Smarx Cloud Security (WEB API) 6.0 White Paper

Purpose of this Solution: Online authentication for many fields of application (Premium or subscription services, updates and more), Access to content or administration panel only for authorized users
Version: WEB API 6.0
Last Update: 31 July 2017 by Steffen Kaetsch
Target Operating Systems: - Client: any browserclient
- Proxy Server: Windows 32 & 64 bit (Linux and macOS on request)
- Server: Any webserver with PHP, JSP or ASP.NET support
Access to source code needed (of protection application): ☑ Yes ☐ No
Supported Programming Tools/Environment: PHP, JSP or ASP.NET
Applicable for Product: CRYPTO-BOX® SC / XS / Versa

Executive summary

Smarx® Cloud Security (WEB API) ensures only authenticated users who are in the possession of a valid CRYPTO-BOX® hardware unit connected to the computer or in the network gain access to a web portal or any kind of content distributed via the Internet/Intranet.

There are many ways in which incorporating Smarx Cloud Security is smart and profitable for your business. Control that support is provided only for paying customers, allow access to content authorized only for service personnel, and much more.

Smarx Cloud Security offers the following possibilities:

1. Manage access to web content with the CRYPTO-BOX

2. Updating subscriptions and licensing information stored inside the CRYPTO-BOX

Smarx Cloud Security offers support for all popular web servers that support PHP, JSP or ASP.NET. The client site supports virtually any browser (Chrome, Firefox, Edge, Internet Explorer, Safari, Opera) and client platform (PCs, Smartphones, tablets), as soon as the Web API Proxy Server is running on a computer in the local network where the CRYPTO-BOX is attached (currently Windows, support for Linux and macOS on request)

Table of Contents

1. Introduction to WEB API....4
2. How does WEB API work?....5
   2.1. Client side....5
      2.1.1. Server knows PIN scenario....6
      2.1.2. Client knows PIN scenario: User Password (PIN/UPW) submission....6
   2.2. Server side....6
      2.2.1. Server side encryption....6
3. WEB API: General Requirements....7
   3.1. Client side requirements....7
   3.2. Server side requirements....8
4. Software Components....8
   4.1. Client Component....8
      4.1.1. WEB API client setup....8
      4.1.2. Client Diagnostic and Troubleshooting....9
   4.2. Web Server application (PHP or JSP based)....9
5. Developer's Notes....9
   5.1. Start building a Secure Server Solution....9
   5.2. Handshake (sides authentication and verification, establishing secure connection)....10
   5.3. WEB API: Client-Server Communication....10
   5.4. WEB API: Client-Server Communication - Notifications....12
   5.5. Web Security Client-Server communication chart....13
   5.6. Server demo sample description....13
      5.6.1. PHP demo sample module description....14
      5.6.2. Java/JSP demo sample module description....16
      5.6.3. ASP.NET demo sample module description....17
   5.7. Login....18
   5.8. Verification....19
   5.9. DataObjects transaction generation....20
   5.10. DataObjects transaction result proceeding....20
   5.11. Error handling....21
      5.11.1. Error messages generated by client....21
      5.11.2. Special Case: Internet Explorer 10 and 11 on client side....21
      5.11.3. Error messages generated by Web Security server....22
      5.11.4. Errors generated by HTTP server or Tomcat....22
   5.12. WEB API reference documentation....22
   5.13. Client component....22
6. Contact and Support....25
7. Alphabetical Index....26

1. Introduction to WEB API

Smarx® Cloud Security (WEB API) ensures only authenticated users (who are in the possession of a valid CRYPTO-BOX) gain access to a web portal or any kind of content distributed via the Internet/Intranet.

There are many ways in which incorporating Smarx Cloud Security is smart and profitable for your business. Control that support is provided only for paying customers, allow access to content authorized only for service personnel, and much more.

Smarx Cloud Security offers the following possibilities:

- **Manage access to web content with the CRYPTO-BOX**

This scenario is useful if you want to allow authorized users to view content only. When accessing the website it will look for a valid CRYPTO-BOX, depending on the status it will allow or deny access to the site.Furthermore, the content of the client's CRYPTO-BOX can be used by Web applications (PHP-, JSP-, or ASP.NET-based) for making decision on providing access to various services to the client.

- **Updating the CRYPTO-BOX**

Smarx Cloud Security can also be used for secure transfer of information to the client's CRYPTO-BOX, and to update the CRYPTO-BOX. Unlike the Remote Update Management System (RUMS, part of the CRYPTO-BOX Protection Kit) there is no need to (manually) send the activation keys. Furthermore, with Online License Management defined update plans can be easily created via the Smarx Application Framework.

Smarx Cloud Security offers support for all popular web servers that support PHP, JSP or ASP.NET. The client site supports Internet Explorer, Chrome, Safari, Opera (Windows) and Firefox (Windows, Linux).

WEB API version 6.0 is based on Smarx® OS, so it's compatible with other Smarx OS programming interfaces and applications. Smarx OS base functionality is used by WEB API for:

- Establishing secure connection;
- Client authentication and transactions encryption;
- Reliable transaction mechanism, based on Smarx OS Data Objects (DO) API;
- Web Security Client shares Smarx OS SSO (Single Sign On) environment

For more details about Smarx OS, please refer to the see **Smarx Compendium**, chapter 10-13.

