASE64 to binary ('Total Commander'). Then the ASN.1 can be decoded using for example 'GUidumpasn'. A complete list of tools can be found in the appendix 8.1: 'Used products'.
SET {
  SEQUENCE {
    OBJECT IDENTIFIER commonName (2 5 4 3)
    PrintableString 'Root Agency'
  }
}
}
SEQUENCE {
  UTCTime '030708184759Z'
  UTCTime '391231235959Z'
}
SEQUENCE {
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER commonName (2 5 4 3)
      PrintableString 'WSE2QuickStartClient'
    }
  }
}
SEQUENCE {
  SEQUENCE {
    OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1
    NULL
    }
  }
}
  BIT STRING 0 unused bits
  30 81 89 02 81 81 00 BE 2F A6 81 F7 1F 76 F2 7A
  18 E3 ED 10 06 C5 E7 C6 44 04 E2 AB 89 97 B7 37
  3D 43 DD 56 EE E4 F4 0D 36 85 11 B8 F3 55 C8 7D
  68 6B 0D 8E 62 EC E1 01 EC A1 96 EE A1 49 A2 D2
  CB 7B C7 9C FD 49 0A 53 00 A0 BF 75 6F 26 BB 36
  9C 4F 71 3B 74 92 58 87 4D CC 00 EE 91 88 7E 0B
  1B 7E 15 45 3A B5 49 03 7A A8 3B 76 AC 22 BC E5
  E1 B8 E4 D1 C8 E2 37 1E A6 D6 4D BB A6 68 B7 5C
  [ Another 12 bytes skipped ]
}
...

Code 4: Using GUDumpASN to view the ASN.1 content of a certificate

The server gets the request (containing the digital signature) from the client, and calculates a DigestValue out of the body, just like the client did. Other DigestValues are also generated and compared with the original DigitalValues.

The server uses the clients' public key to decrypt the SignatureValue it received to extract the DigestValue generated from the SignedInfo tag.

If the two DigestValues match, the message has not been altered and it's valid, so a proper reply can be sent.

```xml
<?xml version="1.0" encoding="utf-8"?>
<log>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse="...oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="...oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <soap:Header>
      <wsa:Action>...StockQuoteRequestResponse</wsa:Action>
      <wsa:MessageID>
```
Code 5: XML file with a possible response without digital signature

```xml
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <wsu:Timestamp
        wsu:Id="Timestamp-6f973f68-2115-4c32-a1d2-7c1eea225c42">
        <wsu:Created>2005-03-07T10:41:25Z</wsu:Created>
      </wsu:Timestamp>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <StockQuotes xmlns=".../wse/samples/2003/06">
      <StockQuote>
        <Symbol>FABRIKAM</Symbol>
        <Last>120</Last>
        <Date>0001-01-01T00:00:00.0000000+01:00</Date>
        <Change>0</Change>
        <Open>0</Open>
        <High>0</High>
        <Low>0</Low>
        <Volume>0</Volume>
        <MarketCap>0</MarketCap>
        <PreviousClose>0</PreviousClose>
        <PreviousChange>5.5</PreviousChange>
        <Low52Week>0</Low52Week>
        <High52Week>0</High52Week>
        <Name>Fabrikam, Inc.</Name>
      </StockQuote>
    </StockQuotes>
  </soap:Body>
</soap:Envelope>
```
3 Installation

This chapter will describe the installation of both the client and the server side on Microsoft Windows. The client side uses Windows XP with Service Pack 2 and the server side uses Windows 2003 with Service Pack 1. The problems concerning getting the hardware running on Windows will be described also. Attempts to the installation on Linux (Debian and Fedora) are described in appendix 8.4: 'Installation of the Smart Card readers on Linux'.

3.1 Client Side

As described in chapter 2.5: 'Interfaces to cryptographic hardware', Smart Cards will be used on the client side to improve security.

The client side is a laptop, which has a built-in O2Micro Smart Card Reader. In fact, a ‘Smart Card Reader’ has possibilities to read and write Smart Cards, but the term ‘Smart Card Reader’ is widespread.

From ReinerSCT, two USB Smart Card readers (ReinerSCT cyberJack pinpad) were received together with some Smart Cards (different types) and installation CD-ROMs, containing drivers and extra software.

After plugging in the reader and installing the drives, the software packages were installed and somehow explored. Only two software packages were interesting for this project and are described in this chapter. Other software includes mateSuite consisting of following items:

- smartMate: show and edit data of different Smart Card types
- cryptMate: save data on encrypted virtual hard disks, protected by a Smart Card
- passMate: save the Internet Explorer passwords (logons from web sites etc.) on a Smart Card
- loginMate: login with a Smart Card on Windows 2000/XP/2003 Server

The ‘smartMate Light’ software package provides the possibility to view and edit information of a cell phone’s SIM card.

3.1.1 cyberJack Device Manager

This package comes with a test tool for Smart Card readers. The test program connects to the PC/SC system (Personal Computer/Smart Card, a specification for Smart Card integration in computing environment) and performs some tests. Both Smart Card readers were recognized correctly and passed the test.
The program also gives a possibility to obtain information about the ‘cyberJack pinpad USB’. Because this program has been made especially for this reader, it is not possible to gather information about the ‘O2Micro Smart Card Reader’.

![Image of cyberJack Device Manager testing a Smart Card reader](image1)

**Figure 21:** Using the cyberJack Device Manager to test a Smart Card reader

![Image of cyberJack Device Manager obtaining information about a connected device](image2)

**Figure 22:** Using the cyberJack Device Manager to obtain information about the connected device

### 3.1.2 SafeSign

ReinerSCT has also provided a CD with SafeSign software from A.E.T Europe. After choosing ‘English’ for the installation, the package is installed in Dutch, the language of the Regional Options on the current computer. Although all language packs were installed, changing the language to e.g. English was not possible.

While searching a solution to this problem, a document concerning hidden options in the registry was found. This document is included on the CD (‘MICROSOFT_Administrator_Guide_SafeSign_v3.2.pdf’), as the SafeSign website doesn’t provide it no more. There are no options to set a language.

An extract of ‘MICROSOFT_TMU_Guide_SafeSign_v1.2.pdf’ (also included on the CD) tells more about this problem:
"Multi-language support has been implemented such, to create utmost flexibility for both administrator and user. <…> In practice, the language of SafeSign will default to the language set in the locale settings of the user’s computer, without the need for the user to change any settings."

Of course, a solution is to set the ‘Regional Options’ to English, configurable in the Windows Control Panel. This option has been chosen to provide screenshots in English.

Only the ‘O2Micro Smart Card Reader’ is recognized at this time. This is very strange since there was no extra software installed for this reader and the SafeSign software is provided by ReinerSCT, which is not recognized.

![Figure 23: The Token Administration Utility](image)

Upon entering a Smart Card, e.g. a ‘ReinerSCT Security Card’, a message is shown that the token is not initialized yet (‘Blanco Token’). The token can be initialized using the Token menu.

Certain fields have to be specified such as PIN and a PUK (Personal Unblocking Key). The PUK code is used in case the PIN is entered incorrectly for three times.

![Figure 24: Initialising a Token](image)

After pressing the OK button, a “Device Error (0x30)” is thrown. Nothing could be found in the manuals concerning this error.
After a while the problem was found: the ReinerSCT Security Card does not work with the SafeSign software. A white card that was provided in the package had to be inserted, because only those cards seem to be able to handle certificates and cryptographic functions. As mentioned earlier, multiple types of cards were provided by ReinerSCT. The unlabeled cards are StarCOS SPK 2.3 cards, a type of cards supported by the SafeSign software. A complete list of supported cards:

“AET middleware supports the STARCOS SPK range of smart cards developed by Giesecke & Devrient GmbH (G&D), the Rainbow iKey 3000 USB token, G&D StarKey100 USB token and Java Card 2.1.1+ compliant smart cards by G&D, GemPlus, IBM, MartSoft, Oberthur, ORGA and Schlumberger.”

Afterinitialising a new token ‘Test1’ on a StarCOS SPK 2.3 card, the status of the token was set to ‘operational’.

Now the token is operational, a certificate can be requested using the SafeSign CSP as described in chapter 2.7: ‘Requesting and issuing certificates’. The PIN has to be entered when the program asks so:

To prove that the certificate is really on the Smart Card, the ‘Token Manager’ can be opened to have a look at the ‘Registered digital IDs’:
As shown in the figure above, the certificate 'Filip Van Lerberge 7' and 'Filip Van Lerberge DemoTest2' are stored on the Smart Card, marked in red. The 'Filip Van Lerberge Soft' and other certificates are stored on the local computer as the icon shows.

Certificates will be added to the personal store ('Current User') when the Smart Card is inserted. When the Smart Card is removed, the certificates will be removed from the store, according to the SafeSign settings in the registry.

Because of possible key logging security issues, the 'ReinerSCT cyberJack pinpad' needs to be installed. As one can see on 'Figure 22: Using the cyberJack Device Manager to obtain information about the connected device', cyberJack PC/SC Service is not active. To make it active, extra drivers need to be installed, which can be found on reiner-sct.com: “PC/SC Treiber für cyberJack®, cyberJack® pinpad und cyberJack® e-com”, version 2.0.0.

