WebIdentity
Developer Guide
v5.0

The WebIdentityDL electronic key has proved compliant with the following regulations:
CEI EN 61000-4-2; CEI EN 61000-4-3; CEI EN 55022
as required by:
CEI EN 61000-6-1, CEI EN 61000-6-2, CEI EN 61000-6-3, CEI EN 61000-6-4
which set forth specific standards for the following tests:
" ESD Immunity Test"
"Immunity Test against radio-frequency broadcast EM fields"
"Broadcast Emission Test"
In compliance with the "Fundamental Requisites" of European Directives EMC 89/336/CE and 2004/108/CE

FCC ID: TFC-AAL

EUTRONSEC S.p.A.
WebIdentityDL
Supply: 5V DC
Absorption: 30 mA

This device complies with Part 15 of the FCC Rules.

Operation is subject to the following two conditions:
(1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

Caution: changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

IMPORTANT REMARKS
Due to the limited space on the product shell, all FCC certification references are on this technical manual.

Table of contents

1 Preface

This chapter describes the contents and the lay-out of this manual and of WebIdentity distribution.

1.1 About this manual

This manual describes the WebIdentity product, its operation, its integration into web services and client/server applications.

This manual appeals to the software developers and system integrators who are willing to implement projects based on data authentication and confidentiality for web services through the development of ASP pages, JSP pages, Servlet Java and client-server applications at large.

1.2 Environments supported by version 5.0

Version 5.0 of this software does not cover all environments that were supported by version 4.0.

Later versions of the software will support such environments.

To verify the actual coverage please refer to chapter Requirements

1.3 How to use this manual

The basic concept and the architecture of WebIdentity are described herein, as well as its implementation as ActiveX and Java classes and the possible operating instructions.

Chapter 1: Preface describes the content and the lay-out of the manual.

Chapter 2: Introduction presents WebIdentity features and introduces its practical operation.

Chapter 3: Requirements lists the hardware and software requirements for using WebIdentity.

Chapter 4: Installation and **Configuration** describes the necessary steps for installing and configuring WebIdentity.

Chapter 5: Features describe the operation and the security specifications of WebIdentity.

Chapter 6: Integration describes the macro activities that are necessary for operating WebIdentity in a project for web application development.

Chapter 7: Examples provides practical examples for using WebIdentity.

Chapter 8: Appendixes

- Symmetric cryptography
- Hashing algorithms
- Glossary
- Bibliography

1.4 Assumptions

For reading this manual the knowledge of web technologies (ASP, VBScript, JavaScript, JSP, COM, ActiveX, EJB, Java, Servlet) that are used for developing web-based and client/server applications is required.

Besides the knowledge of security topics based on symmetric key cryptography is required.

A quick introduction to symmetric cryptography is available in Appendix 8.1.

1.5 Font conventions

In this manual the following conventions for the use of Fonts are agreed:

The *Italics* font is used for:

- Words taking on a specific meaning in the text beyond their own literal meaning. Words that are introduced for the first time

The `Courier` font is used for:

- Sources in HTML, VBScript and JavaScript languages

- Names of classes and objects
- File names

The font is used for:

- Product names
- Company names
- Registered Trademarks

1.6 Interrelated Documents

Here follows the list of further documentation relating to WebIdentity:

EUTRON, "*WebIdentity API Reference*", 2007
Reference for the programmer of WebIdentity-related applications.

EUTRON, "*WebIdentity Java Developer Documentation*", 1998
Electronic guide for the Java developer of WebIdentity in JAVADOC format.

The material supplied with WebIdentity is subdivided into software components and documentation; the lot is supplied within the WebIdentity package.

1.6.1 Software components

The following software components are enclosed with WebIdentity:

`wicli.cab`	Activex client downloadable with Explorer.
`wisrv.cab`	Activex server downloadable with Explorer. It is NOT necessary to put it at the end users' disposal: customers can access the website and therefore do not need to download this cab.
`WIClientInstaller.exe`	Setup which attends to installing each component that may be useful for users in possession of a WebIdentity3P key. This setup installs: the activex client; the plug-in for each non-Explorer browser abovementioned; the driver for the WebIdentity3P Token.
`WIDLClientInstaller.exe`	Setup which attends to installing each component that may be useful for users in possession of a WebIdentityDL. This setup installs: the activex client; the plug-in for each non-Explorer browser abovementioned.
`WIDriverInstaller.exe`	Setup of WebIdentity3P driver. This setup is necessary for an Explorer user in possession of a WebIdentity3P token.
`WIServerInstaller.exe`	Setup for installing the client and server parts. This setup installs: the activex server;the activex client; the plug-in for each non-Explorer browser abovementioned; the driver for the WebIdentity3P Token.
`Webidentity-4.0.0.0-1.i386.rpm`	WebIdentity Plug-In Client for Linux
`Wicli.dmg`	WebIdentity Plug-In Client for Mac OS X

1.6.2 Documentation

The following documentation is enclosed with WebIdentity:

EUTRON, "*WebIdentity Developer Guide*", 2007
Manual for the WebIdentity programmer.

EUTRON, "*WebIdentity API Reference*", 2007
Reference for the programmer of WebIdentity-related applications.

EUTRON, "*WebIdentity Java Developer Documentation*", 1998
Electronic guide for the Java developer of WebIdentity in JAVADOC format.

EUTRON, "*WebIdentity Demo*", 2004
Examples that introduce WebIdentity features in web-based applications with IIS, ASP and ASPNET.

EUTRON, "*WebIdentity Java Demo*", 2004
Examples that introduce WebIdentity features in Java web-based applications with Java 2 Enterprise Edition v1.3 and the use of Java Application Deployment Tool for the illustrative application.

1.6.3 Distribution

The CD-ROM that is supplied with the WebIdentity SDK contains all necessary software for using and developing web-based and client/server applications with WebIdentity use.

1.7 Feedback

The quickest way for getting in touch with EUTRON as regards WebIdentity is sending an email message to the helpdesk: `helpdesk@eutronsec.it`

For any commercial contact it is possible to send an email message to the following address: `info@eutronsec.it.`

Please be reminded that the customer service is exclusively provided to EUTRON's customers who use WebIdentity. No direct assistance is provided to EUTRON's customers' customers.

2 INTRODUCTION

WebIdentity is a USB device which enables the remote identification of a user in a secure way. The software components that are available with WebIdentity enable integrating the token features into web-based applications.

Eutron WebIdentity originated with the intention of making an Internet service user identifiable univocally and guaranteeing access to information contained in the website only and exclusively to the user in possession of a personal hardware device.

Even if the Internet environment is the first operating environment which WebIdentity appeals to, it is possible to use it in a generic client-server environment.

2.1 WebIdentityDL: The new model

WebIdentity has been recently renewed and the main features of the new device are:

1. HID interface: now WebIdentity is a USB device which exports a HID (Human Interface Device) interface, and therefore the native support of the operative system for accessing the token is used in all Windows operating systems (but also Linux and Mac OS). This makes the installation of a specific driver for the device useless, for the benefit of operating user-friendliness. The previous version was a USB device requiring the installation of a standard supplied driver.
2. Authentication via AES 256: The new WebIdentity token is similar to the previous model from the architectural point of view, but instead of carrying out authentication via the Triple-DES algorithm it now uses the new standard AES in the 256 bit key version. This algorithm ensures an even higher security than the previous version.

This new version of WebIdentity has been named WebIdentityDL (WebIdentity driverless). Within this document when this WebIdentity version will be expressly referred to it will be identified with WebIdentityDL whilst the previous version will be identified with WebIdentity3P.

The production of the WebIdentity3P tokens has been discontinued but the software supplied with WebIdentity is compatible with both versions and can therefore support both of them.

Within this document the model WebIdentityDL will always be referred to, unless the older version is expressly specified.

2.2 Secure authentication in web-based applications

The term "authentication" is often used in the broad sense. By authentication the process of verification of a person's identity or of a process identity is meant; within a communication the authentication verifies that the messages coming from a recognized source belong to such a source.

The authentication processes that are used with web application nowadays are mainly based on username and password recognition, whether integrated in the HTTP protocol or controlled by the application itself.

2.3 The WebIdentity solution

WebIdentity has been designed for supplying a high security "strong authentication" without employing any complicated and expensive infrastructure.

WebIdentity solves all problems connected with username- and password-based authentication systems, that is:

- vulnerability: the username and the password are transmitted in plain text mode and are therefore subject to attacks based on packet filtering (sniffing).
- Improper use: it is not possible to control the number of people using the same username and password.

A high security alternative might be the adoption of a PKI (Public Key Infrastructure) which implies a considerable management load, above all within already existing infrastructures. WebIdentity, while ensuring a "strong authentication" that is comparable with the authentication obtainable from a PKI infrastructure, requires a managing and implementing work load that can be compared with the integration of the username- and password-based application-related authentication.

WebIdentity represents the ideal hardware & software solution:

- for Strong Authentication of web-based service users
- for managing protected transactions over the Internet network

2.4 Fields of application

The typical WebIdentity user is anybody interested in the protection of web services, that is in protecting access to reserved information and services that are available via a web server. For instance:

- customized price-lists
- download areas for software update
- on-line newspapers and magazines
- management of Internet subscriptions

The univocal and selective authentication of users enables, among other things, to manage customized user profiles and a really controlled and selective distribution of received information.

2.5 USB Tokens

The WebIdentity product features some functional characteristics that make it the ideal solution for secure authentication when accessing restricted resources.

WebIdentity can identify with absolute certainty the person who is accessing an Internet website thanks to the use of a microchip. Besides, thanks to the univocal data entered in the production phase each WebIdentity token has been produced in a univocal and unrepeatable way.

Thanks to the USB HID interface, WebIdentity affords the advantage of being interfaceable with any available PC without adding any hardware infrastructure. It also offers a reduced dimension that makes it extremely easy to transport, like with a smart card.

2.6 Authentication

WebIdentity originated with the intention of making an Internet service user identifiable univocally and guaranteeing access to information contained in the website only and exclusively to the identified and authorized user.

The WebIdentity, ActiveX and Plug-In software components exploit the storage and cryptography potential of the WebIdentity hardware key, enabling access to a Web site only and exclusively to a user in possession of a WebIdentity and duly initialized by the Internet service provider.

The control on the user's identity is carried out via a process requiring the initialization of a token and the implementation of a challenge/response protocol for authentication:

1. Initialization of the WebIdentity device to be supplied to the service user to protect and generation of a database or addition to the existing one of the token identifying information (User-Id) for each associated user.
2. forwarding of server-generated challenge to the client.
3. Verification of initialized WebIdentity's presence on the client; reading and computation of response; forwarding of response to the server for recognition.
4. Verification of response on the server, identification of the user, forwarding of information to the client if the user turns out to be authenticated.

2.7 Cryptography

Besides identifying the user, Eutron WebIdentity's technology puts another important possibility at the ISPs' disposal: forwarding of encrypted data over the Internet. Indeed it is rather simple, once the client is identified, to be able to forward the hyper-textual content of the pages and receive the data entered in a form on the client by the user securely, that is to say, encrypted. The operation is always carried out by the ActiveX that come with the WebIdentity solution.

2.8 Remote control

This feature enables carrying out the reading and writing operation on the WebIdentity remote device. To this end a protocol based on challenge/response has been developed which enables exchanging data (reading and writing of the remote Token) from the client ActiveX to the server ActiveX securely, that is, ensuring confidentiality, integrity and non-replicability of data.

This function enables storing on the token any kind of information which might concern the user or the service. This might come in useful for instance with recharge-based, score-based or bonus-based services, for which management is carried out in a centralized or partially centralized way; in this situation it is therefore necessary to retrieve or setup data directly from the token and on the token.

3 Requirements

To work properly the WebIdentity product requires the processing environment to meet some minimum requisites in terms of installed hardware and software. The requisites are divided into two distinct parts concerning the client and the server.

3.1 Client

With Microsoft Windows the client component is realized with an ActiveX for Windows Explorer and with a Plug-In for Mozilla Firefox.

This is the list of environments supported by the software 5.0 version:

- Operating system
  - Windows 98SE/ME
  - Windows 2000
  - Windows XP 32 bit
  - Windows Vista 32 bit
- Web Browser
  - Microsoft Internet Explorer 6.0 and 7.0
  - Netscape 8.1.3 or higher
  - Mozilla 1.7.13 or higher
  - Firefox 2.0 or higher

WebIdentityDL also works on XP 64 bit and Vista 64 bit (on AMD64 architecture) with 32 bit browsers.

