WebOTP
Developer’s Manual
v1.2

The WebOTPtime hardware key is in compliance with the following test specification:

EN 61000-4-2; EN 61000-4-3; CISPR22

as required by:

EN 61000-6-1, EN 61000-6-2, EN 61000-6-3, EN 61000-6-4

which are specified for the following test:

“ESD Immunity test”

“Radiated radio-frequency and electromagnetic field immunity test”

“Radiated Emission Verification”

in compliance with the “Essential Requisites” for the EMC Directive 89/336/EEC & 2004/108/EEC

FCC ID: TFC-AAJ

Eutronsec Spa
WebOTPtime
Supply: 5V DC
Absorption: 20 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 layout of this manual and of WebOTP distribution.

1.1 About this manual

This manual describes the *WebOTP* product and how to operate and integrate it into an authentication service.

1.2 How to use this manual

This manual is meant for the developer who needs to integrate the *WebOTP* product into an already existing authentication service or that is underway.

The basic concepts and the architecture of WebOTP are herein described, along with its operating modes.

Chapter 1: Preface describes the content and the layout of the manual.

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

Chapter 3: Getting Started describes the installation of the SDK and its first operation.

Chapter 4: Requirements lists the hardware and software requirements for using WebOTP.

Chapter 5: Authentication describes the different types of authentication used by WebOTP.

Chapter 6: Device describes the features of the hardware device**.**

Chapter 7: SDK describes the features of SDK software.

Chapter 8: Integration describes the integration process between SDK, the device and the authentication service.

1.3 Assumptions

In order to read this manual the knowledge of basic programming concepts is required, as well as of authentication and web service providing.

In particular:

- The knowledge of C language basic concepts can be useful in order to better understand the use of SDK.
- The knowledge of authentication basic concepts via OTP protocols (One Time Password) is useful for fully understanding the services provided by SDK.
- The knowledge of issues connected with web provided services is useful for understanding the issues connected with WebOTP integration into already-existing services.

1.4 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 use for the first time.

The `Courier` font is used for:

- Source
- File names

The SMALL CAPITAL font is used for:

- Product names
- Company names
- Registered Trademarks

1.5 Material supplied

The material supplied with WebOTP is subdivided into hardware and software components and documentation.

1.5.1 Hardware components

The SDK demonstrative release is supplied with a WebOTP device that is working, demonstrative and already initialized.

The demo device is always initialized with the cryptographic *server-key* and *blob-key* set to `202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F` and with the serial set to `1`.

The relevant initialization information is contained in the `WebOTPDemo.csv` file.

1.5.2 Software components

The following software components are enclosed with WebOTP:

SDK for the developer
SDK for running WebOTP in the server platform. Present in the dll/, activex/ and dotnet/directory of distribution.

Server demo
Illustrative example of a server installation for using WebOTP via a web service. Present in the demo/directory of distribution.

The files are organized according to the following structure:

File	**Description**
`WebOTPDeveloperManual.pdf`	This manual for the WebOTP developer.
`WebOTPTest.exe`	A test application for the WebOTP devices.
`WebOTPDemo.csv`	The initialization information of the demo WebOTP devices.
`demo/`	Illustrative example of a web server with authentication carried out by means of WebOTP.
`demo/readme.txt`	Steps for installing and using the illustrative example of a web server.
`dll/`	Library for using the WebOTP features through a DLL.
`dll/webotp.h`	Include file for the C and C++ languages.
`dll/WebOTPDLLReference.chm`	The DLL library documentation in the CHM format.
`dll/amd64/`	The DLL for amd64 architecture.
`dll/i386/`	The DLL for i386 architecture.
`Activex/`	Library for using the WebOTP features through ActiveX.
`Activex/WebOTPActiveXReference.chm`	The ActiveX library documentation in the CHM format.
`Activex/amd64/`	The ActiveX for amd64 architecture.
`Activex/i386/`	The ActiveX for i386 architecture.
`dotnet/`	Library for using the WebOTP features with .NET language.
`dotnet/WebOTPdotNETReference.chm`	The .NET library documentation in the CHM format.
`dotnet/1.1/`	The .NET library for Framework 1.1
`dotnet/2.0/`	The .NET library for Framework 2.0

1.5.3 Documentation

The following documentation is enclosed with WebOTP:

Developer's Manual
This manual is provided in electronic format with file WebOTPDeveloperManual.pdf.

SDK documentation
SDK Documentation of WebOTP provided in electronic format with file `dll/WebOTPDLLReference.chm`, `activex/WebOTPActiveXReference.chm` and `dotnet/WebOTPDotNETReference.chm`.

The documentation can be displayed through the CHM viewer present in every WINDOWS 2000, WINDOWS XP and WINDOWS VISTA installation.

1.6 Feedback

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

It is also possible to call the Customer Service's number +39-35697055 or to send a fax to number +39-35697092.

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

2 Introduction

2.1 What is WebOTP?

WebOTP is a secure authentication device via PC and related USB connection.

WebOTP is especially suited for the authentication of web service users by means of an Internet browser, but it can also be used within all those environments requiring any form of authentication.

The product is made up of an USB device for authentication to assign to users and by an SDK software which provides the necessary services for authentication.

2.2 Use case

The typical WebOTP use case is represented by a user needing to authenticate for using an Internet remote service via web.

The authentication process starts with the presentation by the Internet browser of a page requiring the insertion of the WebOTP device by the user. The user inserts the device in a USB port and the device automatically sends a data code to the server for authentication. If the authentication is successful the browser will display the user a reserved access page.

This occurs *without any interaction by the user*, who is only requested to insert the device in a USB port upon authentication. The whole *acknowledgement, transmission and authentication process takes place silently*.

2.3 Features

The main features of WebOTP that distinguish it from a traditional OTP are as follows:

Usability – The interaction requested to the user is extremely reduced. The user is only requested to insert the device in a USB port upon authentication.

Identification – Besides being authenticated the user is also identified. Therefore it is not necessary to request the user an identifier like a *username* before authentication.

Secure authentication – Authentication is based on 128 information bits and on the AES 256 bits algorithm**.** It is therefore almost impossible to carry out a brute force attack.

2.4 Operation

WebOTP provides several authentication methods for better suiting any kind of need. It is possible to use such protocols as *One Time Password*, *Challenge Response* or SMS-based protocols.

2.4.1 One Time Password

The One Time Password protocol, in short *WebOTP*, is a protocol that allows authentication via a monodirectional communication from the user towards the system.

The protocol is available both in *event-based* and in *time-based* version according to the hardware equipment at disposal.

The main feature of this protocol is the extreme compatibility with all operating systems and Internet browsers without the need of installing software components on the system and of being granted special execution permits.

2.4.2 Challenge Response

The Challenge Response protocol, in short *WebCHR*, is a protocol that allows authentication via a bidirectional communication between the user and the system.