Everything installed so far concerning the Smart Cards was uninstalled, including the hardware, to ensure a clean start. Afterwards, following software components are installed:

- The cyberJack Base Components (latest version on the ReinerSCT web site)
- The PC/SC Components (latest version on the ReinerSCT web site)
- The SafeSign Standard 2.0.3 package, which provides the actual CSP, from A.E.T Europe provided in the ReinerSCT box

After the installation, the ReinerSCT cyberJack pinpad is added to the list in the SafeSign software. It can be used just like the built-in O2Micro PCMCIA Reader.
The installed services in Windows on the client show that the 'cyberJack PC/SC' service has been added.

Another certificate is requested to test the newly installed ReinerSCT cyberJack. There is no possibility to enter the PIN with the computer keyboard: the cyberJack pinpad must be used, which provides more security as mentioned earlier.
3.2 Server Side

From nCipher a PCI HSM named nShield F2 was received. The HSM comes together with a Smart Card reader connected to the nShield PCI device, some Smart Cards and an installation CD-ROM containing the drivers, JAVA KeySafe software and additional tools.

![nCipher nShield F2 HSM PCI card](image)

The necessary software and the drivers for the hardware can be installed using the CD-ROM. A link to the 'nCipher CSP install wizard' will be placed on the desktop.

Using the 'nCipher CSP install wizard', an administrator can create a new ‘Security World’ by setting the module in ‘pre-initialization state’.

![Creating a new Security World](image)

A ‘Security World’ is a framework which maps security policies onto a hardware-based security infrastructure. This way, operational and administrative roles can be separated easily. It permits applications keys that are stored securely as BLOBs (Binary Large Objects) on the host to be shared between modules, backed up and recovered and ensures that they can only be used within the tamper-resistant HSM.

The wizard asks to insert Smart Cards for the administrator and eventually the operator cards and gives the possibility to set a pass phrase on them.
The JAVA KeySafe software can be used to manage the newly created ‘Security World’. Unfortunately, the installation didn’t succeed without problems.

![KeySafe software returning an error](image)

**Figure 30: KeySafe software returning an error**

An error is thrown when the KeySafe software is started. Apparently, a port must be specified in the KeySafe configuration file which is located at ‘c:\nfast\kmdata\config\config’.

![Editing the configuration for KeySafe](image)

**Figure 31: Editing the configuration for KeySafe**

The ‘priv_port’ and the ‘nonpriv_port’ must be set and the nFast service must be restarted. The KeySafe software starts correctly.
There is a possibility to protect the private key of the Certificate Authority, so that one or more Smart Cards must be inserted together with optional PINs when an administrator wants to issue a certificate. The whole CA slows down a lot. When a pass phrase is set to a Smart Card, the CA simply crashes. Protecting the CA with Smart Cards is beyond the scope of this project and therefore it will not be used.

Figure 33: The CA is ‘Not Responding’ when Smart Cards with pass phrases are used
4 Implementation

4.1 Client side

There are several approaches how a client can use XML Digital Signatures together with XML Web Services and WSE 2.0. This chapter will describe two possible approaches: using a Windows executable and using an ActiveX control in Internet Explorer. Both programs are written in the C# programming language using the .NET framework. This chapter only covers the client side. The server side (the actual Web Service) is covered in chapter 4.2: ‘Server side’.

4.1.1 Using Windows Executable

Following items need to be implemented in the test application:
1. Requesting something from the server and get a corresponding answer
2. Possibility to choose a client certificate
3. Sign the request
4. Check whether the response is signed and whether that signature is valid

A new Visual C# Project needs to be created, with a Windows Application as template.

Some components can be placed in the ‘Design View’ of the main .cs file to implement a GUI (Graphical User Interface).

![Figure 34: Implementing a GUI in ‘Design Mode’](image)

The WSE 2.0 extensions have to be added to the project. This can be obtained by right clicking on the project in the ‘Solution Explorer’ and choosing for ‘WSE Settings 2.0’.
A window appears with the possibility to enable the project for WSE 2.0.

![Figure 36: The WSE 2.0 options dialog: enabling WSE 2.0](image)

A new reference called ‘Microsoft.Web.Services2’ will be added to the project. The reference can be included using the corresponding ‘using’ statements in the code of the main .cs file.

```csharp
using Microsoft.Web.Services2;
```

**Code 6: Adding ‘using’ statements for WSE 2.0**

A new ‘Web Reference’ has to be added to be able to use the Web Service using the ‘Add Web Reference’ option in the ‘Project’ menu. The correct URI must be entered and ‘WebReference’ can be used as the name. Just like the other references, Web Reference namespaces can be added by ‘using’ statements. ‘System.Security.Cryptography’ will also be used.

```csharp
using WSEClient2.WebReference;
```

**Code 7: Adding ‘using’ statements for the Web Reference and Cryptography**
The Web Service itself can be instantiated by its name, followed by 'Wse'. If the name of the Web Service is ‘Service1’, the name to call it using WSE will be ‘Service1Wse’. This strange approach can be explained when the ‘Show All Files’ option is set in the ‘Solution Explorer’.

In the Reference.cs file, there is a new class ‘Service1Wse’ added next to the existing class ‘Service1’. Both classes are in fact the same, but instead of deriving from ‘System.Web.Services.Protocols.SoapHttpClientProtocol’, Service1Wse derives from ‘Microsoft.Web.Services2.WebServicesClientProtocol’.

The following code and its comments describe how certificates are used, how the connection to the Web Service is made and how the response from the server is checked for signing.

```
// First the number has to be parsed from string to integer
int number = int.Parse(textBoxNumber.Text);

// Go to the Current User certificate store, open it and look for
// the WSE2QuickStartClient certificate
X509CertificateStore store =
    X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore);
store.OpenRead();
X509CertificateCollection certs =
    store.FindCertificateBySubjectName("CN=WSE2QuickStartClient");

// If there are certificates with this name, select the first one
if (certs.Count > 0)
    signatureToken = new
            certs[0]);

// Define a new object from the Service1Wse type
Service1Wse svc = new Service1Wse();

// Define a new SOAP Context object, add the token and specify
// the security token to sign the message with
SoapContext requestContext = svc.RequestSoapContext;
requestContext.Security.Tokens.Add(signatureToken);

// Execute the request
string serverResponse = svc.getNumber(number);

// Check whether the response from the server is signed or not
SoapContext responseContext = svc.ResponseSoapContext;
{
    if (element is MessageSignature)
{ MessageSignature sig = element as MessageSignature;
 if ((sig.SignatureOptions & SignatureOptions.IncludeSoapBody) == 0)
     MessageBox.Show("The response from the server is not signed");
}

// Set the response from the server in labelOutput
labelOutput.Text = serverResponse;

Code 8: Using a certificate, connecting to the server and checking the server response

At this moment, the client is able to connect to the Web Service using the statically configured WSE2QuickStartClient certificate. Now an option will be added to be able to choose a certificate.

Following code fragments describe what happens when the ‘Change’ button will be pressed. Only small fragments are shown, the whole code can be viewed on the CD.

// Go to the Current User certificate store
X509CertificateStore store =
    X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore);

// Define a new object ccc from the chooseClientCert class
// and show the dialog
chooseClientCert ccc = new chooseClientCert(store);
ccc.ShowDialog();
token = new

Code 9: Code executed when the ‘change’ button is pressed

// Constructor
public chooseClientCert(X509CertificateStore store)
{
    _store = store;
    InitializeComponent();
    _certList.Columns.Add("Name", 200, HorizontalAlignment.Left);
    _certList.Columns.Add("Issued By", -2, HorizontalAlignment.Left);
    _certList.Columns.Add("Full Name", -2, HorizontalAlignment.Left);
    _certList.Columns.Add("Certificate ID", -2, HorizontalAlignment.Left);
    ...

    // Add all certificates from the specified store to the list
    X509CertificateCollection coll = _store.Certificates;
    foreach (X509Certificate cert in coll)
    {
        ListViewItem item = new CertificateListViewItem(cert);
        ...
As it would be nice to be able to obtain more information about a certificate, another approach will be used: the ‘cryptoui’ DLL.

The DLL is unmanaged code (C++) and can be used by managed code such as C#. Therefore, another using statement must be added and the DLL must be imported.

```csharp
using System.Runtime.InteropServices;
...
[DllImport("cryptui", CharSet=CharSet.Unicode, SetLastError=true)]
internal extern static IntPtr CryptUIDlgSelectCertificateFromStore(IntPtr hCertStore, IntPtr hwnd, string pwszTitle, string pwszDisplayString, uint dwDontUseColumn, uint dwFlags, IntPtr pvReserved);
```

One of the disadvantages of this approach is that this DLL can only be used in Windows XP. This way, a check has to be done whether the current Operating System (OS) is Windows XP or higher.