As far as Linux and Mac OS are concerned a future release for supporting WebIdentityDL within such environments is planned for.

If only WebIdentity3P is used it is possible to adopt the previous 4.0 version of the software that is supported also by Linux and Mac OS X.

The 4.0 version supports the following environments:

- Architecture
  - i386 hardware (Microsoft Windows, Linux)
  - PowerPC G3/G4/G5 (Mac OS X)
- Operating system
  - Microsoft Windows 98SE/ME/NT4/2000/XP
  - Linux (kernel 2.4 or higher)
  - Mac OS X
- Browser web
  - Microsoft Internet Explorer 3.0 or higher for Windows
  - Netscape 6.1 or higher for Windows/Linux/Mac OS X
  - Mozilla 1.6 or higher for per Windows/Linux/Mac OS X
  - Firefox 0.7 or higher for Windows/Linux/Mac OS X
  - Camino 0.8 or higher for Mac OS X

Version 4.0 does not support WebIdentityDL

3.2 Server

The 5.0 version of the WebIdentity software provides a COM interface for Windows 32 bit and 64 bit.

At some future date a Java class will also be released for non-Microsoft platforms and for environments supporting Java libraries.

If only WebIdentity3P is used it is possible to adopt the previous 4.0 version of the software that also contains the server's Java library.

3.3 Compatibility between 5.0 and 4.0 versions

The following table sums up the availability or non availability of the support between different software versions on the client and servers side and tokens in use.

	Server 5.0	Server 4.0
WebIdentityDL with Client 5.0	Supported	NOT supported
WebIdentityDL with Client 4.0	NOT supported	NOT supported
WebIdentity3P with Client 5.0	Supported	Supported
WebIdentity3P with Client 4.0	Supported	Supported

4 Installation and Configuration

Using WebIdentity requires installing and configuring two different modules: the client and the server. For each system specific software components to install and configure are provided.

4.1 Client

WebIdentityDL operation on the client requires the installation of a software component: in Windows environment for the Explorer browser it will be an activex, in all other cases it will be a plug-in.

In the event of a WebIentity3p model a driver will also have to be installed.

4.1.1 Example – installation of a web page

The following HTML code enables installing and updating the ActiveX client of WebIdentity in Explorer automatically:

```
<object classid="clsid:878A0D61-48D2-11D3-A75D-00A0245382DE"
id="WIDrvCli" codebase="software/WICli.cab#version=4,0,0,0"
VIEWASTEXT>
<embed type="application/x-wicli-plugin" name="WICli" width="1"
height="1" hidden="true">
</object>
```

The `OBJECT`[1] tag enables inserting in the HTML document the ActiveX univocally identified by means of the class id, which for ActiveX client is `878A0D61-48D2-11D3-A75D-00A0245382DE.` The `ID` attribute identifies the object univocally inside the document by means of a label, thus enabling interaction. The `CODEBASE` attribute is necessary in case ActiveX is not present on the client. Such an attribute specifies the location (URL) from which the component is downloaded automatically. In addition to the URL it is possible to enter the control version (`#version=`) so that it can be downloaded, if more recent than the installed version. The `EMBED` tag enables inserting in the HTML document the client WebIdentity Plug-In univocally identified with the `TYPE application/x-wicli-plugin` tag.

For Microsoft Windows the ActiveX WebIdentity client for Internet Explorer and the Plug-In WebIdentity client for browsers derived from the Gecko engine (Mozilla, Netscape, Firefox) are available.

Hereunder listed are the various cases and the software that it is necessary to install for using WebIdentity.

4.1.2 EXPLORER USERS IN POSSESSION OF A WEBIDENTITYDL DEVICE

WebIdentityDL does not require any drivers; therefore within all operating systems (except for Windows 98) the user just has to connect to the website that employs WebIdentity.

The download of the new activex will be managed by the browser (like with the previous example) which will require the user to provide the necessary confirmations.

No other operations are required, excepting that of using an Explorer configuration that enables downloading and running signed activex.

The name of the cab which contains the activex is wicli.cab

[1] At present supported only by Internet Explorer 4.0+ and following

4.1.3 EXPLORER USERS IN POSSESSION OF A WEBIDENTITY3P DEVICE

For being able to use WebIdentity3P the user needs to install the driver (had it not been previously installed) by using the WIDriverInstaller.exe setup.

The setup must be carried out by administrator-level users. The driver is not signed, therefore the user will have to answer affirmatively to the request for installation of a non-signed driver.

After installing the driver the user can use the device as a WebIdentityDL user.

4.1.4 FIREFOX (or Mozilla and Netscape) USERS IN POSSESSION OF A WEBIDENTITYDL

In order to be able to use the web pages that are protected by the WebIdentity plug-in, download and install the WebIdentity client plug-in: use WIDLClientInstaller.exe.

The versions of those browsers for which installation is guaranteed are:

4.1.5 FIREFOX (or Mozilla and Netscape) USERS IN POSSESSION OF A WEBIDENTITY3P

For Installing the web pages protected by the WebIdentity plug-in download and install WIDLClientInstaller.exe then install WIDriverInstaller.exe; the order in which these setups are run is irrelevant. It is possible to use one single setup for plug-in and driver by using WIClientInstaller.exe.

4.1.6 WINDOWS 98SE USERS IN POSSESSION OF A WEBIDENTITYDL DEVICE

This operating system supports the HID peripherals, but Microsoft does not require any default installation of the support.

Anyway Windows 98 installation disk contains the HID driver related to this operating system.

The related cab is also available at this address.

http://www.eutroninfosecurity.com/shared/test/HID_Win98se/HID_Win98se.zip

The setup of the plug-in or of the client activex is analogous to the operations to carry out on the other operating systems.

4.1.7 LINUX USERS

Please be reminded that with this release Linux support does not support WebIdentityDL; at present only the software of 4.0.0.0 version supporting WebIdentity3P is available.

For Linux the client Plug-in WebIdentity for Gecko browsers is available.

For such browsers an rpm packet (`webidentity-<version>-<build>.i386.rpm`) is available which must be downloaded manually: once the rpm file is downloaded it is possible to use a visual packet manager or the `rpm` command line utility for manual installation. Use the command `rpm -i webidentity-4.0.0.0-1.i386.rpm` for installation, `rpm -e webidentity` for uninstallation and `rpm -q webidentity` for verifying the installation.

4.1.8 MAC OS X USERS

Please be reminded that with this release Mac OS X support does not support WebIdentityDL; at present only the software of 4.0.0.0 version supporting WebIdentity3P is available.

For Mac OS X the client Plug-In WebIdentity is available for Gecko browsers. For such browsers a disk file is present (`wicli.dmg`) which must be downloaded manually; once downloaded it is possible to load it by double-clicking on the file `.dmg,` if not loaded automatically. If the operation fails and the downloaded file is not associated as a disk file, it is necessary to execute “Information” on the file and then choose Disk Copy from the “Open with application” section. With Mac OS X 10.2+ it is possible to use “Open with” from the context menu. Once the file is uploaded, open

it and run it by double-clicking the packet file `wicli.pkg` in order to start the installation; then follow the wizard's instructions for completing it.

4.1.9 ADMINISTRATOR'S PERMITS FOR INSTALLATION

With Microsoft Windows NT/W2K/XP, Linux and Mac OS X it is necessary to have administrator's privileges for carrying out the installation of the client component.

On Windows 2000, XP and Vista the installation of the WebIdentity client component must be carried out at a user's level with administrator's permits.

When using the WebIdentity3P token with Windows 2000 and XP the installation needs to be completed by inserting the token into each of the available USB ports in order to enable every non-administrator user to use the token.

In case of Windows XP the following dialog-box will also appear for installing the driver: installation must be carried out without enabling any research in "Windows Update" (for saving time), then confirm the automatic installation from the following display page; at this point the installation on the USB port will be completed.

4.2 Server

The server configuration is strictly dependent on the Web Server that is to be used.

Within Microsoft Windows 32 bit with i386 architecture just run the installation program `WIServerInstaller.exe` of the (ActiveX) server.

Within a Java environment the copy of the `WEBIDENTITY.JAR` library is provided in order for the java applications (Servlet, for instance) to be able to use the WebIdentity API. It may be necessary to setup a `CLASSPATH` environment variable for identifying the WebIdentity API.

5 Features

This chapter describes the content and the features provided by WebIdentity for integrating with web-based and client-server applications.

5.1 Integrated features

The WebIdentityDL device is a secure, portable and user-friendly hardware key with USB interface.

The device is characterized by the following technical specifications:

- compliant with the USB specifications (Universal Serial Bus) v2.0 low speed devices and HID 1.11.
- equipped with a univocal serial code for each key: each key is individually customized with a factory pre-set identification code, different for each user.
- It runs AES 256 bit On Board.
- It implements secure data storage. The token is provided with about 8KB FLASH memory, processor-internal and externally-accessible only in firmware-controlled mode; besides, no command can modify the serial number. This architecture prevents (i) the possibility to copy the memory from one token to another (it is not possible to clone the token), (ii) the direct modification of the memory content.
- It implements secure data transmission. The data packets passing through the USB bus are protected with AES cryptography and each packet includes a random value so as to avoid any possibility to read and identify the transit commands and information.
- It implements the http communication cryptography with the Blowdish algorithm with 256 bit key.
- Self-powered: it does not use any internal batteries nor any external power supply.

Figure 5.1 displays the schematic make-up of the WebIdentity hardware token.

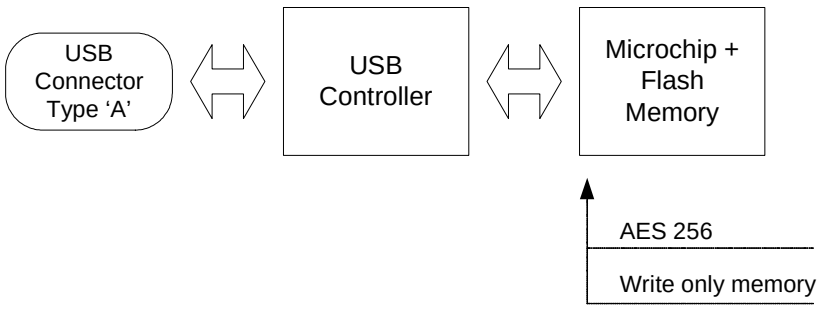

Figure 5.1 – Make-up of WebIdentity token

The AES 256 on board operations have been designed for ensuring maximum security; indeed during the initialization phase the AES key is stored in the write-only memory and used on board for AES computation.

As the key is write-only and executable, it is impossible to extract it from the token and therefore to obtain it unlawfully. Any programs designed for monitoring the data transfer from the token to the PC would not be able to trace back the symmetric key contained in the token in any way.

The flash memory is inside the WebIdentityDL processor; it can be accessed via the implemented commando, that is, it is not possible to access data directly; in particular it is not possible to read the AES keys as there is no command that may carry out such an operation.

Please be reminded that concept of write-only memory with on-board key computation is available also on the WebIdentity3P model, which does not use the AES algorithm but the Triple DES 2EDE algorithm.

On this key a good part of data is assigned to an eeprom memory which, even if external to the processor, is encrypted with a key contained in the processor internal memory space. This makes reading the eeprom memory useless as the cryptography key is not revealed. It is also impossible to clone it because each WebIdentity3P device has a cryptography key which resides on the processor-internal memory partition and is different for each device.

5.2 WebIdentity contents

Three essential items of information are stored securely within the microchip memory, both for WebIdentityDL and for WebIdentity3P:

1. the `Label` is inserted in the initialization phase and exclusively used for identifying the WebIdentity token on the USB bus. It is represented by a string that usually identifies the web service.
2. The User-Identifier (`User-Id`[1]) generated the initialization phase and dependent on the key personal data ad univocal data. The personal data can be made up by any information which might represent the user, for instance, name and surname; the univocal data is represented by the Token Serial Number (univocal serial number assigned to each WebIdentity device).
3. The AES 256 key (3DES per WebIdentity3P) generated in the initialization phase, dependent on the Server Secret (illustrated in the following charter, which can be considered as the generator of all token keys) and on the User-Id; it is stored in the write-only memory partition; the AES key is the symmetric key.

The computation of the AES key is obtained by means of a hashing and AES cryptography procedure. hashing of User-Id and AES computation using the Server Secret as a key. A brief introduction to the hashing algorithm is present in appendix **Errore. L'origine riferimento non è stata trovata.**.

