MARX

2024
EDITION

CRYPTO-BOX®

Smarx® Compendium

SECURING
THE DIGITAL
WORLD™

Windows Linux Mac android iOS

www.marx.com

0-27Nov12sa(SmarxMan_Cover).psd

We highly appreciate and value your comments and suggestions!

Suggestions for improvements will be honored with:

- Free Business Support for 6 months
- Enrollment in our BETA-Tester program

Software security is a growing challenge and requires constant improving - be part of the process!

Please send sugestions and error report to:

- PPK, software/hardware in general and documentation (including this Compendium):

  support@marx.com

- WEB and online ordering system related:

  webmaster@marx.com

Table of Contents

1. What is this Compendium About?

1.1. Introduction

This Compendium illustrates licensing model choices for software and content protection with the CRYPTO-BOX. You will be able to choose between different implementation scenarios, from the simplest automatic application protection (requiring no programming) to highly sophisticated, customized implementation with Smarx OS API calls.

Learn how multiple applications can be protected with one CRYPTO-BOX, on a local PC or in a network environment. Or take advantage of the latest encryption and anti-debugging features, which allow you to store licensing information securely in the CRYPTO-BOX.

1.2. What is New?

The following are some of the latest features included in the Smarx OS Professional Protection Kit:

AutoCrypt Wizard
The new AutoCrypt Wizard makes protection of your Windows .EXE or DLL files even more easy! The Wizard guides you through each step of protection, licensing and CRYPTO-BOX configuration for quick results. You may also export the project for usage with the Smarx Application Framework providing you with some advanced licensing options, including project and user management, or with our command line tools. See chapter *4.4* for details.

Smarx API: Simplified Implementation
The new Smarx API is a high level API for software developers looking for manual integration of customer specific protection and licensing logic into their products. Smarx API exposes simple and user friendly programming interface, significantly reducing implementation efforts compared to existing CRYPTO-BOX interfaces. See chapter *11* for details.

Revised LabVIEW Package
A special package for LabVIEW users includes cross-platform support (Win and macOS) and a prototype application with AutoCrypt-like licensing logic for quick and easy implementation to existing LabVIEW projects.

Support for New Compilers and Programming Languages
Support for all common programming languages under Windows, Linux and macOS.
See chapter *10.12* for a detailed table of supported environments. We are updating our Development Kit regularly to ensure support for the latest compiler versions and new programming languages. The latest table of supported environments can be found in the Control Center in the section "Implementation with API" → “Libraries/Samples” (available after installation of the Smarx OS Protection Kit, see chapter *3.1*).

Binding Software Licenses to a Particular Computer
A new feature is the ability to bind the protected software to a specific computer. For example, this can be beneficial if the use of the protected software outside the company is

not desired. The Protection Kit contains libraries and sample code for C++ and C# demonstrating the binding technology. More details can be found in chapter *14.2*.

1.3. What to Find Where in this Compendium

Task	Remarks	Page
How To Start	How to start with the protection of your digital assets.	18
Automatic Protection	Instant protection of Windows applications in 10 minutes without requiring the source code (see also separate AutoCrypt Application Notes)	23
Implementation with API into Source Code	Implementation of the CRYPTO-BOX into the source code using API calls for Windows, Linux and OS X	24
Document Protection	Secure distribution of documents	31
Remote Update	Update licensing options after your software is in the field	43
Network License Management	Limit the number of running application instances (seats) in a network with only one CRYPTO-BOX	42
Online License Management	Automated distributions of updates, identify all users visiting your web portal	43

Important hints and warnings are marked using these symbols:

ℹ Hint ⚠ Attention ✖ Warning

1.4. Professional Software Protection Secures Revenue

The CRYPTO-BOX – Your Guide in Insecure Markets

Software and data protection ensure that every end user has paid for your product. As you are aware, written agreements and licenses do not always prevent unauthorized use, and enforcing them can create an antagonistic relationship with your end users.

Many violators of copyright and license agreements don't think they are doing anything wrong or illegal, yet most would never steal merchandise from a store.

We've heard all the justifications, including:

- It's only a private backup!
- I will purchase the software later.
- I got it as a free download from the Internet.
- I don't ask for money for my copies, so it's okay.
- Everyone does it.
- This application is way too expensive!
- It's from my company computer. They paid for it.

- I’ve already paid a lot for a single license. Why should I pay more for use in my network?
- It’s from a large company - they don’t care.
- It’s from a company in a foreign country. They won’t know, and they cannot enforce it.
- I have already paid too much for software that didn't work as expected.
- I'm promoting the software for the authors. I help them spread their product and make it well known.
- There was no copyright notice on it.

There is a Short Answer to All these "Excuses":

In addition to preventing piracy, the ability to control license updates and software distribution enables you to generate additional sales, improve customer relations and obtain valuable marketing data by 100% customer registration.

The CRYPTO-BOX Remote Update capability allows you to react quickly and flexibly to end user’s needs, and keep shipping and administration costs to a minimum. Programs can be activated through the Internet, and products can be sold as modular components, authorized after payment is received.

The CRYPTO-BOX System Offers the Following Advantages:

- All end users must pay for the software.
- Every end user is registered - an effective marketing tool!
- Support services are available only to paying end users.
- Additional software modules or license updates are easily activated via remote update.
- Updates are available only to authorized end users, ensuring a constant revenue stream.
- Network usage is controlled.
- Additional licenses bring more revenue.
- Applications, documents and video/audio files can be protected.
- Fast and easy protection with AutoCrypt.
- AutoCrypt requires no source code or programming efforts.
- Support for almost all compilers under Windows, Linux and macOS.
- iOS and Android based devices are supported.
- Professional anti-debug protection, compression and encryption prevent reverse engineering of customer's application.
- Checksum and hash functions are available.
- Provides reliable protection against viruses and tampering recognition.

1.5. The CRYPTO-BOX®Hardware

The CRYPTO-BOX is the "heart" of the software and data protection solutions offered by MARX. The on-board integrated AES encryption algorithm provides reliable security for software and data protection scenarios. The short and robust metal case is only 1.28” (32,5 mm) long. The CRYPTO-BOX USB-C variant is even smaller: only 0.89” (22,5 mm) long and thus ideal for mobile usage.

1.5.1. CRYPTO-BOX Models

The CRYPTO-BOX for the USB Port is Available in Five Different Versions:

- **CRYPTO-BOX SC** (CBU SC): This model is the latest generation of software protection devices. It contains an EAL 4+ certified SmartCard chip with a Crypto Engine supporting RSA up to 2048 bit in hardware and is probably the fastest token on the market. The CRYPTO-BOX SC has 32 KB of internal memory available, e.g. for storing licensing information.
- **CRYPTO-BOX Memory** (CB/M8): This previously released model combines the functionality of the CRYPTO-BOX SC together with an 8GB USB 2.0 flash drive into an attractive designer metal case. It allows you to ship your protected software, data, drivers and tools on the same USB device – no additional CDROM or download is required.
- **CRYPTO-BOX C** (CBUC): The advantage of the CRYPTO-BOX® for USB-C ports is its compact size and the connector which is reversible and can be inserted both ways. And it has the same capabilities as the USB-A variant. It is available as SC, XS and Versa model.
- **CRYPTO-BOX Versa** (CBU VS): This popular model offers all features necessary for software protection on local PCs. Within networks, you can decide if your software is limited to the local PC or if it can be run in the network (without user limitation). It has an internal memory of 4 KB for storing licensing information.
- **CRYPTO-BOX XS** (CBU XS). This model offers a unique serial number for every device. The integrated License Control System (LCS) allows you to decide how many instances of your software may be run at the same time in the network. The CRYPTO-BOX XS is available with up to 64 KB of internal memory and supports a software implementation of the RSA algorithm.

1.5.2. Technical Features of the CRYPTO-BOX®

- On-board encryption of data using the Rijndael algorithm (Advanced Encryption Standard (AES), official successor to the DES algorithm) with a 128 bit key that never leaves the CRYPTO-BOX, in OFB bit-stream cipher mode (output feedback) or CBC mode (chipher-block chaining, CRYPTO-BOX SC only)
- RSA support (key length: 2048 Bits) in hardware (CRYPTO-BOX SC) or on driver level (CRYPTO-BOX XS).
- Access control (PIN-based).
- Every CRYPTO-BOX (except the Versa model) has its own unique serial number.
- Encrypted EEPROM with 4 KBytes of on-board memory, up to 64 KB available for CRYPTO-BOX XS; CRYPTO-BOX SC with 72KB secure memory (approx. 30KB free, high speed access).
- Reliable communication and CRYPTO-BOX identification (Plug & Play).

For detailed information about the technical data of all CRYPTO-BOX models please refer to Appendix A.

2. Protection and Licensing Options

The Smarx OS Professional Protection Kit (PPK) provides you with a comprehensive set of protection techniques and options based on the CRYPTO-BOX. Before choosing a protection strategy, it is important to understand the main concepts. This chapter presents some typical samples and case scenarios describing the practical application of the Smarx OS Protection Kit.

2.1. Overview

Hardware-based protection requires your protected applications and/or data files to have a corresponding CRYPTO-BOX attached to the computer (or a computer within the network) in order to function normally. The protected software will check for the presence of the CRYPTO-BOX. If the CRYPTO-BOX is not found, the program can switch to a demo mode or even refuse to work (depending on your protection strategy). If the CRYPTO-BOX is attached, the program will communicate with it, performing more detailed verification:

- Serial number
- Developer ID
- Access codes
- Hardware-based encryption
- Dataobjects stored in the internal memory

All these, as well as many other unique CRYPTO-BOX features, can be used to build a reliable protection strategy. Data files can be encrypted using the CRYPTO-BOX internal on-board encryption. This approach guarantees an extremely reliable protection model: Encrypted data files can be viewed only when a corresponding CRYPTO-BOX is attached to the end user's computer. More limitations can be added, e.g. expiration dates: The end user will be able to use the software only until a defined date is reached. MARX provides you with a convenient way to update such expiration dates remotely (see chapter *6.1* for more details).

2.2. Automatic Protection and Implementation With API

When protecting your software, you have two basic choices:

- Automatic protection (see chapter *2.4* for details)
- Implementation into source code through API (see chapter *2.5* for details)

The Smarx OS PPK offers both options.

2.3. Benefits of the Smarx®OS Application Framework

The Smarx OS Application Framework (SxAF) makes protection and licensing of your valuable digital assets (software, documents) very convenient:

- The same approach is used for adding protection and licensing to all kinds of digital assets.
- Projects of all supported types (automatic protection, protection with API, document and media protection) are stored in one database – License Management Database (LM/db).
- One application – CRYPTO-BOX Format - is used to program (format) a CRYPTO-BOX to work with a particular project. See chapter *4.9* for more details about CRYPTO-BOX

formatting.

- A particular software or document needs to be protected only once, but CRYPTO-BOX programming/formatting will be performed many times – one box per every end user (Protect once, deliver many).
- All projects support remote updating of Data Objects with licensing information stored in the CRYPTO-BOX memory via Remote Update Management System (RUMS, see chapter *6.1*).
- Existing project settings can be exported from SxAF into an XML script to be used with command line based applications.
- Key SxAF components (AutoCrypt, Data Objects Manager for Implementation with API, CRYPTO-BOX Format, Remote Update) are available as command line utilities, which can be easily integrated into and controlled by 3rd party applications (see chapter *7* for more information).

You will find a detailed description of the Smarx OS Application Framework in chapter *4*.

2.4. Automatic Protection With AutoCrypt

AutoCrypt automatic protection provides a fast, efficient and simple solution to protect Windows applications. You won't need to spend any time learning about CRYPTO-BOX internals and incorporating corresponding code to your program. You won't even need the source code of your program. AutoCrypt Manager can do it all for you.

Your application is compressed and encrypted, then wrapped with a protective layer, preventing it from working unless a valid CRYPTO-BOX is attached. Many additional features and customizations are available.

For technical details about AutoCrypt and its functionality, please refer to chapter *4.4.*

Sample: Use AutoCrypt to Secure Applications and Ensure Payment

Software Vendor A has spent considerable time developing their software and is ready for it to hit the market. There is a tight time line and the application needs to be secured quickly. The company´s sales model for this product is to provide customers with a limited version of the software, valid for ten uses, and offer the option to upgrade to the full version for a one year term, renewable each year. Software Vendor A decided to implement the CRYPTO-BOX and AutoCrypt to ensure that these requirements are met.

Using AutoCrypt, Software Vendor A simply took their existing application and inserted the necessary data-objects provided by AutoCrypt. They created a new project, added their application, selected the execution counter data object (set to 10 uses) and created a license status message with instructions on how to upgrade to a full year at the end of the trial period. With the click of a button, they created a protected version of their software application.

The last step was to format the CRYPTO-BOX units for this protected application. Software Vendor A selected the number of units to format and, within minutes, had a protected version of their software ready for distribution.

After receiving the license status message, the end user requested full usage of the software. Using RUMS (Remote Update Utility, see chapter *6.1*), the end user sent an update request to the Software Vendor and, upon payment, received an Activation Key to update the license.

The same series of events will occur once the expiration date for this customer is reached.

This is only one of many options for software vendors to use AutoCrypt to secure applications and ensure payment.

2.5. Implementation Into Source Code with API

Implementation into source code through the API is a feature targeted at developers who need maximum security and flexibility for their applications. Using the API, you can implement a product-specific and highly efficient protection strategy. You can integrate smart support for demo and full-product versions of the program, online feature activation, remote update scenarios, and much more.

Because Smarx OS allows you to protect multiple applications simultaneously with one CRYPTO-BOX, you can mix applications protected through AutoCrypt and API-protection on one CRYPTO-BOX unit. Currently, Windows, Linux, macOS and Android are supported for API-protection implementations.

For more details regarding implementation with API and an overview about supported environments (IDEs), please refer to chapter *10*.

Sample: Turn Demos Into Paid Packages with API-Protection

Software Vendor B has developed a set of tax preparation applications, using Microsoft Visual Studio .NET. They know that, in order for their software to become widely used by tax consultants, they need to provide an almost fully functional demo. The only limitations required are that the tax documents are not printable and all internet based submission methods must be disabled.

These requirements were easily achieved using the Smarx API. If there is no CRYPTO-BOX attached to the end user´s USB port, the software can be copied and is fully functional, with the exception of the two limitations mentioned, which render the software useless to tax consultants. Once a customer decides to purchase the tax preparation package, Software Vendor B simply provides them with a CRYPTO-BOX.

Software Vendor B is able to protect all applications or application modules with one CRYPTO-BOX. The CRYPTO-BOX supports multiple applications/modules, each with its own licensing information partition in the internal memory.

The example above is a limited scenario where the CRYPTO-BOX API-protection scheme was applied. Since implementation through the API is so flexible, the possibilities are virtually infinite. For a more detailed description of the implementation with API, please refer to chapter *10*.

2.6. Software and Data Licensing

The Smarx OS PPK includes every tool and component you need to support all popular licensing models. Network license management allows you to support multi-user licensing, while the Smarx OS Remote Update interfaces offer the following: software renting, remote activation, subscriptions and Pay-per-Use models. These functions give you the means to dramatically increase sales.

2.7. Network License Management

The License Control System (LCS) for network license management allows end users to run more than one instance of your software product concurrently in the network. Only one CRYPTO-BOX is required to store the network license counter with the number of licenses granted to the end user. This CRYPTO-BOX has to be attached to one of the network computers running Smarx OS Network Server.
The CRYPTO-BOX SC and XS have an integrated license counter that enables you to choose how many instances of your software (between 1 and 254) can be run on the network simultaneously (number of seats – Floating License). The CRYPTO-BOX Versa is network enabled, but does not support license counters/Floating License.

Scenario 1: Use Network Licenses Without Limitation

Software Vendor C has developed a new application for corporations to use in their network environments. For this application, Software Vendor C is not concerned about how many people in the corporation will use the application, as they will purchase a corporate license. However, Software Vendor C wants to ensure that the application can only be used inside the corporate network, and they also need to be able to monitor and update the corporate license for the application. Software Vendor C programs a single CRYPTO-BOX to be used in each corporate network environment, containing all the necessary license information. This CRYPTO-BOX will be configured with a user limit of 255 for unlimited usage (supported by the CRYPTO-BOX SC and XS models). After configuration, Software Vendor C will set up the Network Server (contained in the Smarx OS Professional Protection Kit) on a LAN computer, to which the CRYPTO-BOX is connected. Once this is done, Software Vendor C will be able to ensure that the proper CRYPTO-BOX is found and will be in a position to allow the application to open the box using corresponding API calls. The Server Management Console will also provide the LAN administrator with a convenient way of monitoring the application's status, the attached CRYPTO-BOX, and all of the connected clients, etc.

Scenario 2: Limit the Number of Network Seats

Software Vendor D has developed an expensive application that also targets big corporations. Software Vendor D, like Software Vendor C, wants to ensure that the application can only be used in the corporate network, but also wants to limit access to the application based on the number of licenses purchased. In other words, once a corporation has purchased a given number of licenses, they will have only that number of connections to the application at any given time. Software Vendor D is easily able to issue these licenses, using LCS (License Control System), and grant up to 254 licenses per CRYPTO-BOX in a network. If the corporation requires more licenses, in order to have more concurrent users, they simply contact Software Vendor D, who can update the license information on the CRYPTO-BOX remotely, using RUMS (Remote Update Management System).

Scenario 1 and 2 illustrate two uses of network license management with MARX products. For more information about LCS please refer to our White Paper “Network Licensing”.

2.8. Maintenance of Protected Applications Using Remote Update

The Smarx OS Professional Protection Kit provides a convenient way for remotely updating the memory or the parameters of a CRYPTO-BOX in the field without having to ship the CRYPTO-BOX back and forth.

You can reprogram/update the following values inside the CRYPTO-BOX memory:

- Data Objects with licensing options, such as Expiration Date, Usage Counter or Application Password
- Network licenses
- Individual objects (strings with license information, etc.)

There are two ways to utilize the Remote Updates:

- Using the Remote Update Management Systems (RUMS), part of the Smarx OS Application Framework.
- Using the CBIOS RFP API (Remote Update API)

The first method is of interest for AutoCrypt users or people who do not want to spend time developing their own distribution scenario. They can easily use the Smarx OS Application Framework to program their individual Data Objects into the CRYPTO-BOX and have them updated later by their end users. More information can be found in chapter *6.1*.

The second method is for people who want to implement their own distribution scenario. Given the Remote Update API, they can incorporate all necessary steps into their application through API commands, from update request to update execution. Please refer to chapter *10* for more information.

Benefits of Remote Update:

- Saves time and money because the CRYPTO-BOX doesn't need to be shipped back and forth for reprogramming;
- Remote activation of software features and remote access to your software;
- Remote extension of usage and software leasing and of expiration dates;
- Remote execution of maintenance work and remote exchange of data.