Thanks to the type of communication, this protocol is intrinsically more secure than the corresponding WebOTP, but it requires stricter software requisites.

Using special software with the system allows a multiple query of the device from remote without requiring any interaction with the user.

2.4.3 SMS

The SMS-based protocol, in short *WebSMS*, has been developed for all those cases in which it is not possible to use the WebOTP hardware device, because of damage, loss or simply because a USB connection is not available for connecting it.

Via this protocol the server sends the user an SMS containing a code consisting of letters and/or numbers which the user will be able to use for authenticating.

2.5 Multi-factor authentication

WebOTP is a device that guarantees a state-of-the-art authentication but which cannot guarantee security against such events as theft of the device by itself.

In such cases it is advisable to complement the authentication provided by WebOTP with a second authentication factor. The most common case is using a numerical PIN with a limited number of attempts before being disabled.

The SDK of WebOTP just provides the necessary services for WebOTP authentication and leaves maximum freedom in choosing further authentication factors.

3 Getting Started

This chapter illustrates the use of the WebOTP device

3.1 SDK installation

The WebOTP product is provided with an SDK including the documentation, the libraries for interfacing with the server applications and a test application.

For installing the SDK WebOTP on the Windows platforms run the `WebOTPSDK.exe` installation program.

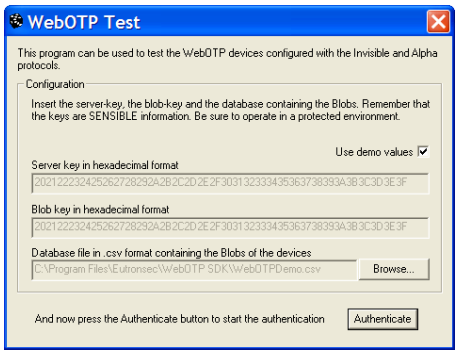
On completion of the installation the SDK content can be accessed by the application menus via the entry *Eutronsec/WebOTP SDK*.

3.2 First operation

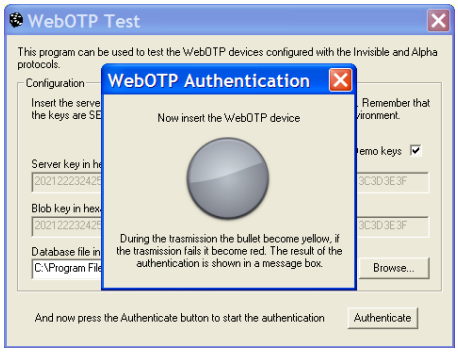
For appreciating the user-friendliness of WebOTP it is possible to use the demo device supplied with the SDK by means of the WebOTPTest application.

The WebOTPTest application enables checking the authentication process of the hardware devices. In order to be able to run, it requires the cryptographic keys and the initialization information of the device contained in a database file with `.csv` extension.

With the demonstrative device it is enough to select the use of the *demo values* via the special checkbox. The correct cryptographic keys and the `WebOTPDemo.csv` database file resident in the same directory of the application will be entered automatically.



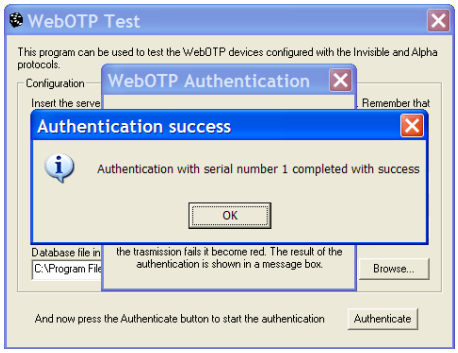
At this point it is possible to start the authentication process by pressing the *Authenticate* button.



To authenticate oneself just insert the WebOTP device in a USB port. On first insertion it will be necessary to wait some seconds in order to allow the operating system enough time for identifying the device.

In order for the authentication to be successful it is necessary to keep the *focus* on the authentication window.

During the communication process between the device and the system the keyboard led will blink and the round icon will become yellow.



At the end of the authentication process the program displays the device serial number.

4 Requirements

This chapter describes the minimum requisites in terms of software and hardware that are compulsory for WebOTP operations.

The WebOTP product features different requirements on the client side and on the server side, according to the authentication protocol in use.

4.1 Client

The hardware device can be used on every system that is provided with a USB 1.1 or USB 2.0 connection.

The WebOTP authentication protocol can be used on any operating system that supports USB connected keyboards, in particular WINDOWS, LINUX and MAC OS X. Also all web browsers that are compatible with JAVASCRIPT, among which INTERNET EXPLORER, FIREFOX, OPERA, MOZILLA and SAFARI are supported. For being operative the system requires no software to install and no special permits are required.

The WebCHR authentication protocol can be used only with WINDOWS operating systems and with INTERNET EXPLORER browser. For being operative the installation of an ACTIVEX component on the system is required. Upon installation of the component the administrator's permits are required. During operation no special permits are required.

The WebSMS authentication protocol only requires the user to own a telephone which can receive an SMS.

4.1.1 Incompatibility

The WebOTP authentication protocols are not compatible with some special configurations:

- WINDOWS 95 and WINDOWS NT – It is advisable to upgrade the operating system.
- MAC OS X 10.2 – It is advisable to upgrade the system to the Mac OS X 10.3 version or higher.
- Browser OPERA 9 for MAC OS X – It is advisable to use the SAFARI or FIREFOX browsers.

The WebCHR authentication protocol is not compatible with some special configurations:

- WINDOWS 95 and WINDOWS NT – It is advisable to upgrade the operating system.

The WebSMS authentication protocol has no chances of incompatibility, as it does not interact with any software.

4.2 Server

For operating at a server level the SDK provides support for WINDOWS 2000, WINDOWS XP, WINDOWS 2003 and WINDOWS VISTA platforms, both with 32 and with 64 bit versions.

On request the SDK can be adapted to other software and hardware architectures as it is based on an easily portable C language.

For the authentication process, it will be necessary to store per each user on the server a data record called *Blob* of about 128 alphanumeric characters. Each Blob is associated with the serial number of the device made up of a 4 byte integer number. The information reserved in the Blob are always encrypted, therefore it will not be necessary to adopt other security measures when storing the database.

The *time-based* devices require an exact time source on the server for carrying out the authentication. If the clock system is used, the clock must be always synchronized with a public *time server* by using an NTP client. Further details are provided in the chapter *Integration.*

4.2.1 Incompatibility

There are no known incompatibilities.

5 Authentication

This chapter describes the authentication type that is carried out from WebOTP.

The WebOTP and WebCHR protocol authentication is based on the use of a symmetric cryptography algorithm and of a secret shared among the hardware devices and the authentication server. The WebSMS protocol authentication is based on the exchange of information via user's telephone instead.