2. How does WEB API work?

2.1. Client side

WEB API contains two components: client's component (end-user side) and web server component (server side). The client component is used to access the CRYPTO-BOX connected either to the local USB port of the computer. It is implemented as service which receives requests (encrypted transactions) from HTML/JavaScript pages, generated by the remote server and downloaded by client's browser. Requests are processed by client's component and encrypted result of the transaction is sent back to the server.

Every Smarx® OS formatted CRYPTO-BOX contains client's private RSA key and distributor's public RSA keys, which are used for the handshake - establishing secure connection and client authentication. In case of the CRYPTO-BOX SC all RSA operations are done in the hardware on the client site. In addition to hardware-implemented 128 bit Rijndael encryption, and software (Open SSL) 256 bit AES encryption are used to secure client-server communication.

For more details about Web Security client-server handshake scenario, see chapter 5.2: ***Handshake (sides authentication and verification, establishing secure connection)***.

Smarx Cloud Security (WEB API) version 6.0 supports two basic scenarios:

1) Server knows PIN: this scenario is specific for online distribution and license management, client side authentication is not important here. For this scenario the client side is only supposed to attach the valid token, while the server will open it remotely (by submitting valid PIN) for further access to token's secure data objects.
2) Client knows PIN: this scenario is common for various security applications, it assumes that the client side must provide the token (CRYPTO-BOX SC, XS or Versa) and enter valid PIN in order to "open" the token for further server side access - two factor local authentication.

For software distribution and license management the "Server knows PIN" scenario has to be used, because exposing the PIN (User Password – UPW) of the CRYPTO-BOX to the user is not desired in that case. Therefore, the latest WEB API online demo and sample code is based on the "Server knows PIN" scenario. However, MARX can provide sample code on "Client knows PIN" scenario on request.

2.1.1. Server knows PIN scenario

This scenario is used mostly for web based distribution and license management, where client side authentication is not important. The handshake scenario requires a CRYPTO-BOX XS or Versa with firmware 2.2 and higher, or a CRYPTO-BOX SC.

2.1.2. Client knows PIN scenario: User Password (PIN/UPW) submission

Local two-factor authentication requires CRYPTO-BOX User Password (PIN/UPW) to be submitted by the client. Further server side authentication (involving RSA encryption/decryption) proceeds after the password is submitted.

2.2. Server side

Smarx Cloud Security server side is supported for different environments:

- PHP (can be deployed on any server platform with PHP support: Win/FreeBSD/Linux, etc.);
- Java/JSP (can be deployed on any server platform with Java/JSP support: Win32/FreeBSD/Linux/Sun Solaris, etc.);
- ASP.NET technology (requires Win platform + IIS).

For Java/JSP solution – Tomcat v7.0 or later is needed on server side. Various development tools, like Borland JBuilder can be used for server-side development.

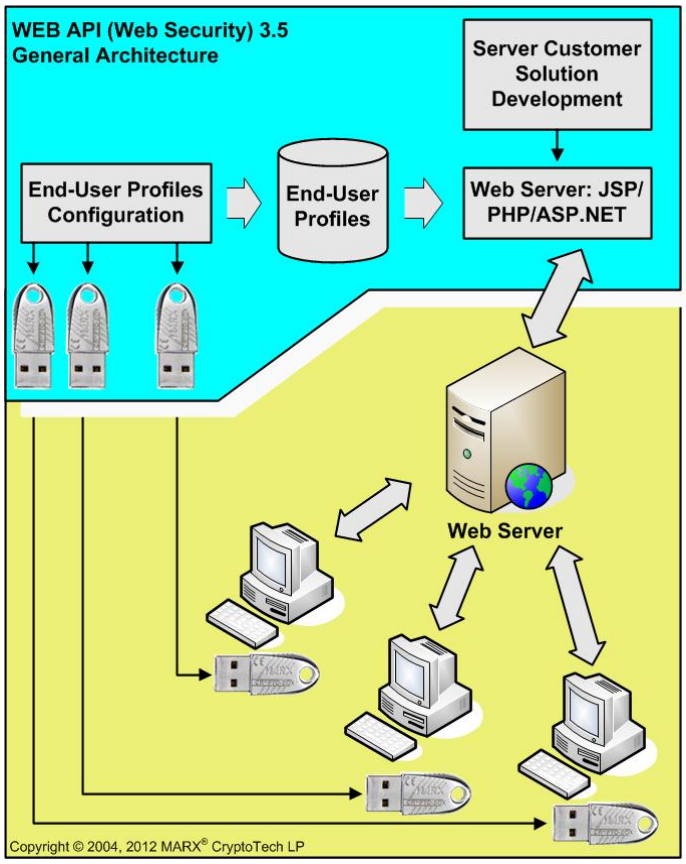
For PHP solution, PHP 5.4 or later must be installed together with the following extensions:

- php_gmp, php_openssl.

For ASP.NET solution Windows is needed with IIS 7.0 or higher.

2.2.1. Server side encryption

Server side encryption required by Web Security includes: 1024 bit RSA, 128 bit AES (hardware-based Rijndael encryption with the CRYPTO-BOX) and 256 bit AES (Open SSL).