Sample: Generate a Transaction Key for Remote Updates in the Field

Software vendor A, from the earlier AutoCrypt Scenario, has had a very successful start with their software product. They now get many requests for upgrades from their limited version to their full version, and existing customers want licenses for terms of longer than just one year. Using the CRYPTO-BOX and RUMS utility, Software Vendor A can quickly and easily process all customer requests. To do so, Software Vendor A’s end users simply run the RUpdate tool to generate a Transaction Key. Then, Software Vendor A processes the request in the Remote Update Manager, which returns an activation code. This code is sent to the customer, who feeds it to the RUpdate tool, automatically updating his software and CRYPTO-BOX.

This demonstration shows just one way the RUMS utility can be used. This method ensures quick and easy license management for most scenarios.

Application Notes with detailed information about usage scenarios of Remote Update can be found at www.marx.com → Support → Documents → Application Notes → RUMS.

Summary:

- Turn an evaluation version into a full-featured product.
- Pay-per-Use distribution models.
- Update network license counter.
- Update information in the CRYPTO-BOX memory (such as license strings, etc.).
- Check usage counters and reset them for software leasing or accounting purposes.

3. Starting with the Professional Protection Kit (PPK)

3.1. Installation

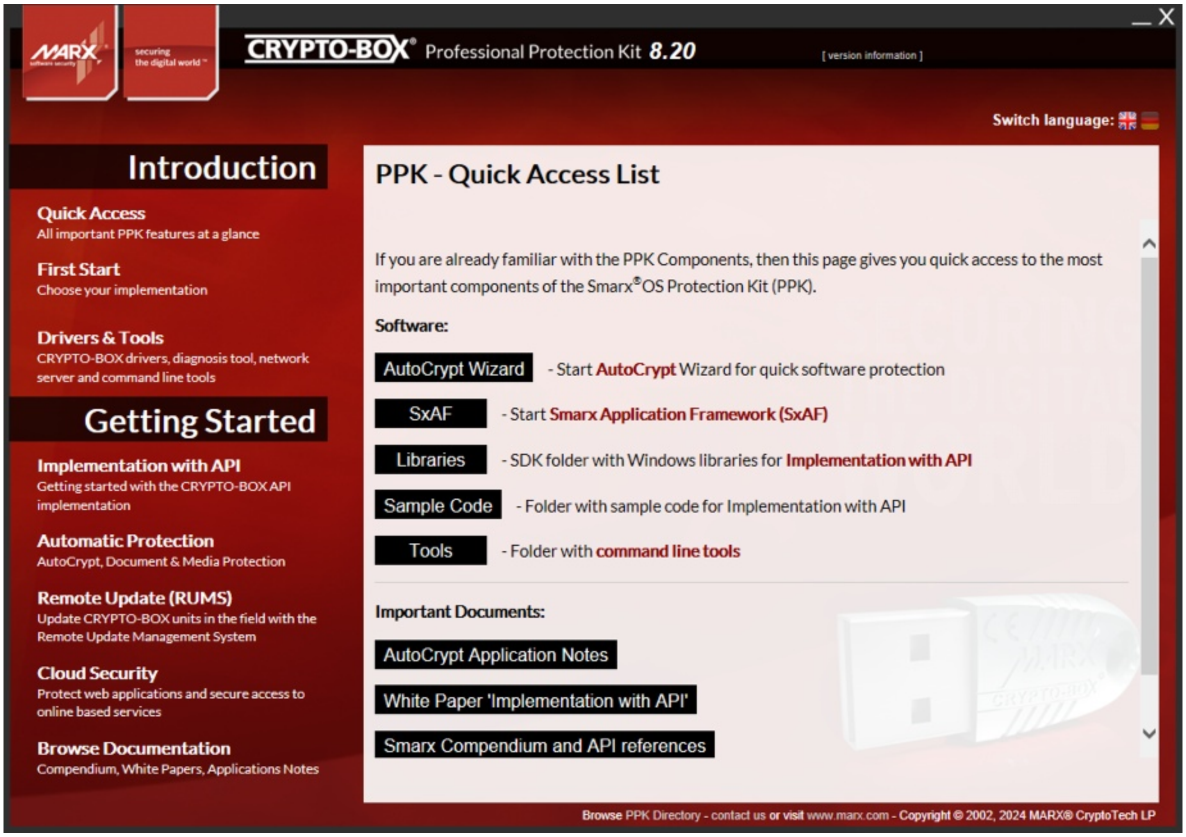
3.1.1. Windows

The Smarx OS Professional Protection Kit (PPK) which includes AutoCrypt for automatic, libraries and sample code for implementation with API as well as tools for CRYPTO-BOX programming and remote update can be downloaded from our web page.

> Visit www.marx.com/downloads to download the latest PPK version (MyMARX registration and valid Support Contract required). For new customers, Economy Support is included for the first 45 days. Open the Setup.exe to start the installation.

Once the PPK installation has finished, attach the CRYPTO-BOX to your PC. Windows will locate the drivers and install them automatically.

The PPK Control Center provides an overview of the components installed, including a brief introduction and links to the components.



CRYPTO-BOX® Professional Protection Kit 8.20 [version information]

Switch language:

Introduction

Quick Access
All important PPK features at a glance

First Start
Choose your implementation

Drivers & Tools
CRYPTO-BOX drivers, diagnosis tool, network server and command line tools

Getting Started

Implementation with API
Getting started with the CRYPTO-BOX API implementation

Automatic Protection
AutoCrypt, Document & Media Protection

Remote Update (RUMS)
Update CRYPTO-BOX units in the field with the Remote Update Management System

Cloud Security
Protect web applications and secure access to online based services

Browse Documentation
Compendium, White Papers, Applications Notes

PPK - Quick Access List

If you are already familiar with the PPK Components, then this page gives you quick access to the most important components of the Smarx®OS Protection Kit (PPK).

Software:

AutoCrypt Wizard - Start AutoCrypt Wizard for quick software protection

SxAF - Start Smarx Application Framework (SxAF)

Libraries - SDK folder with Windows libraries for Implementation with API

Sample Code - Folder with sample code for Implementation with API

Tools - Folder with command line tools

Important Documents:

AutoCrypt Application Notes

White Paper 'Implementation with API'

Smarx Compendium and API references

Browse PPK Directory - contact us or visit www.marx.com - Copyright © 2002, 2024 MARX® CryptoTech LP

Fig. 3.1:
The PPK Control Center

ℹ If you want to update an existing PPK installation to a newer version, it is strongly recommended to make a backup of your Smarx OS Application Framework (SxAF) database. Refer to chapter *4.3* for more details.

3.1.2. Linux

There is a separate “Smarx OS 4 Linux” package available in our download area which contains comprehensive support of Linux platform, including:

- Libraries and sample code for API based implementation for popular programming environments (GCC, Java, Qt and more);
- CBIOS Network Server;
- SmrxProg - CRYPTO-BOX programming tool

Visit www.marx.com/downloads to download the latest “Smarx OS 4 Linux” package (MyMARX registration and valid Support Contract required). Further information on CRYPTO-BOX support under Linux can be found in chapter *10.14.*

3.1.3. macOS

Visit www.marx.com/downloads to download the latest “Smarx OS 4 Mac” package (MyMARX registration and valid Support Contract required).

An overview about libraries and sample code for macOS can be found in chapter *10.15*.

3.2. How to Start

To protect your digital assets with the CRYPTO-BOX, you have to decide first if you prefer to use AutoCrypt or Implementation with API. Chapter *2* gives you a decision guidance here.

If you prefer **AutoCrypt**, please continue with chapter *4.4.*

If you decide for **Implementation with API**, continue with chapter *4.5*.

For **Document Protection**, see chapter *4.6*.

For **Media Protection**, see separate documentation on www.marx.com → Support → Documents

4. Smarx®OS Application Framework

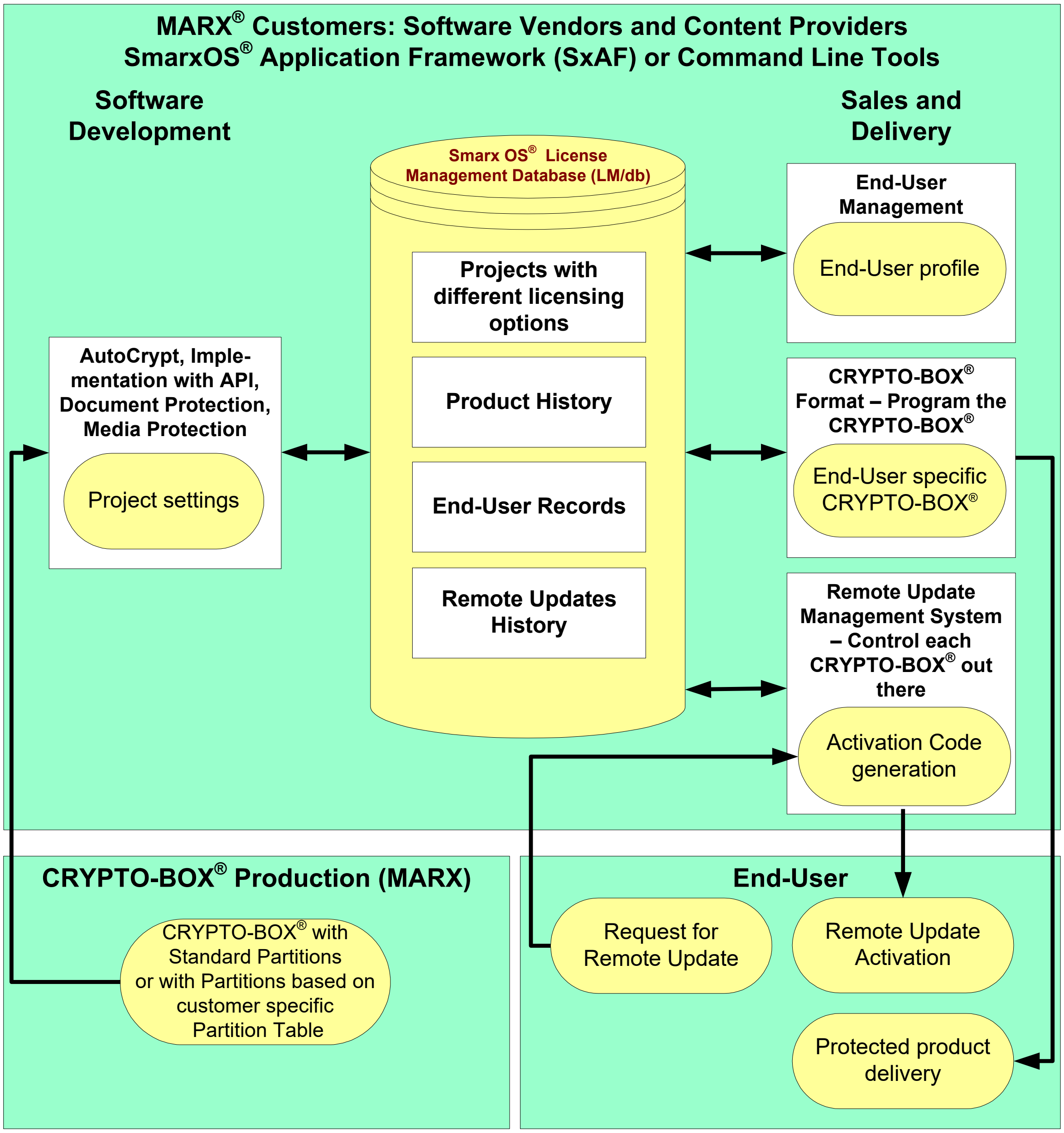

MARX® Customers: Software Vendors and Content Providers
SmarxOS® Application Framework (SxAF) or Command Line Tools
Software Development
Sales and Delivery
Smarx OS® License Management Database (LM/db)
Projects with different licensing options
Product History
End-User Records
Remote Updates History
AutoCrypt, Implementation with API, Document Protection, Media Protection
Project settings
End-User Management
End-User profile
CRYPTO-BOX® Format – Program the CRYPTO-BOX®
End-User specific CRYPTO-BOX®
Remote Update Management System – Control each CRYPTO-BOX® out there
Activation Code generation
CRYPTO-BOX® Production (MARX)
CRYPTO-BOX® with Standard Partitions or with Partitions based on customer specific Partition Table
End-User
Request for Remote Update
Remote Update Activation
Protected product delivery

Fig. 4.1:
Smarx®OS Application Framework

4.1. Overview

The Smarx®OS Application Framework (SxAF) is a project-based environment for software vendors and distributors. It automates software licensing and protection as well as data and media protection:

- Protect and license your software using automatic protection or implementation with API;
- Define the necessary partitions and data objects for implementation with API into the source code;
- Protect and license your documents with Document Protection;
- Create a compilation to combine protection and licensing of your different products into one project and protect them with one CRYPTO-BOX®.
- Configure (program) the CRYPTO-BOX according to your protection and licensing scenario.
- Remotely update licensing data contained in the CRYPTO-BOX distributed to the end-user;
- Manage end-user profiles, etc.

The Smarx OS Application Framework is project-based, meaning that you will need to create a designated project in order to protect your software products, documents or media files.

All projects are stored in the internal License Management Database (LM/db). With the Smarx OS Application Framework, you can open existing projects to program a new CRYPTO-BOX, process remote update requests or even change project settings at any time.

The Smarx OS Application Framework contains the following functional modules:

- AutoCrypt (for automatic software protection); see chapter *4.4*
- Implementation with API - Data Objects Manager; see chapter *4.5*
- Document Protection; see chapter *4.6*
- CB Format (CRYPTO-BOX Format); see chapter *4.9*
- Remote Update Management System (RUMS); see chapter *6.1*
- End-User Management; see chapter *4.10*

Additional options, such as end-user specific formatting and remote update history, can assist with your licensing strategy.

Furthermore, you may choose different Product Editions and Update Plans for each project, depending on your marketing and pricing strategies. You only have to set up the chosen Product Editions for your project and program your CRYPTO-BOX units with corresponding licensing data. See chapter *4.7* for more information on Product Editions.

4.2. Steps and Processes of Software/Document Protection

The Smarx OS Application Framework (SxAF) architecture covers the typical protection and distribution requirements of software vendors and content providers. With SxAF, you can protect your applications and documents, program the CRYPTO-BOX with the licensing data, and process remote updates of licensing data in future.

You, the software vendor or content provider can choose how to protect your application, documents or media files. You may use AutoCrypt for automatic protection, Data Object Manager for implementation with API, or Document/Media Protection. All these components are included into the Smarx OS Application Framework and share the same database (LM/db), which contains information on your selected protection and licensing options.

After choosing your licensing strategy and applying it to your applications or documents, CB Format is used to program the CRYPTO-BOX with the licensing data. Optionally, you can include the Remote Update Utility into the package for your end-users to allow remote updates of licensing data in future, utilizing the Remote Update Management System (RUMS). After formatting, the CRYPTO-BOX is delivered to end-users.

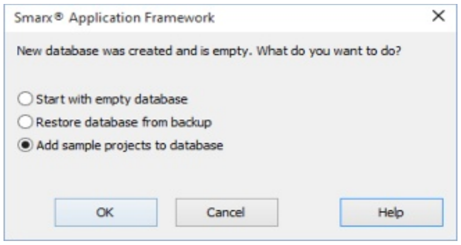
If the end-user wants to update licenses, for example extend the usage time or increase the number of network licenses, they can prepare and send you an update request (transaction key) using the Remote Update Utility.

The history of remote updates is recorded in the database. In addition, it shows what kind of remote updates (packages/plans) were provided for corresponding projects and end-users.

It is important to differentiate project management (selection of protection and licensing options and software/document protection itself) from CRYPTO-BOX programming (using CB Format). Applications/documents can be protected one time, but related CRYPTO-BOX formatting may be performed many times for different end-users. Smarx OS Application Framework contains end-user profile management and remote update history as part of the LM/db, which can be helpful for your own licensing strategy.

4.3. Smarx®OS Application Framework – First Start

When starting the Smarx OS Application Framework (SxAF) for the first time, a new database is created and you will see the following dialog:



New database was created and is empty. What do you want to do?

○ Start with empty database
○ Restore database from backup
● Add sample projects to database

Fig. 4.2:
Creating new database dialog

If you have a database backup file made by a previous installation of SxAF (2.x or later), use the “Restore database from backup” option.

If you have an existing backup of SxAF 1.x database (teosdb.mdb), you need to convert it first with a special conversion tool. You will find it under C:\Program Files\MARX Software Security\SmarxOS PPK\Tools\DBExport. After successful conversion, you can import the converted database with the “Restore" option in "Database" menu.

If you choose “Leave database empty”, the database contains only default “cbu_demo” Hardware Profile and no projects.
The “Create demo projects” option provides two demo projects, one for *AutoCrypt* (local and network) and one *Protection with API* project, intended for Evaluation.

⚠ It is strongly recommended to make regular backups of the SxAF database, especially when you want to update to a new version of the Protection Kit. To backup the database, select “Backup" option in "Database" menu.

When the Smarx OS Application Framework main screen appears, you can create a project according to your desired protection scenario:

- For automatic protection of Windows .exe and .dll files, see chapter *4.4*
- For implementation with API, see chapter *4.5*
- For secure distribution of documents, see chapter *4.6*

4.4. Automatic Software Protection with AutoCrypt

4.4.1. Overview

AutoCrypt protects applications without any programming efforts. It allows you to wrap any existing executable file with a secure layer of protection. The protection process involves incorporating of security code into your application's executable file, including compression and encryption of the original code. AutoCrypt has many features which enhance creative distribution strategies, including expiration dates, usage limits, periodic hardware checks, passwords and much more.

4.4.2. AutoCrypt Wizard, AutoCrypt SxAF, AutoCrypt Command Line

MARX offers 3 AutoCrypt versions:

a) AutoCrypt Wizard

This is the easiest way to protect your .EXE or DLL files. The Wizard guides you through each step of protection, licensing and CRYPTO-BOX configuration for quick results. You may also export the project file for usage with the SxAF (see below) providing you with some advanced licensing options, including project and user management, or with our command line tools.

b) AutoCrypt SxAF

This solution is less intuitive, but provides you with some advanced licensing options contained in SxAF, such as support for Product Editions (see chapter 4.7) as well as project and user management (see chapter *4.10*)

c) AutoCrypt Command Line

The biggest advantage of this solution: the protection process can be controlled within other applications or batch-files. This allows a high grade of automation and deep integration into your own, specific distribution strategy.

ℹ For a step-by-step guide on how to protect your application with **AutoCrypt** (Wizard, SxAF or command line version), please refer to our “AutoCrypt” Application Notes: www.marx.com → Support → Documents → Application Notes → AutoCrypt.

4.5. Using Smarx®OS Application Framework for API Implementation

The Smarx OS Data Objects Manager – part of the Smarx OS Application Framework – provides convenient project management for software that is protected by API (protection implemented into source code). With the Data Objects Manager it is easy to manage smart data objects in CRYPTO-BOX memory partitions, which will be verified in the developer's code. These data objects include expiration dates, usage limits, network licenses, customer-specific memory objects, etc.