The token secret (dependent on the Server Secret) is therefore given by the AES-Key (symmetric key) used in the challenge/response operations and stored in the write-only memory of the microchip itself. Its computation typology ensures the Server Secret and consequently the integrity of the whole service.

In other words if the key entered in the token was recognized, it would not be possible to trace the Server Secret any way.

The key of the token cannot be accessed as long as one the following events occurs at least:

1. AES is fully violated
2. The whole content of the WebIdentity chip memory is violated

The former case appears highly improbable. The latter case might happen only by using extremely expensive devices.

In the latter case a large economic investment might bring to the violation of a single device, not of the whole system, therefore the Server key is secure as long as AES is not fully violated.

Figure 5.2 describes the information contained in the WebIdentity token and their connections.

[1] Also named PIN.

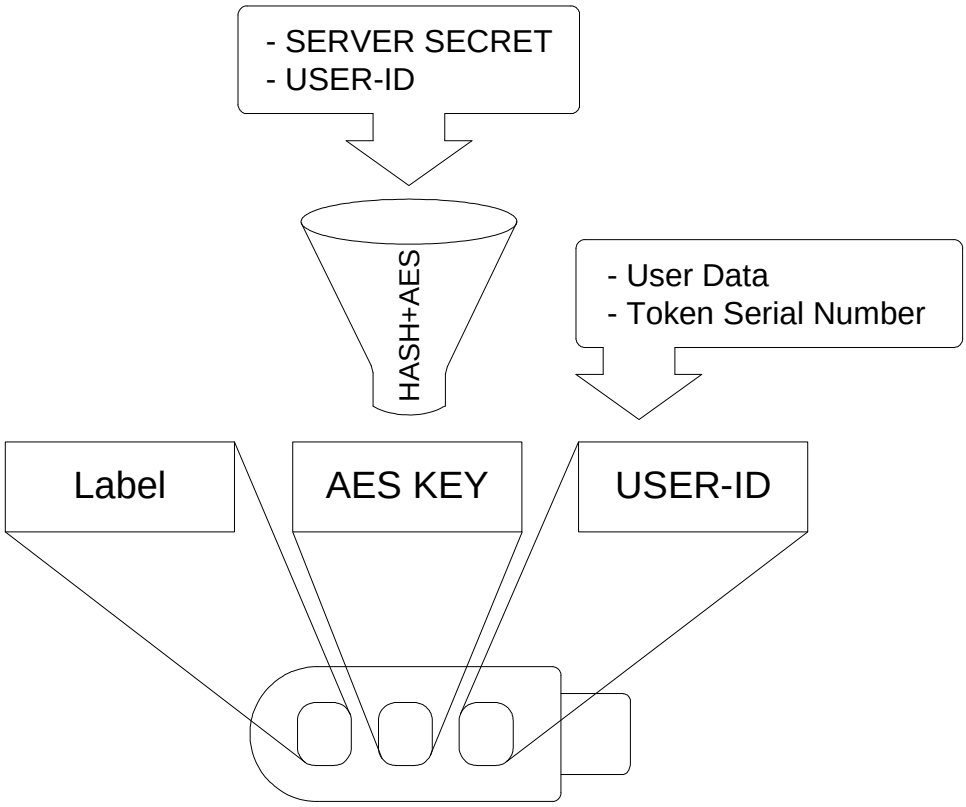

Figure 5.2 – Information provided in the WebIdentity token

5.2.1 Server Secret

The protection model offered by WebIdentity includes the use of one unique secret termed Server Secret (inside the WebIdentity SDK it is termed Server Password). The Server Secret is used during WebIdentity's processing for authentication, cryptography and remote control on the server side; it is also used during the initialization phase relating the WebIdentity hardware devices. Such a secret makes it possible for the service provider to recognized all and only its own WebIdentity keys, and consequently the customer who owns them. Practically speaking, the Server Secret is an alphanumeric string that is chosen in the web service development phase. Il Server Secret is converted into a AES 256 key or DES 2EDE Triplo; an ideal key can be produced by using a string with at least 43 characters, random chosen among the letters of the English alphabet (lower-case and upper-case) and numbers.

The security of the Server Secret is at the web server developer's care who must prevent access to non-authorized persons; for instance, if the Server Secret is stored in the application DB it is advisable to encrypt it. However it is important to take all the necessary precautions for making access to the application as secure as possible, as security depends on the Server Secret, which must be accessible on the server service side.

The Server Secret is used as a AES 256 key (3-DES for WebIdentity3P) for generating the secret key of each token. The input text of the AES computation is the "User-ID" hashing. Therefore each user is assigned a different key, which guarantees the following security level:

1. by knowing the User-ID it is impossible to calculate the token secret key if the Serve Secret is not known
2. by knowing the User-ID and the token secret key (the latter data being virtually inaccessible as reported above) it is impossible to trace the Server Secret.

5.3 Operation

For operating properly it is necessary to structure a website in such a manner as to be able to manage the following.

When a user is connecting to a protected web area, the WebIdentity Server requires the insertion of the device in the client machine in order to verify the information contained therein. The operation is carried out with a Challenge/Response protocol which enables authenticating the client without the information contained in the token being transferred through the network. All WebIdentity-related information exchanged between the server and the client do not require any particular communication protocol and therefore it is possible to use the same protocol of the application. For web-based applications all transactions carried out for authentication, cryptography and remote control are managed with http protocol, which is therefore portable and transparent for the various transfer systems. For client/server applications it is possible to use a DCOM or CORBA for message exchange.

The following paragraphs exemplify WebIdentity authentication, cryptography and remote control operations.

In the following paragraphs the authentication mechanism is illustrated; no clear distinction is made between the functions operated by the WebIdentity software and those carried out by the client application (typically a web application): here attention is drawn to the operating process. As from paragraph 3 an example is provided about how to carry out authentication with a web application; at this point the operations of the WebIdentity software and those relating to the infrastructure will be manifest.

5.4 Authentication

The WebIdentity remote identification mechanism requires a client/server architecture.

Client and server establish a challenge/response dialogue, thus enabling the user's remote identification and guaranteeing secure access to confidential areas/information within the WebServer.

The operations involved are implemented by a ActiveX or by a Java class on the server side component, whilst for the client side component by an ActiveX or a Plug In which communicates with the hardware token.

The following lay-out sums up the authentication operations via WebIdentity in a web ambit:

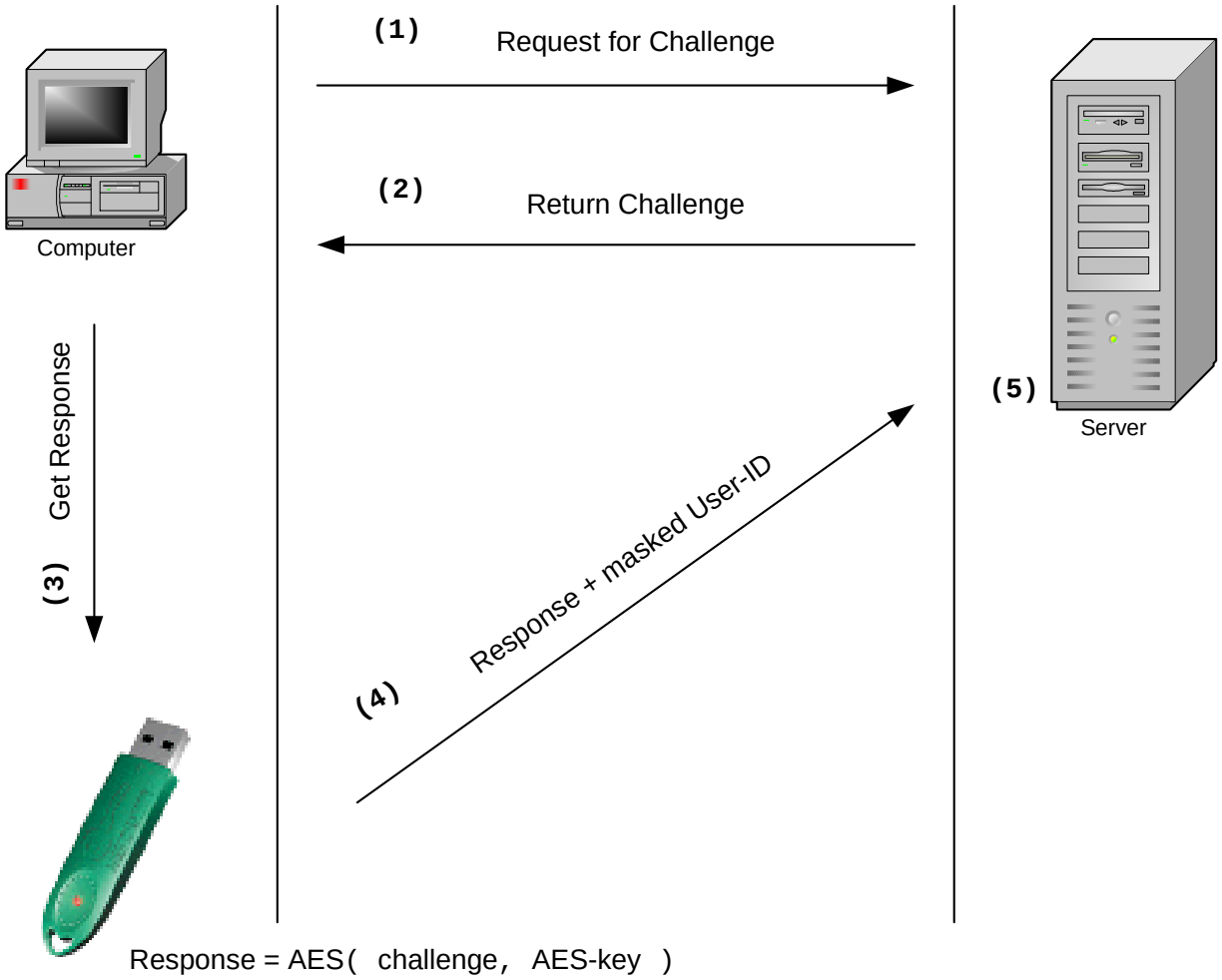

Figure 5.3 – Sequence Challenge/Response for authentication operation

1. In the first phase the client (browser) sends an http request to the web server (get, post).
2. The web server responds to the request by generating a challenge which is sent to the client.
3. The client queries the token by transmitting the challenge sent by the server for generating the response.
4. The token computes the response on board by running: `response = AES ( challenge, AES-key )` and then the client sends the server the response + user-id in encrypted format.
5. The server carries out all necessary operations for recalculating the response. As described at figure 5.4 the server computer the AES key (AES-key) locally by running a hash of the User-Id and then an AES computation by using the Server Secret as a key. The result, that is the client secret key, is used as a key for carrying out the AES of the challenge previously sent by the client. The result is the response recalculated by the server: if it turns out to be identical to the response sent by the client it means that the client is in possession of a valid token, that is, generated and initialized by the same service and therefore containing the same Server Secret. From a conceptual point of view, the computation of the client secret key can be considered as a hashing.