WEB API (Smarx Cloud Security) general architecture

3. WEB API: General Requirements

This section describes the requirements for client and server side.

3.1. Client side requirements

- **Hardware**: Smarx® OS formatted CRYPTO-BOX SC, XS or Versa with firmware version 2.2 or higher;
- **Driver**: CRYPTO-BOX driver needs to be installed for CRYPTO-BOX access on the client site;
- **Client component** - installed via WEB API client setup (see chapter 4.1.1)
- Single Sign On environment (for "Client knows PIN" scenario, optional);
- **Operation System/Browser**:
  - Windows (32 and 64bit): Edge or Microsoft Internet Explorer 11; up-to-date versions of Firefox, Chrome, Safari, Opera
  - macOS and Linux support on request

Important:

The Web API client component supports two modes: local mode and network mode (can be selected during installation, see chapter 4.1.1). In local mode, access to the CRYPTO-BOX is available trough locally installed browser (see above), in network mode virtually any browser and client platform (PCs, Smartphones, tablets) in the local network is supported.

3.2. Server side requirements

- **Application**: Customized Web application (PHP, Java/JSP or ASP.NET based)
- **Web Server with PHP support**:
  - OS: Windows, Linux, FreeBSD or any other PHP enabled system
  - Apache, Xitami, IIS, PWS, ...
  - PHP 5.04 or later
  - Encryption libraries (php_gmp, php_openssl)
- **Web Server with JSP support**:
  - OS: Windows, Linux, FreeBSD or any other JSP/Java enabled system
  - Apache, IIS, …
  - Apache Tomcat v7.0 or later
  - Java JDK v1.8.0 or later
- **Web Server with ASP.NET support**:
  - OS: Windows 10/8/7
  - IIS 7.0 or higher

4. Software Components

4.1. Client Component

Client Component provides access to the CRYPTO-BOX from the browser. Requests generated on the server side are encrypted and MIME-encoded, embedded into HTML/JavaScript page. They will be sent to the client component, where they are decrypted and executed. Transaction results are encrypted, MIME-encoded and sent back to the server.

4.1.1. WEB API client setup

The easiest way to install the client component is to use the WEB API client setup. It installs the CRYPTO-BOX drivers for Windows 32/64 and the WEB API client component.

The latest version of the WEB API client setup can be found on our website:
www.marx.com → Support → Downloads → Network Utilities.
Refer to the readme file included to the installation package for further installation details.

4.1.2. Client Diagnostic and Troubleshooting

You can use MARX Analyzer diagnostics as well as Web API online diagnostic to check/troubleshoot your client configuration.

MARX Analyzer can be downloaded from our website:
www.marx.com → Support → Downloads → Driver and Diagnostic Tools.
For WEB API online diagnostic please visit:
www.marx.com/webapi-check.

4.2. Web Server application (PHP or JSP based)

The server side application, loads the Distributor Private RSA key and Client Public RSA key from binary profile files, obtained from MARX. Client Private RSA key and Distributor Public RSA key are pre-programmed to the CRYPTO-BOX by MARX at the production stage. The usage of these RSA key-pairs (unique for each MARX customer) allows client side CRYPTO-BOX authentication (proving that this CRYPTO-BOX belongs to the customer) and to organize encryption/decryption of commands. All sensitive information have to be stored securely on server-side in the private area.

After successful authentication, a session-unique 128-bit AES Rijndael key is generated randomly and programmed into the CRYPTO-BOX unit of the client. All further commands (transactions requests and results) are encrypted/decrypted with transaction-unique 256-bit AES Rijndael key. This key in turn is transferred, being encrypted with hardware 128-bit AES Rijndael key, stored in the CYPTO-BOX of the client and known to the server site.

5. Developer's Notes

5.1. Start building a Secure Server Solution

This section describes steps of your own Web Security solution development. What do you need for this? Server and client requirements are described in the section 3.

The client side components and WEB API client setup setup were already described in section 4.1.

The end-user management (user profiles) and server side development will be described in further sections of this document.

5.2. Handshake (sides authentication and verification, establishing secure connection)

For the "Server knows PIN" scenario, the Server side needs to know the Distributor/Client's RSA key pairs and Fixed Key (128-bit Rijndael key), as well as the PIN (User Password UPW) and/or Administrative Password (APW) of the CRYPTO-BOX. Those values are provided by MARX in binary files.

If you did not receive the binary files for your CRYPTO-BOX, please contact MARX to obtain them.

When the end-user (Client) accesses the login page, the Server generates a random SessionID. This SessionID and the PIN (User Passwort UPW and/or the Administrative Password APW of the CRYPTO-BOX) will be encrypted with the Rijdael Fixed Key of the CRYPTO-BOX and digitally signed by the distributor's RSA private key and sent to the Client.

Only having a CRYPTO-BOX unit with valid Fixed Key attached, the PIN (UPW and/or APW) can be decrypted and used for further CRYPTO-BOX access, including RSA key pairs verification.

If the CRYPTO-BOX SC model is used, all RSA operations are hardware implemented which increases security on the client side.