```csharp
static bool IsWinXP()
{
    OperatingSystem os = Environment.OSVersion;
    Version v = os.Version;
    if ( os.Platform == PlatformID.Win32NT && v.Major >= 5 && v.Minor >= 1 )
        return true;
    return false;
}

// If Windows XP or higher is the current OS, the DLL can be used
if (IsWinXP())
{
    IntPtr certPtr = CryptUIDlgSelectCertificateFromStore(store.Handle, IntPtr.Zero, "Select Certificate", "Choose a certificate below", 0, 0, IntPtr.Zero);
    if ( certPtr != IntPtr.Zero )
    {
        X509Certificate certificate = new X509Certificate(certPtr);
        return new X509SecurityToken(certificate);
    }
}
```
4.1.2 Moving the code into a Class Library

To move towards a more structured approach and to be able to reuse the client's code in an ActiveX object, a class library (DLL) will be created. This can be done by creating a new project within the current solution with a ‘Class Library’ template. The code from the client, with exception of the Main() method, must be copied into a new User Control in the newly created ‘Class Library’. Because the ‘Class Library’ itself will connect to the Web Service, WSE must be enabled and can be disabled in the previous project. The chooseClientCert.cs can be copied without adaptation.

The original Form1.cs in the ‘Windows Form’ project can be excluded and a new ‘Windows Form’ file can be created. A new using statement must also be included as the ‘Class Library’ will be used.

After compiling the DLL project, the DLL can be included in a new ‘Windows Form’ project. Therefore, the DLL must be added to the toolbox which can be done by the ‘Tools’ menu, ‘Add/Remove Toolbox Items’ and browsing for the DLL. When the new item is added in the toolbox, it can easily be dragged on the ‘Windows Form’ in ‘Design View’.

The last step is to call the UserControl1() after the InitializeComponent() call.

```csharp
using WseDLL;
...
public windowsForm()
{
    InitializeComponent();
    UserControl1 uc = new UserControl1();
}
```
4.1.3 Using an ActiveX control

First a new project must be created with C# as project type and ‘Class Library’ as template. Because ActiveX controls only run properly when they are called on a web server (i.e. using http://server/ instead of file://c:/folder/), the best place to create the project is on the local web server.

After adding a new ‘User Control’ to the project, the previously created UserControl1 (DLL) can be added in ‘Design View’.

After building the solution, all files need to be visible in the current project. This way the bin/debug/ folder can be reached where a new HTML page must be created by choosing ‘HTML Page’ as ‘New Item’. Now we add another ‘New Item’ to the project: a HTML Page called e.g. index.htm. This file must be dragged to the folder where the DLL files are located, which is bin/debug/.

In the HTML code of index.htm, a new object tag must be added between the body tags:

```html
<Object id="Client" name="Client" classid="AX1.dll#AX1.AXWSEClient"
 VIEWASTEXT></Object>
```

The ‘classid’ is the most important part, as this part will tell the browser where to look for the ActiveX control.

When the corresponding URL is opened with Internet Explorer, the ActiveX control will appear on the site:
When the 'Change' button is clicked, users get a 'Security error'.

ActiveX control signing has been tried to solve this problem. This can be done by the 'signcode.exe' which resides in an optional package from Microsoft named 'Authenticode Tools'. A DLL or EXE can be chosen together with a certificate and some other options. ActiveX control signing did not solve the problem described above.

The 'Security error' is thrown by the .NET framework itself. There are two possible solutions to this problem.

The first possibility is to use the global 'caspol.exe' to turn all security checking off.

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322>caspol -s off
Success
```

Code 15: Using 'caspol.exe' to turn the global .NET security settings off
This approach is not really safe. Therefore, another option exists: using the ‘Microsoft .NET Framework 1.1 Wizards’ which can be found under ‘Administrative Tools’ in the ‘Control Panel’.

![Figure 44: The .NET Wizards](image)

The best approach is to set the rights for ‘Trusted sites’, and not for ‘Internet’ or ‘Local intranet’. Therefore, the web site containing the ActiveX control must be added to the ‘Trusted sites’ in Internet Explorer.

![Figure 45: Adding a web site to ‘Trusted sites’](image)

Next, the ‘Adjust .NET Security’ wizard is started and ‘Full Trust’ is given to ‘Trusted Sites’, so all the resources on the local machine can be accessed.

The Web Service can be contacted using an ActiveX control.
4.2 **Server side**

This chapter covers the implementation of the server side, with software certificates and with hardware certificates. In both cases, the service will sign the response and therefore it needs access to a certificate and its corresponding private key.

### 4.2.1 Using software certificates

A new Visual C# Project needs to be created, with an ‘ASP.NET Web Service’ as template. Just like on the client side, the WSE can be enabled by right clicking on the project in the ‘Solution Explorer’ and choosing for ‘WSE Settings 2.0’. The WSE Soap Extensions must also be enabled since this project requires so.

![Figure 47: The WSE 2.0 settings: enabling WSE and WSE SOAP Extensions](image)

Again, using statements have to be added to the Service1.asmx.cs file.
using Microsoft.Web.Services2;
using System.Security;

Code 16: Adding ‘using’ statements for WSE 2.0 and Cryptography

Now a new ‘Web Method’ can be made. As seen in the code from the client, the getNumber(int number) method will be implemented. A certificate will be statically configured because the certificate shouldn't change as much as a client certificate.

[WebMethod(Description="This testing method returns the requested nr." )]
public string getNumber(int number)

Code 17: Making a simple Web Method

The implementation itself is in many parts comparable to the one at the client side.

// Place the current SOAP request in a new object
SoapContext requestContext = RequestSoapContext.Current;

// Check whether the client’s request is signed or not
// and place the token in a new variable so it can be used later
foreach (ISecurityElement element in requestContext.Security.Elements) {
    if (element is MessageSignature) // If context contains a DSIG
    {
        MessageSignature sig = element as MessageSignature;
        if (((sig.SignatureOptions & SignatureOptions.IncludeSoapBody) == 0)
            throw new ApplicationException("The request is not signed.");
        clientToken = sig.SigningToken as X509SecurityToken;
    }
}

// Build a string with a greeting of the requestor and the requested number
outputString += "Hello " + clientToken.Certificate.FriendlyDisplayName;
switch (number) {
    case 1: outputString += " you asked for number 1"; break;
    case 2: outputString += " you asked for number 2"; break;
    default: outputString += " you asked for an unsupported number"; break;
}

// Place the SOAP response in a new object

// Open the ‘Local Machine’ store and search for the certificate
X509CertificateStore store =
    X509CertificateStore.LocalMachineStore(X509CertificateStore.MyStore);
X509CertificateCollection certs =
    store.FindCertificateBySubjectName("CN=WSEQuickStartServer");
if (certs.Count > 0)
    X509SecurityToken serverSignatureToken = new
certs[0]);

// Check whether the certificate was found or not
if (serverSignatureToken == null)
    throw new ApplicationException("Unable to find server certificate.");
// Check if the certificate supports Digital Signatures
if (!serverSignatureToken.SupportsDigitalSignature)
    throw new ApplicationException("Certificate does not allow DSIG.");

// Add the security token
responseContext.Security.Tokens.Add(serverSignatureToken);

// Specify the security token to sign the message with

// Return the string to the client
return outputString;

Code 18: Server side implementation

The code described above compiles without problems and the certificate can be found. However an annoying error is thrown at the client side coming from the server: ‘Server unavailable, please try again later’.

The reason for this behaviour was found with the freeware tools FileMon and RegMon from Sysinternals. Following table is an extract of the FileMon tool and explains what is wrong:

<table>
<thead>
<tr>
<th>Process</th>
<th>w3wp.exe:3708</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>OPEN</td>
</tr>
<tr>
<td>Path</td>
<td>C:\Documents and Settings\All Users\Application Data\Microsoft\Crypto\RSA\MachineKeys\d24d532a235e548ac4925aca66cbea4_5_8c62e76e-2cd0-4fee-ad3e-5b36b77ae262</td>
</tr>
<tr>
<td>Result</td>
<td>ACCESS DENIED</td>
</tr>
<tr>
<td>Other</td>
<td>NT AUTHORITY\NETWORK SERVICE</td>
</tr>
</tbody>
</table>

The ‘Network Service’ account tries to read a file named ‘d24d532a235e548ac4925aca66cbea45_8c62e76e-2cd0-4fee-ad3e-5b36b77ae262’ and it cannot be accessed.

Using the 'WSE X.509 Certificate Tool' which is installed by default by the WSE package, a certificate can be chosen and the corresponding private key properties can be viewed and rights can be altered.
The 'View Private Key File Properties' button must be clicked and a new window will be opened. The 'Network Service' account must be given read rights on the private key.

This way, the Web Service can be used with the 'WSE2QuickStartServer' certificate and no errors appear.
4.2.2 Using hardware certificates

When hardware certificates are used, i.e. certificates protected by the nCipher CSP, the solutions used for the software certificates are not usable: the private keys are protected by the HSM and are in no way readable for the outside world. Another solution needs to be found.

As mentioned before, ASP.NET applications are run by the ‘Network Service’ account. This account has restricted rights and therefore, private keys cannot be accessed. ASP.NET provides the possibility to change the user that runs the code, called Impersonation.

There are two possibilities to use ASP.NET Impersonation:
- Adapting the global machine.config file in the C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\CONFIG\ folder
- Adapting the web.config file that exists per server

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="microsoft.web.services2"
             type="Microsoft.Web.Services2.Configuration.WebServicesConfiguration,
             Microsoft.Web.Services2, Version=2.0.0.0, Culture=neutral,
             PublicKeyToken=31bf3856ad364e35"/>
  </configSections>
  <system.web>
    <identity impersonate="true" userName="supervisor" password="secret" />
  </system.web>
...  

Code 19: Using ASP.NET impersonation with web.config

The last option is chosen because provides more flexibility.

The Web Service can be used with certificates using the nCipher CSP to protect private keys.