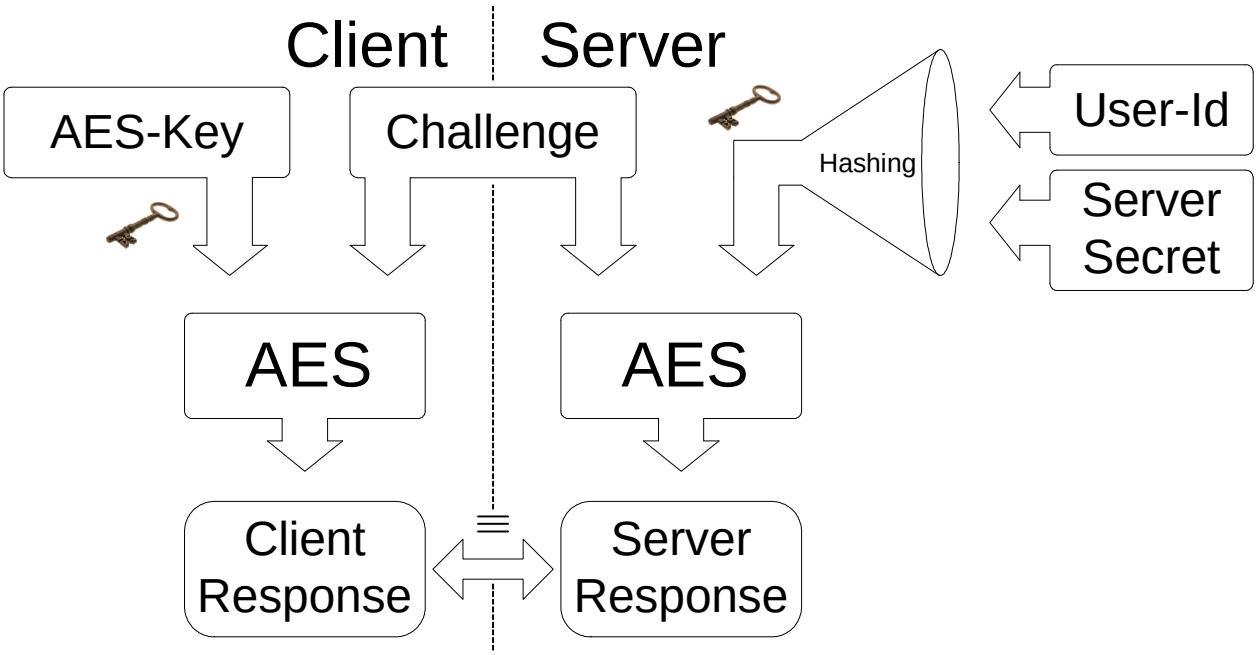

Figure 5.4 – Procedure for client authentication

The challenge is generated by the ActiveX of the server and is termed Random Session String. The Random Session String can depend on the time, on an id-session which typically all application servers make available for identifying the session, and on a string which, if present, contributes to the generation of the Random Session String. Having a string dependent on time and on external pseudo-random variables originates from the need to have univocal and non-repeatable transactions. They are univocal for preventing two transactions from being identical; non-repeatable for preventing the communication from being re-proposed in the attempt to deceive the server ("replay avoidance").

It is to be noticed that the Random Session String can be sent in plain-text format without jeopardizing security absolutely.

5.4.1 Example – log-in from a web page

The following example sums up the essential steps for integrating the authentication with WebIdentity into a web-based application with ASP; hereunder reported are only the descriptions that are strictly necessary for understanding the mere working; for a complete example please refer to section **Errore. L'origine riferimento non è stata trovata.**.

- The first step to take at the beginning of the transaction, after the client's request for a new connection, is the generation of a challenge string by the server including the relevant login page.

```
1→  Session("SessionString") = WISrv.InitRndSessionString (true,
                               Session.SessionID, "FREE-STRING")
```

1. ASP-VBScript code on the server side that is necessary to the generation of the Random Session String dependent on time (with "true" as first parameter, the time value is used as one of the components for the generation of the Random Session String), on the session id (Session.SessionID being the variable provided by ASP for identifying the session) as second parameter and on a random-chosen string as third parameter. The Random Session String is also stored in the session variable `Session("SessionString")` for being used in the client response check.

- Once the client receives the login page containing the challenge the user must insert the token and proceed to the authentication request. The client carries out the computation of the response by means of the token and sends it to the server. The following code describes an operating example for the computation of the response in client-side JavaScript code.

```
1→  function SendPIN()
      {
2→      document.WIDrvCli.RndSessionString =
                                          "<%= Session("SessionString") %>"
3→      document.WIDrvCli.Label = "<%= Application("wi_Label") %>"
4→      document.WebIdData.PIN.value = document.WIDrvCli.ReadPin();
        // Send PIN to server
5→      document.WebIdData.submit();
      }
```

1. Client-side JavaScript function used for the generation and forwarding of the response that is called for by the login page following an authentication attempt. The function must be called for by the web page following an event that occurred by pressing the website enter key, for instance.
2. Initialization of the client object with the Random Session String generated and sent by the server; the entry `RndSessionString` is a variable of the client ActiveX inside the HTML document; the `Session("SessionString")` entry is interpreted by the sever side and then replaced with a "Random Session String" string.
3. Initialization of the client object with the token identification Label; the Label entry is a client ActiveX variable inside the HTML document; the entry `Aplication("wi_Label")`is interpreted by the server side and then replaced with the label chosen for the service.
4. Call of `ReadPin` method by the client ActiveX inside the HTML document for the generation of the response (response + masked User-Id) to be sent to the server; the response value is assigned to the `PIN` variable contained in the `WebIdData` Form inside the HTML document.
5. Explicit call to the submit of the `WebIdData` form for actual forwarding via a GET or a POST of the PIN variable containing the response.

- After the server has received the response from the client it must verify its authenticity. The following server-side ASP-VBScript code describes the verification part of the response authenticity.

```
1→  WIDSrv.Password = Application("wi_Password")
2→  WIDSrv.RndSessionString = Session("SessionString")
3→  WIDSrv.DecryptPIN( Request.Form("PIN") )
4→  If WIDSrv.GetLastError() <> 0 then
          Response.Redirect "index.asp"
    endif
```

1. Initialization of the server object with the Server Secret; the `Password` entry is a server ActiveX variable inside the ASP page; the `Application("wi_Password")` entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
2. Initialization of the server object with the Random Session String generated at the start of transaction; the `RndSessionString` entry is a variable of the server ActiveX inside the ASP page; the `Session("SessionString")` entry is a session variable maintained by ASP and used for storing the Random Session String.

3. Call of `DecryptPIN` method of the server ActiveX for decrypting the response sent by the client and returned by the call to `Request.Form("PIN").` As illustrated in figure 5.4 the server computes the response with the User-Id and the Server Secret again. If the verification is successful the User-Id is stored in the `PIN` variable of the server ActiveX.
4. With the call to `GetLastError` the correct execution of the la test call to the server ActiveX is checked; in this case the response decryption is checked. In this case the user is verified for the possession of a WebIdentity token properly initialized by the same service (that is, initialized with its Server secret `Application("wi_Password")`).

Table 5.4 – Association of software functions /methods for authentication.

Function	*Method*	*Description*	*Component*
Challenge Generation	`InitRndSessionString`	Generation of time-variable Random Session String.	Server
Response Computation	`ReadPin`	Computation of response by using the WebIdentity token	Client
Response Verification	`DecryptPin`	Re-computation of the response for comparison with the client-sent response and extraction of the User-Id.	Server

5.5 Cryptography

With WebIdentity it is possible to execute web page encryption; in particular it is possible to encrypt single items of information both from the server towards the client and vice versa. Indeed some ad hoc scripts are used both on the server side and on the client side; they enable the server to encrypt the confidential information to be sent to the client and to decrypt the received information; on the client side they enable decrypting the information received by the server and encrypting the information to be sent. This way it is at the developer's discretion to choose which information is confidential and which is not.

In order to execute the WebIdentity encryption and decryption a symmetric algorithm named Blowfish [SCH96] is used. Blowfish uses a session key that is shared between the client and the server; it is generated by WebIdentity each time this is requested. The session key in use is 256 bit type; it is generated in run-time mode both by the server and by the client.

The client side generates the Blowfish symmetric key on board by using the AES 256 key that is stored in the token (3DES for WebIdentity3P) and the Random Session String generated and received by the server; the server generates the symmetric key with the User-Id provided in the authentication phase, the Server Secret stored by the server and by the Random Session String generated by the server itself. Figure 5.5 describes the process that is necessary for computing the session key on the client side and on the server side. Here follows the formula used for the session key computation.

```
Blowfish Key = AES ( AES-Key, Random Session String )
```

The Random Session String can depend on time, on a session-id (optional but strongly recommended) which typically all application servers put at disposal for identifying the session; it can also depend on a password (optional) which, if present, contributes to the generation of the Random Session String. The generation of the Random Session String can be different for each session but also when sending each page; important is that the same value is used for completing the challenge response operations; for instance when the server sends the client an encrypted data with a Random Session String the client must use the same Random Session String generated by the server for decrypting the data.

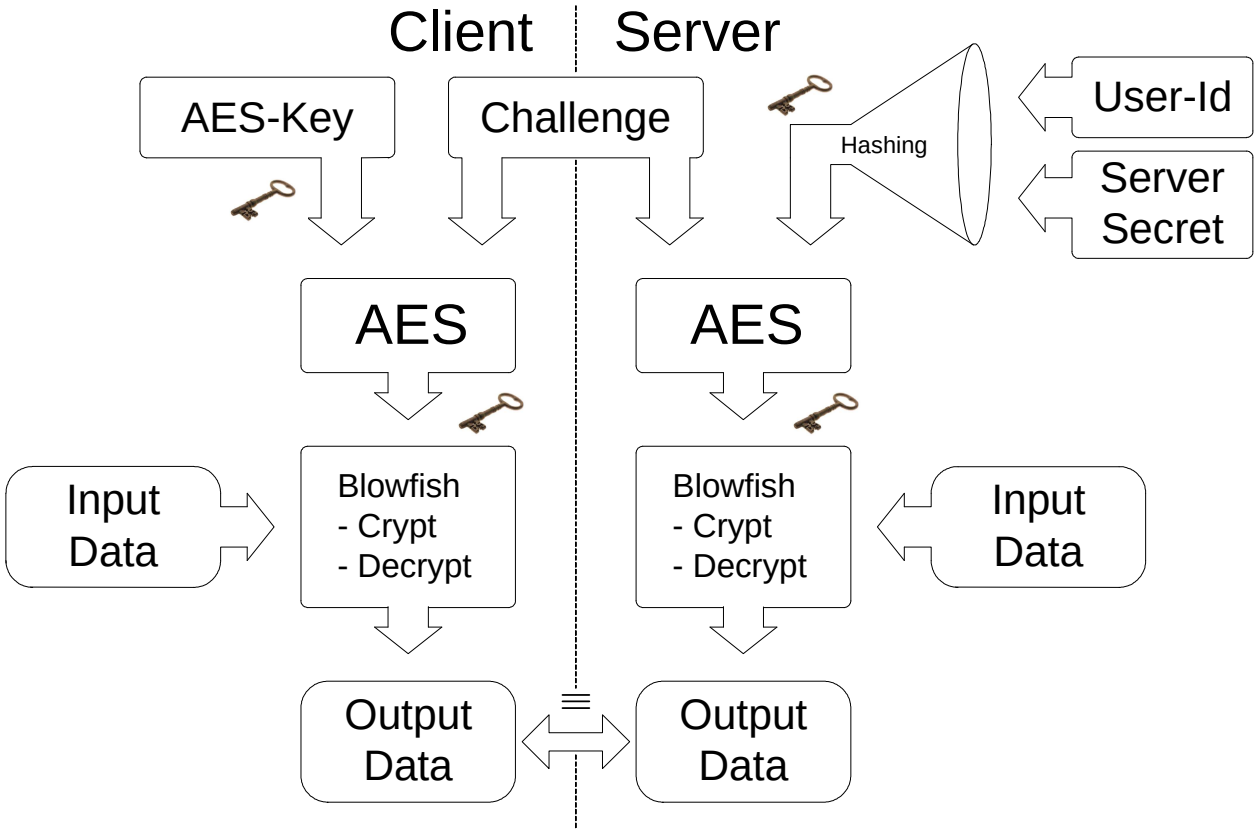

Figure 5.5 – Procedure for data cryptography

5.5.1 Example – cryptography from server to client

The following example illustrated the essential steps to take for integrating the WebIdentity cryptography into a web-based application with ASP; the following example shows the passage of an encrypted data from the server to the client; only the parts that are strictly necessary for describing the operation are reported hereunder; a complete example is provided at section **Errore. L'origine riferimento non è stata trovata.**.

- From the server side (ASP-VBScript code on server side) just a call to the `Crypt` method is necessary for encrypting the data in order to send it to the client.

```
1→  WIDSrv.pin = Session("user_PIN")
2→  WIDSrv.RndSessionString = Session("SessionString")
3→  WIDSrv.password = Application("wi_Password")
4→  CryptConfidentialData = WIDSrv.Crypt( ConfidentialData )
```

1. Initialization of WebIdentity server object with the User-Id transmitted during authentication by the client and stored in the session variable `Session("user_PIN")`; the entry `Pin` is an ActiveX server variable.
2. Initialization of the WebIdentity server object with the Random Session String; the entry RndSessionString is a variable of the server ActiveX; the `Session("SessionString")` stores the previously generated Random Session String.
3. Initialization of the server object with the Server Secret; the `Password` entry is a server ActiveX variable inside the ASP page; `the Application("wi_Password")` entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
4. Call of the `Crypt` method of the server ActiveX for encrypting the data to send to the client (`ConfidentialData`). The encrypted value is stored n the vbscript variable `CryptConfidentialData` to be transmitted to the client.