5.1 Security

In the WebOTP and WebCHR protocols the authentication is based on the use of the AES 256 bit symmetric cryptography algorithm, as opposed to the common hash algorithm used by traditional OTP devices.

Using a symmetric cryptography algorithm is possible for WebOTP thanks to the USB connection which does not prescribe any length limits in the authentication code. Symmetric cryptography requires operating with an authentication code of at least 128 bits, many more bits than a traditional OTP display with few figures can display.

The advantages of using a symmetric cryptography algorithm during OTP authentication are manifold:

Identification – Besides authentication the user, it is also possible to identify him/her. It is therefore possible for the user to avoid identification upon authentication. By comparison a traditional OTP always demands to know who the user is in advance.

Security – The authentication code contains 128 bit of security information. By comparison, the traditional display-equipped OTP's use maximum 40 bits, often fewer[1].

Speed – The authentication check is easy and quick. The authentication server will be submitted to a much reduced workload[2]. By comparison a traditional OTP with hash function requires a process by attempts for *guessing* the time values or the events used by the device.

Resistance to *Brute Force Attack* – Given the high security level of the authentication, it is not necessary to block access to the users after a certain number of unsuccessful attempts. By comparison, a traditional OTP is obliged to use block techniques for preventing brute force attacks.

Resistance to *DoS Attack* – The authentication server is especially proof against *Denial Of Service* – type attacks, which are programmed for requiring a high number of fictitious authentications in the attempt of blocking user access. The authentication check is very quick also in case of failure. By comparison, with a traditional OTP based on hash function a failed authentication case is always the worst possible event in terms of performance time.

Effective error management – In case of authentication failure, it is possible to know the exact reason of the failure. In particular it is possible to distinguish between errors due to faulty devices and errors due to attack attempts. For instance, it is possible to continue to use a *time-based* device with a flat battery as if it was an *event-based* device. By comparison, a traditional OTP with hash function has always got only one type of authentication failure and no further information is possible to infer from the error.

In the WebSMS protocol, the authentication is based on the user sending a random code up to 64 bits via the user's private telephone number. By using the received code, the user will be able to perform the authentication.

5.2 Keys

The identification and authentication process is based on sharing a secret between the WebOTP device and the authentication server.

For that reason every device contains two secret 256 bit keys: the *server-key* and the *device-key*.

- The *server-key* is a key shared by all devices that will be used with a certain authentication server; it enables the server to identify the user that owns the device. Only the server which knows the *server-key* will be able to identify the user.
- The *device-key* is a key which differs for each device; it enables the server to authenticate the user. Only the device that contains the correct *device-key* will be able to authenticate itself.

[1] An 8-figure numerical display represents about 27 information bits.

[2] With a *Pentium Core Duo 2* workstation it is possible to perform about 200000 authentication checks per second.

6 Device

This chapter describes the WebOTP hardware devices.

WebOTP is a hardware device with a USB 2.0 connection which uses the HUMAN INTERFACE DEVICE (HID) standard for communicating with the PC. The device contains a microprocessor, which is able to carry out authentication operations by using the AES256 cryptography standard which is the state-of-the-art in terms of security.

6.1 Event-based and time-based

The devices are provided in two different versions. With battery for the *time-based* version and without battery for the *event-based* version.

The event-based version, which does not avail itself of a battery, is extremely small-sized and boasts unlimited duration in time. The hermetic sealing of this device makes it also *waterproof.*

The *time-based* version avails itself of a battery used by the device for storing the time information for various years. The battery duration does not depend on the number of times the device is used; indeed, while in operation the device is fed by the USB connection.

When battery loading is completed the *time-based* device continues to work like an *event-based* no-battery device.

6.2 Serial number

Each device is associated with a serial number printed on a label that is applied to the device. The serial number is present both in numerical format and in bar code format in order to simplify the management of large quantities.

The code bar is encoded according to the 6-figure no-check digit *Interleaved 2/5* standard.

6.3 Authentication information

When supplied the WebOTP devices are already initialized; they come with a database containing the associated authentication information.

The database is in COMMA SEPARATED VALUES (CSV) format and contains the serial number of the device that is associated to an alphanumeric string of about one hundred characters, containing the authentication information. Such a string, termed *Blob*, will have to be imported as it is in the user database and provided to the SDK functions that require it.

An example of such a file is the following:

```
161315,M/4Om9A+g3W6XPPm0ihdmx2CxaGelIaOyCxJhIK7SL…
161316,2lP4Xa96D16WuykZHjIZ2Swtzee69UFMlD1nBtPpzP…
161317,L+9LIPyCGpBsBzYSKQuKFPkbOPK5ETw3QAK1i4OV+E…
161318,0rnbtTxAFjPgP1w9b/MCddOrSFznxsfGczCPDjK+wD…
161319,0W2QCGoSCfU542XT4LfpUrcVX6TBAuWWOAPiDFAdYG…
```

The confidential information contained in this file, among which the generated *device-keys*, are encrypted by using a key named *blob-key*[3]. The file can then be transmitted also via insecure channels.

6.4 Protocols

The devices are configured at the moment of production for supporting one or more authentication protocols. Such a configuration will not be changed and will be preserved for the whole life of the device.

The devices using the WebOTP protocol communicate with the system by simulating pressing a special sequence of keys on the system. The device is recognized as a USB keyboard and treated as such. Two different information encodings are possible: *Invisible and Alpha.*

The devices using the WebCHR protocol communicate with the system as proprietary HID devices.

[3] If for security reasons it is not acceptable that the *server-key* or the *device-key* are recognized in the production phase it is possible to provide non-initialized devices and the software necessary to the setting of such reserved information.

Possible configurations are:

Configuration	Description
WebOTP Invisible	The most widely used and best usable configuration. The authentication occurs in a totally transparent way for the user.
WebOTP Alpha	The configuration that best enables integrating with already existing password-based systems. The authentication occurs by transmitting an alphabetical string which can be easily recognized by already existing applications.
WebCHR	This configuration enables using a bi-directional authentication for device multiple queries.
WebOTP Invisible + WebCHR	This configuration supports both protocols in the same device. With this configuration the device exports two USB interfaces. With WINDOWS platforms the operating system requires more time for recognizing the device, both during insertion and during removal. For guaranteeing maximum usability it is advisable to use this configuration only when strictly necessary.

The details of the communication process are illustrated below in case it is necessary to integrate the WebOTP authentication in an application. For operating with a web server just use the JAVASCRIPT example provided in the chapter *Integration.*

6.4.1 Invisible WebOTP Communication

The Invisible communication has been designed for hiding the authentication code transmission to the user and making the process absolutely silent.