4.2.3 Expanding the client/server to support Data Encryption

So far, the client signs a request, sends it to the server and gets a signed response. As explained in 2.2: ‘Why not using SSL (only)?’, following items are achieved:
- **Non-repudiation**: the client cannot deny that he sent the message, nor can the server
- **Authenticity**: Both the client and the server know who they are corresponding with
- **Integrity**: The data is not altered

To accomplish this, the client uses its private key to generate a signature. It sends the message together with its public key to the server. The server uses the public key of the client to check whether the signature is valid or not. If the signature is not valid any more, the message is altered and will be discarded.

Imagine a service using critical user data such as credit card information. The message must be signed but the need for another item arises: the data needs to stay confident. Therefore, data encryption is used.
To accomplish data encryption, the client needs to use the public key of the server. The client encrypts its message with this key. The only possibility to decrypt the data is to use the corresponding private key, which the server has.

After decrypting the request, the server uses the public key of the client (sent along with the request) to encrypt the response. When the data arrives at the client side, the client’s private key is used to decrypt the message.

WSE includes the possibility to use data encryption. Both the client and the server side need to be extended. First, some extra components will be placed on the client in ‘Design View’:

- An option to encrypt the request
- An option to encrypt the response

![Figure 51: Expanding the WSE Client to encrypt the request and/or the response](image)

Next, some code will be added to encrypt the request. When the signature is added, the data will be encrypted according to the checkbox. The Web Service will be called with an extra parameter (boolean) telling whether the response should be encrypted or not.

```csharp
// Specify the security token to sign the message with

// If the checkbox for encryption is checked, the data will be encrypted
if (checkBoxEncReq.Checked)
{
    // Open the current user store and look for a certificate named
    // ‘WSE nC ServerAuth 2’
    X509CertificateStore store =
        X509CertificateStore.CurrentUserStore(X509CertificateStore.MyStore);
    X509CertificateCollection certs =
        store.FindCertificateBySubjectName("CN=WSE nC ServerAuth2");
```
  // Make a new EncryptedData object and add it to the request context
  EncryptedData enc = new EncryptedData(serverToken);
  requestContext.Security.Elements.Add(enc);
}

// Call the Web Service with an extra param concerning response encryption
string serverResponse = svc.getNumber(number, checkBoxEncResp.Checked);

Code 20: Expanding the client to use encryption

At the server side, less code will be added because the client sends its public key to the server due to the digital signing. This way the server side can use the clientToken variable already defined in Code 18: Server side implementation'. As mentioned before, an extra parameter will be added to the Web Method.

[WebMethod(Description="This testing method returns the requested nr.")]
public string getNumber(int number, bool encrypt)
{
  ...
  if (encrypt)
  {
    // Check whether the client token supports data encryption
    if (!clientToken.SupportsDataEncryption)
      throw new ApplicationException("No encryption support");

    // Use the client token (public key) to encrypt the response
    EncryptedData enc = new EncryptedData(clientToken);
    responseContext.Security.Elements.Add(enc);
  }
}

Code 21: Expanding the server to use encryption

While testing the data encryption, strange errors occurred. Encryption for the request is not a problem, but when the response needs to be encrypted, an error occurs.

This is a bug in the implementation of WSE 2.0: WSE looks for the certificate (and the corresponding private key) in the wrong certificate store. As Code 20: Expanding the client to use encryption' describes, the certificate should be looked for in the 'Current User' store. When the certificate is placed in the 'Machine Store', the response encryption works without problems.
4.3 Signing Benchmark Tool

When the nCipher CSP is used to perform cryptographic operations, it seems to be slower than using the Microsoft Enhanced CSP. This doesn’t seem normal, as the hardware has its own processor especially designed to perform cryptographic operations; thus, it should be faster.

According to the online nCipher nShield F2 specs, 150 RSA operations per second should be possible.

To compare the time needed to do cryptographic operations, a real-world-simulating benchmark-tool in C# has been made:

This tool can do two kinds of benchmarking:
- **Hash signings** using the SignHash(byte[], string oidHash) method from the RSACryptoServiceProvider class in the Microsoft.Web.Services2.Security.Cryptography Namespace
- **XML Digital Signatures** by calling a web service using message signing with the WSE 2.0 extensions
At the Web Service side, a simple doBench() method is implemented to check whether the SOAP body is signed or not.

```csharp
[WebMethod(Description="XML Signatures Benchmarking Method")]
public void doBench()
{
    SoapContext requestContext = RequestSoapContext.Current;
    if (requestContext == null)
        throw new ApplicationException("Non SOAP-body recieved");

    // Check whether the client’s request is signed or not
    {
        if (element is MessageSignature)
        {
            MessageSignature sig = element as MessageSignature;
            if ((sig.SignatureOptions & SignatureOptions.IncludeSoapBody) == 0)
                throw new ApplicationException("The request is not signed.");
        }
    }
}
```

**Code 22: Implementing a simple Web Service**

At the Benchmark tool side, the same certificate dialog as in the WSEClient has been implemented. Next, the two different benchmark options are implemented.

- **For the Hashing:**

```csharp
    token.Certificate;

// Defining a plain text to be hashed
string plainText = "X0X0X0X0X0";
byte[] plain = new UnicodeEncoding().GetBytes(plainText);
byte[] cipher;
string cryptoconfig = CryptoConfig.MapNameToOID("SHA1");
for (int i = 0; i < amount; i++)
{
    // textBoxOutput.Text += i+1;
    // if (i < amount-1)
    //    textBoxOutput.Text += ", ";
    // textBoxOutput.Update();
    cipher = rsa.SignHash(plain, cryptoconfig);
}
```

**Code 23: Implementing the hashing benchmarks**

It is possible to show the current hashing number in real-time in the output box, but this slows down the application quite a lot. Therefore these lines are commented out in the code shown above.

The amount of seconds is calculated by subtracting two DateTime objects to achieve a TimeSpan object (not shown in the code).
For the XML Signatures:

```csharp
Service1Wse svc = new Service1Wse();
SoapContext requestContext = svc.RequestSoapContext;

for (int i = 0; i < amount; i++)
{
    // Add the security token and specify the signing token
    requestContext.Security.Tokens.Add(token);
    requestContext.Security.Elements.Add(new MessageSignature(token));
    svc.doBench();

    // Add the security token
    // Specify the security token to sign the message
}
```

**Code 24: Implementing the XML Signatures benchmarks**

Two certificates were used to test:
- "BenchTest with Microsoft CSP": a certificate using the MS Enhanced CSP
- "BenchTest with nCipher CSP": a certificate using the nCipher Enhanced CSP

The benchmarking was done on an Intel Pentium 4 at 3.0 GHz with Hyper Threading enabled.

The calling of the Web Service was tested over a GigaBit network and on localhost. There was no difference between the results.

In the following table the results of the benchmarks are shown:

<table>
<thead>
<tr>
<th>Type</th>
<th>Certificate</th>
<th>nFast server</th>
<th>10x</th>
<th>100x</th>
<th>1000x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash</td>
<td>Microsoft</td>
<td>On</td>
<td>0.1 s</td>
<td>1.06 s</td>
<td>10.6 s</td>
</tr>
<tr>
<td>Hash</td>
<td>nCipher</td>
<td>On</td>
<td>0.1 s</td>
<td>1.06 s</td>
<td>10.5 s</td>
</tr>
<tr>
<td>Hash</td>
<td>Microsoft</td>
<td>Off</td>
<td>0.03 s</td>
<td>0.35 s</td>
<td>3.6 s</td>
</tr>
<tr>
<td>XML</td>
<td>Microsoft</td>
<td>On</td>
<td>0.35 s</td>
<td>3.35 s</td>
<td>33.5 s</td>
</tr>
<tr>
<td>XML</td>
<td>nCipher</td>
<td>On</td>
<td>0.45 s</td>
<td>4.6 s</td>
<td>46 s</td>
</tr>
<tr>
<td>XML</td>
<td>Microsoft</td>
<td>Off</td>
<td>0.26 s</td>
<td>2.6 s</td>
<td>26 s</td>
</tr>
</tbody>
</table>

There are some very strange results:
- The hashing using a certificate enrolled with the MS CSP is three times faster when the nFast server is turned off.
- There is a clear difference between XML signing using a certificate enrolled by the MS CSP and the one enrolled by the nCipher CSP.
- There is no difference between the Hash signings.

nCipher has been informed about these problems, but until now they didn't provide an answer solving all questions above.

Finally, 'csptest.exe', a tool provided with the nCipher nShield, has been run with the nFast server turned on and off and the results were compared. The numbers in the table represent signature operations per second in an average of 10 seconds.
<table>
<thead>
<tr>
<th>CSP</th>
<th>nFast off</th>
<th>nFast on</th>
</tr>
</thead>
<tbody>
<tr>
<td>nCipher Enhanced Cryptographic Provider</td>
<td>Error 0x80090011</td>
<td>149.4</td>
</tr>
<tr>
<td>nCipher Enhanced DSS and Diffie-Hellman</td>
<td>Error 0x80090011</td>
<td>295.6</td>
</tr>
<tr>
<td>Cryptographic Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nCipher DSS Signature Cryptographic Provider</td>
<td>Error 0x80090011</td>
<td>299.2</td>
</tr>
<tr>
<td>Microsoft Enhanced RSA and AES Cryptographic Provider</td>
<td>184.4</td>
<td>150.2</td>
</tr>
<tr>
<td>Microsoft Base Cryptographic Provider v1.0</td>
<td>1558.4</td>
<td>650</td>
</tr>
<tr>
<td>Microsoft Base DSS and Diffie-Hellman</td>
<td>418.6</td>
<td>299.3</td>
</tr>
<tr>
<td>Cryptographic Provider</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Enhanced DSS and Diffie-Hellman</td>
<td>367.7</td>
<td>298.9</td>
</tr>
<tr>
<td>Cryptographic Provider</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Again, there is a big difference between using a MS CSP with the nFast server turned off, and the server turned on. The nShield seems to slow down the MS CSP a lot.
5 Conclusion

All parts of the topic succeeded: in the end I am able to sign and/or encrypt the request sent to the Web Service using X.509 certificates residing on a Smart Card. A PIN must be entered using the pinpad, this way the Smart Card is safe for key logging. For the server side, signing and/or encrypting the response using X.509 certificates protected by the nCipher HSM is no problem.