- From the client side (Javascript code on the client side) just a call to the `Decrypt` method is necessary for decrypting the data sent by the server.

```
1→  document.WIDrvCli.RndSessionString =
                                    "<% = Session("SessionString") %>";
2→  document.WIDrvCli.Label = "<% = Application("wi_Label") %>";
3→  ConfidentialData = document.WIDrvCli.Decrypt("<%= CryptConfidentialData %>");
    window.document.write(ConfidentialData);
```

1. Initialization of the WebIdentity client object with the Random Session String generated and sent by the server; the entry RndSessionString is a variable of the client ActiveX inside the HTML document; the Session("SessionString") entry is interpreted by the sever side and then replaced with a Random Session String.
2. Inizializzazione of the client object with the Label for identifying the token; the Label entry is a client ActiveX variable inside the HTML document; the entry `Application("wi_Label")` is interpreted by the server side and the replaced with the label chosen for the service.
3. Call of the Dec`rypt` method of the server ActiveX for decrypting the data sent by the server (`ConfidentialData`). In this case the decrypted value is displayed with a call to the Javascript `Write` function inside the HTML page.

Table 5.5 – Association of software functions / methods for cryptography.

Function	*Method*	*Description*	*Component*
Server side encryption	`Crypt`	Encryption with the use of Random Session String, of Server Secret and User-Id which the confidential data is to be sent.	Server
Server side decryption	`Decrypt`	Decryption with the use of Random Session String, of Server Secret and User-Id which the confidential data has been received from.	Server
Client side encryption	`Crypt`	Encryption with use of Random Session String.	Client
Client side decryption	`Decrypt`	Decryption with use of Random Session String.	Client

5.6 Remote control

The remote control feature and in the case in point the possibility to read from and write on the WebIdentity token memory is carried out with simple calls to the server ActiveX methods for creating the desired command and with the call of a client ActiveX method for the actual execution of the requested command. The command and all the necessary information for execution are encapsulated inside an encrypted string by the server ActiveX and decrypted and executed by the client ActiveX. The response containing the command status and all that concerns the result is encapsulated into a string encrypted by the client ActiveX and decrypted and controlled by the server ActiveX subsequently.

The commands that it is possible to execute on the token are the synchronization between the server and the token, the reading of the remote token memory and the writing into the remote token memory.

5.6.1 Synchronize Command

A counter has been introduced for preventing write commands from being reused on the key, as they might be contained in scripts that are locally stored on the computer. This counter is incremented at each reading on the token and it is necessary to state the value subsequent to the current one at each writing attempt. The Synchronize command enables synchronizing such a value, that is, knowing which number will be associated during the next writing into the memory. During the authentication phase, in addition to the authentication itself the writing counter is also synchronized, therefore it should not be necessary to use the Synchronize command explicitly unless in case of special needs.

The response of the synchronize command is an object containing the value of the progressive content on the remote token.

5.6.2 Read Command

The Read command enables reading from the remote token the full memory area or a specific memory area. The available memory on the token is about 8 KB. In the read command the start position of the reading and the length of readable data is indicated.

The response of the read command is an object containing the operation result, the initial position of said data, the length of the read data and the read data themselves.

5.6.3 Write Command

The Write command enables writing on the remote token the full memory area or a specific memory area. For security reasons a progressive counter is used for writing management; it prevents the repeated execution of the same command. In the write command the value subsequent to the current progressive value is specified, as well as the start position for writing, the writable data length and the data themselves.

The write command response is an object containing the result of the operation with the counter value for any subsequent writing operation.

All commands and information exchanged between the client and the server are encrypted with Blowfish following the same procedure that is described under section **Errore. L'origine riferimento non è stata trovata.**.

The following layout illustrates the operating sequence for the remote control:

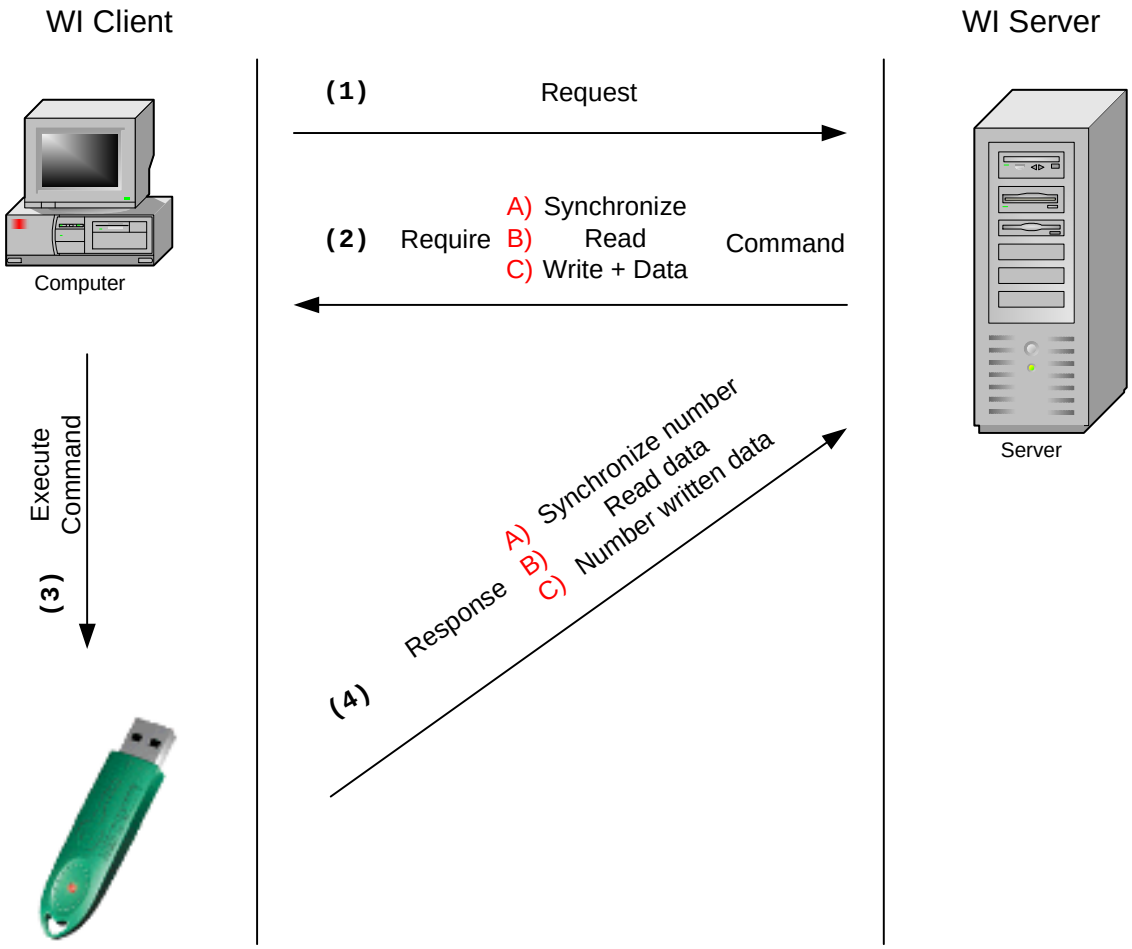

Figure 5.6 –Challenge/Response Sequence for remote control

1. In the first phase the client sends a request to the server.
2. The server answers the request by generating a response containing the request for a command in the shape of an encrypted string; these are some possible commands:
    a. `Synchronize` enables synchronizing the server with the progressive number of the remote token
    b. `Read` enables carrying out reading operations from the token memory
    c. `Write` enables carrying out writing operations on the token; together with the command also the writable data is transmitted
3. The client decrypts and interprets the string transmitted by the server and executes the command requested.
4. The client stores the result of the command into an encrypted string and sends it to the server, which can control the execution of the command; the response depends on the type of requested command and is described by the following list:
    a. The response to the Synchronize command is the progressive number stored by the token.
    b. The response to the Read command is the information requested by the token memory.
    c. The response to the Write command is the operation status.

5.6.4 Example – writing on the remote token

The following example sums up the essential steps to take for using the writing features on the remote token. Hereunder reported are the parts that are strictly necessary for the description of the operations.

The first step to take is the generation by the server side (server side ASP-VBScript code) of the write command; it is possible to write on the remote token in the desired location, therefore the parameters for the write command are the start position where data and data length are to be written.

```
    Var StartPosition, Length, DataString
    StartPosition = 1
    Length = 4
    DataString = "DATA"
1→  WIDSrv.pin = Session("user_PIN")
2→  WIDSrv.RndSessionString = Session("SessionString")
3→  WIDSrv.password = Application("wi_Password")
4→  WIDSrv.SyncNumber = Session("wi_SyncNumber")
5→  CmdString = WIDSrv.CommitWrite(StartPosition, DataString, Length);
```

1. Initialization of WebIdentity server object with the User-Id transmitted during authentication by the client and stored in the session variable Session("user_PIN"); the entry Pin is an ActiveX server variable.
2. Initialization of the server object with the Random Session String; the entry `RndSessionString` is a variable of the server ActiveX; the `Session("SessionString")` entry stores the previously generated Random Session String.
3. Initialization of the server object with the Server Secret; the Password entry is a server ActiveX variable inside the ASP page; the Application("wi_Password") entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
4. Initialization of the server object with the Sync Number; the `SyncNumber` entry is a server ActiveX variable; the `Session(“wi_SyncNumber”)` entry is a session variable maintained by ASP, where the Sync Number value has been previously stored. The Client and the Server can synchronize the Sync Number via the authentication procedure or via the explicit synchronization remote command.
5. Call of `CommitWrite` method of server ActiveX for generating the write command; the first parameter is the start position where data is to be written, the second parameter is the writable data itself and the third parameter is its length.

On the client side (client side Javascript code) it is enough to call the `CmdExecute` method of the WebIdentity client ActiveX for actually executing the command.

```
1→  document.WIDrvCli.RndSessionString =
                                    "<% = Session("SessionString") %>";
2→  document.WIDrvCli.Label = "<% = Application("wi_Label") %>";
3→  CmdResultString = WIDrvCli.CmdExecute( "<%= CmdString %>" );
4→  if ( WIDrvCli.GetLastError() != 0) {
        alert( WIDrvCli.GetLastErrorDescription() );
    }
```

1. Initialization of the WebIdentity client object with the Random Session String generated and sent by the server; the entry RndSessionString is a variable of the client ActiveX inside the HTML document; the `Session("SessionString")` entry is interpreted by the sever side and then replaced with a Random Session String.
2. Initialization of the WebIdentity client object with the Label for token identification; the Label entry is a client ActiveX variable inside the HTML document; the entry `Aplication("wi_Label")` is interpreted by the server side and then replaced with the label chosen for the service.
3. Call of ActiveX client `CmdExecute` method for executing the command; the transmitted parameter represents the string generated by the ActiveX WebIdentity server `CommitWrite` method. The result of the operation is stored in the `CmdResultString` Javascript variable for being sent to the server as an operation confirmation. `CmdExecute` executes the specified command inside the encrypted string.
4. With the `GetLastError` call it is possible to control the result of the write command for notifying the client in case the problem requires an intervention by the user, for instance token insertion, and therefore repeat the operation.

The last step necessary for completing the remote write operation is the control by the server of the result of the operation carried out by the client (server side ASP-VBScript code). The control is carried out by calling the ActiveX server `CmdResult` method which returns an object describing the operation status and sums up the operations executed on the remote token.

```
    Dim ObjRemoteCmdResponse
1→  WIDSrv.pin = Session("user_PIN")
2→  WIDSrv.RndSessionString = Session("SessionString")
3→  WIDSrv.password = Application("wi_Password")
    ‘ Check of the result (CmdResultString) sent from the client
4→  Set ObjRemoteCmdResponse = WIDSrv.CmdResult( CmdResultString )
5→  If ObjRemoteCmdResponse.status <> 0 then
          Response.Redirect "error.htm"
          Response.End
    End If
```

1. Initialization of WebIdentity server object with the User-Id transmitted during authentication by the client and stored in the session variable Session("user_PIN"); the entry Pin is an ActiveX server variable.
2. Initialization of the server object with the Random Session String; the entry RndSessionString is a variable of the server ActiveX; the Session("SessionString") entry stores the previously generated Random Session String.
3. Initialization of the server object with the Server Secret; the Password entry is a server ActiveX variable inside the ASP page; the Application("wi_Password") entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
4. The call to the ActiveX server `CmdResult` method for the actual control of the operation executed by the client ActiveX on the remote token. The trasnmitted parameter represents the string generated by CmdExecure of the client ActiveX which encapsulates the operation result. The return value represents a `RemoteCmdResponse` type object, describing the operation carried out and the result status.
5. For controlling the operation result it is just necessary to control the `Status` variable of the `ObjRemoteCmdResponse` object that is returned by the `CmdResult`.