The transmission simulates pressing two special key combinations in order to represent the transmission of one bit at a time. The two codes have been chosen in such a way as to not to have undesired effects in case they are intercepted by generic applications and to guarantee maximum compatibility with all platforms.

The key combinations in use are as follows:

Keys	Meaning
SHIFT+PAUSE	Representation of bit 0
NUM LOCK	Representation of bit 1

The actual codes of the keys received are different according to the execution context. Usually you only need to record the pressing sequence of the single keys PAUSE and NUM LOCK which the standard keyboard codes correspond to.

Key	Code	Bit
PAUSE	19	0
NUM LOCK	144	1

Within a generic Windows application, if the codes are received via Windows WM_KEYDOWN messages, such codes will be:

Key	Character Code	Bit
PAUSE	VK_PAUSE	0
NUM LOCK	VK_NUMLOCK	1

Within a generic JAVASCRIPT, if the codes are received via the `onkeydown` event, such codes might differ according to the operating system or the browser in use. However it is possible to have a univocal decoding by using the following conversion:

Key	Code	Bit
PAUSE	19, 126	0
NUM LOCK	0, 12, 144	1

As the NUM LOCK key is used, during transmission the relevant keyboard led will blink.

The authentication data are embedded into a bit sequence which provides a special header and a special footer:

	Size	Description
Header	4 bit	Sequence for recognizing the transmission start and for identifying the device type in connection and the protocol in use. For the WebOTP protocol the sequence is always 0001.
Data	20*8 bit	Data to transmit for authentication. The single bytes are sent starting from the least meaningful bit.
Footer	2 bit	2 parity bits used for always making the number of key pressing sequence even.

In the recording process it is advisable not to confine oneselves to recording the character sequence though, but to carry out some special operations for minimizing any error possibility due to interferences from the user's side during keyboard management.

The algorithm must behave like this:

- On reception of the fourth bit, if the string does not correspond to 0001 delete the first bit and keep only the last 3 bits. The sequence always begins with '0001'.
- If in 200 milliseconds no character is received, consider the sequence as interrupted and restart from the reception process again.
- After receiving 166 binary codes the transmission is to be deemed concluded.

When reception is completed the received string can be interpreted; if necessary, check the header/footer fields before using it for authentication.

An example of implementation in JAVASCRIPT is to be found in chapter *Integration*.

6.4.2 WebOTP Alpha communication

Alpha communication has been designed for use in those already-existing contexts where an alphabetical password input is required.

The transmission is based on the simulation of sixteen special letter codes, where each code is assigned the representation of a bit group.

The characters in use are:

Character	Meaning
B	0000
D	0001
E	0010
F	0011
G	0100
H	0101
I	0110
J	0111
K	1000
L	1001
N	1010
R	1011
T	1100
U	1101
V	1110
Q, A	1111

The letters chosen correspond to keys which have the same position both on the QWERTY, QWERTZ and AZERTY keyboards, with the only exception of the ‘Q’ letter, which on the AZERTY keyboards will be interpreted as ‘A’. This choice is fundamental for preventing the generated string from depending on the operating system keyboard international configuration.

Besides, the characters can be received both in capital letters and in small letters, according to the CAPSLOCK and SHIFT key status.

Therefore when interpreting the keys no difference shall be made between receiving capital letters and small letters, and the 'Q' and 'A' letters shall be treated likewise.

The authentication data are embedded into a bit sequence which provides a special header:

	Size	Description
Header	4 bit	Sequence for recognizing the transmission start and for identifying the device type in connection and the protocol in use. For the WebOTP protocol the sequence is always 0001.
Data	20*8 bit	Data to transmit for authentication. The single bytes are sent starting from the least meaningful bit.

The reception algorithm must behave like this:

- Ignore every character until receiving letter 'D'. The sequence always starts with letter 'D'.
- If in 200 milliseconds no character is received, consider the sequence as interrupted and restart from the reception process again.
- Ignore the differences between capital letters and small letters.
- Ignore the difference between letters 'Q' and 'A'.
- After receiving 41 codes the transmission is to be deemed concluded.

When reception is completed the received string can be interpreted and used for authentication.

6.4.3 WebCHR Communication

During WebCHR the system queries the device using the HID *SetFeature* and *GetFeature* commands that are available in all operating systems, also on applications that are run without special permits.

The WebCHR features can therefore be implemented within any client program by using the *WebCHR SDK* library, provided on request.

When using the web, communication occurs via ActiveX instead, supplied with the standard distribution.

6.5 Installation

The devices are recognized by the system as USB HUMAN INTERFACE DEVICE (HID) devices without requiring any installation of proprietary drivers in the operating system.

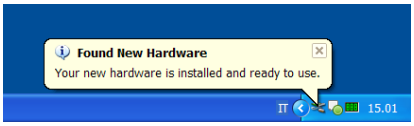
According to the operating system in use, on first insertion the user may be requested to carry out some actions. In particular, Mac OS X operating system requires a minimum configuration process on first insertion of each HID device.

After the possible configuration process, on subsequent insertions the device is always recognized by all operating systems immediately.

6.5.1 Windows

With WINDOWS ME, WINDOWS 2000, WINDOWS XP, WINDOWS 2003 and WINDOWS VISTA operating system the device requires an automatic installation process on first insertion. Such a process may take some seconds and neither requires any interaction with the user nor any administrative permits.

The user just needs to confine himself/herself to waiting for the *"Found New Hardware/Your new hardware is installed and ready to use"* message to appear.



When installation is completed the authentication process starts automatically.

The installation process is requested on first insertion in every USB port.

6.5.2 Windows 98

Windows 98 operating system requires installing a driver that is resident in the original CD-ROM of the operating system on first insertion.

The user must proceed like this:

1. The user inserts the device

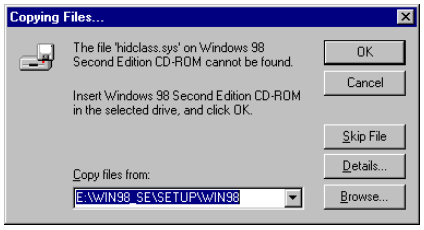
2. The operating system starts the installation wizard. The user presses "*Next*".

3. The user presses "*Next*".

4. The user presses "*Next*".

5. The user presses "*Next*".

6. The user inserts the Windows 98 original CD-ROM, keeps selecting the installation default directory and presses "*OK*".



7. The user presses “Next”.

8. The user presses “*Finish*” and the configuration process is completed.

When installation is completed the authentication process starts automatically.

This process is requested only on first insertion on any USB port.