Since topics like Web Services and security are becoming more and more important, I’m very glad Prof. Dr. Heuert proposed this project to me and it was a pleasure to work together. I learned lots of new technologies last three months from different subjects and how they fit together to form a coherent whole.

Of course sometimes problems occurred. They were often very unclear: the problem could arise from the Operating System, from the Visual Studio, from the WSE 2.0 implementation or from something else. This is mainly due to the fact that the used technologies and products are all quite recent and therefore sometimes not so good documented.

Because this was the first time that the nCipher HSM was used together with .NET (as mentioned before, the HSM is mainly used for speeding up and securing SSL tunnels), several questions are still not solved at this moment of writing. nCipher has been contacted several times and in the last reply they told that they will contact us for their .NET support in the future:

You know, I may be asking for your help later as I shall be involved with ensuring that we support .NET as best we can for the future. I’ll be in touch again as soon as I can.
6 Lists

6.1 List of figures

Figure 1: Using SSL to encrypt network transport ......................................................... 6
Figure 2: Using XML Digital Signatures to assure long-term authenticity, integrity and non-repudiation ................................................................. 7
Figure 3: A Smart Card used for the health system .......................................................... 8
Figure 4: An example of a Certificate Chain using an offline Root CA............................ 10
Figure 5: X.509 version 3 certificate structure ................................................................. 10
Figure 6: X.509 version 3 certificate with only version 1 fields shown ............................ 11
Figure 7: X.509 version 3 certificate with only the extensions shown ............................ 12
Figure 8: Setting the certificate properties when the EKU extension is not set .............. 12
Figure 9: Different certificate store locations in MMC with their corresponding folders .. 13
Figure 10: Requesting a certificate using the certificates snap-in ...................................... 14
Figure 11: An Active Directory must be installed ............................................................ 14
Figure 12: Requesting an E-Mail Protection Certificate .................................................... 15
Figure 13: Selecting an installed CSP ................................................................................ 16
Figure 14: Defining a special OID using the ‘Advanced Certificate Request’ ..................... 16
Figure 15: Issuing the certificate in the CA ........................................................................ 16
Figure 16: Installing certificates involves a potential security risk .................................... 17
Figure 17: Certificate Revocation List store locations ...................................................... 18
Figure 18: The content of a CRL shows different revoked certificates ............................. 18
Figure 19: Running the sample X509Signing program ..................................................... 20
Figure 20: Enabling ‘Message Trace’ in the WSE 2.0 options dialog ............................... 20
Figure 21: Using the cyberJack Device Manager to test a Smart Card reader ................. 27
Figure 22: Using the cyberJack Device Manager to obtain information about the connected device ................................................................................. 27
Figure 23: The Token Administration Utility ................................................................. 28
Figure 24: Initialising a Token ......................................................................................... 28
Figure 25: Entering a PIN ............................................................................................... 29
Figure 26: List of certificates and where they reside ......................................................... 30
Figure 27: The Token Administration Utility with the ReinerSCT cyberJack ................. 31
Figure 28: The nCipher nShield F2 HSM PCI card ......................................................... 32
Figure 29: Creating a new Security World ....................................................................... 32
Figure 30: KeySafe software returning an error ............................................................... 33
Figure 31: Editing the configuration for KeySafe ............................................................. 33
Figure 32: The KeySafe software introduction page ....................................................... 34
Figure 33: The CA is ‘Not Responding’ when Smart Cards with pass phrases are used .... 34
Figure 34: Implementing a GUI in ‘Design Mode’ ............................................................ 35
Figure 35: The ‘WSE Settings 2.0’ menu ....................................................................... 36
Figure 36: The WSE 2.0 options dialog: enabling WSE 2.0 ............................................. 36
Figure 37: The ‘Solution Explorer’ with all files shown .................................................. 37
Figure 38: Choosing a certificate ..................................................................................... 38
Figure 39: Choosing a certificate using CryptoUI ........................................................... 39
Figure 40: Placing a ‘Class Library’ in the ‘Design Mode’ ................................................ 40
Figure 41: Placing an index.htm file in the bin/Debug/ folder ........................................ 41
Figure 42: Opening the ActiveX control in Internet Explorer ........................................... 42
Figure 43: Security Error from the .NET platform ......................................................... 42
Figure 44: The .NET Wizards ......................................................................................... 43
Figure 45: Adding a web site to ‘Trusted sites’ ............................................................... 43
Figure 46: Using the ActiveX control in Internet Explorer: showing the certificates ......... 44
Figure 47: The WSE 2.0 settings: enabling WSE and WSE SOAP Extensions ............... 44
Figure 48: An error thrown by the server .................................................................46
Figure 49: Using the WSE X.509 Certificate Tool .................................................. 47
Figure 50: Setting rights on the private key for the 'Network Service' account ...... 47
Figure 51: Expanding the WSE Client to encrypt the request and/or the response 49
Figure 52: Unable to retrieve the referenced token ................................................. 50
Figure 53: Screenshot of the certificate benchmark tool .......................................... 51
Figure 54: Visiting a HTTPS site with a valid certificate page using Internet Explorer 70
Figure 55: Importing the CRL in Mozilla Firefox ..................................................... 70
Figure 56: Visiting a HTTPS site with an 'Unspecified' revoked certificate using Mozilla Firefox .................................................................................................................. 71
Figure 57: Enabling the server certificate revocation ................................................. 71
Figure 58: Visiting a HTTPS site with a 'Certificate Hold' revoked certificate using Mozilla Firefox .................................................................................................................................... 71
Figure 59: Choosing a digital certificate for the client side ....................................... 72
Figure 60: Finding out the Identifier from a Web Site in IIS6 ..................................... 72
Figure 61: The Token Administration Utility in Debian Linux with its error ............... 74
Figure 62: The content of a certificate request .......................................................... 78
Figure 63: Importing a certificate .............................................................................. 79
Figure 64: Comparing two certificates using 'Total Commander' ............................. 80
Figure 65: Configuring the security settings of an email account in Microsoft Outlook Express ........................................................................................................................................... 81
Figure 66: Signing and/or Encrypting an email in Microsoft Outlook Express .......... 81
Figure 67: The advanced options panel in Mozilla Thunderbird ............................... 82
Figure 68: Adding a PKCS#11 device to Mozilla Thunderbird ................................. 82
Figure 69: The 'Device Manager' shows the newly added module ............................ 82
Figure 70: Configuring a certificate for email signing and/or encryption .................. 83
Figure 71: Sending a digitally signed email using Mozilla Thunderbird .................... 83
Figure 72: Trusting the CA to identify email users ..................................................... 83
Figure 73: Enabling the project for WSE .................................................................. 84
Figure 74: Security settings for certificates .............................................................. 85
6.2 List of code fragments

Code 1: XML file with the request and its digital signature ......................................................... 23
Code 2: Calculation of DigestValue .......................................................................................... 23
Code 3: Calculation of SignatureValue .................................................................................... 23
Code 4: Using GUIDumpASN to view the ASN.1 content of a certificate ............................... 24
Code 5: XML file with a possible response without digital signature ....................................... 25
Code 6: Adding ‘using’ statements for WSE 2.0 ....................................................................... 36
Code 7: Adding ‘using’ statements for the Web Reference and Cryptography ....................... 36
Code 8: Using a certificate, connecting to the server and checking the server response .......... 38
Code 9: Code executed when the ‘change’ button is pressed .................................................. 38
Code 10: Small fragment out of the chooseClientCert code .................................................... 39
Code 11: Importing a DLL written in unmanaged code ............................................................. 39
Code 12: Checking the OS version and using the DLL if it can be used .................................... 40
Code 13: Calling the newly created DLL ................................................................................... 41
Code 14: Adding a new Object to the HTML source ................................................................ 41
Code 15: Using ‘caspol.exe’ to turn the global .NET security settings off ............................... 42
Code 16: Adding ‘using’ statements for WSE 2.0 and Cryptography ....................................... 45
Code 17: Making a simple Web Method ....................................................................................... 45
Code 18: Server side implementation ......................................................................................... 46
Code 19: Using ASP.NET impersonation with web.config ....................................................... 48
Code 20: Expanding the client to use encryption ........................................................................ 50
Code 21: Expanding the server to use encryption ..................................................................... 50
Code 22: Implementing a simple Web Service ............................................................................ 52
Code 23: Implementing the hashing benchmarks ....................................................................... 52
Code 24: Implementing the XML Signatures benchmarks ......................................................... 53
Code 25: Converting Fedora to Debian packages with the ‘alien’ tool ...................................... 73
Code 26: Errors concerning shared libraries .............................................................................. 73
Code 27: Creating a symbolic link to the shared library ‘libtiff.so’ ............................................. 73
Code 28: Creating symbolic links to other shared libraries ....................................................... 73
Code 29: Error concerning GTK ............................................................................................... 74
Code 30: A possible patch for the GTK problem ........................................................................ 75
Code 31: Usage of OpenSSL ..................................................................................................... 76
Code 32: Example of an OpenSSL configuration file ................................................................. 77
Code 33: Requesting a certificate using OpenSSL .................................................................... 78
Code 34: Comparing two certificates using OpenSSL ............................................................... 79
Code 35: The ‘app.config’ file with its settings ........................................................................ 85
7 References

References for this project were mainly found on the Internet because relative new technologies are used. First the link to web sites will be given together with a short description or title and then the Adobe PDF files will be listed. These PDF files were also found on the Internet, but are included on the CD.