After that, a random Session Key value (128-bit Rijndael key) is generated on the client side and submitted to the CRYPTO-BOX. The CRYPTO-BOX S/N, (optionally model, memory size, etc.) is encrypted with this session key and the session key itself is encrypted with Distributor's RSA public key and digitally signed with Client's RSA private key. This package is sent back to the server. Only trusted server side can decrypt the Session Key and obtain the CRYPTO-BOX information.

Finally, both sides are considered as verified and trusted and ready for secure communication. For further secure transactions the Session Key (known to both sides) is used for encryption and SessionID is sent unencrypted.

5.3. WEB API: Client-Server Communication

After a secure connection is established, the server side can submit encrypted requests to be processed by the client side: get/update information in the internal memory of the CRYPTO-BOX . The results of the request are encrypted and returned to the server.

WEB API is based on Smarx OS Data Objects API. Every request contains the following data fields:

- App ID (partition number #999 is used for online demo);
- Memory zone (RAM1/RAM2/RAM3);
- Administrator Password (reserved for future customer specific implementations);

- Memory Offset
- Data Object Type (Expiration Date, Counter, Memory Object, etc...)
- Data Object Size (for Memory Object)
- Data Object Action
- Extra Parameters (reserved)

Each transaction request is encrypted with randomly generated Transaction Key (256 bit AES). The Transaction Key in turn is encrypted with the Session Key (obtained during client-server handshake and valid for all transactions of current session).

On the client side transaction request is decrypted using hardware Session Key and then software Transaction Key. After this, the request is executed. The result of the transaction and/or the error code is encrypted with a newly generated Transaction Key, which is encrypted in turn with the Session Key and the result of transaction is sent back to the server.

The transaction result is decrypted on the server side and used for further server side processing.

Executing Data Object command (by the client) implies forming request and analyzing result (returned by the client) by the server side code. Forming request consists of calling the following sequence of methods on PHP/JSP/ASP.NET page:

1. call **Command** constructor;
2. submit DO transaction by calling **submitDOCommand**(Command command)
3. commit DO transaction by calling **commitDOCommand**()

There are two constructors used to initialize DO command:

```
public Command(short sPartitionId,      // application partition Id
                int iDOId,              // Data Object Id
                int iDOType,            // Data Object type
                int iDOOperation,       // Data Object operation
                int iRAMBank,   // RAM bank (RAM1 = 1, RAM2 = 2, RAM3 = 3)
                int iOffset,            // offset in application partition
                int iReserved); // reserved
```

and:

```
public Command(short sPartitionId,      // application partition Id
                  int iDOId,    // Data Object Id
                  int iDOType,  // Data Object type
                  int iDOOperation,// Data Object operation
                  int iRAMBank, // RAM bank (RAM1 = 1, RAM2 = 2, RAM3 = 3)
                  int iOffset,  // offset in application partition
                  byte []bDOData,       // Data Object data
                  int iDOSize,  // Data Object data size
                  int iReserved);       // reserved
```

First is used to form most of the DO commands, the second to form DO commands performing reading/writing to/from the internal CRYPTO-BOX memory.

For detailed information on Data Objects API, please refer to the **Smarx Compendium**, chapter 14 and the **DO API Reference**.

To analyze DO command results returned by the client the:

proceedExecuteResults(String executeResult, ArrayWraper result)

is called on the next PHP/JSP/ASP.NET page.

Where:

executeResult – MIME string returned by the client;

result – wrapped byte array returned by the client (if data is returned).

If the method returns non-zero, error took place during DO command execution. Error code can be further analyzed by calling **getErrorMessage(int errCode)**.