6.5.3 Linux

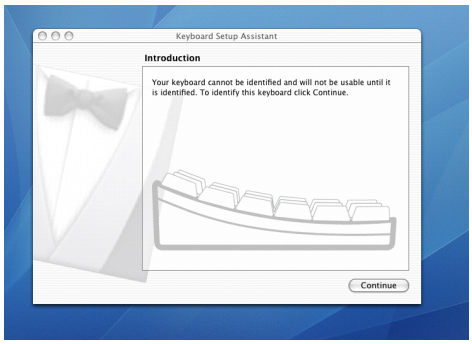
Linux operating system recognizes the device immediately. There is no difference between the first insertion and the following ones.

6.5.4 Mac OS X

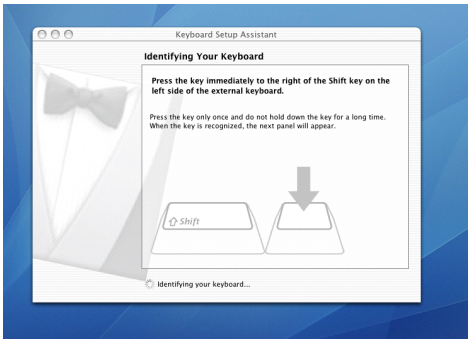
Mac OS X operating system requires an interaction with the user for configuring the device on first insertion. As soon as the device is inserted the system displays the user a request for manual configuration.

The user needs to proceed like this:

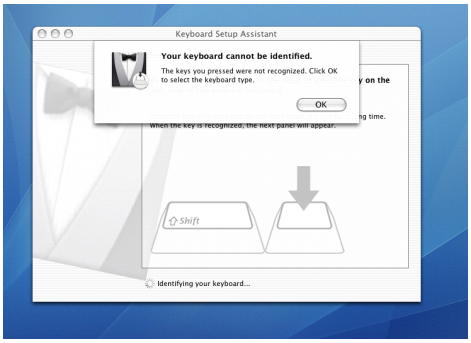
1. The user inserts the device.
2. The system displays a request for manual configuration that requires pressing a *“Continue”* screen button with the mouse.

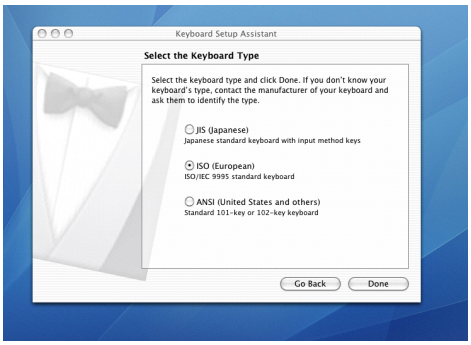


3. The user presses “*Continue*”.
4. The system requests the user to press the key next to the “*Shift*” key.



5. The user just waits for the device to generate a sequence of recognition keys. This sequence is generated only once after five seconds form insertion. It is therefore essential for the use to open this panel within 5 seconds.
6. The system detects the recognition sequence but does not identify the device, thus it signals the user such a condition by requesting a manual selection of the keyboard type.





7. The user selects any type of keyboard and the configuration process is completed.
8. The user removes and reinserts the device for generating the authentication sequence.

This process is requested only on first insertion in any USB port.

7 SDK

This chapter describes the WebOTP SDK.

The WebOTP product is distributed with an SDK which puts at the developer's disposal the necessary services for using the WebOTP features.

The WebOTP features can be subdivided into the following categories:

Initialization – Initialization features**.**

WebOTP – Features for WebOTP authentication.

WebCHR – Features for WebCHR authentication**.**

WebSMS – Features for WebSMS authentication**.**

Utilities - Generic utility features**.**

Users – Customization features.

Configuration – Configuration features.

The SDK is distributed in different versions for use via DLL, ACTIVEX and .NET interface. The following description refers to the DLL version for use via C language; the other versions offer the same features but they are suited to the various environments.

7.1 Initialization

The initialization functions enable preparing and configuring the SDK by setting up a data context where all necessary information is stored for maintaining the SDK internal status.

The use of each function of the SDK will be based on this context, the format of which is hidden to the user, who shall just confine to provide it at every function call up that requests it. If the C language is used, it is possible to use the *webotp_context* type for declaring the context. With any other language just supply a memory area of 4096 bytes.

The initialization functions must be called up only upon the application start and completion. In particular it is essential that the moment such functions are used there are no other threads which might make simultaneous calls to other functions of the SDK using the same data context.

```
int webotp_init(struct webotp_context* context, unsigned char* server_key,
unsigned char* blob_key);
```

The initialization of the context occurs via *webotp_init()* function which needs to be provided with the *server-key* too, on which all authentication operations will be based and the *blob-key* used for the Blob cryptography operations.

```
void webotp_done(struct webotp_context* context);
```

On completion of each operation with the SDK, the context must be destroyed, calling up the *webotp_done()* function and providing the context to destroy.

7.2 Authentication

The authentication functions enable obtaining authentication through an information exchange between the server and the users.

According to the authentication type in use it may be the case of a mono-directional communication (WebOTP) or bidirectional communication (WebCHR and WebSMS). In case of mono-directional communication, it will be necessary to call up a single function which will check the information coming from the user. In case of bidirectional communication it will be necessary first to call up a function for preparing the information which the server has to send to the user. In a second phase it will be necessary to call up a second function that will check the answer received from the user.

All such functions are *thread-safe*.

The information that is necessary for authentication is stored into a data packet called Blob which is associated to every user. The Blob must be stored into an external database and associated with the corresponding serial number of the device provided to the user.

The Blob must be updated after each successful use by the authentication functions. When a function ends in error, the Blob has not been modified and therefore it is not necessary to update it[4].

7.2.1 WebOTP

The WebOTP authentication takes place through a single communication from the user to the server. The server receives a 20 byte-long data string which is checked via function *webotp_authenticate()*.

```
int webotp_authenticate(struct webotp_context* context, webotp_time_t now, const
unsigned char* packet_ptr, char* blob_ptr, unsigned blob_size);
```

The *webotp_autenticate()* function authenticates the user by using the received data packet and by using the time information present in the Blob.

```
int webotp_identify(struct webotp_context* context, const unsigned char*
packet_ptr, unsigned* id);
```

The *webotp_identify()* function is used for extracting the user's identity directly for the information received by the user before performing the authentication.

Using this function is needless in case you already know which user is being dealt with and the Blob is already at disposal for authentication. If the user is unknown instead, it is possible to obtain the serial number of the device and from this datum identify the Blob to use for authentication.

7.2.2 WebCHR

The WebCHR authentication occurs via a double communication between server and user. The server generates an 8 bytes random data packet which is then sent to the user and is used for calling up the *webchr_setup()*function. The user answers with a 20 byte-long data packet which the server checks through the *webchr_authenticate()* function.

```
int webchr_setup(struct webotp_context* context, webotp_time_t now, const
unsigned char* random_ptr, char* blob_ptr, unsigned blob_size);
```

The *webchr_setup()* function is used for storing the random data packet sent to the user in the Blob, which will then be used for authentication. The data packet can be generated through the function *webotp_random()* or through a random data external source.

```
int webchr_authenticate(struct webotp_context* context, webotp_time_t now, const
unsigned char* packet_ptr, char* blob_ptr, unsigned blob_size);
```

The *webchr_authenticate()* function is used for checking the answer received from the user by means of the information contained in the Blob.