7.1 Web Sites

7.1.1 XML Web Services and its security

- http://loop.interop.com/comments.php?id=131_0_1_0_C
  LOOP : Securing your enterprise: Web services in a suspicious world

  nCipher Security Insights: Securing Web Services

  Overview of the Web Services Enhancements

  Securing Web Services with WSE 2.0

  WSE 2.0 Tracing Utility

- http://www.rsasecurity.com/rsalabs/node.asp?id=2124
  RSA Security - Public-Key Cryptography Standards (PKCS)

  Cryptomathic Technical Articles - Cryptography In A Hole - Cryptographic Service Providers

  SOAP Security Extensions: Digital Signature

7.1.2 XML Signatures and WSE 2.0

- http://www.w3.org/TR/xmlsig-core/
  XML-Signature Syntax and Processing

  NO-SOAP Xml Signature with WSE and Certificates

  Vordel SOAPbox - free Web Services security test tool

  Web Services Security (WS-Security)
• http://pages.infinit.net/ctech/20040316-0936.html
  Poupou on "Xml Digital Signature Status" (3/16/2004)

• http://webservices.xml.com/pub/a/ws/2003/01/15/ends.html
  webservices.xml.com: Securing Web Services

• http://www.geminisecurity.com/guidumpasn.html
  GUIDumpASN Download Page

  Technical Articles on MSDN concerning XML DSIG

• http://msdn.microsoft.com/msdnmag/issues/04/08/xmlfiles/default.aspx
  The XML Files: What's New in WSE 2.0 -- MSDN Magazine, August 2004

  The Code Project - An Introduction To Web Service Security - Part II - C++ Web Services

• http://www.codeproject.com/cs/webservices/WSSecurity.asp

• http://blogs.msdn.com/johanl/articles/272768.aspx
  Sign a SOAP message with certificates stored on a SmartCard!

• http://www.windowsitlibrary.com/Content/1219/06/3.html
  Secure Web Services with WS-Security

• http://sys-con.com/story/?storyid=45103&de=1
  Web Services Journal

• http://www.thedatafarm.com/blog/CategoryView.aspx?category=WSE
  Julia Lerman Blog - Don't Be Iffy... - WSE

• http://www.devx.com/dotnet/Article/19986/0/page/1
  Securing .NET Web Services with the WS-Security Protocol

  Securing Web Services on MDSN

  X.509 Certificate CLIENT on MSDN

  X.509 Certificate SERVER on MSDN
7.1.3 Certificates and cryptography

Creating, Viewing, and Managing Certificates

About Cryptography on MSDN
7.1.4 ActiveX

7.1.5 Smart Cards, A.E.T Europe, SafeSign and ReinerSCT

  Smart Card Functions on MSDN

  Smart Cards on MSDN

  How can I call API SCardTransmit(........) from C#

  Advanced Encryption Technology Europe BV

  Security :: PKI :: doc :: Smart Cards :: StarSign

- http://www.motechno.com/produkte/STARCOSSPK2.3.htm
  MoTechno : Kartenleser, Fingerprint, Smart Card Reader

  XT-Commerce - Kryptokarte + Software Kryptokarte

- http://electronics.howstuffworks.com/question332.htm
  Howstuffworks "What is a "smart card"?"

7.1.6 CRL and OCSP

- http://weblogs.asp.net/israelio/archive/2005/03/14/394513.aspx
  Certificate Revocation List (CRL) - Client's Cache

  Digitally Signing .NET Assemblies & ocsp

  .NET Renaissance adsutil
7.1.7 Linux

- http://www.mono-project.com/Cryptography
  Cryptography - Mono

- http://wiki.debian.net/?Mono
  Mono - DebianWiki

- http://64.179.4.149/questions/history/215559
  LinuxQuestions.org - Can't compile pgadmin3

  Re: undefined symbol

  Patch: Comment out calls to _gtk_accel_group_attach

  Patch for gtk_accel_group_attach

7.1.8 Other

- http://www.sysinternals.com/
  Sysinternals Freeware

- http://www.openca.org/
  The OpenCA project

  Install OpenSSL on Windows; Create, Sign Root Certificate and CSR

- http://csp11.labs.libre-entreprise.org/
  Cryptographic Service Provider #11: a wrapper between PKCS#11 and a CSP
7.2 Adobe PDF files

Following alphabetic list of documents can be found in the ‘References’ folder on the CD. The document name is not altered since it was downloaded from the internet.

- **02_WSE_c.pdf**
  WSE Routing
  http://www.cs.auckland.ac.nz/compsci734s1t/lectures/radu/02_WSE_c.pdf

- **AET.SafeSign_KCA6.pdf**
  RSA Keon Ready Implementation Guide For PKI 3rd Party Applications

- **a-security-tool-for-web-services.pdf**
  TulaFale: A Security Tool for Web Services
  http://research.microsoft.com/users/adg/publications/a-security-tool-for-web-services.pdf

- **Bedienanleitung.pdf**
  Installation manual ReinerSCT CyberJack (German)
  http://support.reiner-sct.de/downloads/Dokumente/Bedienanleitung.pdf

- **ClipcodeSnippetsForWSE2.pdf**
  Clipcode Snippets for WSE 2.0 (and explanation of the WSE 2.0 option dialog)
  http://www.clipcode.biz/stream/ClipcodeSnippetsForWSE2.pdf

- **DCOCEFInalRepTechOv.pdf**
  Digital Certificate Operation in a Complex Environment: final report
  http://www.dcoce.ox.ac.uk/docs/DCOCEFInalRepTechOv.pdf

- **distributed_net_oop_2005.pdf**
  Distributed .NET, OOP 005, München

- **Entrust.SafeSign_v1.0.pdf**
  SafeSign Integration and Configuration Guide For Entrust

- **Implementing a PKI_rev1.pdf**
  Implementing a PKI
• **pkitutorial.pdf**
  Everything you Never Wanted to Know about PKI but were Forced to Find Out

• **S3-020366.pdf**
  CMP and CMC Comparison

• **sample.pdf**
  XML Digital Signatures

• **toolkits.pdf**
  CipherTools Integration Guide
  nCipher website (registered users part)

• **web_services.pdf**
  Web Services Authentication

• **whatisacrl.pdf**
  What is a CRL? (and how do I use one?)

• **WhitepaperSafeSignv1.0 EN.pdf**
  SafeSign Middleware: White Paper
  http://www.introcom.nl/download/producten/safesign/WhitepaperSafeSignv1.0%20EN.pdf

• **ws-wima03.pdf**
  A Signing Proxy for Web Services Security
  http://www.mathematik.uni-ulm.de/sai/ws03/webserv/ws-wima03.pdf
8 Appendix

8.1 Used products

To accomplish this project, following products were used:

- Intel Pentium IV 3.0 GHz with HT and 512MB RAM
- Intel Pentium Mobile 1.6 GHz and 512MB RAM
- Microsoft Windows 2003 Enterprise with Service Pack 1
- Microsoft Windows XP with Service Pack 2
- Debian Sarge Linux
- Fedora FC3 and FC1
- Microsoft Visual Studio .NET 2003
- Microsoft WSE 2 SP3
- nCipher nShield PCI F2
- ReinerSCT cyberJack pinpad
- O2Micro SmartCardBus Reader
- STARCOS 2.3 Smart Cards
- Ethereal 0.10.10
- GUIDumpASN 1.03
- OpenSSL 0.9.7c
- Sysinternals FileMon 7.0
- Sysinternals RegMon 7.0
- Total Commander 6.51 and 6.52
- HashCalc 2.01
### 8.2 Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programmable Interface</td>
</tr>
<tr>
<td>ASN.1</td>
<td>Abstract Syntax Notation one</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large Object</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>CDP</td>
<td>Certificate Distribution Point</td>
</tr>
<tr>
<td>CMC</td>
<td>Certificate Management Messages over CMS</td>
</tr>
<tr>
<td>CMS</td>
<td>Cryptographic Message Syntax</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate Revocation List</td>
</tr>
<tr>
<td>CSP</td>
<td>Cryptographic Service Provider</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Linked Library</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>EKU</td>
<td>Enhanced Key Usage</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standards</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>ICC</td>
<td>Integrated Circuits Card</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ITU-T</td>
<td>ITU Telecommunication Standardization Sector</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>MMC</td>
<td>Microsoft Management Console</td>
</tr>
<tr>
<td>OCSP</td>
<td>Online Certificate Status Protocol</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PHP</td>
<td>PHP: Hypertext Preprocessor</td>
</tr>
<tr>
<td>PIN</td>
<td>Personal Identification Number</td>
</tr>
<tr>
<td>PKCS</td>
<td>Public Key Cryptography Standard</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RSA</td>
<td>Cryptographic algorithm for public key encryption by Rivest, Shamir and Adleman</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
<tr>
<td>SO</td>
<td>Shared Object</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPKAC</td>
<td>Signed Public Key And Challenge Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TCL</td>
<td>Tool Command Language</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transport Control Protocol / Internet Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Discovery Description and Integration</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Indicator</td>
</tr>
<tr>
<td>URL</td>
<td>Universal Resource Location</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSE 2.0</td>
<td>Web Service Enhancements version 2</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XML DSIG</td>
<td>XML Digital Signatures</td>
</tr>
</tbody>
</table>
8.3 Implementation of certificate revocation lists

In this test-case scenario, two certificates are issued: one for ‘Client Authentication’ and one for ‘Server Authentication’. The certificates are installed on the correct location (Current User store for the client certificate and Local Machine store for the server certificate) and the web site is tested with HTTPS using both client and server side certificates.