4.5.1. Steps for Protecting Applications with API

To protect your application with API, we recommend the following steps:

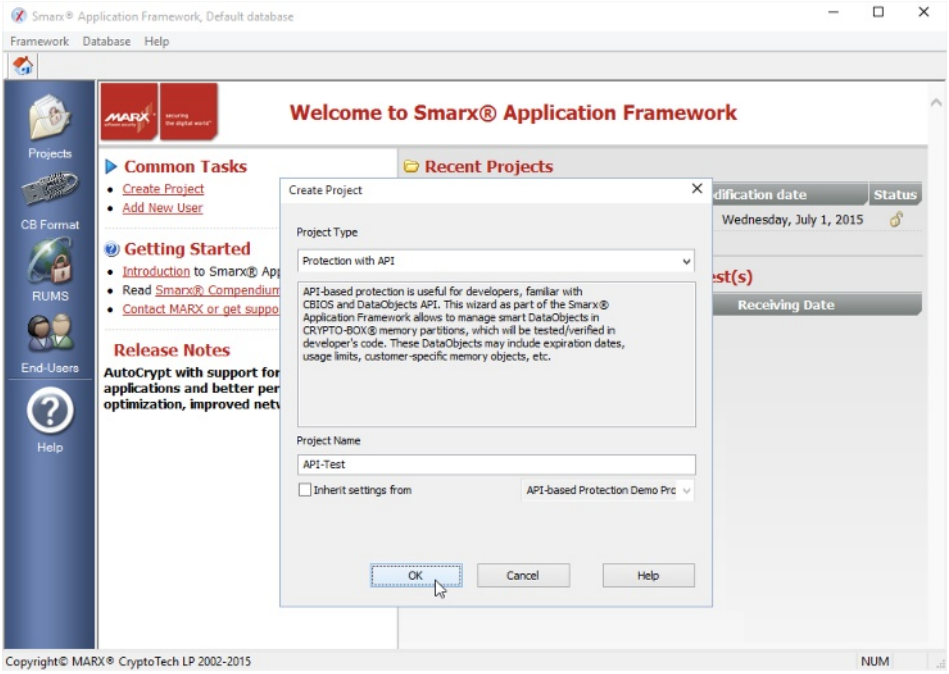
1. MARX supports almost all popular programming environments. We recommend that you familiarize yourself with our available APIs for developers by reading chapter *10* in this Compendium and evaluating the sample code in the PPK (see chapter *10.12.2* for a table of supported environments). Then choose your own protection strategy.
2. Define a new SxAF project, specifying *Implementation with API* as project type. A project includes all information used for programming the CRYPTO-BOX and is stored in the SxAF database (LM/db).
3. Choose the project-specific values for the CRYPTO-BOX, such as label and AES keys.
4. Select your project's licensing strategy by defining one or more partitions to hold data objects with licensing information, which can be expiration dates, counters, network licenses and/or customer specific memory objects (see chapter *4.5.5)*.
5. Use CB Format (see chapter *4.9*) to format your CRYPTO-BOX units with the project settings). Optionally, you can export your project settings into an XML file to use it with our command line based tool “SmrxProg” for CRYPTO-BOX formatting which provide more automation options (see chapter *4.8*). Under Linux and OS X, SmrxProg is the only option for CRYPTO-BOX formatting.
6. If you plan to update your CRYPTO-BOX later at your end-user's site, you can create the Remote Update Tool for this project and ship it together with the CRYPTO-BOX to your end-users (see chapter *4.9.3* for more information).
7. Test all licensing options carefully.
8. Ship your protected application, along with the CRYPTO-BOX and supplemental files (drivers, network server for network licensing if applicable). For Windows platform, device drivers (and network server in case of protection in networks) for the CRYPTO-BOX has to be shipped together with your protected application. MARX provides easy-to-use setup tools and even Windows Installer Merge Modules for this task. See chapter *8* for more details. Linux and OS X users will find more details on prerequisites in the readme files of the Linux/OS X package (see chapter *3.1*).

4.5.2. Creating, Deleting and Selecting projects

Start SxAF: In the PPK Control Center, choose “Quick Access” → “SxAF”. On the main screen of the Smarx OS Application Framework, you can create a new project or work with an existing project. If you work with an existing project (or you want to evaluate one of the demo projects), click the “Projects” tab on the left navigation bar and select an existing project. Or select “Create Project” to create a new project.

Enter a project name. Then select “OK”, to continue by defining the project settings.

The “Inherit settings from” option allows you to create an independent copy of an existing project. All project and partition settings will be taken from the existing project.



AutoCrypt with support for
applications and better per
optimization, improved netv

Create Project

Project Type: Protection with API

API-based protection is useful for developers, familiar with CBIOS and DataObjects API. This wizard as part of the Smarx® Application Framework allows to manage smart DataObjects in CRYPTO-BOX® memory partitions, which will be tested/verified in developer's code. These DataObjects may include expiration dates, usage limits, customer-specific memory objects, etc.

Project Name: API-Test

□ Inherit settings from: API-based Protection Demo Prc

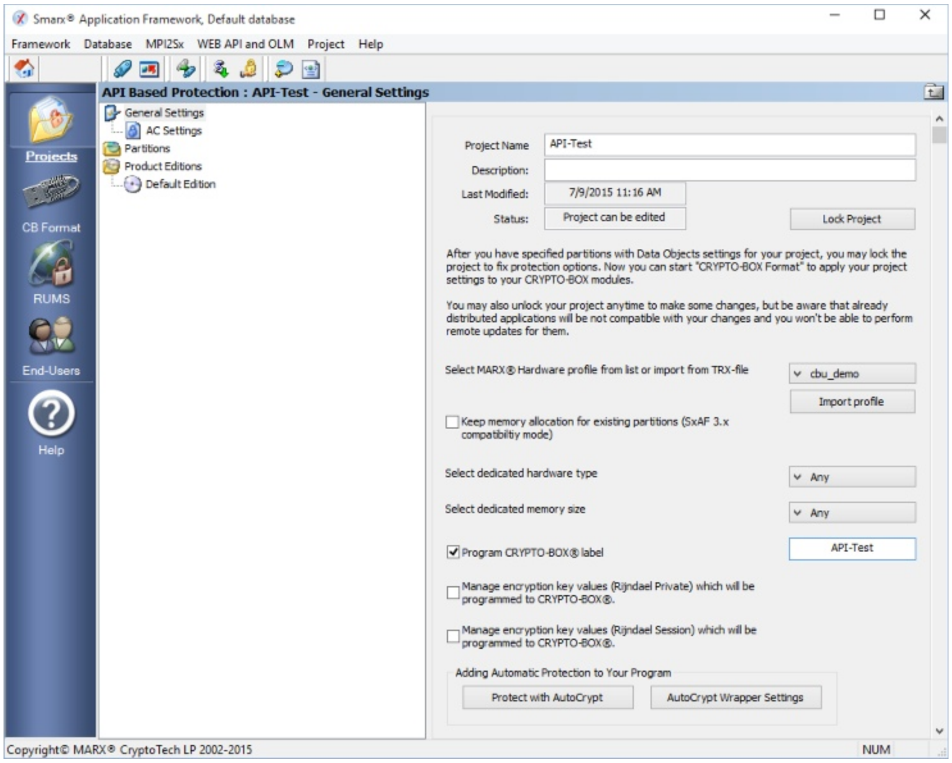
Fig. 4.3:
Creating a new API-based project

4.5.3. General Project Settings

With the “General Settings” tab in the navigation tree on the left side you can edit settings for the selected project.

You can change/edit the project name and the description. You can also lock this project (use it in read-only mode). This avoids accidental changes to the project. It is also possible to unlock the project for editing. But be careful if you already produced (formatted) CRYPTO-BOX modules with this project: You will not be able to process Remote Updates for these CRYPTO-BOX modules.

In the lower part of the window, you need to select your projects CRYPTO-BOX hardware profile. This profile contains the codes needed to access the CRYPTO-BOX. MARX distributes this to customers as a TRX file. Simply select the “cbu_demo” profile for the Evaluation Kit CRYPTO-BOX or click “Import profile” to import the profile you received with your customer-specific CRYPTO-BOX.



Project Name: API-Test

Description:

Last Modified: 7/9/2015 11:16 AM

Status: Project can be edited

After you have specified partitions with Data Objects settings for your project, you may lock the project to fix protection options. Now you can start "CRYPTO-BOX Format" to apply your project settings to your CRYPTO-BOX modules.

You may also unlock your project anytime to make some changes, but be aware that already distributed applications will be not compatible with your changes and you won't be able to perform remote updates for them.

Select MARX® Hardware profile from list or import from TRX-file: cbu_demo

□ Keep memory allocation for existing partitions (SxAF 3.x compatibiltiy mode)

Select dedicated hardware type: Any

Select dedicated memory size: Any

☑ Program CRYPTO-BOX® label: API-Test

□ Manage encryption key values (Rijndael Private) which will be programmed to CRYPTO-BOX®.

□ Manage encryption key values (Rijndael Session) which will be programmed to CRYPTO-BOX®.

Adding Automatic Protection to Your Program

Fig. 4.4:
API-based project: project settings

If you have existing partitions in the CRYPTO-BOX, for example created with another project or with the Extended API (see chapter *16*), and you do not want to change them, you can check the "Keep memory allocation for existing partitions (SxAF 3.x compatibility mode)" option. When you do so, CB Format will use the existing partition structure to write the project data (see chapter *4.9*). If this option is not checked, CB Format will assign the memory size dynamically, according to project requirements. For example, the CRYPTO-BOX already contains a partition #200 with 50 bytes assigned to RAM1, but the data objects written to this partition will occupy only 40 bytes of RAM1. By default, CB Format will dynamically assign those 40 bytes to save space. If the option "Keep memory allocation for existing partitions (SxAF 3.x compatibility mode)" is checked, it will write the data objects to the existing RAM1 memory, which was already assigned, and keep the 50 bytes RAM1 size for this partition intact.

If you wish to ensure that your project is only accessible with a particular type of CRYPTO-BOX hardware, you can use the "Select dedicated hardware type" to specify your choice: CBU SC (CRYPTO-BOX SC) or CBU (CRYPTO-BOX XS and Versa). You can use the "Select dedicated memory size" option to limit the memory size to be used with this project, and/or if you want to make sure that the project settings will fit to a particular CRYPTO-BOX. If the default settings are used, SxAF will try to adapt the project settings to the attached CRYPTO-BOX during formatting.

With the check box "Program CRYPTO-BOX label" you can set/change the label. The label is part of the CBIOS_BOX_INFO structure which can be obtained via the CBIOS_GetBoxInfoAdvl API command.

The two “Manage encryption key values...” options define the values for the AES Private/ Session Key and Initialization Vector of the CRYPTO-BOX hardware based AES encryption. The AES encryption can be used with the CBIOS_CryptPrivate and CBIOS_CryptSession API commands. By default, SxAF will generate new, random key values. To update key values with TRX file settings (factory default key settings for your CRYPTO-BOX), click the “Update” button.

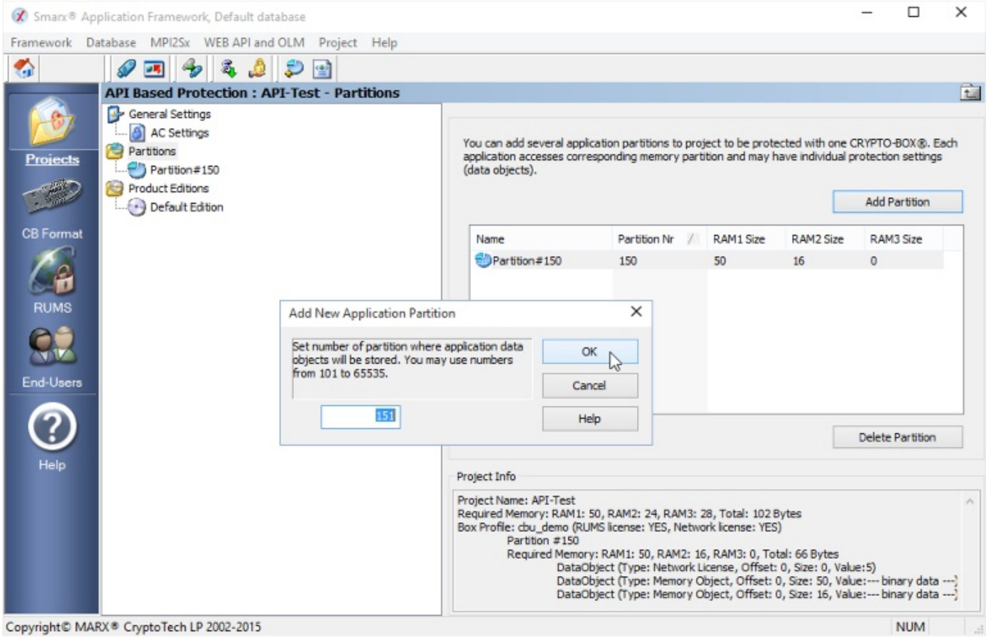
The "Protect with AutoCrypt" button will add automatic protection to any EXE/DLL module of your program. This can be used for additional security. Please refer to chapter *4.4 Automatic Software Protection with AutoCrypt* for more details.

4.5.4. Adding Partitions to the Project

Click the “Partitions” tab on the left navigation tree to start managing partition settings. Here you can add, edit and remove partitions; partitions are used for storing sets of licensing data (data objects) defining licensing logic for protected applications.

⚠ We strongly recommend to read chapter *10.2* for details on CRYPTO-BOX partitions – it is important to understand this concept!

To add a new partition to the project, click the “Add Partition” button and specify the number of your new CRYPTO-BOX partition. The number of a partition must be in the range between 101 and 65535.



API Based Protection : API-Test - Partitions

You can add several application partitions to project to be protected with one CRYPTO-BOX®. Each application accesses corresponding memory partition and may have individual protection settings (data objects).

Name	Partition Nr	RAM1 Size	RAM2 Size	RAM3 Size
Partition#150	150	50	16	0

Add New Application Partition

Set number of partition where application data objects will be stored. You may use numbers from 101 to 65535.

151

Project Info

```
Project Name: API-Test
Required Memory: RAM1: 50, RAM2: 24, RAM3: 28, Total: 102 Bytes
Box Profile: cbu_demo (RUMS license: YES, Network license: YES)
        Partition #150
        Required Memory: RAM1: 50, RAM2: 16, RAM3: 0, Total: 66 Bytes
                DataObject (Type: Network License, Offset: 0, Size: 0, Value:5)
                DataObject (Type: Memory Object, Offset: 0, Size: 50, Value:--- binary data ---)
                DataObject (Type: Memory Object, Offset: 0, Size: 16, Value:--- binary data ---)
```

Copyright© MARX® CryptoTech LP 2002-2015

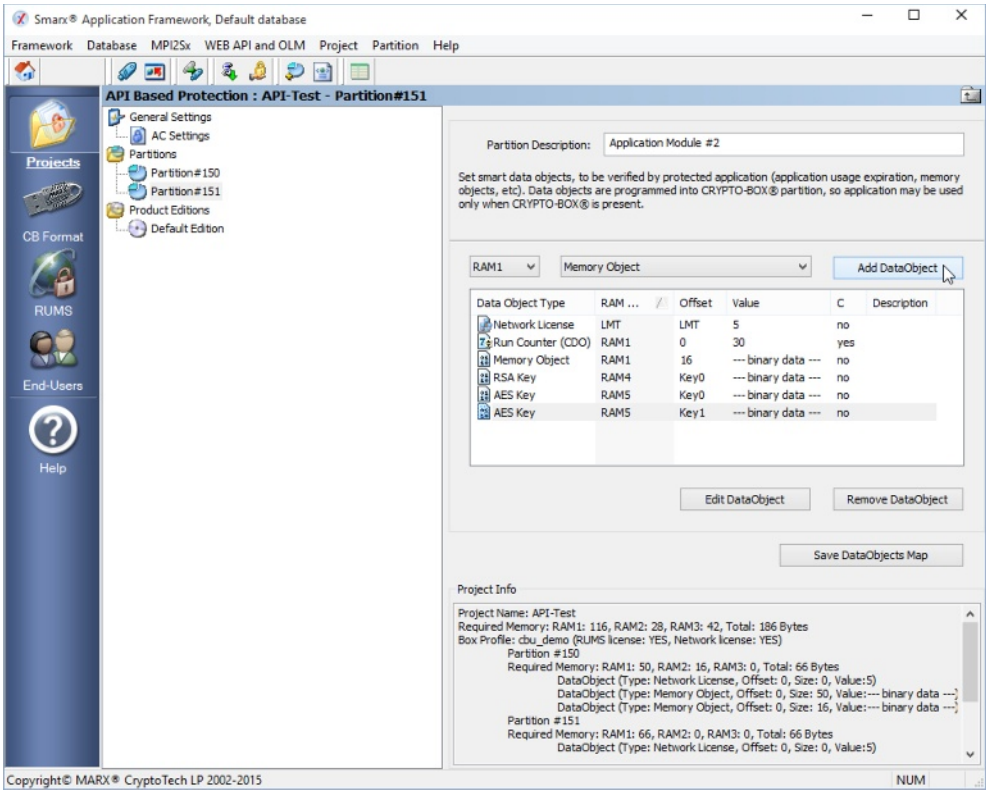
Fig. 4.5:
Adding a new partition to the project

To delete an existing partition, select the partition from the list and click the “Delete Partition” button.

To change the data objects settings of a partition, double-click the desired partition or select it from the left navigation bar.

4.5.5. Defining Data Objects

Here you can set application-specific data objects that will be programmed into the selected CRYPTO-BOX partition. Select the desired data object from the list and click the “Add Data Object” button to add the data object to the selected partition.



Partition Description: Application Module #2

Set smart data objects, to be verified by protected application (application usage expiration, memory objects, etc). Data objects are programmed into CRYPTO-BOX® partition, so application may be used only when CRYPTO-BOX® is present.