7.2.3 WebSMS

The WebSMS authentication, similarly to the WebCHR, occurs via a double communication between server and user. The server generates a random data packet, encodes it through the *websms_encode()* function in order to send it to the user under alphanumeric string format and uses the *websms_setup()* function for setting up the authentication. The user answers with the same received string, that the server decodes through the *websms_decode()* functions and check with *websms_authenticate()*.

```
int websms_encode(const char* charset, unsigned char* random_ptr, unsigned
random_bit, char* string_ptr, unsigned string_size);
```

The *websms_encode()* function encodes a data field into a character string. It is possible to specify the number of bits to use and the character set for encoding. This function deletes any unused bits from the *random_ptr* parameter, so that the next call up to *websms_setup()* takes only the actually used bits into account. The random data packet can be generated through the function *webotp_random()* or through a random data external source.

```
int websms_setup(struct webotp_context* context, webotp_time_t now, const
unsigned char* random_ptr, char* blob_ptr, unsigned blob_size);
```

The *websms_setup()* function is used for storing the random data packet sent to the user in the Blob, which will then be used for authentication.

```
int websms_decode(const char* charset, const char* string_ptr, unsigned
string_size, unsigned char* random_ptr);
```

---

[4] The only exception is the `websms_authenticate()` function which requires to limit the number of access attempts.

The *websms_decode()* function carries out the reverse operation of *websms_encode()* by decoding the character string received by the user in a data field.

```
int websms_authenticate(struct webotp_context* context, webotp_time_t now, const unsigned char* packet_ptr, char* blob_ptr, unsigned blob_size);
```

The *websms_authenticate()* function is used for checking the answer received from the user by means of the information contained in the Blob. In this function the Blob must be saved also in case of INVALID_PACKET error. This is necessary for limiting the number of failed attempts. This limitation is used for preventing brute force attacks which might take place if an authentication code of few bits is used. In any case it is advisable to always use 24 bits for the authentication code at least.

7.3 Utility

The utility functions supply some auxiliary services that can come in useful when using the authentication functions. When during the authentication phases the current time or the generation of random data are requested, it is possible to use such functions or other sources, if available.

All such functions are *thread-safe*.

```
webotp_time_t webotp_time(void);
```

The *webotp_time()* function supplies the number of seconds elapsed from midnight (00:00:00) of 1st January 1970 in Coordinated Universal Time (UTC) [5]. The SDK uses all information of the system data for obtaining such a value. It is therefore essential that the system time and time zone information are correct.

In case a more precise source for date and time is available, it is possible to use it instead of the *webotp_time()* function.

```
int webotp_random(unsigned char* random_ptr, unsigned random_size);
```

The *webotp_random()* function is a source of random numbers to be used each time random information on the authentication process is requested. The SDK uses the random number generator within Windows.

In case more advanced system are available for the generation of random numbers it is possible to use them instead of the *webotp_random()* function.

7.4 Users

The users' management functions enable modifying the authentication information of a user partially. Typically this is not necessary when the SDK is operated normally.

All such functions are *thread-safe*.

```
int webotp_user_reset(struct webotp_context* context, unsigned property, char* blob_ptr, unsigned blob_size);
```

The *webotp_user_reset()* function can be used for setting some information in the Blob. In particular it is possible to reinitialize the management related to the event counter and to the WebOTP protocol time; it is also possible to cancel the temporary information related to the WebCHR and WebSMS protocols.

It is important to underline the such operations are to be carried out only under extraordinary circumstances and only after contacting the user directly. The possible causes related to similar problems are mainly due to faulty devices and therefore may not be solved at the server level.

7.5 Configuration

The authentication process can be customized by varying some parameters controlling the width of the input windows of the event and time counter. Typically this is not necessary as the default values have been devised for ensuring maximum usability on the user's part without invalidating the authentication security. To this end the following function is used:

```
int webotp_configure(struct webotp_context* context, unsigned configure_property, unsigned value);
```

At each call a parameter can be set by specifying the type of property that is to be set by using one of the following codes:

[5] It is the same format used in the C language by the time() function.

RANGE_COUNTER	Maximum leap ahead of the event counter from the latest authentication. The value is expressed in units. The default value is 1500.
RANGE_TIME_PAST	Maximum fixed component of time error for devices with a past time value (device clock is slower than normal). The value is expressed in seconds. The default value is 30.
RANGE_TIME_FUTURE	Maximum fixed component of time error for devices with a future time value (device clock is faster than normal). The value is expressed in seconds. The default value is 30.
FACTOR_TIME_PAST	Maximum variable component of time error for devices with a past time value (device clock is slower than normal). The actual error value is given by the time elapsed from the latest authentication multiplied by the specified factor. The value is expressed in ppm (parts per million). The default value is 40.
FACTOR_TIME_FUTURE	Maximum variable component of time error for devices with a future time value (device clock is faster than normal). The actual error value is given by the time elapsed from the latest authentication multiplied by the specified factor. The value is expressed in ppm (parts per million). The default value is 10.
DELTA_TIME_CHR	Maximum time elapsed between the generation of the CHR authentication code and its use. Beyond this time it will not be possible to authenticate oneself by using the generated code. The value is expressed in seconds. The default value is 60.
DELTA_TIME_SMS	Maximum time elapsed between the generation of the SMS authentication code and its use. Beyond this time it will not be possible to authenticate oneself by using the code sent via SMS. The value is expressed in seconds. The default value is 180.
DELTA_COUNTER_SMS	Maximum number of attempts allowed for an SMS authentication. Beyond this number of unsuccessful authentications it will not be possible to authenticate oneself by using the code sent via SMS. The value is expressed in units. The default value is 10.

7.5.1 Input window for the counter field

The input window of the counter field with WebOTP and WebCHR protocols for *event-based* and *time-based* devices is given by the RANGE_COUNTER parameter.

An error E of the counter field is accepted only in a window equalling:

$$0 < E < \texttt{RANGE\_COUNTER}$$

The width of the window does not represent a particularly meaningful element from the security point of view as the counter is used only for avoiding reusing the authentications and not for guaranteeing authenticity.

7.5.2 Input window for the time field

The input window of the time field with WebOTP and WebCHR protocols for *time-based* devices is given by the RANGE_TIME_PAST, FACTOR_TIME_PAST, TIME_FUTURE and FACTOR_TIME_FUTURE parameters.