The server certificate can be revoked using the Certificate Authority. As reason ‘Certificate Hold’ is specified, so the certificate can be un-revoked at a later time. Everything still works after revoking the server certificate. Even after choosing an explicit ‘Publish’ of the CRL in the CA, the page can still be reached.

There is caching of the CRL on the client side: In the directory C:\Document and Settings\Username\Application Data\Microsoft\CryptnetUrlCache\Metadata, the caching files can be found and deleted.

Even after a restart of the client, it still doesn’t throw an error like it should.

In the search for a solution Mozilla Firefox was used, because this browser has a different behaviour concerning certificates: the Netscape-approach.

First the ‘Microsoft Certificate Services’ web page is visited and the option to ‘Download a CA certificate, certificate chain, or CRL’ is chosen. The CRL will be imported correctly:

Even in this browser, the client still accepts the server’s certificate:
Figure 56: Visiting a HTTPS site with an ‘Unspecified’ revoked certificate using Mozilla Firefox

There seems to be a different behaviour between certificates revoked with a different ‘reason code’. When using ‘Certificate Hold’ as reason, the certificate is not really revoked: it is still usable. Another reason must be specified, for example ‘unspecified’. This way, the certificate cannot be un-revoked at a later time, and the server certificate will be rejected by Mozilla Firefox.

Still, Microsoft Internet Explorer accepted the server certificate. In this browser, the check whether the server certificate is revoked or not is standard disabled. To enable it, an option has to be turned on in the ‘Advanced’ tab of ‘Internet Options’: ‘Check for server certificate revocation’.

Figure 57: Enabling the server certificate revocation

In Mozilla Firefox, the checking is done by default and cannot be disabled:

Figure 58: Visiting a HTTPS site with a ‘Certificate Hold’ revoked certificate using Mozilla Firefox

So far client side revocation checking has been achieved: the client checks whether the server’s certificate is revoked or not. Now for the server side revocation checking: the IIS server must be able to check if a client certificate is still valid. The IIS server is set to require client certificates. The client’s browser will ask which certificate to use, or will select one automatically, depending on the browser’s settings.
According to Microsoft, there is a way to set the server to check for revoked certificates in IIS version 6 using the ‘adsutil.vbs’. This Visual Basic script can be found in the Inetpub/Adminscripts/ directory.

First the Identifier of the Web Site has to be figured out. When the Internet Information Services console is opened, the Identifier for 'Default Web Site' is set to 1 as shown in the following screenshot:

![Finding out the Identifier from a Web Site in IIS6](image)

Now the identifier can be used in the ‘adsutil.vbs’ script using the ‘Windows Script Host’ 'cscript.exe'.

```plaintext
C:\Inetpub\AdminScripts>cscript adsutil.vbs GET W3SVC/1/CertCheckMode
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
The parameter "CertCheckMode" is not set at this node.
C:\Inetpub\AdminScripts>cscript adsutil.vbs SET W3SVC/1/CertCheckMode 0
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.
CertCheckMode : (INTEGER) 0
C:\Inetpub\AdminScripts>
```

After restarting the IIS server, the revoked client certificate is still allowed. The setting failed. Since CRL checking is beyond the scope of this project and a solution was not found within a reasonable time, it is skipped.

A possible solution could be found using the following documents on Microsoft’s MDSN:

- ‘How To Perform CRL Checking with CAPICOM’
- ‘Enabling Certificate Revocation Checking in Internet Information Server 4.0’
8.4 Installation of the Smart Card readers on Linux

To test platform independency and operability, the ReinerSCT cyberJack was tried under Linux also.

8.4.1 Debian Sarge

A version of Debian Sarge has been installed. Debian Sarge is still in development, but because it's already very mature and robust and it has a 2.6 kernel instead of a 2.4 (which enhances the USB support), this version was installed.

First the SafeSign software was tried. Although the SafeSign web site describes that Linux is supported, only distribution-specific packages could be found on the CD-Rom. Here fore, the RPM packages for Fedora Linux Core 1 were converted to Debian packages with the ‘alien’ tool without problems.

```
debian:/home/fuss/safesign/SafeSign-STANDARD-2.0/Linux/Fedora Core 1# ls -al *.deb
-rw-r--r--  1 root root  663746 2005-04-07 13:05 pcsc-lite_1.1.2-1_i386.deb
-rw-r--r--  1 root root  259916 2005-04-07 13:07 pcsc-lite-debuginfo_1.1.2-1_i386.deb
-rw-r--r--  1 root root  477382 2005-04-07 13:04 safesign-pkcs11_2.0.0-11_i386.deb
-rw-r--r--  1 root root 1491106 2005-04-07 13:04 safesign-tokenadmin_2.0.0-7_i386.deb
```

Code 25: Converting Fedora to Debian packages with the ‘alien’ tool

Next, they were installed using the ‘dpkg’ tool. Following error was thrown after starting the newly installed program.

```
debian:/home/fuss/safesign/SafeSign-STANDARD-2.0/Linux/Fedora Core 1# tokenadmin
tokenadmin: error while loading shared libraries: libtiff.so.3: cannot open shared object file: No such file or directory
```

Code 26: Errors concerning shared libraries

The system was searched for libtiff.so.3 in /usr/lib and the file /usr/lib/libtiff.so was found. A symbolic link was created. Later on, an error of the same kind appeared.

```
debian:/home/fuss/safesign/SafeSign-STANDARD-2.0/Linux/Fedora Core 1# ln -s /usr/lib/libtiff.so /usr/lib/libtiff.so.3
debian:/home/fuss/safesign/SafeSign-STANDARD-2.0/Linux/Fedora Core 1# tokenadmin
tokenadmin: error while loading shared libraries: libssl.so.4: cannot open shared object file: No such file or directory
```

Code 27: Creating a symbolic link to the shared library ‘libtiff.so’

The creation of symbolic links was repeated until there were no “cannot open shared object file” errors any more:

```
# ln -s /usr/lib/libssl.so.0.9.7 /usr/lib/libssl.so.4
# ln -s /usr/lib/libcrypto.so.0.9.7 /usr/lib/libcrypto.so.4
# apt-get install libpcsclite1
# ln -s /usr/lib/libpcsclite.so.1 /usr/lib/libpcsclite.so.0
```

Code 28: Creating symbolic links to other shared libraries
Tokenadmin was tried again, but there was another error, this time thrown by the program itself:

![Token Administration Utility in Debian Linux with its error](image)

Figure 61: The Token Administration Utility in Debian Linux with its error

This is a strange error because Linux doesn’t use DLL files. The ‘aetpkss1.dll’ was pointed to the corresponding library ‘aetpkss1.so’ file using the configuration files in ‘/etc/safesign’ and the folder ‘.safesign/’ in the home directory, but this did not solve the problem. Because both ReinerSCT and SafeSign officially support Fedora Linux, the attempts to install ReinerSCT and SafeSign on Debian were dropped.

### 8.4.2 Fedora Linux Core 3

After the installation of this Linux version, there were in fact no SafeSign packages for Fedora Core 3, only for Core 1. On the ReinerSCT side, it’s just the opposite: only Fedora Core 3 is supported, not Core 1. Because it would make no sense to install such an old version of Fedora, the installation of SafeSign Fedora Core 1 RPM’s on the Fedora Core 3 machine was tried.

After the installation of the SafeSign software, there was an error concerning an undefined symbol in the GTK libraries:

```plaintext
undefined symbol: _gtk_accel_group_attach
```

Code 29: Error concerning GTK

Some research was done about this error and the main problem was found: Fedora Core 3 uses GTK+2.4, while Fedora Core 1 uses GTK+2.2. SafeSign tries to use a function of GTK which has been marked as private in the GTK+2.4 release, because applications shouldn’t use it directly.