RAM1 | Memory Object

Data Object Type	RAM ...	Offset	Value	C	Description
Network License	LMT	LMT	5	no	
Run Counter (CDO)	RAM1	0	30	yes	
Memory Object	RAM1	16	--- binary data ---	no	
RSA Key	RAM4	Key0	--- binary data ---	no	
AES Key	RAM5	Key0	--- binary data ---	no	
AES Key	RAM5	Key1	--- binary data ---	no	

Project Info

```
Project Name: API-Test
Required Memory: RAM1: 116, RAM2: 28, RAM3: 42, Total: 186 Bytes
Box Profile: cbu_demo (RUMS license: YES, Network license: YES)
        Partition #150
        Required Memory: RAM1: 50, RAM2: 16, RAM3: 0, Total: 66 Bytes
                DataObject (Type: Network License, Offset: 0, Size: 0, Value:5)
                DataObject (Type: Memory Object, Offset: 0, Size: 50, Value:--- binary data ---}
                DataObject (Type: Memory Object, Offset: 0, Size: 16, Value:--- binary data ---}
        Partition #151
        Required Memory: RAM1: 66, RAM2: 0, RAM3: 0, Total: 66 Bytes
                DataObject (Type: Network License, Offset: 0, Size: 0, Value:5)
```

Fig. 4.6:
Adding Data Objects

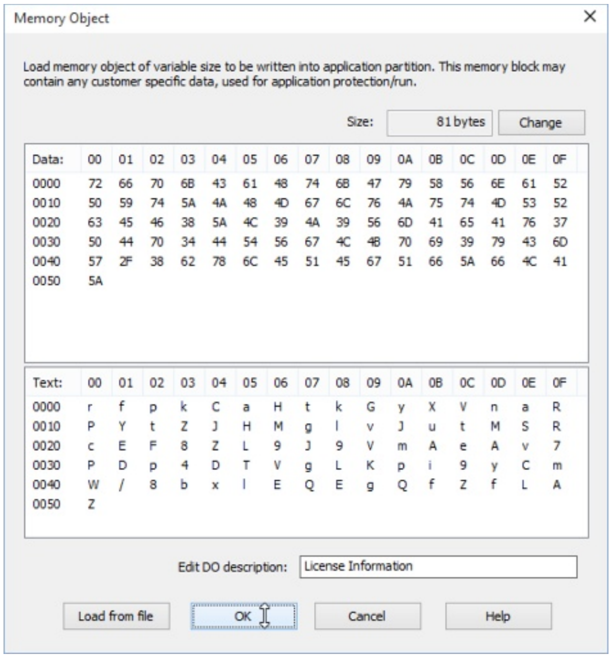
The following data object types are supported (see also chapter *14* for more information on Smarx OS Data Objects API*)* :

- **Run Counter:** specifies the number of application executions (runs). This counter decrements one step for each execution. As soon as the run counter is 0, the license is expired.
- **Expiration Date & Time:** fixed expiration date and time of the application's license expiration. After this date, the application can no longer be started.
- **Expiration Date:** fixed expiration date of the application's license expiration. After this date, the application can no longer be started. This type of data object is obsolete and preserved only for compatibility purposes. We recommend that you use “Expiration Date & Time” instead.
- **Expiration Days:** flexible expiration of license in number of days. The license counter decrements one step per day. After the defined number of days has expired, the application can no longer be started.
- **Expiration Time:** real-time expiration of a license in hours, minutes, and seconds. The license counter decrements during execution each time the periodic check is called.

- **Password:** an application launch password. It is queried every time the application is launched.
- **Network License:** number of times the protected application can be simultaneously launched in the network (number of "seats", Floating License). Multiple instances of the protected application can be launched from one or more network computers with one CRYPTO-BOX connected to the server (see chapter *42* for more information).
  The "License sharing rule" option can be activated to specify different license sharing options. "Sharing" means that, when a specified condition is met, a predefined group of application instances can use the same license (instead of the usual one license per application). IMPORTANT: if the "License sharing rule" option is activated CBIOS_LockLicenceExt API call must be used.

> The "Network License" data object is stored in a special area of the CRYPTO-BOX memory. It can hold network license information for multiple partitions (different applications). That means you can define independent network license counters for each partition (application) in the CRYPTO-BOX. See chapter *5* for more information.

- **Data Encryption Key Object:** data object that supports Data Encryption.
  More information on Data Protection API can be found under the corresponding entry in the Protection Kit Control Center.
- **AES Key:** AES Encryption Key programmed in RAM5 zone. Key values cannot be read from this zone, they can be used for hardware-based AES encryption only. This type of data object is supported exclusively by the CRYPTO-BOX SC. For more information using this key type, please refer to chapter *10.10*.
- **RSA Key:** RSA Encryption Key programmed in RAM4 zone. Key values cannot be read from this zone, they can be used for hardware-based RSA encryption only. This data object type is supported exclusively by the CRYPTO-BOX SC. For more information using this key type, please refer to chapter *10.10*.
- **AES Descriptor:** AES Encryption Key Descriptor to be programmed to application partition. Allows programming of key type (CBU SC AES or Internal AES) as well as AES algorithm specific data.
- **RSA Descriptor:** AES Encryption Key Descriptor to be programmed to application partition. Allows programming of key type (CBU SC RSA or Internal RSA) as well as RSA algorithm specific data (like padding).
- **Signature:** Combination of two RSA encryption key descriptors used for signing routine.
- **Binding to local PC:** Locks the software to the computer on which it is running or to the network server where the CRYPTO-BOX is attached.
- **Memory Object:** customer-specific data objects (memory block of variable size) to be written into a CRYPTO-BOX partition and to be read/updated by the protected application. Memory objects may be loaded from a file prepared beforehand (e.g. from a .txt or .ini file) using the "Load from File" button.



Load memory object of variable size to be written into application partition. This memory block may contain any customer specific data, used for application protection/run.

Size: 81 bytes

Data:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	72	66	70	6B	43	61	48	74	6B	47	79	58	56	6E	61	52
0010	50	59	74	5A	4A	48	4D	67	6C	76	4A	75	74	4D	53	52
0020	63	45	46	38	5A	4C	39	4A	39	56	6D	41	65	41	76	37
0030	50	44	70	34	44	54	56	67	4C	4B	70	69	39	79	43	6D
0040	57	2F	38	62	78	6C	45	51	45	67	51	66	5A	66	4C	41
0050	5A															

Text:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0000	r	f	p	k	C	a	H	t	k	G	y	X	V	n	a	R
0010	P	Y	t	Z	J	H	M	g	l	v	J	u	t	M	S	R
0020	c	E	F	8	Z	L	9	J	9	V	m	A	e	A	v	7
0030	P	D	p	4	D	T	V	g	L	K	p	i	9	y	C	m
0040	W	/	8	b	x	l	E	Q	E	g	Q	f	Z	f	L	A
0050	Z															

Edit DO description: License Information

Fig. 4.7:
Adding Memory Object

Memory objects cannot be larger than the space available in the CRYPTO-BOX memory. If the size of memory objects exceeds the space available, CRYPTO-BOX Format will display an error message during formatting.

You may add one or more data objects and edit/change their values. Data objects are located in the partition memory; their offsets are calculated dynamically.

If you need to have different data object settings for different customers, you do not need to create a separate project for each customer! Just create different **Product Editions** of your project with different data object settings. Please refer to chapter *4.7 Product Editions* for more information.

The “Protection with API” project is stored automatically in the SxAF database. The data objects will be programmed into the CRYPTO-BOX using the CB Format component for CRYPTO-BOX programming (see chapter *4.9 CRYPTO-BOX®Format: Configuring and Programming*) so the application may be used only when the CRYPTO-BOX is present.

Later, various protection options, set by Smarx OS Data Object Manager, can be updated remotely using the Remote Update Management System (RUMS).

A more detailed decription of these data objects and how to access them using API commands provided by the Smarx OS DO API can be found in chapter *14*. You may want to try different protection and licensing options to determine the best protection strategy for your application.

After you have specified your desired project settings, you need to format your CRYPTO-BOX units with the project settings, using CB Format. See chapter *4.9 CRYPTO-BOX®Format: Configuring and Programming* for more information.

> If you plan to update your CRYPTO-BOX remotely, we strongly recommend that you create additional (empty) data objects and reserve them for future use. Later, you can use SxAF RUMS to update their content (see chapter *4.9.3)*. Because RUMS in SxAF can only update existing data objects! If you need to add new memory objects later, you can use the command line based RU_Tool.exe, see chapter *7.5*.

4.5.6. Export of Data Objects Map and Smarx®API License File

With the Data Object Manager, you can export the data objects map to a file to load it with an external application. A sample illustrating the evaluation of the map file and testing product specific licensing logic is included to the Protection Kit (Visual Studio project). See Control Center → Implementation with API → Demo Code → DODemoApp fur further details.
To export the data objects map, click the “Save DataObjects Map” button.

See chapter *11.2* for instructions on how to export a license file for usage with the Smarx API.

4.6. Document Protection

Document Protection offers a secure way to distribute digital documents. This solution converts customer-specific documents to PDF format and then encrypts them to ensure they cannot be viewed without an attached CRYPTO-BOX. All common text formats (such as .DOC, .TXT, .RTF) are supported. Additionally, protected documents may contain an expiration date, which can later be updated with the Remote Update Management System (RUMS, see chapter *6.1*).

Document Protection consists of two components: The *Document Protection* component of Smarx OS Application Framework and the *PDF Viewer*, which is used to view the protected documents.

> If you look for protection of digital media (audio and video files with the CRYPTO-BOX, Media Protection is available as separate product, see www.marx.com → Shop → Solutions → Media Protection for more details.

Both Media Protection and Document Protection do not require programming.

4.6.1. Steps for Protecting Digital Documents

Document Protection is accomplished through the following steps:

1. Define a new SxAF *Document Protection* project. A project includes all information that is used for programming the CRYPTO-BOX. The projects are stored in the Smarx OS License Management Database (LM/db).
2. Select appropriate licensing strategy for the project by defining one or more licensing groups and/or product editions with optional expiration date specified for every group and edition defined for the project. Groups are used to organize documents. For

example, you can create a group for all documents relating to your English manual, one group for the German manual and one group for extra documents. So it is very easy to add complete groups to a project.

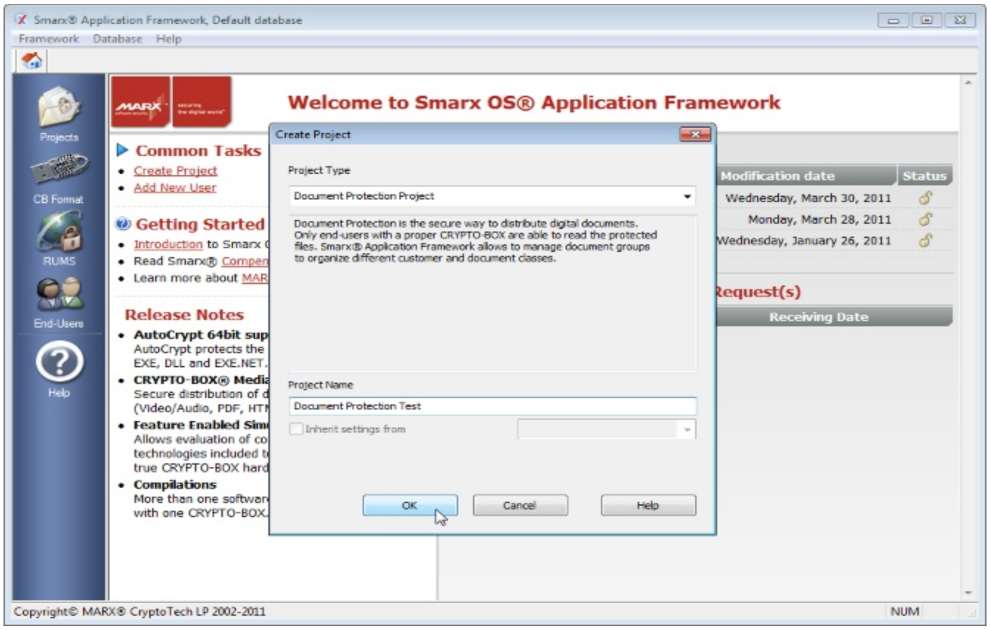
3. Individually add documents to one of the project groups and select conversion options for each document.
4. Protect the documents.
5. Use the *CB Format* component of SxAF (see chapter *4.9*) to format the CRYPTO-BOX units with the project settings).
6. With the *PDF Viewer*, end-users are able to view protected documents only when a CRYPTO-BOX properly formatted for this project is attached to the end-user’s computer.
7. If you plan to update licensing options later at your end-user's site (prolong usage time by modifying the expiration date), you can create the Remote Update Tool and ship it together with the CRYPTO-BOX to your end-users (see chapter *4.9.3* for more information).
8. Ship your protected documents together the CRYPTO-BOX and the necessary supplemental files (PDF Viewer setup, CRYPTO-BOX driver setup). MARX provides an easy-to-use redistribution setup. See chapter *4.6.7* for more details.

4.6.2. Creating a New Project or Selecting Existing Projects

On the main screen of the Smarx OS Application Framework, you can either create a new project or work with an existing project. If you work with an existing project, click the “Projects” tab on the left navigation bar and select an existing project. Or click the “Create Project” button to create a new project.

Enter a project name. Then press the “OK” button to continue by defining the project settings.

With the “Inherit settings from” option you can create an independent copy of an existing project. All project and partition settings will be taken from the existing project.



Create Project

Project Type: Document Protection Project

Document Protection is the secure way to distribute digital documents. Only end-users with a proper CRYPTO-BOX are able to read the protected files. Smarx® Application Framework allows to manage document groups to organize different customer and document classes.

Project Name: Document Protection Test

☐ Inherit settings from

Fig. 4.8:
Creating a new Document Protection project

4.6.3. General Project Settings

With the “General Settings” tab (in the navigation tree on the left side) you can edit settings for the selected project.

You can change/edit the project name and the description. You can also lock this project (use it in read-only mode). This avoids accidental changes to the project. It is also possible to unlock the project for editing. But be careful if you already produced (formatted) CRYPTO-BOX unis with this project: after editing the project you might not be able to process Remote Updates for these CRYPTO-BOX units.

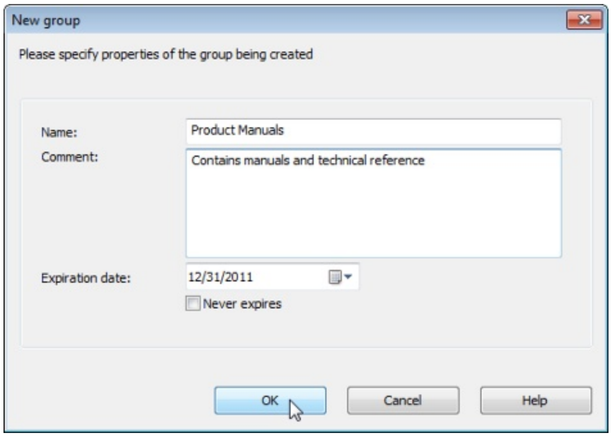
In the lower part of the Window, you need to select your projects CRYPTO-BOX hardware profile. This profile contains the access codes that the protected application needs to access the CRYPTO-BOX. You will receive it as a TRX format file with your CRYPTO-BOX shipment from MARX. Click on “Import profile” to import the profile you received together with your customer specific CRYPTO-BOX units. To evaluate a CRYPTO-BOX Evaluation Kit simply select the “cbu_demo” profile.

4.6.4. Creating a Document Group

Document protection projects consist of one or more groups, each with one or more documents. Each group may have its own expiration date, or you can use the “never expires” option, which means that the documents in this group will be available for reading without expiration.

If you do not want to create different groups of documents, you can simply add your documents to the project without creating a group. Then, global settings of the project for the expiration date will be valid.

Click the Add Group“ button to create a new group of documents. In the following window, specify a name for the document group you want to create. The “Comment” field can be used to add a detailed description to the document group. Additionally, you need to choose an expiration date for the documents in this group (or check the option “Never expires”).



Please specify properties of the group being created

Name: Product Manuals

Comment: Contains manuals and technical reference

Expiration date: 12/31/2011

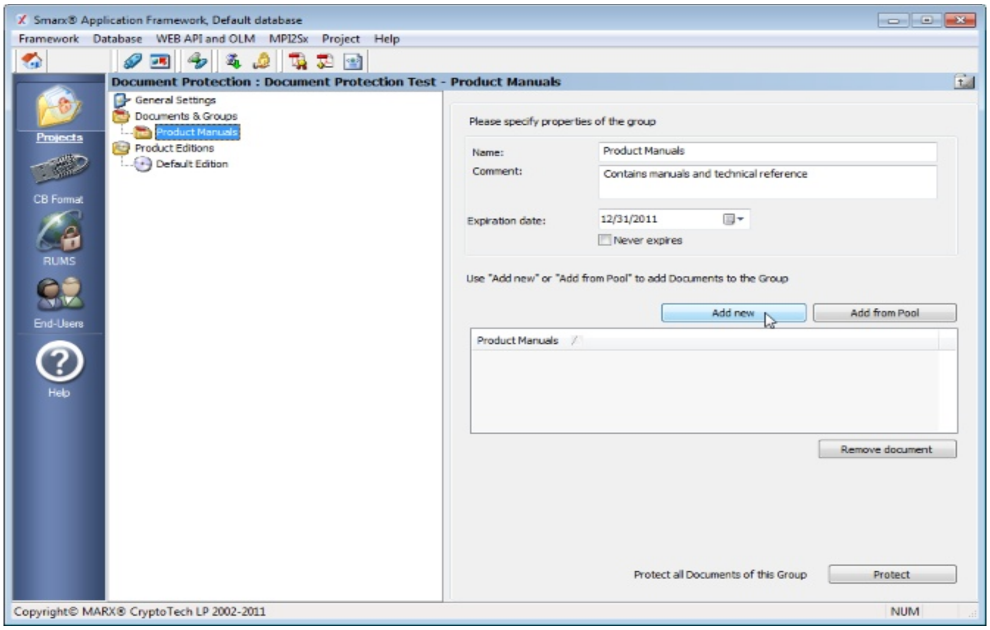
□ Never expires

Fig. 4.9:
Creating a new group of documents

Click “OK” to create the group and go back to the main screen. You will see that the new group you created will appear under the "Documents & Groups" tab in the navigation tree.

4.6.5. Adding Documents to the Group

Now you can add a document to the group being created or delete a document from the group. Also, this page allows you to convert documents using MARX PDF Converter Printer, upload them to the Document Pool and attach (add) them to the group. The Document Pool is the document-related part of the SxAF database (LM/db). It contains folders organized in tree structure and documents. All the contents of the Document Pool reside in the LM/db. When you delete a document from the project, it is being deleted only from the project, not from the Document Pool.



Smarx® Application Framework, Default database

Document Protection : Document Protection Test - Product Manuals

Please specify properties of the group

Name: Product Manuals

Comment: Contains manuals and technical reference

Expiration date: 12/31/2011

□ Never expires

Use "Add new" or "Add from Pool" to add Documents to the Group

Protect all Documents of this Group

Copyright© MARX® CryptoTech LP 2002-2011

Fig. 4.10:
Adding new documents

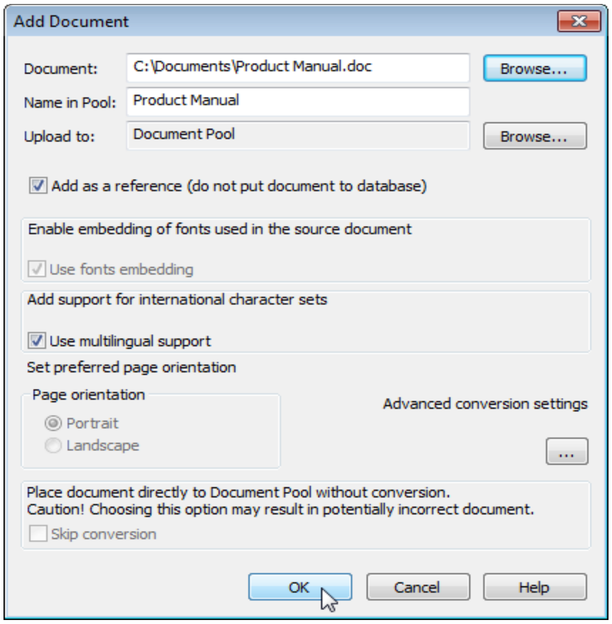
To add a new document (and upload it to the Document Pool), click the “Add new” button. You will see the “Convert Document” dialog. In this dialog, you need to provide document’s path, name the document will have in the Document Pool, and the Document Pool folder the document will be uploaded to.