5.4. WEB API: Client-Server Communication - Notifications

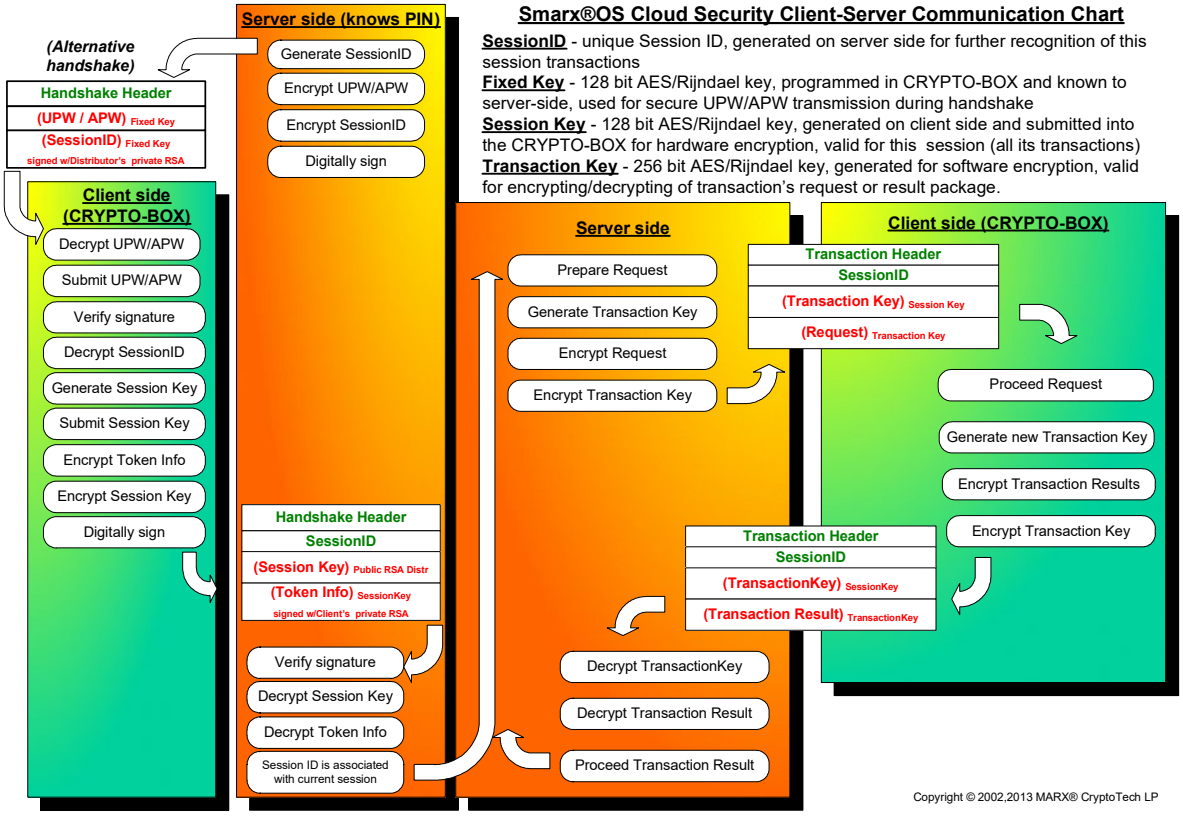
To receive notifications on CRYPTO-BOX attachment/detachment the following technique can be used.

Example:

```
<script language='javascript'>
WEBSEC.registerNotification(handlerR);
function handlerR(notification) {
    if (notification.isError()){
        WEBSEC.registerNotification(null);

        // handle connection problem
        ...
    }
    else{
        if (!loggedin)
            searchBoxes();
        else{
            if (notification.isRemoved()){
                searchBoxes();
            }
            else if (notification.isAttached()){
                searchBoxes();
            }
        }
    }
}
</script>
```

5.5. Web Security Client-Server communication chart



Smarx Cloud Security (WEB API) Client-Server communication chart

Files, containing Distributor RSA Private key and Client RSA Public Key for the Demo CRYPTO-BOX (part of the CRYPTO-BOX Evaluation Kit) are included to the sample code and can be used for testing (sample.d.prv.bin and sample.c.pub.bin). Key files for customer specific CRYPTO-BOX units will be provided when buying WEB API.

5.6. Server demo sample description

The data exchange between server and client is performed the following way: The server generates pages, with client component calls, embedded into JavaScript. The encrypted transaction string, transformed into MIME64 format is transmitted as parameter for component functions. The result of client execution is also an encrypted string, which is sent to the server as form field value, using POST method.

Below is a description of the modules contained in the different samples available for the server site (PHP, JSP, ASP.NET).

5.6.1. PHP demo sample module description

For a live test of the PHP demo sample, please visit **cloudsecurity.marx.com/php/**

PHP classes:

WebSec.php	- WebSec main class.
AuthStr.php	- authorization string class. (is used to handle authorization)
BCMath.php	- provides set of math functions, which are used by RSA.php under Win32 platform
GMP.php	- provides set of math functions, which are used by RSA.php under FreeBSD platform
Command.php	- Data Object Command class. (describes Data Object operation to be performed)
Constants.php	- constants class
CryptParam.php	- cryptographic parameters loader (loads RSA keys)
DOCommandResults.php	- Data Object command result class. (proceeds results of Data Object operation returned by client)
DOCreateTransactStr.php	- exception class
JSTag.php	- class is used to generate client side JavaScript code
Rijndael.php	- CRYPTO-BOX® compatible Rijndael implementation
RSA.php	- CRYPTO-BOX® compatible RSA implementation
StringHeader.php	- common header used to wrap/unwrap authorization and transaction string
Token.php	- contains information on current Token
Tools.php	- used to perform data conversions
TransactStr.php	- class is used to securely wrap/unwrap DO operation data
WrongPwdLength.php	- exception class

Sample:

index.php	- the startup file of the online demo ("Server knows PIN" scenario). It helps to choose CRYPTO-BOX® (MARX® hardware), prepares a handshake procedure. By clicking the <Login> button client initates the handshake.

verifyResult.php - this file validates the handshake results. In case of successful validation it shows basic CRYPTO-BOX info: model, location, SerialNumber.

It prepares commands for client info extraction and transaction, i.e.:

1. Get client info
2. Increment visit counter
3. Get the counter value
4. Get last visit date
5. Update the last visit date

transaction.php, *transactionEdit.php*, *transactionUpdate.php* - displaying Client Info, read from client's CRYPTO-BOX:

1. Full information about the client,
2. Number of visits
3. Last visit date and time (with time zone)

binding.php, *bindingStart.php* - Binding support. Demonstrate how to make bind, activate, unbind commands.

expiration.php, *expirationStart.php* - Demonstrates how to work with Expiration License.

how_to_use.html - brief usage info (displayed on Main Page)

securityviolation.php and *servererror.php* - handle possible security violations of the online demo: different errors found in process of the response string decryption.