Table 5.6 – Association of software functions / methods for remote control.

Function	*Method*	*Description*	*Component*
Read command	`CommitRead`	It enables generating the read command by stating the start position and the readable data length.	Server
Write command	`CommitWrite`	It enables generating the write command by stating the start position, the data length and the writable data.	Server
Synchronization command	`Synchronize`	It enables generating the synchronization command for aligning the counter of the writing number.	Server
Command execution	`CmdExecute`	It enables the actual execution of the command generated by the server on the token.	Client
Command execution control	`CmdResult`	It enables controlling and extracting the status and the return values of the remote command.	Server

6 Integration

The integration of WebIdentity into an existing project or its adoption into a new project includes a sequence of operations to carry out for identifying the necessary points on which work is to be focused.

First of all it is necessary to identify the positions inside the application where a control of the connected user identity is required and of how such a user can interact with the application. It is therefore necessary to create a connection between the user and the WebIdentity token by entering a reference into the user profile and positioning the user authentication scripts where necessary.

It is important to choose the Server Secret in an imaginative way so that it is not "guessed" easily. For a secure choice it is possible to use a random string with a length of at least 52 characters consisting of upper-case letters, lower-case letters and numbers. This instruction is given for providing a key with a bit information quantity that corresponds to the AES 256 bit key used by WebIdentityDL.

The location where to store and protect the Server Secret must also be chosen; it is also necessary to choose a Label for identifying the service and therefore for distinguishing it from other services; its value can be unique for the whole application or it can have different values for the same application for special needs.

The application part that requires more attention is the token initialization (a necessary operation for using the token) which must be carried out outside the application logic preferably.

Another important consideration is the possibility to maintain the user status inside the transaction in order to be able to manage the authentication in one single service point; this is possible with the aid of session persistence provided by the application server as described in the next paragraph.

Lastly, some expedients are not to be neglected, such as the security-related expedients, in particular the possibility to associate the client strong authentication provided by WebIdentity with the server strong authentication provided by SSL; such complementary feature is illustrated hereunder.5

6.1 Management of user-token association

To associate the token to its own user a new field shall be inserted inside the user database: it is the User-Id related to the WebIdentity token returned during the token initialization phase; this way each single user will be associated with the relevant WebIdentity token for service access.

In addition to the User-Id field it will be possible to insert other fields containing information relating to the policies (expiry dates, restrictions, etc.) made out for creating more complex user profiles. It is also possible to enter the information relating to the Label and the Server Secret in the database for making the application independent of the specific service; this way it will be possible to use the same web service scripts or programs for various web applications.

It is also possible, according to the case, to associate the WebIdentity key-based authentication with a further protection including the insertion of an alphanumeric string (Password) for obtaining a *two factor authentication*, that is:

1. the first factor is possessing something
2. the second factor is knowing something

Therefore for accessing the service it is necessary to possess a hardware token and know a password. A typical example is the Bancomat (cash dispenser) working; for being operated it requires knowing the password and possessing a Bancomat card.

6.2 Token initialization

The key initialization represents the implementation start-up of a web-based WebIdentity-protected service; by using the libraries provided with the WebIdentity SDK it is possible to initialize the “keys” by univocally associating them with the protected service and then with each user authorized to the service in question. The key initialization phase associates one single WebIdentity key with one service and one user univocally.

For initializing the keys it is necessary to use the WebIdentity server ActiveX after duly initializing it with the following parameters:

- Label – A label used for identifying the token on the USB bus, thereby identifying the service

- Server Secret[1] – Secret of the server for security control
- User Data – Identifying data of the user; for instance name, surname, etc.

As described by figure 5.2 the Label, the Server Secret and the User Data are used for computing and initializing the WebIdentity token. The label is inserted and used for identifying the token; the Server Secret is the secret of the server and jointly with the User-Id contributes to the creation of the symmetric key, which is the secret part of the token; the User Data is the data of the user and jointly with the Token Serial Number it contributes to the creation of the User-Id. The resulting data is entered in the write-only memory of the WebIdentity key.

Special attention must be given to the application for the token initialization, as during the generation phase the server side ActiveX must be used, together with the Server Secret; if used on an insecure channel this might seriously jeopardize the application security.

Differently from authentication and encryption, the initialization just requires using the Server ActiveX after duly initializing it with the Server Secret.

The ideal implementation might be a standalone application expressly prearranged for ensuring the Server Secret security and confidentiality. If a web-based application is used, it is necessary to adopt some expedients for guaranteeing the Server Secret protection; for ensuring confidentiality it is possible to use the HTTPS or WebIdentity, for authentication security a client SSL3 or WebIdentity certificate can be used instead.

6.2.1 Example –token initialization

This example clearly illustrates the sequence of necessary operations for writing a procedure for WebIdentity token initialization. The example is realized HTML with Javascript, therefore it is possible to use a browser (Internet Explorer) for execution. The WebIdentity server ActiveX is used after being downloaded and automatically initialized by the browser. The HTML page requires inserting the Server Secret, the Label and the User Data as requested for initialization.

```
    <html>
    <head>
    <title>Token Initialize</title>
    </head>
    <body>

1→  <object classid="clsid:A811A602-4655-11D3-A75A-00A0245382DE" id="WIDrvSrv"
          codebase="software/WISrv.cab#version=4,0,0,0"  VIEWASTEXT>
    </object>

    <script language="JavaScript">
    <!--
2→    function InitToken()
      {
3→      WIDrvSrv.Label = window.TxtLabel.value;
4→      WIDrvSrv.Password = window.TxtPwd.value;
        // Initialize Token
5→      window.TxtUserId.value = WIDrvSrv.InitDongle( window.TxtUser.value );
      }
    //-->
    </script>
      <p>Server Secret</p>
      <input id=TxtPwd name=TxtPwd size=24 value=""><br>
```

[1] Also called Server Password.

```
  <p>Label</p>
  <input id=TxtLabel name=TxtLabel size=24 value=""><br>
  <p>User Data</p>
  <input id=TxtUser name=TxtUser size=24 value=""><br>
  <input type="button" language="javascript" onclick="return InitToken()"
       value="Init Token"><br>
  <p>User-Id</p>
  <input id=TxtUserId name=TxtUserId size=50 value="">
</body></html>
```

1. Declaration of the WebIdentity Server ActiveX object by the HTML OBJECT tag; the OBJECT[1] tag enables inserting in the HTML document the ActiveX univocally identified by means of the class id, which for the client ActiveX is 878A0D61-48D2-11D3-A75D-00A0245382DE. The attribute identifies the object univocally inside the document by means of a label thus enabling interaction. The `CODEBASE` attribute is necessary in case ActiveX is not registered on the current machine. Such an attribute specifies the location (URL) from which it is downloaded. In addition to the URL it is possible to enter the control version (`#version=`) so that it can be downloaded, if more recent than the installed version.
2. Javascript `InitToken` function used for initializing the token.
3. Initialization of the server object with the Label for token identification; the Label entry is a server ActiveX variable inside the HTML document.
4. Initialization of the server object with the Server Secret; the `Password` entry is a server ActiveX variable inside the HTML document.
5. Call to ActiveX Server InitDongle method for the actual initialization of the hardware token; the parameter that is transmitted is the User Data entered in the HTML form; the User-Id that is returned by the `InitDongle` is assigned to the `TxtUserId` variable of the HTML form for display.

The example illustrated below the association of the (token) User-Id with the user profile has been left out, as previously explained at paragraph **Errore. L'origine riferimento non è stata trovata.**. However the user-token association is also important as it enables identifying the user following authentication.

6.3 Session operation

WebIdentity requires the use of certain server side variables such as, for instance, the Label, the Random Session String etc. For an easy management of such variables it is possible to use the *session* feature, which is available in all application servers nowadays. In addition to the server side variables it is possible to store also the user status, so as to be able to manage the authentication phase just on one application point or in specific areas of the service itself. However it is possible to implement the authentication at each step (page) if this turns out to be necessary. Using the session is a very useful method for managing the persistence of data between one page and another but it requires a mechanism which enables interrupting the session on request or after a certain time lapse; it is therefore advisable to insert – even in each page – the possibility to run the logoff manually on the user's request and to configure the session timeout suitably inside the application server or the web server (e.g. IIS) avoiding too small values, which hinder the service usability, as well as too big values which might jeopardize security in some cases.

[1] At present supported only by Internet Explorer 4.0+. For Netscape it is necessary to use a Javascript for installing the relevant Plug-In.

6.4 SSL operation

The components or properties characterizing a system for secure information exchange based on cryptography are data confidentiality, client / server authentication and information integrity.

Confidentiality means that the exchanged information is not understood by anybody outside the people involved in the transaction.

Integrity means that particular function which enables guaranteeing that personal data is original, that is, it has not been modified in any way by the version sent by the sender. It is to be noticed that integrity can be combined with confidentiality but also it may not; if information is not confidential it can be transmitted in plain text mode but the main thing is that it must be possible to check that it corresponds to the original message exactly.

Client secure authentication means the possibility to check and control for certain that the client is what it declares to be and therefore its authenticity.

Server secure authentication means the possibility for the client to verify for certain that the interlocutor server is actually what it declares to be and not, for instance, another server simulating the requested server.

Using WebIdentity provides secure client authentication and confidentiality. To guarantee also the other two security components, one possible standard-based solution is using SSL (Secure Sockets Layer) which manages secure server authentication, integrity and confidentiality. Therefore the two solutions can be considered complementary and in a position to provide an ideal solution for confidentiality, secure client authentication, secure server authentication and integrity.

Two possible operating solutions with SSL are displayed in the following table:

	Webidentity	SSL		Webidentity	SSL
Confidentiality		◯		◯	
Client Authentication	◯			◯	
Server Authentication		◯			◯
Integrity		◯			◯
	Combination 1			Combination 2	

In the first combination WedIdentity is used for client strong authentication and SSL for confidentiality, server strong authentication and integrity. In point of fact the whole SSL potential is used with the addition of client authentication which SSL[1] does not provide; in such a combination, development and integration time is reduced to a minimum, as the modifications to bring about are limited to the management of authentication whilst confidentiality is provided by SSL in a transparent mode.

In the second combination WebIdentity is entrusted with data confidentiality and client strong authentication whilst SSL is to provide server strong authentication and integrity. In this case WebIdentity cryptography function is used for encrypting single critical items of information, as distinct from SSL which encrypts everything or nothing. It is also to be considered that WebIdentity encryption enables keeping the information secure even when this is stored in the browser cache or stored locally: indeed, for displaying the pages that have been stored locally in encrypted format it is necessary to insert the token, on the contrary case their content would be unreadable as far as the WebIdentity encrypted

[1] SSL version 2

parts are concerned. In the second combination, for adding the WebIdentity confidentiality feature it is necessary to write ad hoc scripts for managing the encryption from the server side with the client side decryption and vice versa, that is to say, a greater integration effort.

Security at large

It is important to notice that adopting solutions that can provide encryption for guaranteeing confidentiality, authentication and integrity is just one of the aspects concerning the overall security of a web-based or client-server service. Some other important aspects are for instance the expedients to adopt during the actual implementation of the service , such as for instance the control of subjects for information exchange; a typical error is buffer overflow, which enables executing a code on a remote service under special circumstances. Attention must also be given to constant and timely updating of the web server and of the application server by downloading the relevant available updates and patches. Finally it is important to adopt a firewall system for protecting the web services and for avoiding any attacks directed at protocol vulnerability.

Security at large

It is important to notice that adopting solutions E' importante notare che l'adozione di soluzioni in grado di fornire crittografia per garantire confidenzialità, autenticazione e integrità sono solo alcuni degli aspetti che riguardano la sicurezza generale di un servizio web-based o client-server. Altri aspetti importanti sono per esempio gli accorgimenti da adottare durante l'implementazione effettiva del servizio come per esempio il controllo degli argomenti per lo scambio di informazioni; un tipico errore è il buffer overflow che permette in alcune circostanze l'esecuzione di codice sul server remoto. Attenzione va data anche all'aggiornamento continuo e tempestivo del web server e dell'application server con i rispettivi aggiornamenti e patches disponibili. Infine è importante adottare un sistema di firewall in grado di proteggere i servizi web ed evitare attacchi rivolti alle vulnerabilità dei protocolli.