If the "Add as a reference" option is checked, SxAF will not upload the source document to the database to save space.

The option "Skip conversion" allows you to add existing PDF files directly to the Document Pool without using the MARX PDF converter. This option is not available for documents which need to be converted to PDF first (for example Microsoft Word documents).

Use multilingual support

When this option is selected, any characters that are not included in the Western European and US character sets will be converted. Additionally, all document fonts will be embedded to ensure that the document is displayed correctly. Keep in mind, however, that this conversion is less efficient and files tend to be larger in size.



Add Document

Document: C:\Documents\Product Manual.doc

Name in Pool: Product Manual

Upload to: Document Pool

☑ Add as a reference (do not put document to database)

Enable embedding of fonts used in the source document

☑ Use fonts embedding

Add support for international character sets

☑ Use multilingual support

Set preferred page orientation

Page orientation

● Portrait

○ Landscape

Advanced conversion settings

Place document directly to Document Pool without conversion.
Caution! Choosing this option may result in potentially incorrect document.

□ Skip conversion

Fig. 4.11:
Convert document and upload it to the Document Pool

Advanced conversion settings

This option enables you to edit some more conversion settings:

- **Use fonts embedding:** By selecting this option you will enable inclusion of all True Type fonts used by the document in the output PDF file. This results in larger files but ensures a consistent appearance of the document on any platform. You have to activate this option if you want to convert a PDF file with embedded fonts. If the "Use multilingual support" option is selected, fonts will be embedded automatically.
- **Page resolution:** defines the resolution of the converted document in DPI. Higher resolutions result in better quality. However, the document can become very large.
- **JPEG compression:** defines the compression level of pictures. "High" means strong compression but less picture quality, "Low" means good picture quality but larger document size.

> You can add a document of any type (not only PDF, but also DOC, RTF, TXT). It will be converted to the Portable Document Format (PDF). Please note, that some document formats (like .doc files) require to have the source application (e.g. Microsoft Word) installed on your PC to convert them.

To start document conversion, press OK.
After the conversion process is finished, the converted document is automatically added to the selected group (or to the root folder) and also appears under the "Documents & Groups" tab in the navigation tree.

4.6.6. Protect Documents

After you finish adding all the documents to the project and set up the document groups, click "Protect". All documents in the current project will be protected and stored in the selected target folder.

> If you are currently in a group folder, Document Protection will only protect files of this group. If you have selected the project root folder (the "Documents & Groups" tab in the navigation tree), Document Protection will protect all documents in the project.

After you have finished protecting all required documents in your project, you can start to program the CRYPTO-BOX units you want to distribute with the protected documents. Please use CRYPTO-BOX Format to do this (see chapter *4.9 CRYPTO-BOX®Format: Configuring and Programming*).

4.6.7. PDF Viewer

The PDF Viewer is intended for end-users to access documents protected with Document Protection. To view protected documents, the end-user has to attach a valid CRYPTO-BOX (formatted with the settings of the Document Protection project, see chapter *4.9*) to the USB port and open the file (with PPD extension).

> Refer to chapter *8* and *8.3* for more details on CRYPTO-BOX driver and PDF Viewer installation at the end-user site.

4.7. Product Editions

Every SxAF project (AutoCrypt, Implementation with API or Document/Media Protection) is associated with some license: a set of data objects with initial values defining licensing for software product or document(s) protected with this SxAF project.

When protecting a software or media product, you may want to define more than one license per project, or in other words, have more than one edition of your product.

Some examples for product editions:

- Advanced Local Edition: for 1 year
- Unlimited Local Edition: unlimited

or

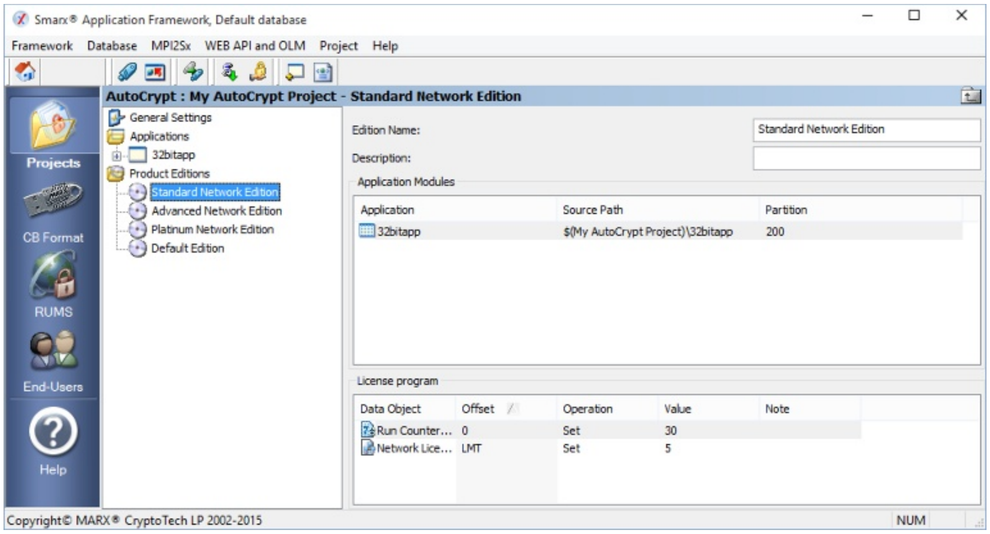
- Standard Network Edition: 5 network licenses for 6 months
- Advanced Network Edition: 10 network licenses for 1 year
- Platinum Network Edition: 15 network licenses, unlimited time

And so on.

Such editions can help with your marketing and pricing strategies. All you have to do is to set up the desired Product Editions for your project and format your CRYPTO-BOX units with those corresponding settings using CRYPTO-BOX Format (see chapter *4.9*). Product Editions are supported for all types of SxAF Projects (AutoCrypt, API based Protection, Document

Protection and Media Protection). If you do not create your own editions, the standard edition serves as the default option.

To add a new Product Edition, click the "Product Editions" tab on the left navigation tree. Choose "Add Edition" to create a new one or double-click on an existing partition to change its settings.



AutoCrypt : My AutoCrypt Project - Standard Network Edition

Edition Name: Standard Network Edition

Description:

Application Modules

Application	Source Path	Partition
32bitapp	$(My AutoCrypt Project)\32bitapp	200

License program

Data Object	Offset	Operation	Value	Note
Run Counter...	0	Set	30	
Network Lice...	LMT	Set	5	

Fig. 4.12:
Adding Product Editions

You can only change data objects in the Product Edition settings which were initially created in your project settings. Therefore, please be sure you have created the corresponding data object in your project before you create Editions with different settings. For Document/Media Protection projects there is only one data object type supported: Expiration Date.

4.8. Generating XML Script for Use with Command Line Tools

If you want to integrate application protection and CRYPTO-BOX formatting with your own administration/ distribution strategy, choose the “Generate script for SmarxTools“ in the “Project“ menu. This allows you to export the project data to a XML script file for further usage with AC_Tool.exe (in case of AutoCrypt project), Doc_Tool.exe (in case of Document Protection Projects) and/or SmrxProg.exe.

AC_Tool (for protecting applications, see chapter *7.2*), Doc_Tool (for protecting documents, see chapter *7.3*) and SmrxProg (for configuring CRYPTO-BOX modules for all type of SxAF projects, see chapter *7.4*) are console applications. They are controlled via command line switches and can be called up by applications or scripts.

4.9. CRYPTO-BOX®Format: Configuring and Programming

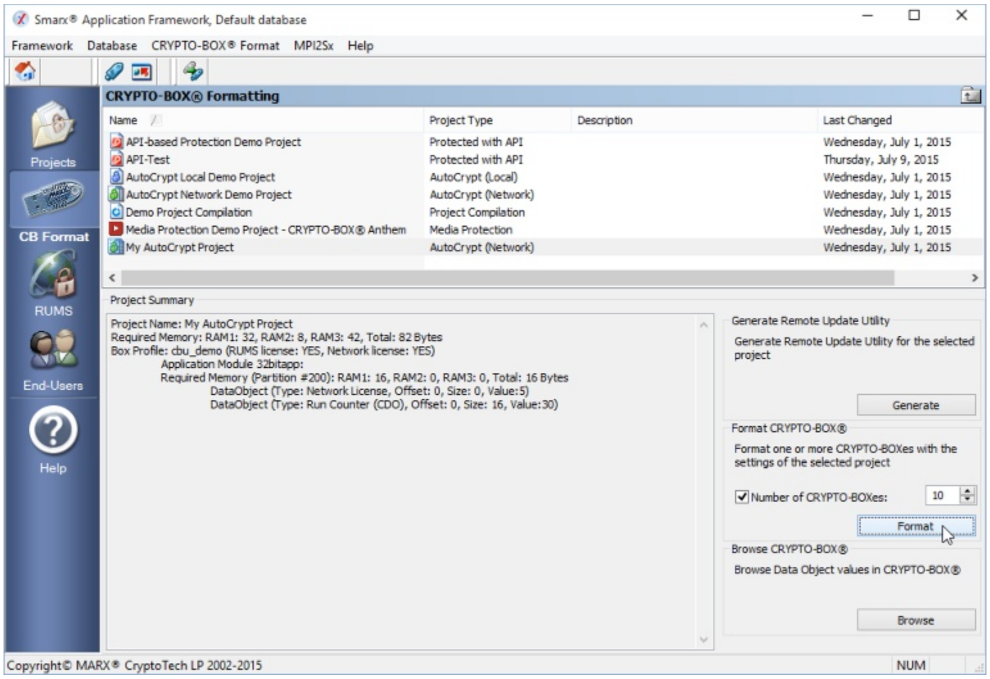
CRYPTO-BOX Format (CB Format) as part of the Smarx OS Application Framework provides CRYPTO-BOX formatting for projects stored in the SxAF database (LM/db). The following project types are supported:

- AutoCrypt projects (see chapter 4.4)
- Protection with API projects (see chapter 4.5)
- Document Protection projects (see chapter 4.6)

To start CRYPTO-BOX Format click “CB Format” in the Smarx OS Application Framework.

4.9.1. Selecting Projects to Format

Select an existing project in the upper window. The project summary field below allows you to review the project settings.



Name	Project Type	Description	Last Changed
API-based Protection Demo Project	Protected with API		Wednesday, July 1, 2015
API-Test	Protected with API		Thursday, July 9, 2015
AutoCrypt Local Demo Project	AutoCrypt (Local)		Wednesday, July 1, 2015
AutoCrypt Network Demo Project	AutoCrypt (Network)		Wednesday, July 1, 2015
Demo Project Compilation	Project Compilation		Wednesday, July 1, 2015
Media Protection Demo Project - CRYPTO-BOX® Anthem	Media Protection		Wednesday, July 1, 2015
My AutoCrypt Project	AutoCrypt (Network)		Wednesday, July 1, 2015

Project Summary

```
Project Name: My AutoCrypt Project
Required Memory: RAM1: 32, RAM2: 8, RAM3: 42, Total: 82 Bytes
Box Profile: cbu_demo (RUMS license: YES, Network license: YES)
          Application Module 32bitapp:
          Required Memory (Partition #200): RAM1: 16, RAM2: 0, RAM3: 0, Total: 16 Bytes
                    DataObject (Type: Network License, Offset: 0, Size: 0, Value:5)
                    DataObject (Type: Run Counter (CDO), Offset: 0, Size: 16, Value:30)
```

Generate Remote Update Utility

Generate Remote Update Utility for the selected project

Format CRYPTO-BOX®

Format one or more CRYPTO-BOXes with the settings of the selected project

☑ Number of CRYPTO-BOXes: 10

Browse CRYPTO-BOX®

Browse Data Object values in CRYPTO-BOX®

Fig. 4.13:
CRYPTO-BOX Format: "Select Project" dialog

4.9.2. Formatting CRYPTO-BOX Modules

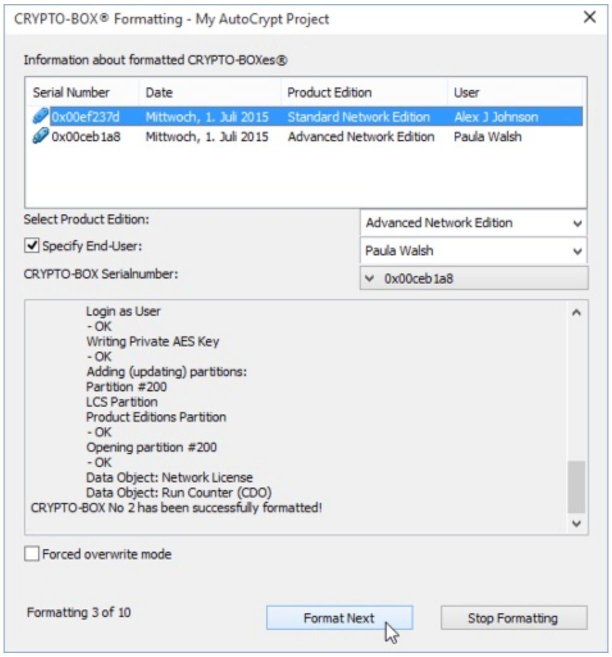
In the lower right corner, select the number of CRYPTO-BOX units you want to format. Plug in the first CRYPTO-BOX and click the “Format” button. In the next window, you will see the following options:

- **Information about formatted CRYPTO-BOXes** - shows you the CRYPTO-BOX units you already formatted, including Serialnumber, selected Product Edition and assigned end-user
- **Select Product Edition** - choose the desired Product Edition (see chapter *4.7* for more information on Product Editions)
- **Specify End-User** - select this box if you want to assign the formatted CRYPTO-BOX to a certain end-user (see chapter *4.10* for more information on end-user management). This allows you to identify the end-user later during Remote Update. If you have not specified any end-users earlier, this field will be grayed out.

- **CRYPTO-BOX Serialnumber** - displays the serial number of the currently attached CRYPTO-BOX.
- **Forced overwrite mode** - will delete all existing partitions from the CRYPTO-BOX, except the project specific ones.

Click the “Format Next” button to start formatting the first CRYPTO-BOX. A message box shows the status of the formatting process. After all CRYPTO-BOX units have been formatted, click “Stop Formatting” to close the window.

> If you get an error message during formatting that Administrator login to the CRYPTO-BOX has failed, check if you specified the correct CRYPTO-BOX hardware profile (TRX file) in your project. Please refer to the description in the "General Project Settings" chapter, depending on your selected project type.



Information about formatted CRYPTO-BOXes®

Serial Number	Date	Product Edition	User
0x00ef237d	Mittwoch, 1. Juli 2015	Standard Network Edition	Alex J Johnson
0x00ceb1a8	Mittwoch, 1. Juli 2015	Advanced Network Edition	Paula Walsh

Select Product Edition: Advanced Network Edition

☑ Specify End-User: Paula Walsh

CRYPTO-BOX Serialnumber: 0x00ceb1a8

```
          Login as User
          - OK
          Writing Private AES Key
          - OK
          Adding (updating) partitions:
          Partition #200
          LCS Partition
          Product Editions Partition
          - OK
          Opening partition #200
          - OK
          Data Object: Network License
          Data Object: Run Counter (CDO)
CRYPTO-BOX No 2 has been successfully formatted!
```

□ Forced overwrite mode

Formatting 3 of 10

Fig. 4.14:
CRYPTO-BOX Format: "Select Project" dialog

> The command line tool SmrxProg allows you to automate CRYPTO-BOX formatting. See chapter *4.8* for more information.

After you have protected your applications, documents or media files and formatted the CRYPTO-BOX, you can ship it together with supplemental files (drivers, network server for network licensing if applicable) to your end-users. MARX provides an easy-to-use redistribution setup. See chapter *8* for more details.

> Do not change the project settings after you have formatted CRYPTO-BOX units for your end-users. If you need to update licensing information we recommend that you use Remote Update (RUMS).

4.9.3. Creating Remote Update Utility

If you want to allow updates of your end-user’s CRYPTO-BOX later, you need to generate the Remote Update Tool. This program encapsulates the project and CRYPTO-BOX specific data and can be distributed to the end-user together with the CRYPTO-BOX. Click the “Generate” button on the CB Format main screen (See Figure 4.20) and select the path and filename the Remote Update Utility should be extracted to. A detailed description of how to update licenses remotely on the End-User side can be found in chapter *6.1*.

The Remote Update functionality is available as an option. See www.marx.com → Shop → Solutions → RUMS for more information.

4.10. End-User Management

With the Smarx OS Application Framework, you can assign CRYPTO-BOX units which have been formatted with corresponding project settings, such as AutoCrypt, Protection with API or Document Protection, to particular end-users. This assignment can be done during CRYPTO-BOX Format (see chapter *4.9*) if the end-users are already defined in the database. Click the “End-Users” button on the SxAF main screen to add end-user descriptions to the database.

There is no need to fill in all the fields for every end-user. For user selection in CRYPTO-BOX Format and for identification during Remote Update, filling in the name fields is sufficient.

If you want to take customer information and licensing information from your own, customized database instead of using the License Management Database available in SxAF, you can implement CRYPTO-BOX configuration into your own system by using command line based tools. See chapter *7* for more information.

5. Network License Management

5.1. Introduction

Network licensing is ideal for cost-effective software licensing in (corporate) networks. The software vendor determines how often the application is allowed to run in a network - with just one CRYPTO-BOX per network.

Furthermore, it allows software licensing not only for PCs and laptops, but also for environments without the possibility to connect a dongle directly:

- Mobile devices (Tablets, Smartphones)
- IoT devices
- Virtual machines (Windows/Citrix Terminal Server)

Please refer to our White Paper “Network Licensing” for more details.

6. Updating Licenses Remotely

6.1. Remote Update Management System - RUMS

The Remote Update Management System (RUMS) provides a convenient way to perform remote updates of licensing data objects programmed inside the CRYPTO-BOX. It can be used with:

- AutoCrypt (automatic protection, see chapter *4.4*);
- Implementation with API (API based software protection, see chapter *4.5*);
- Document/Media Protection

without any programming efforts.

> See the RUMS Application Notes for a detailed description of all options offered by RUMS. The Remote Update functionality is available as an option (one-time fee). See www.marx.com → Shop → Solutions → RUMS – Remote Update for more information and pricing.

6.2. Online License Management - OLM

6.2.1. Introduction

The Online License Management technology based on Smarx Cloud Security (WEB API) provides automated online update for application/document/media licenses (set of data objects, programmed to application specific CRYPTO-BOX partitions). OLM is used in cooperation with the Smarx OS Application Framework which provides generation of license update scripts for all types of supported projects:

- AutoCrypt (automatic software protection);
- Implementation with API (API based software protection);
- Document/Media Protection.

Compared to RUMS, which generates an activation code for every individual remote update transaction according to update requests obtained from the end-user side, OLM uses remote update plans generated once and deployed on web server. All end-user update requests are processed automatically on server side.

The OLM technology helps MARX customers to automate remote updates and significantly decrease human efforts spent at software vendor’s side.