The sample demonstrates the following steps of client authentication, transaction:

hardware selection, login → CRYPTO-BOX® info → client info, binding info, expiration license info

Furthermore, the sample demonstrates the following features:

- ajax web development techniques
- network or local hardware selection, including support for FEST (allows testing of core

features of the CRYPTO-BOX system without the need to a physical CRYPTO-BOX – ideal for immediate testing)
- license management via WEB API (including binding & activation)

The brief functionality and design of each page is described starting with chapter 5.7. See WEB API reference documentation (contained in the /doc subfolder of the WEB API package) for a detailed description of the PHP classes.

5.6.2. Java/JSP demo sample module description

For a live test of the Java demo sample, please visit **cloudsecurity.marx.com/java/**

Java packages:

com.marx.jsmrxweb	- basic WebSec.class
com.marx.jsmrxweb.crypt	- encryption algorithms
com.marx.jsmrxweb.exception	- exceptions
com.marx.jsmrxweb.Script	- HTML/JavaScript tags generation
com.marx.jsmrxweb.transaction	- transaction generation/processing
com.marx.jsmrxweb.util	- service classes
org.bouncycastle.crypto	- encryption algorithms**
org.bouncycastle.util.encoders	- service classes**

** - some modified and original encryption services, developed by **The Legion Of The Bouncy Castle (www.bouncycastle.org)** are used.

Sample:

index.jsp	- the startup file of the online demo ("Server knows PIN" scenario). It helps to choose CRYPTO-BOX® (MARX® hardware), prepares a handshake procedure. By clicking the <Login> button client initates the handshake.
verifyResult.jsp	- this file validates the handshake results. In case of successful validation it shows basic CRYPTO-BOX® info: Model, Location, SerialNumber. It prepares commands for client info extraction and

transaction, i.e.:

1. Get client info
2. Increment visit counter
3. Get the counter value
4. Get last visit date
5. Update the last visit date

transaction.jsp, *transactionEdit.jsp,* *transactionUpdate.jsp*	- displaying Client Info, read from client's CRYPTO-BOX: 1. Full information about the client, 2. Number of visits 3. Last visit date and time (with time zone)
binding.jsp, *bindingStart.jsp*	- Binding support. Demonstrate how to make bind, activate, unbind commands.
expiration.jsp, *expirationStart.jsp*	- Demonstrates how to work with Expiration License.
how_to_use.html	- brief usage info (displayed on Main Page)
securityviolation.jsp and *servererror.jsp*	- handle possible security violations of the online demo: different errors found in process of the response string decryption.

The sample demonstrates the following steps of client authentication, transaction:

index.jsp (hardware selection, login) → verifyresult.jsp (CRYPTO-BOX® info) → [transaction.jsp (client info) | binding.jsp (binding info) | expiration.jsp (expiration license info)]

If a transaction error occurs, it is processed on *servererror.jsp* page.

The brief functionality and design of each page is described starting with chapter 5.7. See WEB API reference documentation (contained in the /doc subfolder of the WEB API package) for detailed description of Java packages/classes.

5.6.3. ASP.NET demo sample module description

For a live test of the ASP.NET demo sample, please visit **cloudsecurity.marx.com/asp.net/**

\WebSec.sln	- Smarx Cloud Security ASP.NET solution
\Web.config	- Web configuration

1. WebSec project

\WebSec*.*	- Smarx Cloud Security sample files
\WebSec\bin*.*	- Dll files
\WebSec\classes\ *.*	- Smarx Cloud Security ASP.NET classes
\WebSec\data\ *.*	- RSA demo key files
\WebSec\netsetup*.*	- ASP.NET client diagnostic files
\WebSec\pics*.*	- sample images files

2. WebSec.Cryptography project

\WebSec.Cryptography*.*	- Smarx Cloud Security Cryptography files
\WebSec.Cryptography\bin*.*	- Dll files
\WebSec.Cryptography\RSA*.*	- Dll files

The sample demonstrates the following steps of client authentication, transaction:

Index.aspx → VerifyResult.aspx → [Transaction.aspx | Binding.aspx | Expiration.aspx]

The brief functionality and design of each page is described starting with chapter 5.7. See WEB API reference documentation (contained in the /doc subfolder of the WEB API package) for detailed description of the ASP.NET solution.

5.7. Login

The login (index) page is the start page of the sample. It is generated by the server-side, using `com.marx.jsmrxweb.WebSec` class functions: `getHTMLHeader()`, `getMozillaCode()`, `getHandshakeCode()`, `getHandshakeSubmitButton()`, etc. (see WEB API reference documentation package for details). For JSP, ASP.NET there is WEB API (MARX) tag library implementing above functionality in compact form.