An error E of the time field is accepted only in a window equalling:

$$\texttt{-(RANGE\_TIME\_PAST + FACTOR\_TIME\_PAST * T)} < E < \texttt{(RANGE\_TIME\_FUTURE + FACTOR\_TIME\_FUTURE * T)}$$

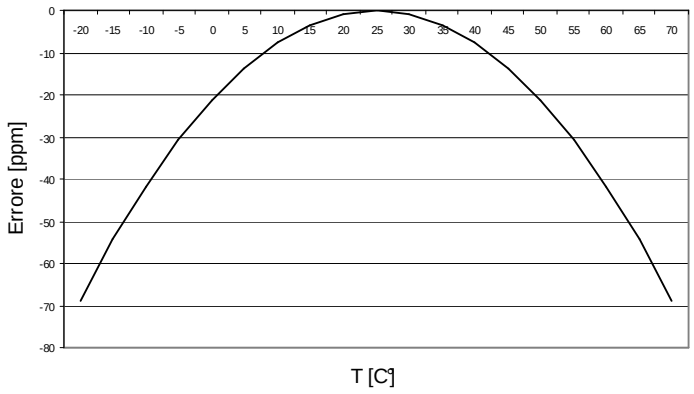
Where T is the time elapsed from the latest authentication. The width of the input window is therefore dependent on the time elapsed from the latest authentication. The less time from the latest authentication, the narrower the window.

The width of the window is mainly due to the physical characteristics of the devices. The default values have been chosen for guaranteeing operation even in the worst possible conditions of work and storage.

The devices are calibrated in the production phase with an error lower than 2 ppm (parts per million). The storage temperature can increase the error though, as regards the period of exposure to such a temperature.

The progress of the error according to the temperature is expressed in the following table.

T [C°]	E [ppm]
-10	-42
-5	-31
0	-21
5	-14
10	-8
15	-3
20	-1
25	0
30	-1
35	-3
40	-8
45	-14
50	-21
55	-31
60	-42



In proportion with the temperature variation the induced error always tends to slow down the clock. It is therefore advisable to use an asymmetric input window.

For instance, storing the device for 10 hours at the temperature of 60° C brings about an error accumulation of 1.5 seconds.

$$E = 10 * 3600 \text{ [seconds]} * 42 / 1000000 = 1.5 \text{ [seconds]}$$

7.6 Error management

All SDK functions return an error code which indicates whether the operations has been completed successfully or not. The generic error codes that are common to all functions are:

Error code	**Description**
SUCCESS	The operation has been completed successfully.
INVALID_ARGUMENT	One of the past arguments is invalid.
INVALID_STATE	It is not possible to carry out the operation requested in the current state.

During the authentication process further errors are possible suggesting the type of error incurred, the cause and the possible remedy.

Typically after one of such errors it is advisable to always allow the user to attempt authentication again in order to prevent *Denial Of Service* – type attacks aiming at blocking the rightful users. Given the security characteristics of the WebOTP and WebCHR protocol, brute force attacks are not effective and there is no reason for blocking the rightful user.

Error code	**Description**
SUCCESS	The operation has been completed successfully. **The user is authenticated.**
TIME_OVERFLOW	The time information used in the authentication process has undergone an unexpected lead backward. In that case it is important to check that the time source used on the server is working properly.

	The user is NOT authenticated.
INVALID_BLOB	The datum supplied as Blob is invalid. The data in the Blob are both encrypted and validated. Every variation not carried out by the SDK is identified and makes the Blob unserviceable. In that case it is necessary to reset a valid Blob in the database. It is also possible that an incorrect Blob-key is being used. **The user is NOT authenticated.**
INVALID_PACKET	The received packet is invalid. Probably it is a transmission error or a fraudulent access attempt without using a real device. In this case the user needs to be required to retry and carry out the authentication. It is also possible that an incorrect Server-key is being used. **The user is NOT authenticated.**
INVALID_COUNTER_REUSE	The received packet had already been used previously. Probably it is a fraudulent access attempt through the reuse of authentication codes that were previously used and stolen someway. In this case the user needs to be required to retry and carry out the authentication. **The user is NOT authenticated.**
INVALID_COUNTER_OVERRUN	The received packet contains an event counter field with an extremely high leap ahead It is probably a damaged device or a device that has undergone a brute force attack attempt. In this case the damaged device must be replaces, or the counter field must be reset by using the dedicated function. The user cannot authenticate until the problem is solved. **The user is NOT authenticated.**
INVALID_TIME_REUSE	The received packet had already been used previously. It is probably a fraudulent access attempt. In this case the user needs to be required to retry and carry out the authentication. It is also possible that the clock used for authentication is not correct. **The user is NOT authenticated.**
INVALID_TIME_OVERRUN	The received packet contains a time field with an extremely high leap ahead. It is probably a damaged device. In this case the damaged device needs to the replaced. The user cannot authenticate until the problem is solved. It is also possible that the clock used for authentication is not correct. **The user is NOT authenticated.**
INVALID_TIME_NOBATTERY	The device results authenticated except for the time comparison which is not available due to the flat battery. In this case the device must be replaces or considered to all effect as an *event-based* device and no more as a *time-based* device. **The user can be considered authenticated or not at discretion.**
INVALID_TIME_LATE	The authentication has failed because too long time has elapsed between the first and the second authentication step. It is probably a user error. In this case the user needs to be required to retry and carry out the authentication.

	The user is NOT authenticated.
INVALID_COUNTER_LATE	The authentication has failed because too many unsuccessful attempts at authentication have been made. It is probably a user error or a brute force attack attempt. In this case the user needs to be required to retry and carry out the authentication. **The user is NOT authenticated.**

7.7 LIbraries

7.7.1 DLL

The DLL version of the SDK is present in the `dll/` directory of the package and can be used by every environment which is able to call up a Windows standard DLL.

In the `dll/webotp.h` is present the declaration of the functions exported into C langue. For using the DLL in other languages the same functions must be defined, following the specific format of the language in use. In case the function call convention is to be chosen, keep in mind that is being used the `stdcall` convention which is shared by all Windows DLLs.

The DLL is supplied in several versions according to the hardware architecture that is to be used:

i386 – Version for WINDOWS 2000, WINDOWS XP, WINDOWS SERVER 2003, WINDOWS VISTA for 32 bit processors.

amd64 – Version for WINDOWS XP X64, WINDOWS SERVER 2003 X64 and WINDOWS VISTA X64 for INTEL EM64T and AMD X86-64[6] processors.

For a detailed description please refer to the documentation enclosed in the SDK in the `dll/WebOTPDLLReference.chm` file.

7.7.2 ActiveX

The ACTIVEX version of the SDK is present in the `activex/` directory of the packet and can be used by every environment which is able to call up an ACTIVEX Windows standard.