6.2.2. How Does it Work ?

Online License Management is based on Smarx Cloud Security (Server knows PIN scenario) and contains client-side components (used to access the CRYPTO-BOX) and server-side components (to be deployed on web server). OLM can be considered as a higher abstract layer built on top of Smarx Cloud Security (WEB API).

OLM can be used only with CRYPTO-BOX XS or Versa firmware 2.2 (or higher) or CRYPTO-BOX SC, and it requires a an ASP.NET web server where OLM server side components and license update scripts will be deployed and integrated into customer specific web environment.

The client-side component is the same as used for any Smarx Cloud Security scenarios: it is implemented as a browser plug-in for Internet Explorer, Firefox, Chrome, Safari and Opera. The component provides access to the CRYPTO-BOX from within client's Web browser. License update commands are generated on the server side, encrypted and MIME-encoded, embedded into HTML/JavaScript page, sent to the Web browser, decrypted and executed by the client component. Commands execution results are encrypted, MIME-encoded and sent back to the server.

OLM is a solution on top of Smarx Cloud Security (WEB API). For more details on Smarx Cloud Security basics and documentation, please refer to the "Smarx Cloud Security (WEB API)" section in the Smarx Control Center. In contrast to OLM, WEB API requires manual implementation of update mechanism, but is more flexible in system requirements (also supports PHP and JSP technology). More details can be found below.

6.2.3. Client-Side Requirements

- Hardware: Smarx OS formatted CRYPTO-BOX (firmware 2.2 or higher);
- Driver: CRYPTO-BOX driver must be installed;
- Client component:
  - SMRXWEB COM-object for Microsoft Internet Explorer;
  - npWebSec plug-in for Firefox, Chrome, Safari or Opera;
- Operation System/Browser:
  - Windows: Microsoft Internet Explorer, Firefox; up-to-date versions of Chrome, Safari or Opera
  - Linux: Firefox

MARX provides an easy-to-use setup for installing all required components (CRYPTO-BOX drivers and browser plugin) on the end-user's computer. See chapter *8.4* for more information.

6.2.4. Server-Side Requirements

- Application: Customized Web application with OLM module (ASP.NET 2.0);
- Web Server with ASP.NET 2.x support
  - OS: Windows
  - IIS 6.0 or higher
- **WEB API (without OLM)** supports Web Servers with PHP (5.04 or higher) as well as JSP (Java JDK 1.4.2 or higher), too.
  - OS: Windows, Linux, FreeBSD or any other PHP/JSP enabled system

  - Apache, IIS, PWS, …

6.2.5. License Update Scripts Generation

The Smarx OS Application Framework generates the script files, which are necessary for automated license updates. These files are generated in XML-format and could be of two types:

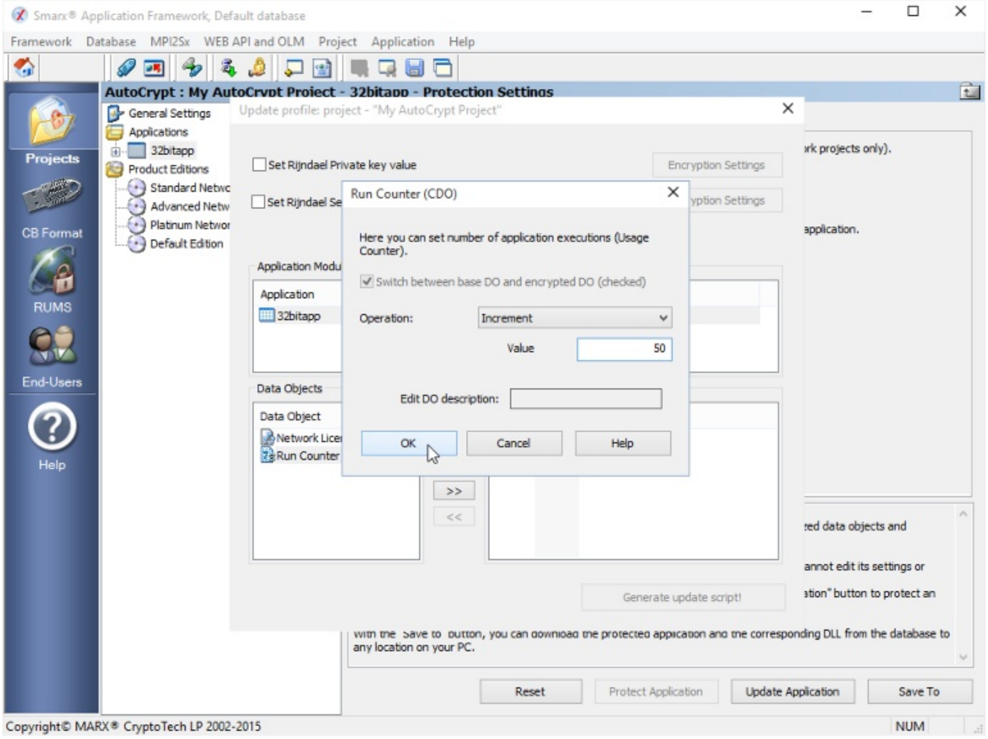
- CRYPTO-BOX access codes scripts
- License update plans scripts

CRYPTO-BOX access codes (UPW, APW, Fixed AES Key/IV, Client Public RSA Key, Distributor Private RSA Key) are needed to perform client-server handshake authentication, CRYPTO-BOX verification and read/write access to partitions and data objects.

To generate the access codes script, launch the Smarx OS Application Framework (SxAF), open desired project and select menu item “Export access codes” in the “OLM” menu. The required access code data will be taken from the TRX-file assigned to the chosen project. Select the folder where you want to store the XML file with access codes and click the "Save" button. The generated file has to be deployed on the Web Server, to be used with OLM application. Usually, only one access codes script file is needed because it corresponds to the CRYPTO-BOX profile, which is unique for every end user.

License update plans scripts contain lists of license check conditions (performed before update) and license update operations to be done. It is possible to generate one or more license update plans for each project in the Smarx OS Application Framework. In other words, each project could have several license update plans, depending on licensing and distribution policy.

To generate the license update access codes script, launch the Smarx OS Application Framework, open desired project and select menu item “Generate update script for OLM” in the “WEB API and OLM” menu. Select an application (in case of AutoCrypt) or partition (in case of Implementation with API) at the top of the window, then select the data object(s) to be updated (changed) from the left part of the window and click the “>” button to add the data object to the update sequence (right window). A new window opens which allows you to change data object settings.



Run Counter (CDO)

Here you can set number of application executions (Usage Counter).

☑ Switch between base DO and encrypted DO (checked)

Operation: Increment

Value 50

Edit DO description:

OK Cancel Help

With the "Save to" button, you can download the protected application and the corresponding DLL from the database to any location on your PC.

Fig. 6.1:
Generate OLM license update script dialog

Repeat the steps above for other applications or data objects and click on the “Generate update script!” button. Generated files (one or more) have to be deployed on web server, to be used with OLM application. Each license update script file corresponds to a unique remote update plan for selected project.

6.2.6. OLM Evaluation Demo

The OLM demo is implemented as MS VS2005 project and contains the following files:

\OLM-Demo.sln	solution file
\OLM\Login.asp	the startup page of the online demo
\OLM\Upload.aspx*	license update script upload page. This page applies to the evaluation demo only; in the customer’s solution a set of license update plans will be predefined
\OLM\CheckLicense.aspx	CRYPTO-BOX validation and license check script generation page
\OLM\LicenseStatus.aspx	license check results processing and submit ticket form generation page
\OLM\ UpdateLicense.aspx	ticket validation and license update script generation page
\OLM\UpdateStatus.aspx	the last page of the online demo, displays the results of license update

\OLM\App_Code\*.*	customizable interface classes (Constants,cs JSTag.cs, OLM.cs)
\OLM\bin\WebLM.dll	OLM module implemented as .NET DLL, which encapsulates internal functionality
\OLM\data\access.xml	demo CRYPTO-BOX access codes script
\OLM\pics\*.*	images used in the online demo

MARX Online License Management scenario consists of several steps, demonstrated in the evaluation demo.

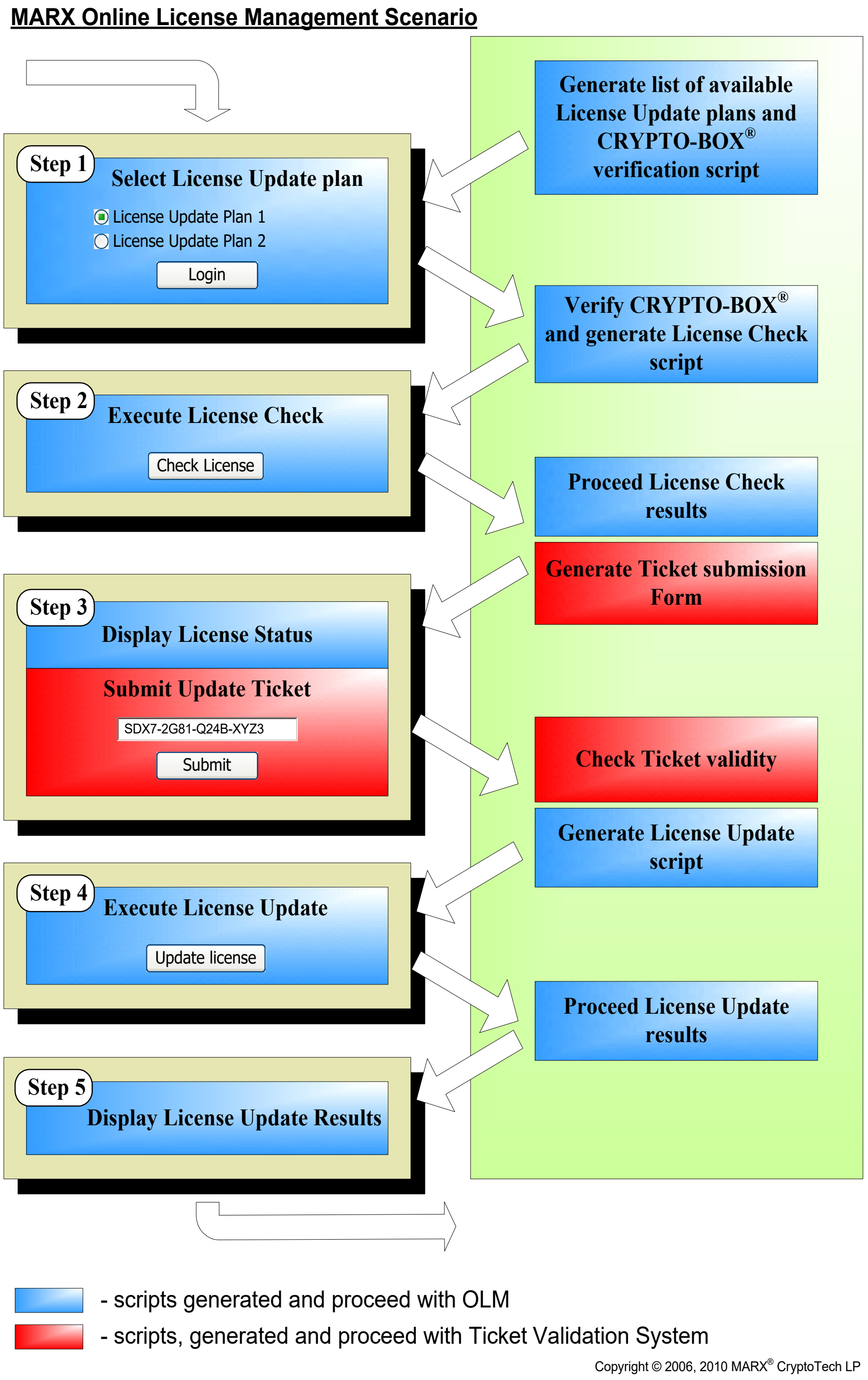

MARX Online License Management Scenario
Step 1
Select License Update plan
● License Update Plan 1
○ License Update Plan 2
Login
Generate list of available License Update plans and CRYPTO-BOX® verification script
Verify CRYPTO-BOX® and generate License Check script
Step 2
Execute License Check
Check License
Proceed License Check results
Generate Ticket submission Form
Step 3
Display License Status
Submit Update Ticket
SDX7-2G81-Q24B-XYZ3
Submit
Check Ticket validity
Generate License Update script
Step 4
Execute License Update
Update license
Proceed License Update results
Step 5
Display License Update Results
🟦 - scripts generated and proceed with OLM
🟥 - scripts, generated and proceed with Ticket Validation System
Copyright © 2006, 2010 MARX® CryptoTech LP

Fig. 6.2:
Generate OLM license update script dialog

Step 1.

At this time, end-user selects a license update plan to be applied from the list of license update plans available (previously uploaded to the server). A set of license update plans can be associated with each customer’s product. For evaluation purposes, it is possible to upload license update scripts, generated in SmarxOS Application Framework. Thus, OLM demo can be evaluated by MARX customers with a demo CRYPTO-BOX. In the customized OLM solution, license update plans will be predefined by the customer, generated and deployed on customer’s Web server.
After license update plan has been selected, end-user clicks on login button for client-server handshake and CRYPTO-BOX verification. The results of CRYPTO-BOX verification and selected license update plan are sent to server-side. OLM server module verifies CRYPTO-BOX and generates license check scripts.

Step 2.

At this time, license check scripts are executed. License check should be performed for each application module (partition) to be updated. In simple cases, partition existence and its memory size are checked to allow operations with data objects. For more sophisticated licensing strategies, product edition (special data object value) check may be required.
The license check results are sent to server-side. OLM server module proceeds with license check results and generates form for update of ticket submission.

Step 3.

At this time, end-user submits the unique ticket value received from software vendor after payment confirmation. The ticket is sent to the server site, where license update and validation are demonstrated in this demo and are a part of customer's web solution, separate from OLM functionality.

Step 4.

At this time, license update scripts are executed. License update is performed for each application module (partition) listed in update script. The results of license update are sent to server-side. OLM server module proceeds with license update results and outputs them on the next page.

Step 5.

At this time, license update results are displayed.

7. Command Line Utilities

7.1. Introduction

In addition to the Smarx OS Application Framework (SxAF), which allows you to set up your protection strategy and configure your CRYPTO-BOX modules via an easy-to-use graphical interface, the Protection Kit contains a set of command line based tools duplicating functionality of SxAF components. The advantage of these tools is that they can be controlled from within other applications and therefore can be easily integrated into your own, specific distribution strategy. They allow a high grade of automation when protecting applications, configuring CRYPTO-BOX units individually for each customer, or performing remote updates.

> All available command line utilities can be found in the "Drivers & Tools" section of the Protection Kit Control Center.
> AutoCrypt and SmrxProg are available for Linux and macOS, too. Please refer to corresponding SmrxOS4Linux or SmarxOS4Mac packages at www.marx.com → Support → Downloads. Some settings in the XML file are different for Linux and macOS, please refer to the included readme files for details.

7.2. AutoCrypt - Command Line Version

The command line version of AutoCrypt, AC_Tool.exe, allows you to protect Windows applications and DLLs. The AC_Tool can be called from within your application or batch file. AC_Tool.exe can be found in the folder *<SmarxOS PPK root>\Tools\AC_Tool*.
AC_Tool is used in combination with SmrxProg, which performs CRYPTO-BOX programming. See chapter *7.4* for more information on SmrxProg.

Parameter description:
AC_Tool.exe <TRX file> <XML file>
where:

<TRX file> TRX file provided to you by MARX with your customer specific CRYPTO-BOX (cbu_demo.trx for demo CRYPTO-BOX shipped with the Evaluation Kit)

<XML file> XML file with application protection settings and CRYPTO-BOX configuration also used by SmrxProg for further CRYPTO-BOX programming (see AC_Test.xml as an example)
Also initial XML script file can be generated for the active SxAF project (menu “Project”->”Generate Script for SmrxTools”). Even if not using SxAF for protection and license management it still can be useful for XML prototype file creation.

Short explanation on how to use AC_Tool:

1. Take an XML file that was generated in Smarx OS Application Framework (see chapter *4.8*). Or use a text editor to customize XML data (AC_Test.xml, AC_Local.xml or AC_Network.xml may be used as prototypes)
2. Place the TRX file (you received it with your CRYPTO-BOXes), the XML file created on the previous step and the AC_Tool.exe file into the same directory.
3. Run the following command from the console: AC_Tool.exe <TRX file> <XML file>
4. Resulting output will be displayed on the console and also saved to AC_TOOL.LOG file.

Have a look at the readme.txt file in AC_Tool folder for detailed information and return code description.

7.3. Document Protection - Command Line Version

Doc_Tool is a command line utility for automatic conversion and protection of documents. It provides similar functionalities as its GUI-based counterpart in the Smarx OS Application Framework. It can be found in the folder *<SmarxOS PPK root>\Tools\Doc_Tool*.

Besides protection of documents so that they can be viewed only if a valid CRYPTO-BOX is attached to the computer, Doc_Tool provides the following options:

- License management - "Expiration Date" utilizing DataObjects;
- Customized dialogs (license agreement, expiration error).

Documents are aggregated in Document Groups which hold license options.

> To convert documents, "MARX PDF Converter" printer must be installed on the computer. This is done automatically during Protection Kit installation.

Doc_Tool is used in combination with SmrxProg, which performs further CRYPTO-BOX programming.

Parameter description:

Doc_Tool.exe <TRX file> <XML file>
where:

<TRX file> TRX file is provided by MARX with your customer specific CRYPTO-BOX (cbu_demo.trx for CRYPTO-BOX shipped with the Evaluation Kit)

<XML file> XML file with application protection settings and CRYPTO-BOX configuration also used by SmrxProg for further CRYPTO-BOX programming (see Doc_Tool.xml as example)

Short explanation on how to use Doc_Tool:

1. Take an XML file that was generated in Smarx OS Application Framework (see chapter *4.8*). Or use a text editor to customize XML data (Doc_Tool.xml may be used as prototypes).
   XML-file format notes:
   Leaving SOURCE_PATH or TARGET_PATH of GROUP empty tells the system to construct them from corresponding PROJECT fields adding group's NAME (except group with FLAG_IS_ROOT=1 which inherits project fields "as is");
   If EXPIRATION_DATE is set to UNLIMITED - license will never expire.
   All documents placed within certain group inherit its expiration date.
   Document NAME can be set in the form of a mask (like *.pdf) - describing bunch of files.
   If FLAG_IS_CONVERTED is set to 1, document is considered to be converted (thus needs no conversion)
2. Place TRX file, XML file obtained in the previous step and Doc_Tool.exe into the same directory.
3. Run command line: Doc_Tool.exe <TRX-file> <XML-file>
4. Results will be displayed on console and output to Doc_Tool.LOG file

Have a look at the readme.txt file in Doc_Tool folder for more information (including return codes).

7.4. SmrxProg - Command Line Based CRYPTO-BOX Formatting

SmrxProg is a command line utility for CRYPTO-BOX formatting (programming) through command line switches. Duplicating CRYPTO-BOX Format (GUI-based component of CRYPTO-BOX formatting in Smarx OS Application Framework) functionality, SmrxProg can be efficiently used for customer specific scenarios of CRYPTO-BOX programming.