7 Examples

The following examples introduce code pieces for WebIdentity integration, for authentication and for encryption in a web-based application. The examples are written in VBScripts for ASP on the server side and in Javascipt and HTML on the client side; for use it is necessary to have IIS and the properly-configured Access driver on the server side whilst for the client side it is necessary to have Microsoft Internet Explorer.

In the following examples some values stored in the session variables and in the application variables are used:

`Session("SessionString")`	Value of Random Session String that is initialized where requested inside the ASP pages:
	`Session("SessionString") =`
	`  WIDS.InitRndSessionString ( true,`
	`Session.SessionID,`
	`"FREE-STRING" )`

`Session("user_PIN")`	Value of User-Id stored in the authentication phase for use inside the http session.
`Application("wi_Label")`	Variable declared in `global.asa` and initialized by taking the value from the database. It is used for identifying the token, which is service-related and therefore unique for the whole application.
`Application("wi_Password")`	It is Server Secret, used for security management; this variable is declared in `global.asa` , the value of which is taken from the database.

For storing the information an Access database is used with a *User* table containing the user's personal data (PIN, Name, ...)

7.1 Authentication

This example clarifies the ASP pages that are necessary for using WebIdentity for authentication purposes.

The authentication process follows the challenge/response sequence introduced in section 5.4 and is implemented with the aid of two ASP pages:

- The former page is used as a response to the client authentication request and is useful for generating the challenge on the server side and a form necessary for the authentication request by the user, for using the token for response generation and for sending the response to the server.
- The latter page is used for effectively controlling the authentication by interpreting the response sent by the client by verifying the User-Id validity.

The next page generates one form and the challenge that is necessary to the user for fulfilling recognition. First thing it generates the challenge univocally with the aid of the server ActiveX and then it created the form to send to the client, containing the challenge and the scripts that are necessary for querying the token for response generation.

The `index.asp` page is implemented in HTML with Javascripts for execution on the client side and VBScripts for execution of the server side.

```
<%
Dim WIDS
1→ Set WIDS = Server.CreateObject("WISrv.WebIdSrv")
2→ Session("SessionString") = WIDS.InitRndSessionString(1, Session.SessionID,
                                  "AeCdEfGhIlMnOpQrStUvZ")
Set WIDS = Nothing
%>
<html>
<head>
<title>User Data</title>
</head>
<body>
3→ <object classid="clsid:878A0D61-48D2-11D3-A75D-00A0245382DE" id="WIDC"
        codebase="software/WICli.cab#version=4,0,0,0"  VIEWASTEXT>
   <embed type="application/x-wicli-plugin" name="WIDC" width="1" height="1"
        hidden="true">
</object>
<script language="JavaScript">
<!--
4→ function SendPIN()
{
5→   document.WIDC.RndSessionString =
                                   "<%= Session("SessionString") %>"
6→   document.WIDC.Label = "<%= Application("wi_Label") %>"
7→   document.WebIdData.PIN.value = document.WIDC.ReadPin();
8→   document.WebIdData.submit();
}
//-->
</script>
<form action="login.asp" method="POST" name="WebIdData" >

<p>Site Entry</p><br>
9→   <input type="button" language="javascript"
        onclick="return SendPIN()" value="Enter">
10→  <input type="hidden" name="PIN">

</form>
</body></html>
```

1. Creation of ActiveX WebIdentity Server object necessary for the univocal generation of the challenge.
2. Generation of the Random Session String dependent on time (with “true” as first parameter, the time value is used as one of the components for the generation of the Random Session String), on the session id (Session.SessionID being the variable provided by ASP for identifying the session) as second parameter and

on an option string as third parameter. The Random Session String is stored in the session variable `Session("SessionString")` for being used in the client response check.

3. Request of the WebIdentity Client ActiveX object; the OBJECT tag enables inserting in the HTML document the ActiveX univocally identified by means of the class id, which for the client ActiveX is `878A0D61-48D2-11D3-A75D-00A0245382DE.` The `ID` attribute identifies the object univocally inside the document by means of a label thus enabling interaction. The CODEBASE attribute is necessary in case ActiveX is not present on the client. Such an attribute specifies the location (URL) from which the ActiveX is downloaded automatically. In addition to the URL it is possible to enter the control version (`#version=`) so that it can be downloaded, if more recent than the installed version. The `EMBED` tag enables inserting in the HTML document the client WebIdentity Plug-In univocally identified with the `TYPE application/x-wicli-plugin` tag.
4. `SendPIN` function used during the submit phase for reading the User-Id from the token and for sending the response to the web server.
5. Initialization of the WebIdentity client object with the Random Session String generated and sent by the server; the entry RndSessionString is a variable of the client ActiveX inside the HTML document; the `Session("SessionString")` entry is interpreted by the sever side and then replaced with a Random Session String.
6. Initialization of the WebIdentity client object with the Label for token identification; the Label entry is a client ActiveX variable inside the HTML document; the entry `Aplication("wi_Label")` is interpreted by the server side and then replaced with the label chosen for the service.
7. Call of ReadPin method by the client ActiveX inside the HTML document for reading the encrypted User-Id form the WebIdentity token; the value of the User-Id is assigned to the PIN variable contained in the WebIdData Form inside the HTML document. The ReadPin is the function that generates the response to send to the server.
8. Explicit call to the submit of the WebIdData form for actual sending to the web server via a POST of the PIN variable containing the response.
9. Button inside the HTML document for authentication request by the user after inserting the WebIdentity token.
10. Variable hidden inside the HTML document used for storing the encrypted User-Id to send to the web server.

The following page manages the necessary operations for user verification and authentication; first thing it verifies the validity of the response sent by the client and extracts the User-Id; then it control the presence of the User-Id inside the local database for reading the profile of the token-related user. The connection between the user profile and the token is carried out with the User-Id returned by the client and the User-Id stored in the database. If the user turns out to be valid the control is entrusted with the actual home page of the service; otherwise the previous authentication page is re-proposed.

The ASP (login.asp) page is written in VBScript with the aid of ADO for interfacing with the Access database.

```
<%
Dim WIDS
1→ Set WIDS = Server.CreateObject("WISrv.WebIdSrv")
2→ WIDS.Password = Application("wi_Password")
3→ WIDS.RndSessionString = Session("SessionString")
4→ WIDS.DecryptPIN( Request.Form("PIN") )
5→ If WIDS.GetLastError() <> 0 Then
       Response.Redirect "index.asp"
       Response.End
End If
Dim rsUser, sqlUser, fOk
Set rsUser = CreateObject("ADODB.Recordset")
sqlUser = "SELECT * FROM User WHERE User.PIN = '" &
          Server.HTMLEncode(WIDS.pin) & "'"
6→ rsUser.Open sqlUser, Application("Users_ConnectionString")
if Not rsUser.eof Then
       Session("user_PIN") = WIDS.pin
       fOk = 1
Else
       fOk = 0
End if
rsUser.Close
Set rsUser= Nothing
Set WIDS = Nothing

If fOk=1 Then
       Server.Transfer "main.asp"
Else
       Response.Redirect "index.asp"
End If
%>
```

1. Creation of the Server WebIdentity ActiveX object that is necessary for response verification and interpretation.
2. Initialization of the server object with the Server Secret; the `Password` entry is a server ActiveX variable inside the ASP page; the `Application("wi_Password")` entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.

3. Initialization of the server object with the Random Session String; the same string previously communicated to the client; the entry RndSessionString is a variable of the server ActiveX; the Session("SessionString") stores the previously generated Random Session String.
4. Call of `DecryptPIN` method of the server ActiveX for decrypting the response sent by the client and retrieved by the call to `Request.Form("PIN")` which returns the corresponding value of the `PIN` variable of the HTML form. The User-Id is stored in the `PIN` variable of the server ActiveX.
5. Verification of the User-Id read operation; if the result is 0 it means that the client is in possession of a token properly initialized by the same service.
6. The previous control has verified the token validity; to verify the user status (user profile) a token-user connection has been created in the user information file in the local database. The research in the received User-Id local database enables associating the user with his/her profile and his/her current status. In this case if the user proves registered in the database he/she can be authenticated, otherwise he/she is readdressed to the login page.

```
<%
Dim Name, rsUser, sqlUser
Set rsUser = CreateObject("ADODB.Recordset")
sqlUser = "SELECT * FROM User WHERE User.PIN = '" &
          Server.HTMLEncode( Session("user_PIN") ) & "'"
rsUser.Open sqlUser, Application("Users_ConnectionString")
if rsUser.eof Then
       Response.Redirect "index.asp"
       Response.End
End If
Name = rsUser("Name")
rsUser.Close
Set rsUser= Nothing
Dim WIDS
Set WIDS = Server.CreateObject("WISrv.WebIdSrv")
Session("SessionString") = WIDS.InitRndSessionString(1, Session.SessionID,
                           "AeCdEfGhIlMnOpQrStUvZ")
WIDS.pin = Session("user_PIN")
WIDS.RndSessionString = Session("SessionString")
WIDS.password = Application("wi_Password")
Dim CryptName
CryptName = WIDS.Crypt( Name )
Set WIDS = Nothing
%>
```

7.2 Cryptography

The following example shows the use of WebIdentity for data encryption both by the server to the client and vice versa. WebIdentity is used for encrypting the `Name` information (containing the name associated to the user retrieved from the `User` table of the database) both when it is passed by the server to the client for display and when the client passes it to the server for modification. The application retrieves the requested content from the local database, in this case the Name field associated with the user, and encrypts it by using the WebIdentity server ActiveX object with an expressly-initialized Session Random String. Hereunder the HTML page is generated; it contains the requested form with the encrypted datum and a series of Javascripts for proper running. While running the HTML page on the client browser the Javascripts `OnLoad_Populate()` function is automatically executed; with the aid of the client ActiveX is decrypts the information passed by the server and enters the decrypted value in the special field. During field modification by the user and during the request for modification transfer to the server the `OnSubmit()` Javascripts function is called for encrypting the data to send and carrying on with transfer. The encrypted data arrives to the server which decrypts it and enters it in the local database.

The web page, prearranged for sending confidential information to the client, is represented by the following listing (`userdata.asp`):

```
1→
2→
3→
4→
5→
6→
e
7→
```

```
<html>
<head>
<title>User Data</title>
</head>
<object classid="clsid:878A0D61-48D2-11D3-A75D-00A0245382DE" id="WIDC"
        codebase="software/WICli.cab#version=4,0,0,0"  VIEWASTEXT>
   <embed type="application/x-wicli-plugin" name="WIDC" width="1" height="1"
        hidden="true"></object>
<script language="JavaScript">
<!--
function OnLoad_Populate()
{
  var Name
  document.WIDC.RndSessionString = "<% = Session("SessionString") %>";
  document.WIDC.Label = "<% = Application("wi_Label") %>";
  Name =  document.WIDC.Decrypt( "<%= CryptName %>" );
  document.WebIdData.Name.value =  Name;
}
function OnSubmit()
{
  var Name
  document.WIDC.RndSessionString = "<% = Session("SessionString") %>";
  document.WIDC.Label = "<% = Application("wi_Label") %>";
  Name = document.WebIdData.Name.value;
  document.WebIdData.Name.value = document.WIDC.Crypt(Name);
  document.WebIdData.submit();
}
//-->
</script>
<body onload="OnLoad_Populate()">
<form action="changeuserdata.asp" method="POST" name="WebIdData" >
  <p>Name:</p>
  <INPUT TYPE="text" NAME="Name">
  <INPUT TYPE="button" VALUE="Confirm" ONCLICK="return OnSubmit()">
</form>
</body></html>
```