On this page client may need to submit password (if it is required by Web Sec Scenario).

```
<form name="form1" method="post" action="verifyresult.jsp">
<input type="password" name="password" value="demo">
```

There should be input fields on this form holding client command execution results to be sent back to the server.

```
<input type="hidden" name="WEBSECHandshakeResult" value="-">
```

The "handshake" code getting information about the CRYPTO-BOX, attached to the client computer and doing CRYPTO-BOX verification is also embedded. The results of the handshake process are sent to the server.

```
<input type="button" name="websecsubmit" value="Login"
onClick='doHandshake(this.form);'>
<script language='javascript'>
function doHandshake(frm) {
<script language="javascript">
function doHandshake(frm) {
 if(WEBSEC) {
   WEBSEC.Handshake(<mimed handshake code generated on server with
   getHandshakeCode()>);
   WEBSEC.set('WEBSECHandshakeResult');
   // post to server
   WEBSEC.post('verifyresult.jsp');
 }
}
</script>
```

After the "Login" button is clicked, the handshake code is executed and the handshake result `(form.WEBSECHandshakeResult)` is sent to the server. The server receives and processes this data. If the CRYPTO-BOX was successfully verified the verification page is generated.

For the demo sample, each demo-formatted CRYPTO-BOX matches (sample files contain demo values of Distributor RSA Private Key and Client RSA Public Key, used for verification). However in your own solution – it's a good idea to limit number of matching units with unique criteria, for instance the serial number of the CRYPTO-BOX. In this case verification can only be continued if the end-user has a CRYPTO-BOX with the proper Serial Number (this information has to be stored on the server side).

5.8. Verification

Verification page contains information about the CRYPTO-BOX, found on client-side and result of its verification. Handshake results are preceded, using `com.marx.jsmrxweb.WebSec`

class function: `proceedHandshakeResults()` (see WEB API reference documentation for details).

For JSP, ASP.NET there is WEB API (MARX) tag library implementing above functionality in compact form.

```
String handshakeResults = request.getParameter("WEBSECHandshakeResult");
int r = websec.proceedHandshakeResults(handshakeResults);
if(r != 0){
// handle error
...
```

If the CRYPTO-BOX was verified successfully, server and client sides are ready to execute memory transactions. However client can break connection by clicking the "Logout" button, and return to the start page.

If none of the attached CRYPTO-BOX units was verified (encryption key pairs don't match), a page with an error message is generated, informing that client was not verified. If client was not verified, all further work with the CRYPTO-BOX is terminated.

5.9. DataObjects transaction generation

Scripts necessary for CRYPTO-BOX DataObjects transactions are only generated after successful CRYPTO-BOX verification. `com.marx.jsmrxweb.transaction.Command` class and the following functions of `com.marx.jsmrxweb.WebSec` class are used on server side for page generation:

`submitDOCommand, commitDOTransact, getTransactionSubmitCommandScript`, etc.

For JSP, ASP.NET there is WEB API (MARX) tag library implementing above functionality in compact form.

Resulting page would look like this:

```
<form name="form1" method="post" action="transaction.jsp">
<input type="hidden" name="WEBSECExecuteResult" value="-">
<input type="button" name="websecsubmit" value="Client info"
onClick='doTransaction(this.form);'>
<script language='javascript'>
function doTransaction(frm) {
  if(WEBSEC) {
    WEBSEC.Execute(<mimed transaction code generated on server with
commitDOTransact>);
    WEBSEC.set('WEBSECExecuteResult');
    WEBSEC.post('transaction.jsp');
  }
```



```
</script>
```

The result of transaction execution is sent back to the server with the `form.WEBSECExecuteResult` field.

5.10. DataObjects transaction result proceeding

The transaction page is generated as the result of transaction execution on client side. `getHTMLHeader(), getMozillaCode()` and `proceedExecuteResults` functions of `com.marx.jsmrxweb.WebSec` class are used on server side for page generation – see WEB API reference documentation (contained in the /doc subfolder of the WEB API package) for details.

5.11. Error handling

Java and JSP mechanisms are used to control errors on server side. Here is a list of the most probable errors and how they are handled:

5.11.1. Error messages generated by client

The result of transaction contains an error code, if an error occurs on the client side. The return code can be used with the getErrorMessage() method to get an error description. It is a customers' decision how to handle erroneous situation – either the last page should be generated again or any other steps should be taken (i.e. to alert the system administrator, to lock the account, to redirect the client to the NetSetup or driver's update page, ..). Some errors should be processed locally on client-side, displaying alert messages (for example – check if compatible CRYPTO-BOX is attached, etc…)

5.11.2. Special Case: Internet Explorer 10 and 11 on client side

Starting with Internet Explorer 10, Microsoft has changed the logic of AJAX post processing. A discussion about this issue can be found here:
http://stackoverflow.com/questions/11235613/jquery-ajax-post-not-working-ie10

As suggested in the above discussion the problem can be fixed with adding the following string to the code:

```
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE9" >
```

With the Internet Explorer released in November 2013, Microsoft changed the way the browser is detected. When being asked about its type, the Internet Explorer 11 does not identify itself as MSIE type browser (Microsoft standard definition for previous Internet Explorer versions). Instead it defines itself as a Gecko-type browser.