SmrxProg supports:

- (re)programming CRYPTO-BOX® Label;
- creating partitions in CRYPTO-BOX memory (supports partition numbers from 101 to 65535);
- programming Data Objects and network licenses to particular partitions.
- executing extended partitions operations, like update, delete etc. (see "Extended script format" section in the SmrxProg readme.txt for details).
- (re)programming of CRYPTO-BOX encryption keys (Private/Session AES Key/IV);
- (re)programming User Password (UPW) (see "Extended script format" section in the readme.txt for details).

SmrxProg and its readme file can be found in the folder:

<SmarxOS PPK root>\Tools\SmrxProg

Parameter description:

SmrxProg.exe <TRX file> <INI file>
or
SmrxProg.exe <TRX file> <XML file>

where:

<TRX file> TRX file provided to you by MARX distributor with your customer specific CRYPTO-BOX hardware (cbu_demo.trx for demo CRYPTO-BOX shipped with the Evaluation Kit)

<INI-file> INI file with CRYPTO-BOX configuration (see Test.ini as example)

<XML file> XML file with application protection settings and CRYPTO-BOX configuration also used by SmrxProg for further CRYPTO-BOX programming (see AC_Test.xml as example)
Also a prototype XML file can created with SxAF for the active project (menu “Project”->”Generate Script for SmrxTools”). Even if not using SxAF for protection and license management this option can still be useful for automatic creation of the prototype XML script.
Another option is to use Partition Editor (PE) utility for extended XML (script) file generation.

Short explanation on how to use SmrxProg:

1. Take an XML file that was generated in Smarx OS Application Framework (see chapter 4.8). Or use a text editor to customize XML file (the .xml sample files in SmrxProg folder can be used as prototypes).
2. Place the TRX file distributed by MARX, the XML file obtained on the previous step, and SmrxProg.exe in the same directory.
3. Run the following command from the console:
   SmrxProg.exe <TRX file> <XML file>
4. Results will be displayed on the console and directed to the SMRXPROG.LOG file.

Have a look at the readme.txt file in SmrxProg folder for more information (including return code and script format description.

7.5. RU_Tool - Command Line Utility for Remote Update Management

7.5.1. Overview

RU_Tool.exe allows you to perform remote update of the CRYPTO-BOX hardware. Duplicating RUMS (GUI-based component to perform CRYPTO-BOX updates in Smarx OS Application Framework, see chapter 6.1) functionality, RU_Tool can be integrated to your application to automate customer specific scenarios of remote update. RU_Tool can be used in combination with SmrxProg.exe, Doc_Tool.exe and AC_Tool.exe (see chapter *7.2 - 7.4*).

See the RUMS Application Notes, chapter 3 for a detailed description of RU_Tool. The Remote Update functionality is available as an option (one-time fee). See www.marx.com → Shop → Solutions → RUMS – Remote Update for more information and pricing.

8. Distributing Your Software

8.1. Installing CRYPTO-BOX Support on the Target System

After selecting the protection and licensing strategy best suited to your needs, protecting your software and configuring the CRYPTO-BOX hardware, it is time to send everything to the end user. At this point, it is important to make sure that the target system is configured properly to work with the CRYPTO-BOX.

> Please refer to the “Driver Installation” Application Notes for for information on installing CRYPTO-BOX support under Windows.
> Under Linux and macOS, there are no CRYPTO-BOX drivers required. Please see instructions in the readme files of the corresponding “Smarx OS 4 Linux” and “Smarx OS 4 Mac” packages available at www.marx.com → Support → Downloads for further details on configuring CRYPTO-BOX access under these systems.

8.2. CRYPTO-BOX Network Server Installation

The CRYPTO-BOX Network Server (CBIOS Network Server) is available for different platforms:

- Windows 32 and 64 bit versions (Windows XP and up)
- Linux 32 and 64 bit (x86/amd64/armhf/arm64)
- macOS

> Please refer to the “Network Licensing” White Paper for details on installing and configuring the Network Server on your target system.

8.3. Document Protection PDF Viewer Installation

The PDF Viewer is intended for end-users to access documents protected with Document Protection, a solution to secure the distribution and sharing of digital documents (see chapter *4.6* for more details). MARX provides a setup package which can be shipped together with the protected documents. It installs the PDF Viewer application and the CRYPTO-BOX device drivers on the computer, plus sets file association for .PPD files to the PDF-Viewer. The setup supports Windows 32 and 64 bit platforms (Windows XP and up).

> The latest version of the PDF Viewer Setup can be found on our website: www.marx.com → Support → Downloads → Driver and Diagnostic Tools.

8.4. Smarx Cloud Security and OLM Client Component

Smarx Cloud Security (WEB API) and Online License Management (OLM) ensure that only authenticated users with a valid CRYPTO-BOX will be able to access a web portal or other web-based online distribution solution. They also allow automated license updates via the Internet. More details can be found in chapter *6.2*.

In order to access the CRYPTO-BOX attached to the client's computer with the WEB API server component, it is required to install the WEB API client component on this computer.

The setup installs the CRYPTO-BOX drivers for Windows and the WEB API client component, depending on the installed browser type.

The latest version of the WEB API client setup can be found on our website:
www.marx.com → Support → Downloads → Network Utilities

9. Troubleshooting with MARX® Analyzer

9.1. Introduction

MARX Analyzer is a powerful tool that simplifies the process of troubleshooting for CRYPTO-BOX hardware and MARX components, providing intuitive and comprehensive diagnostics. Its clearly structured, tree-view based output enables you to identify and resolve problems quickly.

MARX Analyzer performs a number of tasks: It runs extensive operating system diagnostics, detects Smarx OS libraries, components and device drivers, tests hardware and generates detailed reports, which you can submit to our Technical Support.

9.2. Features

MARX Analyzer performs the following tasks:

- Detection of attached CRYPTO-BOX hardware.
- Analysis of installed MARX libraries, device drivers and components.
- Comprehensive network diagnostics: UDP broadcasting can be used for automated server search or server address can be specified directly.
- Identifying problems and providing instructions for resolution.
- Report generation. A report can be saved to a file or e-mailed right from the application.

MARX Analyzer supports all Windows operating systems from Windows 7 and up.

9.3. Using MARX® Analyzer

MARX Analyzer works like a Wizard guiding you step-by-step through the diagnostic process. Click the "Start Diagnostics" button on the upper right side to begin.

9.3.1. Standard or Extended Diagnostic (Hardware Profile required)

By default, MARX Analyzer will always try to detect if the CRYPTO-BOX is attached to the local USB port of the computer. It will also check LPT and COM ports for legacy CRYPTO-BOX devices.
If a CRYPTO-BOX is found on the local computer, MARX Analyzer will ask for the hardware profile (TRX file for Smarx OS formatted CRYPTO-BOX or MRX file for MPI) to perform extended diagnostics and provide more detailed information on the CRYPTO-BOX to MARX customers (not intended for end-user usage!).

The hardware profile is provided to you by MARX together with your first CRYPTO-BOX order. Never send this file to your end-users!
MARX Analyzer will not ask for the hardware profile if a CRYPTO-BOX with demo configuration (from the Evaluation Kit) is attached, or in network mode (see section 9.3.2).

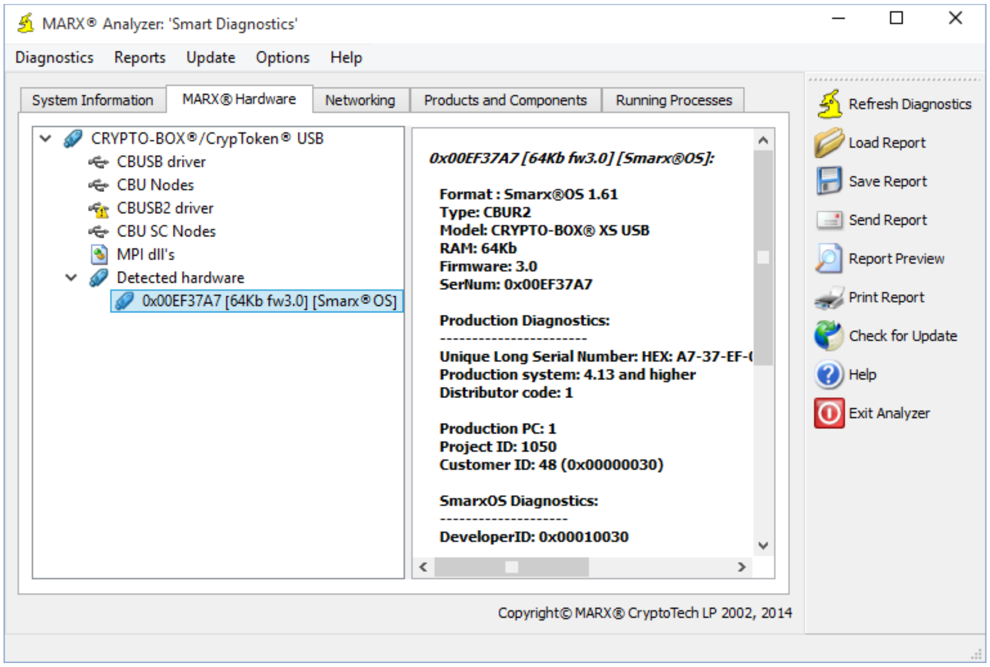
Specify the location of the hardware profile by clicking the browse button "...", then click "Rescan" to continue. Or press "Skip" if you do not have the hardware profile.

9.3.2. Network Diagnostic

In case the CRYPTO-BOX is available via network, MARX Analyzer can locate the network server where the CRYPTO-BOX is attached. If your CRYPTO-BOX is attached to the local computer and does not work in network mode, click the "Skip" button to skip the network test. Otherwise, you can either try automatic search via UDP broadcasting (works only if the server is located in the same sub-network) or specify the server name or IP address manually. After that, click the "Search" button to continue.

9.3.3. Diagnostics Results

The diagnostics process may take a while, after it you will see a screen with results.



```
0x00EF37A7 [64Kb fw3.0] [Smarx®OS]:

  Format : Smarx®OS 1.61
  Type: CBUR2
  Model: CRYPTO-BOX® XS USB
  RAM: 64Kb
  Firmware: 3.0
  SerNum: 0x00EF37A7

  Production Diagnostics:
  ------------------------
  Unique Long Serial Number: HEX: A7-37-EF-(
  Production system: 4.13 and higher
  Distributor code: 1

  Production PC: 1
  Project ID: 1050
  Customer ID: 48 (0x00000030)

  SmarxOS Diagnostics:
  --------------------
  DeveloperID: 0x00010030
```

Fig. 9.1:
MARX Analyzer diagnostics result

On top of the window, you will see different tabs:

System Information

This tab provides more details about the computer MARX Analyzer is running on (CPU, chipset, network card, USB ports, etc.).

MARX® Hardware

This tab shows information about installed CRYPTO-BOX drivers and detected CRYPTO-BOX units. If one (or more) CRYPTO-BOX was found, additional information about it will be shown when looking under the "Detected hardware" tree entry on the left side of the window.

A yellow exclamation mark on the CBUSB (for CRYPTO-BOX XS/versa) or CBUSB2 driver (for CRYPTO-BOX SC) means that this driver is installed, but not active. So when you have a CRYPTO-BOX XS or Versa, it is normal that the CBUSB2 driver is not running (see Fig. 9.1 above as example).

Networking
This tab lists network servers found by MARX Analyzer (if network test was selected). Furthermore, it will show the settings of the server, and information about attached CRYPTO-BOX units and client applications.

Products and Components
This tab displays information about MARX components found on the computer (libraries and plugins). Missed components will be shown with an exclamation mark.

Not all components are necessary; it depends on the field of application. For instance, the data filtering driver is only required if the application uses data protection technology.

Running Processes
This tab shows all processes currently running on the computer which may have influence on the communication with the CRYPTO-BOX.

9.3.4. Report Generation

The diagnostic results found by MARX Analyzer during hardware and software analysis can be saved to a file and/or sent to our technical support team for further investigation in case of difficulties with the CRYPTO-BOX or one of its software components. There are several buttons on the right side of the window for this purpose:

Refresh Diagnostics
Starts the diagnostic process again with the same settings as specified before

Load Report
Allows you to load an existing MARX Analyzer report for review purposes.

Save Report
Saves the generated diagnostic report to a file (eg. for sending it to our Technical Support).

Send Report
Allows you to send the report as an e-mail to our Technical Support. When clicking on this button, MARX Analyzer will ask you for contact information first, then it launches the installed e-mail client and attaches the report to the e-mail.

Report Preview
Displays the information which will be included in the diagnostic report.

Print Report
Creates a hard copy of the report

10. Smarx®OS API for Developers

10.1. Overview

This part of the Compendium is intended exclusively for developers. It introduces developers to Smarx OS architecture and internals as well as to the application programming interfaces (APIs) supported. In this chapter and the following ones, all Smarx OS APIs are introduced and discussed, while the latest syntax, usage details and error codes are provided in separate e-documents.

This chapter also presents a typical scenario for developers: how to start using the Smarx OS Protection Kit for their needs. Next, all supported platforms and environments are discussed, including available samples, technical notes and advices on programming.

Here is an overview about all available Smarx OS APIs for the CRYPTO-BOX system:

Smarx®OS Interface	Platform	Language	Environment
Smarx API Simple protection API with SxAF projects	Windows, Linux, macOS, Android, iOS (*)	C++ 11, C# 4.0+	MSVS 2013+, gcc 4.8+, Xcode 5+, QT 5+
CBIOS API, DO API Advanced protection API	Windows, Linux, macOS, Android, iOS	C#, F#, C/C++, Java, Delphi, VB, VBA, Swift, LabVIEW, MATLAB, VFP, Scala, DMD, IVFortran, DarkBASIC, REALbasic	MSVS 6+, Builder 6+, Delphi 5+, gcc 4+, Xcode 4+ and others
RUMS API Simple remote update API with SxAF projects	Windows, Linux, macOS, Android, iOS (*)	C/C++, Delphi	MSVS 6+, Builder 6+, Delphi 5+
RFP API Advanced remote update API	Windows, Linux	C#, C/C++, Delphi, VB	MSVS 6+, Builder 6+, Delphi 6+

* Windows platform is supported now, other platforms will be supported in future or on request.

See chapter *10.12* for a table of libraries provided in different formats for various target platforms/IDEs.

To preserve a better overview, every API providing certain functionality is placed into a separate chapter:

API	Remarks	Chapter
Smarx API	High level API for the CRYPTO-BOX which exposes a simple and user friendly programming interface to developers than other Smarx OS based APIs (CBIOS/DO API).	11
CBIOS API	Standard API for the CRYPTO-BOX SC, XS and Versa models. Includes functions for CRYPTO-BOX search and identification, access to its internal memory and encryption functions.	12
CBIOS Networking	Special subset of the CBIOS API allowing access the CRYPTO-BOX on networks and perform network licensing - defining a number of running instances of the protected application to be launched run in a network	13
DO API	Data Objects API, subset of the CBIOS API which provides a convenient way to create and access various objects for licensing purposes, such as expiration dates, counters, passwords or self-defined objects.	14
RFP API	Remote Update API to update the CRYPTO-BOX directly on the end-user side. Intended for customers who prefer advanced API integration instead of using tools provided by MARX (RUMS component in SxAF or "RU_Tool.exe" command line tool)	15
CBIOS4NET/ Smarx4NET	Implementation of all APIs mentioned above as object oriented, components based approach for .NET developers. For detailed description and Developer's Guide, see PPK Control Center, section "Implementation with API" → "API Components".	10.13.2
Extended API (XSMRX)	Provides CRYPTO-BOX formatting features for customers who prefer API integration instead of using tools provided by MARX (SxAF or "SmrxProg.exe" command line tool)	16

Further APIs (documentation provided separately):

API	Remarks
DP API	Data Protection API, allows to encrypt sensible data coming with your application (such as databases, calculations, logs, statistics, documents, video/audio, etc.) on the fly, so that they cannot be read without having proper CRYPTO-BOX attached. For detailed description and Developer's Guide, see see PPK Control Center, section "Implementation with API" → "API Components" → "e) Data Protection (DP) API".
WEB API (Smarx Cloud Security)	Authenticate users via Internet/Intranet and update the CRYPTO-BOX. Ideal for online licensing and subscription services. For detailed description and Developer's Guide, see "Cloud Security" section in the PPK Control Center.

MPI-formatted CRYPTO-BOX units cannot be accessed with Smarx OS based API functions, but can be converted in most cases. Please contact support@marx.com for more information on MPI to Smarx OS conversion.

10.2. Sharing CRYPTO-BOX® Memory Between Different Applications

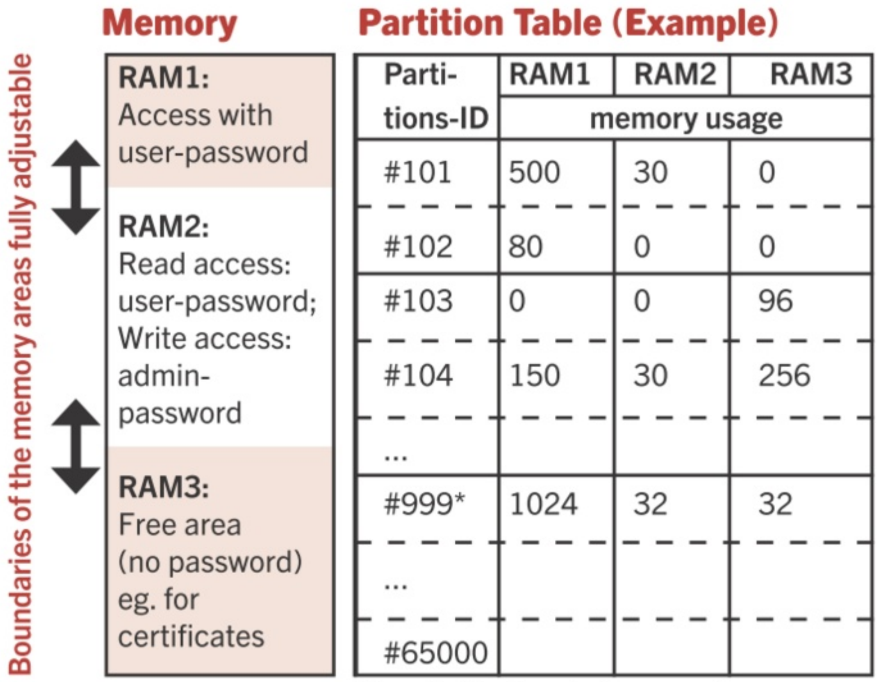
Smarx OS solves this task by providing every application with its own partition(s). Partition is a set of memory areas in the CRYPTO-BOX RAM1/2/3 memory zones, storing protection and licensing data of this application. Partition can be considered as a virtual CRYPTO-BOX allocated for this application. This eliminates memory conflicts with other applications. Every Smarx OS partition:

- is associated with an application;
- is identified through its partition number.