1. The `Name` datum to pass to the client is retrieved from the local database `User` table; the User-Id is used that was retrieved and stored in the `Session("user_PIN")` session variable during authentication for retrieving the user profile.
2. Creation of the ActiveX WebIdentity Server object necessary for encrypting the data to send.
3. Generation of the Random Session String dependent on time (with "true" as first parameter, the time value is used as one of the components for the generation of the Random Session String), on the session id (Session.SessionID being the variable provided by ASP for identifying the session) as second parameter and on an option string as third parameter. The Random Session String is also stored in the session variable `Session ("SessionString")` for being used for decrypting the information sent by the client.
4. Initialization of WebIdentity server object with the `User-Id t`ransmitted during authentication by the client and stored in the session variable `Session("user_PIN")`; the entry Pin is an ActiveX server variable.
5. Initialization of the server object with the Random Session String; the entry RndSessionString is a variable of the server ActiveX; the `Session("SessionString")` entry stores the previously generated Random Session String.
6. Initialization of the server object with the Server Secret; the Password entry is a server ActiveX variable inside the ASP page; the `Application("wi_Password")` entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
7. Call of the `Crypt` method of the server ActiveX for encrypting the data to send to the client (`Name`). The encrypted value is stored in the vbscript variable CryptName to be transmitted to the client.
8. Request of the ActiveX WebIdentity Client object; the OBJECT tag enables inserting in the HTML document the ActiveX univocally identified by means of the class id, which for the ActiveX client is `878A0D61-48D2-11D3-A75D-00A0245382DE`. The `ID` attribute identifies the object univocally inside the document by means of a label, thus enabling interaction. The `CODEBASE` attribute is necessary in case ActiveX is not present on the client. Such an attribute specifies the location (URL) from which the control is downloaded automatically. In addition to the URL it is possible to enter the control version (`#version=`) so that it can be downloaded, if more recent than the installed version. The EMBED tag enables inserting in the HTML document the client WebIdentity Plug-In univocally identified with the TYPE application/x-wicli-plugin tag.
9. `OnLoad_Populate` function used in the form generation phase for initializing the content fields sent by the server in encrypted format; the function sees to decrypting the data sent by the server and to setting the `Name` field with the corresponding value.
10. Initialization of the WebIdentity client object with the Random Session String generated and forwarded by the server; the entry RndSessionString is a variable of the client ActiveX inside the HTML document; the `Session("SessionString")` entry is interpreted by the sever side and then replaced with a Random Session String.
11. Initialization of the client object with the token identification Label; the Label entry is a client ActiveX variable inside the HTML document; the entry `Aplication("wi_Label")`is interpreted by the server side and then replaced with the label chosen for the service.
12. Call of the `Decrypt` method of the client ActiveX for decrypting the data sent by the server (`Name`).
13. Setting of variable in the form with the user name in plain text.
14. `OnSubmit` function used during modification of the name value for transmission to the server.
15. Initialization of the client object with the Random Session String generated and sent by the server; the entry RndSessionString is a variable of the client ActiveX inside the HTML document; the `Session("SessionString")` entry is interpreted by the sever side and then replaced with the "Random Session String" string.

16. Initialization of the client object with the token identification Label; the Label entry is a client ActiveX variable inside the HTML document; the entry `Aplication("wi_Label")` is interpreted by the server side and then replaced with the label chosen for the service.
17. Call of the `Crypt` method of the client ActiveX for encrypting the data to send to the client (`Name`). The encrypted value is stored in the `Name` variable of the `WebIdData` form.
18. Explicit call to the form submit for sending the encrypted `Name` variable to the web server with POST.

The web page, prearranged for receiving the encrypted information by the client, is represented by the following listing (`changeuserdata.asp`):

```
    <%
    Dim WIDS
1→  Set WIDS = Server.CreateObject("WISrv.WebIdSrv")
2→  WIDS.pin = Session("user_PIN")
3→  WIDS.RndSessionString = Session("SessionString")
4→  WIDS.password = Application("wi_Password")
    Dim Name
5→  Name = WIDS.Decrypt( Request.Form("Name") )

    Dim objConn, strSQL
    Set objConn = Server.CreateObject("ADODB.Connection")
    objConn.Open Application("Users_ConnectionString")

    strSQL = "UPDATE User SET Name='" & Name & "' WHERE User.PIN = '" &
             Server.HTMLEncode(WIDS.pin) & "'"

6→  objConn.Execute strSQL
    objConn.Close
    Set objConn = Nothing
    Set WIDS = Nothing
    %>
    <html>
    <head>
    <title>User Data</title>
    </head>
    <body>
      <p>OK</p>
    </body></html>
```

1. Creation of the ActiveX WebIdentity Server object necessary for decrypting the information transmitted by the client.
2. Initialization of WebIdentity server object with the `User-Id` transmitted during authentication by the client and stored in the session variable `Session("user_PIN")`; the entry Pin is an ActiveX server variable.
3. Initialization of the server object with the Random Session String; the RndSessionString entry is an ActiveX server variable; the `Session("SessionString")` entry stores the previously generated Random Session String which is identical to the string passed to the client for encrypting the information.
4. Initialization of the server object with the Server Secret; the Password entry is a server ActiveX variable inside the ASP page; the `Application("wi_Password")` entry is an application variable maintained by ASP, where the Server Secret value has been previously stored.
5. Call of the `Decrypt` method of the server ActiveX for decrypting the data sent by the client (`Name`).
6. Storage of the value in the `Name` field of the `User` table in the local database.

8 Appendixes

8.1 Symmetric cryptography

Symmetric algorithms use a single numeric key used both for encrypting and decrypting data. A secure message is encrypted for the receiver by using a key that is known only to the sender and the receiver. The principle on which the algorithm is based is that by encrypting a plain text with a certain key and by decrypting the encrypted text with the same key it is possible to obtain the same plain text again (see figure 8.1).

$$D_k(E_k(P_k)) = P$$

Where:

P = plain text

E = encrypting operation

D = decrypting operation

K = key

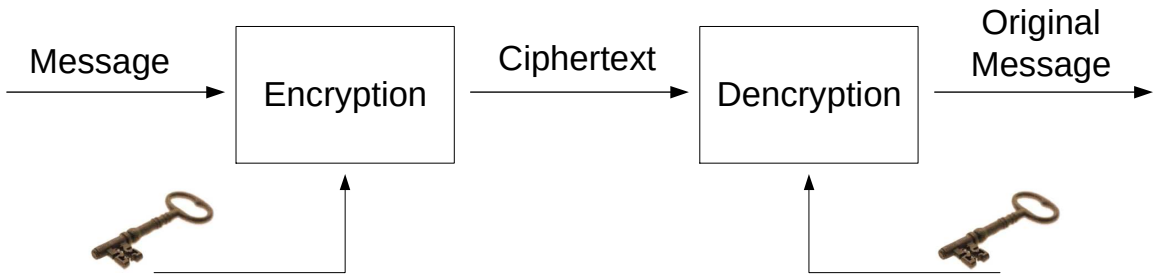

Figure 8.1 – Operating layout of a symmetric algorithm

The most common symmetric algorithms are AES, CAST, DES, 3DES, RC2, RC4, RC5, IDEA, Blowfish.

8.2 AES

The Advanced Encryption Standard (AES),

is a block encryption algorithm used as standard by the USA government

AES is the implementation of a more general algorithm known as Rijndael.

AES can have 128, 192, or 256 bit encryption keys. WebIdentity uses 256 bit encryption.

To date is is considered an extremely secure algorithm and it can be estimated that it will be used successfully all over the world as it happened to its predecessor, the Data Encryption Standard (DES).

See [AES197] or:

http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

http://it.wikipedia.org/wiki/Advanced_Encryption_Standard

8.3 DES

The Data Encryption Standard (DES) is an encryption algorithm chose as standard by the Federal Information Processing Standard (FIPS) for the USA government in 1976 and then adopted worldwide.

DES is reckoned to be insecure for a number of applications. Its insecurity derives from the key used for encrypting the messages, which is only 56 bit.

http://en.wikipedia.org/wiki/Data_Encryption_Standard

http://it.wikipedia.org/wiki/Data_Encryption_Standard

8.4 Triple DES

Triple DES is a block encryption system based on the repetition of \ref DES for three times.

Triple DES was introduced for making the DES secure. This algorithm expands the key by preventing a brute force attack even if it makes the algorithm virtually vulnerable to attacks, which are not feasible in practice.

Triple DES 2EDE is made up of a 128 bit key (only 112 bits are useful) where the encrypted data is obtained by applying the DES with the first 56 bits of the key, then decryption is applied with the second part of the key, and finally a new encrypting is carried out with the first part of the key.

8.5 Hashing algorithms

By hash functions those functions generally defined as « one-way hash function » are meant; hereunder a definition is provided.

Suppose that M is an arbitrary length message and h a fixed length binary string.

The “one way hash function” H function is defined as follows:

h = H(M)

Besides, for H it is true:

1. Given M, it is « easy » to calculate h.
2. Given h, it is “difficult” to calculate M so that H(M) = h.
3. Given M, it is “difficult” to find another M’ message so that H(M) = H(M’).

In many applications the following property is also required:

4. it is « difficult » to find two such M and M’ messages that H(M) = H(M’).

By “easy” a calculation that any computer can carry out in an extremely short time is meant, whilst by “difficult” a calculation that it is impossible to carry out in reasonable time is meant at the state-of-the-art (in terms of days, weeks, months) even with the aid of exceptionally powerful computation resources.

Points 1 and 2 of the above-given definition identify the unidirectional characteristic (one way) required to Hash algorithms: given an x value it is easy to calculate $f(x)$, but given $f(x)$ it is difficult to extract x from a computational point of view.

Point 3 states the collisions are improbable, that is, difficult.

It is understood as computationally impossible for the same hashing value to correspond to two different values. Given an x and y value so that x is different from y then the x ($f(x)$) hashing value will be different from the y ($f(y)$) hashing value.

The value obtained from a hashing algorithm is called *message digest.*

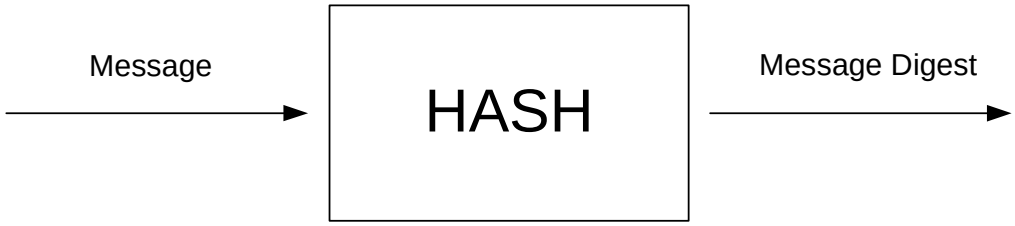

Figure 8.2 – Operating layout of hash algorithms

Examples of hashing algorithm are SHA-1, MD2, MD4, MD5.

8.6 Glossary

User-Id	User identity dependent on the key personal data ad univocal data.
PIN	Alternative name for indicating the User-Id. Please refer to the User-Id definition.
Label	Identifier assigned to each service that is necessary for identifying the token on the bus.
Server Secret	Server secret that allows secure operations.
Password Server	Alternative name for indicating the Server Secret. See the definition of Server Secret.
password	Alternative name for indicating the Server Secret. See the definition of Server Secret.
Challenge	Random Session String; the most univocal string possible used for generating authentications that are always univocal and unforeseeable.
Response	Response that the client processes following an authentication attempt.
Hash	Function describer in the appendix **Errore. L'origine riferimento non è stata trovata.**.
Sync Number	Synchronization number used in the write remote operation.

8.7 Bibliography

[AES197] Announcing the ADVANCED ENCRYPTION STANDARD - AES FIPS PUB 197, November 2001.

[RFC1319] B. Kaliski, “The MD2 Message-Digest Algorithm”, Request for Comment 1319, April 1992.

[RFC1320] R. Rivest, “The MD4 _Message-Digest Algorithm”, Request for Comment 1320, April 1992.

[RFC1321] R. Rivest, “The MD5 _Message-Digest Algorithm”, Request for Comment 1321, April 1992.

[FLAN98] D. Flanagan, “JavaScript The Definitive Guide - Third Edition”, O’REILLY, June 1998.

[BASP99] D. Bruser, J. Kauffman, J. T.Llibre, B. Francis, D. Sussman, C. Ullman, J. Duckett, “Beginning Active Server Pages 3.0”, Wrox Press Ltd., 1999

[PASP99] A. Homer, D. Sussman, B. Francis, D. Esposito, A. Chiarelli, S. Robinson, R. Anderson, G. Reilly, C. McQueen, C. Blexrud, D. Denault, J. Schenken, B. Kropog, D. Sonderegger, M. Gibbs, “Professional Active Server Pages 3.0”, Wrox Press Ltd., 1999

[SCH96] B. Schneier, “Applied Cryptography, Protocols, Algorithms and Source Code in C”, Second Edition, Iohn Wiley & Sons Inc., 1996.

[SMIT97] R. E. Smith, “Internet Cryptography”, Addison-Wesley, 1997.