While doing some research, a possible solution has been found because other programs had the same error in the past, but mostly patches were provided. Since the SafeSign software is not Open Source, there was no possibility to apply a patch. A possible solution is described below.

```c
...  GtkAccelGroup *accel;
      
      accel = gtk_menu_get_accel_group (GTK_MENU (menu_item->submenu));
      _gtk_accel_group_attach (accel,
          G_OBJECT (gtk_widget_get_toplevel (GTK_WIDGET(menu_item))));
```

...
GtkWidget *menu;

ptr1 = NSA_GET_PTR (env, obj);
ptr2 = NSA_GET_PTR (env, menuitempeer);
if (key)
{
    gtk_widget_add_accelerator (GTK_WIDGET (ptr2), "activate",
        gtk_menu_get_accel_group (GTK_MENU (menu)), key,
        (GDK_CONTROL_MASK | ((shift) ? GDK_SHIFT_MASK : 0)),
        GTK_ACCEL_VISIBLE);
...
}

Code 30: A possible patch for the GTK problem
## 8.5 Using OpenSSL

OpenSSL is an open source implementation of the SSL and TLS protocols. Of course OpenSSL is much more: it has a crypto library, an ASN.1 parser and builder, it offers high-level PKI functionality, etc. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

First OpenSSL needs to be downloaded and installed. A Windows installer can be found on the Internet. OpenSSL will be installed in `C:\Program Files\GnuWin32\`.

This chapter will describe two possible usages of OpenSSL: requesting a certificate and comparing two certificates.

Other options of OpenSSL include:

```plaintext
OpenSSL> help
openssl:Error: 'help' is an invalid command.

Standard commands
asn1parse ca ciphers crl crl2pkcs7
dgst dh dhparam dsa dsaparam
cnc engine errserr gendh gendsa
genrsa nseq ocsp passwd pkcs12
pkcs7 pkcs8 rand req rsa
rsautl s_client s_server s_time sess_id
smime speed spkac verify version
x509

Message Digest commands (see the `dgst` command for more details)
md2 md4 md5 mdc2 rmd160
sha sha1

Cipher commands (see the `enc` command for more details)
aes-128-cbc aes-128-ecb aes-192-cbc aes-192-ecb aes-256-cbc
aes-256-ecb base64 bf bf-cbc bf-cfb
bf-cfb bf-ofb cast cast-cbc cast5-cbc
cast5-cfb cast5-ecb cast5-ofb des des-cbc
des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb
des-ede-ofb des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb
des-ofb des3 desx idea idea-cbc
idea-cfb idea-ecb idea-ofb rc2 rc2-40-cbc
rc2-64-cbc rc2-cbc rc2-cfb rc2-ecb rc2-ofb
rc4 rc4-40 rc5 rc5-cbc rc5-cfb
rc5-ecb rc5-ofb

OpenSSL>
```

---

**Code 31: Usage of OpenSSL**

---

Securing XML Web Services with WSE 2 and cryptographic hardware
### 8.5.1 Requesting a certificate

To be able to request certificates, OpenSSL requires a configuration file. Several examples can be found on the Internet. A short, modified example is given below.

```plaintext
[ ca ]
default_ca  = CA_default  # The default ca section

[ CA_default ]
certs       = certs  # Where the issued certs are kept
crl_dir     = crl     # Where the issued crl are kept
database    = database.txt  # Database index file.
new_certs_dir = certs  # Default place for new certs.
certificate = cacert.pem  # The CA certificate
serial      = serial.txt # The current serial number
crl         = crl.pem   # The current CRL
private_key = cakey.pem  # The private key
RANDFILE    = private.rnd # Private random number file
x509_extensions = x509v3_extensions  # extenstions to add to the cert
default_days = 36  # How long to certify for
default_crl_days = 30  # How long before next CRL
default_md    = md5    # Which md to use.
preserve     = no     # Keep passed DN ordering

[ policy_anything ]
countryName   = optional
stateOrProvinceName = optional
localityName  = optional
organizationName = optional
organizationalUnitName = optional
commonName    = supplied
emailAddress  = optional

[ req ]
default_bits  = 1024
default_keyfile   = privkey.pem
distinguished_name = req_distinguished_name
attributes   = req_attributes

[ req_distinguished_name ]
countryName   = Country Name (2 letter code)
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
localityName   = Locality Name (eg, city)
0.organizationName = Organization Name (eg, company)
organizationalUnitName = Organizational Unit Name (eg, section)
commonName    = Common Name (eg, your website's domain name)
commonName_max = 64
emailAddress  = Email Address
emailAddress_max = 40

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 3
challengePassword_max = 20
```

*Code 32: Example of an OpenSSL configuration file*
After saving 'openssl.conf' in the OpenSSL 'bin/' directory, the actual request can be made.

```
C:\GnuWin32\bin>openssl req -newkey rsa:1024 -keyout key.pem -out req.pem -config openssl.conf
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
...............+++++
.+++++
writing new private key to 'key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase: 

You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be left blank.

Country Name (2 letter code) []:BE
State or Province Name (full name) []:
Locality Name (eg, city) []:
Organization Name (eg, company) []:
Organizational Unit Name (eg, section) []:
Common Name (eg, your websites domain name) []:Filip Van Lerberge OpenSSL
Email Address []:filip.vanlerberge@kahosl.be

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:

C:\Program Files\GnuWin32\bin>
```

**Code 33: Requesting a certificate using OpenSSL**

A challenge password is not set. At this point two files are created in the bin\ dir: 'key.pem' and 'req.pem'. Only the 'req.pem' is needed. The 'key.pem' holds the private key for later usage.

This base64 encoded PKCS#10 file can be pasted in Internet Explorer in the 'Advanced Certificate Request' on the CA pages.

**Figure 62: The content of a certificate request**

The 'certnew.cer' file can be saved and 'certmgr.msc' must be opened. The certificate 'Import' must be chosen.
After the import, the certificate is installed correctly and can be used just as other installed certificates.

8.5.2 Comparing two certificates

First the two certificates need to be transformed to text-format, so that they can be compared properly. The two certificates in this example are named ‘WSE2QSS_b64.cer’ and ‘WSE2Copy1_b64.cer’.

```bash
C:\GnuWin32\bin>openssl x509 -text -in WSE2QSS_b64.cer -out WSE2QSS_b64.txt
C:\GnuWin32\bin>openssl x509 -text -in WSE2Copy1_b64.cer -out WSE2Copy1_b64.txt
```

**Code 34: Comparing two certificates using OpenSSL**

Now the two text files can be compared using e.g. ‘Total Commander’. The main difference between the two certificates is the extensions.
Figure 64: Comparing two certificates using ‘Total Commander’
8.6 Email and Digital Signatures

This chapter will describe the process how email can be secured using Digital Signatures. Both Microsoft Outlook Express and Mozilla Thunderbird will be described.

8.6.1 Microsoft Outlook Express

Microsoft Outlook Express provides the possibility to choose a certificate in the properties panel of an account. The ‘Filip Van Lerberge 10’ certificate is stored on a Smart Card and Outlook Express can directly access the ‘Personal Certificate Store’.

![Figure 65: Configuring the security settings of an email account in Microsoft Outlook Express](image)

When someone wants to send a message, the Digitally Signing and/or Digitally Encryption can be chosen.

![Figure 66: Signing and/or Encrypting an email in Microsoft Outlook Express](image)

8.6.2 Mozilla Thunderbird

In Mozilla Thunderbird, a new security device needs to be added, because this program doesn’t work together with the ‘Personal Certificate Store’ or CSPs, instead it works with PKCS#11.

The options panel provides the possibility to manage those security devices.
A new PKCS#11 Device needs to be added. The DLL needed is the SafeSign DLL, located at 'c:\windows\system32\aetpkis1.dll'.

After supplying a password, the SafeSign module will be added.

After adding the device, the correct certificate can be chosen to sign the message per account.
When someone wants to send a message, Digitally Signing and/or Digitally Encryption can be chosen.

To make sure that the email can be sent, the ‘Root CA’ must be trusted in Mozilla Thunderbird to identify email users. This can be done by opening the ‘Certificate Manager’ in the options panel.

Figure 70: Configuring a certificate for email signing and/or encryption

Figure 71: Sending a digitally signed email using Mozilla Thunderbird

Figure 72: Trusting the CA to identify email users
8.7 WSE 2.0 Options Dialog

The WSE 2.0 options dialog is in fact just a skin around the ‘app.config’, ‘web.config’ and ‘policyCache.config’ files. Every change made in the WSE 2.0 options dialog represents a change in the corresponding .config file.

- web.config is used with web applications. web.config will by default have several configurations required for the web application. A web.config can exist for each folder of a web application.
- app.config is used for Windows applications. When an application is built in Visual Studio .NET, it will be automatically renamed to <appname>.exe.config and this file has to be delivered along with the application.

The GUI was made to avoid typing errors in the config files and to make the programmers' life easier.

The WSE 2.0 options dialog consists of several tabs with its configuration settings. The first tab (‘General’) gives the programmer the possibility to enable the project for WSE.

![Figure 73: Enabling the project for WSE](image)

The second tab (‘Security’) permits the programmer to set security options concerning the certificate and security token managers.
Figure 74: Security settings for certificates

The corresponding ‘app.config’ file shows the settings.

```xml
<configuration>
  <configSections>
    <section name="microsoft.web.services2"
  </configSections>
  <microsoft.web.services2>
    <diagnostics>
      <trace enabled="false" />
    </diagnostics>
    <security>
      <x509
        storeLocation="CurrentUser"
        allowTestRoot="false"
        allowUrlRetrieval="false" />
    </security>
  </microsoft.web.services2>
</configuration>
```

Code 35: The ‘app.config’ file with its settings

The ‘Policy’ tab permits the programmer to set filters on the incoming and outgoing certificates using a wizard. This can be useful if for example only a certain person (certificate) is allowed to use the Web Service.

Other tabs include message tracing (‘Diagnostics’), routing handlers (‘Routing’) etc.