The ACTIVEX is distributed in the `WebOTPCom.DLL` file and must be registered before use. It is possible to register the component ACTIVEX through the use of the Windows `REGSVR32.EXE` utility with the following syntax:

```
regsvr32 WebOTPCom.DLL
```

For registering the component administrative rights are required.

For a detailed description of the interface of the ACTIVEX please refer to the documentation enclosed in the SDK of the `activex/WebOTPActiveXReference.chm` file.

7.7.3 . NET

The .NET version of the SDK is present in the `dotnet/` directory of the packet and can be used by the.NET environment.

The SDK version for .NET exports the interface `Eutronsec.WebOTP.dll` which redirects.NET calls to the ACTIVEX through the relevant primary interoperability assembly `Eutronsec.WebOTPComLib.dll`. It is therefore necessary to have installed the ACTIVEX component too.

For a detailed description of the .NET interface please refer to the documentation enclosed in the SDK in the `dotnet/WebOTPdotNETReference.chm` file.

[6] They are the 64 bits system sold at this time. Do not confuse them with the INTEL ITANIUM systems.

8 Integration

This chapter describes how to integrate the WebOTP device into an authentication service.

8.1 Integration into a web service

WebOTP has been designed for making the use of a web service as user-friendly as possible. Both in case a new authentication service is to be created and in case such a service is to be integrated into an existing one.

The integration of the WebOTP authentication service into a web service requires the following activities:

- Definition of the new authentication system.
- Integration of the SDK into the server application and the user database.
- Adaptation of the client part for reception and transmission of authentication codes.

8.1.1 Authentication system

According to one's needs and the possible already-existing authentication system different operating scenarios can be foreseen.

The most common scenario is using WebOTP both for identifying the user and for authenticating him/her, adding the request for a *PIN* as second authentication factor in order to protect oneself against any possible theft of the device. In case an identification system via *userid* and *password* is already present it is possible to add WebOTP as second authentication factor.

After completing authentication it is possible to use the *application server* session management for enabling the user to surf the reserved pages.

It is also possible to require a further insertion of the WebOTP device as a confirmation of operations that are regarded as especially important. The request for a manual operation by the user enables a greater control on transactions, preventing any possible malware present in the user's system from carrying out an arbitrary number of fraudulent operations.

8.1.2 Server

At the server level it is necessary to integrate the use of the SDK for checking the authentication information and for modifying the database containing the user's information in order to add the authentication Blob, the device serial number and the possible PIN as second authentication factor.

During the SDK initialization and operation it will be necessary to provide the *server-key* and the *blob-key*. It is particularly important to keep in mind where such keys are stored. It is essential that the applications that use them are carried out within a protected environment. In those cases in which security is an especially critical factor, all authentication operations should be carried out via remote on a special server residing in a highly secure environment.

The information necessary for the SDK for carrying out the authentication is the Blob and the serial number of the device. Such information must be stored into a user database. In case such a database already exists, just add the relevant fields. It is to be noticed that the database does not require special security expedients as the information contained in the Blob are always encrypted via the *blob-key*.

During the authentication process of the *time-based* devices it is necessary to provide the correct time. If the clock in use is not correct the authentication of the *time-base* devices might fail. If the system clock is used use an NTP client for maintaining and synchronizing it with a public *timer server*.

In the WINDOWS environment it is possible to use the W32TIME service for keeping the system clock updated. Please refer to the MICROSOFT documentation for further details on the service configuration, in particular the *Knowledge Base*:

- *How to configure an authoritative time server in Windows 2000*- http://support.microsoft.com/kb/216734
- *How to configure an authoritative time server in Windows XP* - http://support.microsoft.com/kb/314054
- *How to configure an authoritative time server in Windows 2003* - http://support.microsoft.com/kb/816042

Within a UNIX environment it is possible to use a periodically executed NTP client or to install an NTP demon. An OPENSOURCE implementation of both is available on the website http://www.ntp.org

8.1.3 Client

At the client level it is necessary to modify the login page in order to register the information sent to the device. It is also possible to modify other pages of the website, among which the *home page*, for recognizing the insertion of the device and authenticate the user without any intermediate phase.

The recommended system is using a script for recording the key pressure and for sending them to the server.

```
<script language="JavaScript">

var last = new Date();
var data = "";
var data_count = 0;

function handler_down(e)
{
      var code;

      if (!e) {
            e = window.event;
      }
      if (e.keyCode) {
            code = e.keyCode;
      } else {
            code = e.which;
      }

      if (code == 19 || code == 126 || code == 0
            || code == 12 || code == 144
      ) {
            var now = new Date();

            // reset the recording on a pause
            if (data_count != 0 && now.getTime() - last.getTime() > 200) {
                  data = "";
                  data_count = 0;
            }

            last = now;

            // add a new bit
            if (code == 19 || code == 126) {
                  data += "0";
            } else {
                  data += "1";
            }
            data_count += 1;

            // check the start of trasmission
            if (data_count == 4 && data != “0001”) {
                  data = data.substr(1);
                  --data_count;
            }

            // check the end of trasmission
            if (data_count == 166) {
                  …
                  // send the data at the server
                  …
            }

            // stop the event
```

```
            e.cancelBubble = true;
            return false;
      }

      return true;
}

// trap the keydown events
document.onkeydown = handler_down;
</script>
```

9 Technical specifications

9.1 WebOTP event-based

Size: 34x16x8 mm

Interconnection: USB 2.0 FullSpeed

Protocol: HID 1.11

Connector: USB Male Type A

Power supply: fed by USB port

Temperature: -20 - +80°C

Humidity: 20-95% relative humidity

9.2 WebOTP time-based

Size: 55x20x9 mm

Interconnection: USB 2.0 FullSpeed

Protocol: HID 1.11

Connector: USB Male Type A

Power supply: fed by USB port during operation, by battery for time keeping

Battery life: 5 years

Temperature: -10 - +60°C

Humidity: 20-95% relative humidity

9.3 Bar Code

6-figure bar code *Interleaved 2 of 5* without check digit.

10 Appendixes

10.1 Glossary

One Time Password	Generic authentication protocol requiring the use of dynamic non-reusable passwords.
Challenge Response	Generic authentication protocol requiring the use of a bi-directional information exchange between client and server.
Brute Force Attack	Malicious attack with aiming at accessing reserved information in every way possible
DoS, Denial Of Service Attack	Malicious attack aiming at making a public service inoperative, even only temporarily
Time-Based	Type of OTP devices for authentication using time-based information in the authentication process. The time-based devices are objectively more secure that event-based devices.
Event-Based	Type of OTP devices for authentication using incremental-based information in the authentication process.
Thread-safe	A function that can be executed correctly and simultaneously by several threads.
NTP, Network Time Protocol	Internet protocol for synchronizing the computer clocks.
Time server	Server implementing the NTP protocol.