This issue was addressed with the following series of changes in our WEB API sample code

(PHP and JSP):

- index.php and \classes\JSTag.php:

```
if(navigator.appName.substring(0,9) \"Microsoft\")
```

was replaced with:

```
if(navigator.userAgent.indexOf('Trident') > -1)
```

- \netsetup\netsetup-???.php:

```
if((strpos(strtolower($_SERVER['HTTP_USER_AGENT']), "msie")=false
```

was replaced with:

```
strpos(strtolower($_SERVER['HTTP_USER_AGENT']), "trident")===false
```

- \netsetup\BrowserUtils.php:

```
else if (false !== strpos($userAgent, 'Trident'))
```

was changed to:

else if (false !== strpos($userAgent, 'MSIE '))

If you are integrating WEB API code using iframe, the IE9 compatibility mode can be enforced from within iframe window by including this string:

```
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE9" >
```

to the parent page containing iframe.

More details can be found in this discussion:
http://stackoverflow.com/questions/3717932/will-an-iframe-render-in-quirks-mode

5.11.3. Error messages generated by Web Security server

In case of erroneous situation on the server side, the server will log all the errors to stderr using System.err.println(). If there are some problems with the server part of this solution, look through the log files of Tomcat.

5.11.4. Errors generated by HTTP server or Tomcat

There is always the possibility that errors occur on JSP server level. If problems persist even after a server restart, it is recommended to check the server configuration files (server.xml for Tomcat) and to monitor the log files of Tomcat and the HTTP server.

5.12. WEB API reference documentation

Please refer to the JavaDoc-generated WEB API reference documentation package, which reflects the most up-to-date Java classes (see /doc subfolder in the WEB API package).

5.13. Client component

Client component performs client side operations and is called from JavaScript code on JSP pages.

It supports the following methods:

Handshake(Command);

Method accomplishes initial handshake.

Input parameter	- MIME string sent by the server.
Output	- resulting MIME string, sent by the client to the server.
Compatibility	- used for both scenarios ("client knows PIN"; "server knows PIN")

Execute(Command);

Method accomplishes Data Object transaction.

Input parameter	- MIME string sent by the server.
Output	- resulting MIME string, sent by the client to the server.
Compatibility	- used for both scenarios ("client knows PIN"; "server knows PIN")

isSSOInstalled();

Method determines if SSO is installed.

Compatibility	- used only for scenario "client knows PIN"

SSOLogin();

Method realizes client login by means of SSO.

Compatibility	- used only for scenario "client knows PIN"

WebLogin(UPW);

Method realizes client login by standard means.

Compatibility	- used only for scenario "client knows PIN"

GetCTFirmware();

Method determines the firmware version of the attached CRYPTO-BOX

Compatibility	- used only for scenario "server knows PIN" (Firmware v2.2 and higher is needed)

GetVersion();

Method returns client component's version.

Compatibility - used for both scenarios ("client knows PIN"; "server knows PIN")

Logout();

Method realizes client component's logout.

Compatibility - used for both scenarios ("client knows PIN"; "server knows PIN")

SearchBoxes(Command, UPW);

Method searches local computer or network for available CRYPTO-BOX units specified by

Command (in human-readable format, i.e.
"search=network,connect=broadcast,broadcastport=8766").
Result lists what was found, UPW is optional (only for "client knows PIN" scenario).

SetBox(Command, UPW);

Same as SearchBoxes except that only one CRYPTO-BOX (first-best) will be chosen for further work.

6. Contact and Support

USA
MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518
U.S.A.
www.marx.com

Sales: sales@marx.com
Support: support@marx.com
Phone: (+1) 770-904-0369
Fax: (+1) 678-730-1804
E-Mail: contact@marx.com

Germany
MARX Software Security GmbH
Vohburger Str. 68
D-85104 Wackerstein
Germany
www.marx.com

Sales: sales-de@marx.com
Support: support-de@marx.com
Phone: +49 (0) 8403 9295-0
Fax: +49 (0) 8403 1500
E-Mail: contact-de@marx.com

Italy
CS Computers S.r.l.
Via Indipendenza, 4-12
I-47033 Cattolica (FO)
Italia
www.cscomputers.it

Contact: Giorgio del Bene
Phone: +39 0 541/963-801
Fax: +39 0 541/953-847
E-Mail: cscomp@cscomputers.it

Poland
BCSG Sp. z o.o.
Św. Michała 43
61-119 Poznań
Poland
www.bcsg.pl

Contact: Grzegorz Bigos
Phone: +48 (61) 2785830
E-Mail: biuro@bcsg.pl

7. Alphabetical Index

A
AES Rijndael encryption 11
Apache 10
ASP.NET 6

C
Chrome 9
Client component 24
Client knows PIN 7
Contact Information 27
Counter 13

D
Data Objects (DO) API 12
Diagnostic 11

E
Error handling 23
Expiration Date 13

F
FEST 17
Firefox 9

H
Handshake 12

I
IE10/11 Issues 23
IE9 compatibility mode 24
IIS 10
Internet Explorer 9

J
Java sample 18
Java Server Pages 6, 18

L
Linux 10
Login 20

M
macOS 9
MARX Analyzer 11

O
Opera 9

P
PHP 6
PHP sample 16
PIN 7

R
Remote Update 6

S
Safari 9
Server knows PIN 7, 12
SessionID 12
Support 27

U
User Password (UPW) 7

V
Verification 21

W
WEB API Client Component 10
WEB API reference 24
Windows 9

0-15Mar013ks(WEB_API_Reference).odt