Every partition has associated memory allocated in one or more memory areas – depending on the protection logic required:

- RAM1 area – user password (UPW) is required for read/write access (ideal for data which might be changed on the end-user side during application rumtime, such as counters);
- RAM2 area – UPW is required for read access and APW must be provided for write access (ideal for data which will not be changed after initial CRYPTO-BOX formatting, such as customer information);
- RAM3 area – free access area.

The CRYPTO-BOX SC (CBU SC) contains 2 additional memory areas: RAM4 and RAM5. RAM4 can be used to store RSA keys for CBU SC hardware-based RSA encryption, RAM5 stores additional AES keys for hardware-based AES encryption. See chapter *10.10.2* and *10.10.3* for more information.


Memory
Partition Table (Example)
Boundaries of the memory areas fully adjustable
RAM1:
Access with user-password
RAM2:
Read access: user-password;
Write access: admin-password
RAM3:
Free area (no password) eg. for certificates

Parti-tions-ID	RAM1	RAM2	RAM3
	memory usage		
#101	500	30	0
#102	80	0	0
#103	0	0	96
#104	150	30	256
...			
#999*	1024	32	32
...			
#65000			

* Demo-Partition (for evaluation purposes)

Fig. 10.1:
CRYPTO-BOX memory allocation concept

The following table shows partitions associated with important applications and services. Partitions related to predefined components and services are printed in gray:

AppID	Name	Description	Occupied memory (by default)		
			RAM1	RAM2	RAM3
4	AutoCrypt	Automatic protection of Windows applications	200	0	0
5	LCS	License Management Table of Smarx OS License Control System: network license management for protected applications	0	240	0
6	DOC Protection	Document Protection	300	0	0
...	...	...			
10	Product Edition	Smarx OS Product Edition functionality	48	0	0
...	...	...			
98	RUMS	Remote Update Management System (RUMS) – special internal zero-sized partition	0	0	0
...	...	...			
100	User defined	Partitions with AppID 100 and larger are customer specific			

...	...	...			
999	MARX samples	This partition is used for all MARX samples and demos	1024	32	32

Create your own partitions in the CRYPTO-BOX memory to store licensing information! Partitions (and their content) can be added to the CRYPTO-BOX creating an “Implementation with API” project in the Smarx OS Application Framework (see chapter *4.5*) or using the SmrxProg command line tool (see chapter *4.8*). These tools will adjust RAM1/2/3 sizes automatically depending on required space for the specified Data Objects. Alternatively, the XSMRXCOM API (see chapter *16*) can be used to distribute the CRYPTO-BOX memory. Later, this content can be queried via API with your preferred IDE (see chapter *10.12*).

The partition number (AppID) range for customer specific partitions can be 100 up to 65000. The maximum number of partitions per CRYPTO-BOX is 32. If this is not enough, you can share one partition with multiple applications.

The only limitation for RAM1/2/3 sizes of each partition is the total physical memory of the CRYPTO-BOX (4/32/64 KB, depending on the CRYPTO-BOX model).

10.3. Access to One CRYPTO-BOX for Different Processes/Threads

By using special OS level kernel objects Smarx OS allows concurrent processes or even threads of one process to access CRYPTO-BOX concurrently. This means that if one thread/process A is trying to access CRYPTO-BOX which is busy at the moment processing another request from a different thread/process B, then process A will wait until Smarx OS finishes the currently processing request.

This approach allows a concurrent usage of MPI and Smarx OS applications.

10.4. Caching CRYPTO-BOX Calls

To increase operating speed during concurrent access a special encrypted read cache is implemented. All data to be read from or written to the box are saved to this cache and are retrieved from cache for further read operations. The cache is implemented as a read cache only, write behind is not supported.

10.5. CRYPTO-BOX Plug In/Plug Out Notifications

Smarx OS kernel (CBIOS – CRYPTO-BOX Input/Output System) “listens” for OS hardware notifications on USB events and refreshes its internal global tables automatically. Thus CBIOS does not perform time-consuming physical box scanning, but only returns the number of currently attached boxes, when the application asks for the current MARX hardware configuration. It is also possible to receive CBIOS notifications by defining a callback function or as Windows messages.

10.6. MARX Digital Signature

MARX digital signature is a special system object residing in every Smarx OS formatted CRYPTO-BOX. It prevents unauthorized duplication of a digitally signed CRYPTO-BOX.

10.7. Establishing Secure Communication Channel, Document Submission, Remote Update

Every Smarx OS formatted CRYPTO-BOX/ CRYPTO-BOX SC contains two additional system objects: two RSA keypair for establishing secure communication channels/document exchange. In particular they are used for Remote Update (RUMS) technology and WEB API.

You as a software vendor will obtain two unique RSA key pairs from MARX. One of them defines your side, another will define the end user side: their CRYPTO-BOX. A CRYPTO-BOX that was formatted for Smarx OS and programmed within one project of one customer has the same end user side RSA key pair. The end user side RSA keys (two RSA keys stored in the CRYPTO-BOX memory) are:

- the private RSA key of the CRYPTO-BOX (Client's Private Key);
- the public RSA key of the Distributor (Distributor's Public Key).

In turn you as the software distributor will have two other RSA keys (the distributor's side RSA keys):

- your private RSA key;
- public RSA key of the CRYPTO-BOX.

Using these system objects together with Remote Update Management System or WEB API (Web Security) offers high flexibility when integrating the CRYPTO-BOX into your commercial solutions. Features such as remote authentication, remote update, and secure document submission can be easily implemented based on these objects and the CBIOS API.

10.8. Symmetric Encryption (AES/Rijndael)

All CRYPTO-BOX models for USB port include a hardware implementation of the 128 bit AES (Rijndael) symmetric encryption algorithm – OFB (output feedback) mode. Three encryption keys are embedded into the firmware: Session, Private and Fixed.

Every key contains: the encryption key itself (16 bytes value) plus the associated Initialization Vector (IV) that takes another 16 bytes.

There is no way to read the Rijndael encryption key value or associated IV (initialization vector) from the CRYPTO-BOX. It can only be used for encryption. This logic allows developers to implement strong authentication, verification, general software and/or data protection by comparing footprints, encrypting/decrypting some key data structures, etc. The Session Key (and its IV) can be used and reprogrammed by any application without password submission. The Private Key (and its IV) requires the User Password for usage and the Admin Password for reprogramming. The Fixed Key and its IV is hard coded and cannot be reprogrammed. Every MARX customer has the same values for hardware Rijndael keys initially programmed to all his CRYPTO-BOX units.

> The CRYPTO-BOX SC supports more than 3 AES key slots, as well as CBC mode. See chapter *10.10.2* for details.

10.9. Asymmetric (RSA) Encryption

MARX associates two unique RSA key pairs with every customer: The distributor's key pair and the client's key pair.

Every Smarx OS formatted CRYPTO-BOX programmed for a particular customer will contain the client's private key and the distributor's public key. The customer receives two keys (the distributor's private and the client's public key) included in the TRX file. The keys can also be provided as binary files. These two key pairs are used by Smarx OS in RFP (Remote Field Programming) and WEB API interfaces to establish and support secure communication channels. They can also be used for various customer specific "distributor <-> end user" scenarios.

There are two special CBIOS calls working with these keys:
CBIOS_EncryptInternalRSA()/CBIOS_DecryptInternalRSA()

> For the CRYPTO-BOX SC these internal RSA calls are based on hardware RSA implementation (see chapter *10.10.3*), for the CRYPTO-BOX XS and Versa a software-based implementation (on driver level) is used.

The RSA encryption for the CRYPTO-BOX XS and Versa (CBU) is implemented inside the system level software: driver and low-level library. This implementation is supported by the Smarx®OS API and used in various MARX solutions and technologies, such as: WEB API, OLM and Remote Update.

The RSA implementation in the CBU driver uses different padding rules comparing to other popular RSA implementations (e.g. OpenSSL, WinCrypt, etc.), which use PKCS#1 padding rules:

PKCS#1 Padding Rules

D: data
P: padding, (k - 3 - length of D) bytes, where k: key length in bytes

Encryption

- Public Key encryption:
  00 || 02 || P || 00 || D
  P: pseudorandom, nonzero bytes

- Private Key encryption:
  00 || 01 || P || 00 || D
  P: all bytes have value 0xFF

Decryption

- Public Key decryption (message was encrypted with private key):

- after decryption checks for block starting with 00 || 01
- checks for 0xFF padding bytes
- gets beginning of data from 0x00 separator

- Private Key decryption (message was encrypted with public key):

- after decryption checks for block starting with 00 || 02
- gets beginning of data from 0x00 separator

While the actual (pure) RSA operations are just Encryption and Decryption, however there are four RSA operations after taking these standard padding rules into account:

RSA Public Encryption
RSA Private Encryption
RSA Public Decryption
RSA Private Decryption

Padding Rules Used in CBU RSA Implementation

The RSA implementation in the CBU driver uses a slightly different RSA padding:

D: data
L: length of DATA
P: padding, (k - 4 - length of D) bytes, where k: key length in bytes

- Encryption:
00 || 01 || L || P || 00 || D
P: all bytes have value 0xFF

- CBU Decryption

- after decryption checks for block starting with 00 || 01
- gets L
- checks for 0xFF padding bytes
- checks for 0x00 separator

As a consequence of this approach the CBU RSA implementation is not compatible to standard RSA implementation conforming to the PKCS#1 padding rules. However if a crypto library has API calls for the plain RSA operations without any padding, they can be used to implement CBU compatible padding.

For the CRYPTO-BOX XS and Versa (CBU) MARX provides RSA implementation sources with CBU compatible padding for such popular environments as Java, PHP, and C#.NET as a part of the Smarx Cloud Security (WEB API) solution. These sources are based on standard RSA implementation adjusted for CBU RSA padding rules:

- included to GMP.PHP and BCMath.PHP
- provided by: http://www.bouncycastle.org for Java and C#.

> The CRYPTO-BOX SC supports additional RSA key slots, as well as standard PKCS#1 padding rules. See chapter *10.10.3* for details.
> The Smarx OS API also includes pure software based RSA for both padding modes (identical to CBU SC hardware implementation) - it allows to consider CBU XS and Versa models as RSA key storage for solutions with entry level security requirements.

10.10. CRYPTO-BOX®SC Specific Functions

10.10.1. Compatibility of CRYPTO-BOX XS/Versa and CRYPTO-BOX SC

The CRYPTO-BOX SC (CBU SC) is fully supported by Smarx OS. Being 100% backward compatible with the CRYPTO-BOX XS and Versa (CBU XS/Versa), the CBU SC can simply replace it for all existing applications and solutions where the CBU XS or Versa models are currently used. Even this simple replacement will mean: ultra-fast CBU SC with 32KB of RAM. Plus hardware based RSA implementation, which automatically increases security for Remote Updates of licenses. The Smarx Cloud Security, OLM (Online License Management) and RU (Remote Update) solutions are built on top of RSA.

However 100% backward compatibility should be considered as an entry level only. Besides CBU XS/Versa compatibility, the CBU SC brings a lot of brand new features and options for Smarx OS customers, in particular:

10.10.2. CRYPTO-BOX SC AES Encryption Extension

While the CRYPTO-BOX XS and Versa (CBU XS/Versa) models include only three dedicated AES keys (Fixed, Private and Session, see chapter *10.8*), the CRYPTO-BOX SC (CBU SC) additionally contains a special new memory zone (RAM5) which can securely hold as many extra AES keys as required (only limited by the memory size). Moreover, besides OFB (output feedback) mode of AES encryption supported by CBU XS and Versa, the CBU SC allows you to choose between OFB and CBC (cipher block chaining) encryption modes for AES keys stored in RAM5 memory zone.

The RAM5 memory zone only allows you to write the key values and use them for encryption; there is no way to read the key values. During creation of the keys in this zone, it is possible to specify access rights for these keys: User Password (UPW) or Admin Password (APW) is required to use this key for encryption or to change its value. Furthermore, it is possible to lock the key (for instance, in case a cracking attempt was detected).

This new functionality is used by AutoCrypt to protect more than one application with one CRYPTO-BOX SC using separate encryption. In case of API based protection developers are able to use as many AES keys as necessary by the product licensing logic. Also such MARX technologies as Document Protection, Data Protection, Media Protection and Web based communication benefit from this functionality, providing higher level of security and customization.

To help developers incorporating the extended AES functionality to their programs the DO API includes a special data object type - "AES key"- TEOSDO_AES.

10.10.3. Using Hardware Based RSA of the CRYPTO-BOX SC

The CBU SC includes RSA implementation in its firmware. It contains a special memory zone (RAM4) storing RSA keypairs. There is no way to read key values; they can only be used for encryption. Special access restrictions can be added: APW/UPW login requirements for encryption and/or changing key values.

The CRYPTO-BOX XS and Versa (CBU XS/Versa) uses non-standard padding rules for RSA encryption (see chapter 10.9 for details). For compatibility purposes the CBU SC firmware RSA implementation uses the same padding rules as the CBU XS/Versa. In addition it also includes standard PKCS#1 padding rules, so developers can integrate CBU SC hardware based RSA to a wide range of standard and customer specific security solutions. The API related calls include a special parameter (dwMode) reserved for defining padding rules: CBU / PKCS#1 modes.

OpenSSL or any other open RSA implementation (for example, see www.bouncycastle.org) can be used on the server (trusted side), while CBU SC based RSA will be used by clients. Besides, an extended number of RSA key-pairs securely stored in RAM4 allow developers to use them also for software protection and licensing needs.

To help developers incorporating this new RSA functionality to their programs the following improvements are introduced in the Smarx OS API:

- In addition to currently supported driver level RSA implementation the Smarx OS API also supports hardware based RSA for the CRYPTO-BOX SC
- A special encryption mode fully compatible with PKCS#1 padding rules is supported in addition to the existing RSA support with non-standard padding mode
- CBIOS starting from version 1.5 also includes software based RSA for both padding modes (identical to CBU SC hardware implementation) - it allows to consider CBU XS and Versa models as RSA key storage for solutions with entry level security requirements
- A new data object - "RSA keypair" (TEOSDO_RSA) is introduced in DO API helping developers with RSA integration

10.11. Smarx®OS API: Local and Network Modes

The Smarx OS API covers two scenarios:

- Local access: The CRYPTO-BOX is attached to the local computer (i. e. the computer on which the application runs).

- Network access: The CRYPTO-BOX is attached to a remote computer (server); the application running on the local computer (client) accesses the remote computer through the network.

Local and network access is supported for all available Smarx OS APIs (see next chapter). A special case is Smarx4NET which works solely in network mode.

For more details on network access refer to chapter *13*.

10.12. Using Smarx®OS Under Different Platforms

10.12.1. Overview

Depending on the platform (OS) and programming environment used, Smarx OS APIs (CBIOS local and network, DO, RFP) are provided in different formats, as:

- Static libraries;
- Dynamic libraries (DLL);
- .NET assembly (Managed DLL);
- COM;
- Native DLL/SO.

Static libraries are the most secure way of linkage. They are provided for most of supported programming environments under Windows, Linux and OS X platforms, including: Microsoft C/C++, Borland C Builder, Delphi environments, and GCC.

Dynamic libraries (DLLs) allow easy, but less secure linkage. DLL based implementation should be considered only if for some reasons no other options can be used (static library, COM). When using DLL try to improve the level of protection and licensing logic for your application (using hardware based encryption, keeping vital data in the CRYPTO-BOX, using parallel threads, etc.), making it difficult to emulate this logic by replacing the DLL. DLLs are provided for some environments of Windows (x86 and x64) platform where static linkage is not applicable, there is a special DLL (CBIOSVB6.DLL) for Visual Basic 6.0 environment.

For .NET developers, Smarx4NET or CBIOS4NET is the most recommended approach. It provides .NET developers with an object oriented, component based approach, simplifying integration of protection and licensing to .NET applications (see 10.13.2 for more details).

COM/ActiveX is the Windows platform specific interface standard. This interface format is universal and can be used from almost any Windows programming environment. Required Smarx OS ActiveX objects are included to CBUSetup.exe driver installation utility (see chapter 8) and are installed and properly registered together with CRYPTO-BOX driver. Native DLL/native SO are specific to Java environment (Windows and Linux correspondingly).

10.12.2. Table of available Smarx®OS Libraries

The following table summarizes al available Smarx OS libraries and provides details on their target audience and supported environments:

Smarx®OS library	Target audience	Smarx OS Interfaces	Platform	Language	Environment
SmarxCPP static library	If you develop apps in C++ 11, you can validate license with only one call using higher abstract layer Smarx API or develop your licensing model with enhanced C++ classes	Smarx, *RUMS, CBIOS, DO API	Win, Linux, macOS, *Android, *iOS	C++ 11	MSVS 2013+, gcc 6+, Xcode 9+, QT 5+
CBIOS static library	For C/C++, Delphi, Swift, COBOL, MATLAB, IVFortran developers	CBIOS, DO, RUMS API	Win, Linux, macOS, Android	C/C++, Delphi, Swift, COBOL, MATLAB,	MSVS 6+, Builder 6+, Delphi 5+, gcc 6+, Xcode 9+

				IVFortran	and others
CBIOS dynamic library	For: LabVIEW, VFP, DMD, DarkBASIC, REALbasic developers	CBIOS, DO API	Win, Linux, macOS	LabView, VFP, DMD, DarkBasic, REALbasic	*
CBIOS4NET assembly	For .NET developers See chapter *10.13.2* for details and differences between Smarx4Net and CBIOS4NET Note: Smarx API (Higher abstract layer) is implemented only for CBIOS4NET	Smarx, *RUMS, CBIOS, DO, RFP, DP API	Win x86, x64	C#, VB, C++.NET	MSVS2005+
Smarx4Net assembly		CBIOS, DO, RUMS API	Any CPU	C#, VB, C++.NET	MSVS 2013+, Mono C#
JNI CBIOS dynamic library	It is for Java, Scala developers	CBIOS, DO API	Win, Linux, macOS	Java, Scala	Java 6+ SDK, Eclipse SDK 3.7+
Smrxw COM library	Obsolete COM model, except for using it with VBA	CBIOS, DO API	Win	(Any) VBA, C#, VB, C++.NET, Delphi	*
RFP static library	RFP API allows to update the CRYPTO-BOX directly on the end-user side. In contrast to RUMS (see chapter *6.1*) it provides maximum flexibility. Available for C/C++ and Delphi.	RFP API	Win, Linux	C/C++	MSVS 6+, gcc 4+
RFP dynamic library			Win	Delphi	Delphi 6+
Smarx®OS Data Protection	If you distribute your software together with sensitive and valuable data files, you will require reliable protection not only for your app itself but also for the associated data files your app works with.	DP API	Win	C#, Delphi	MSVS 2005+, Delphi 7+

* To be implemented

10.13. Supported Environments: Windows

10.13.1. Microsoft Visual C/C++ 6.x and up

If you develop in C++ 11.0 or higher, you may consider **Smarx API**, a highly abstracted layer which makes implementation very easy (see chapter *11*)

If you need more features and full control, have a look at our object oriented **SmarxCpp** static library. For more details on SmarxCpp and sample code, see PPK Control Center → Implementation with API → Libraries/Samples

For Microsoft Visual C/C++ starting from version 6.0, Smarx®OS API is implemented as static **CBIOS.LIB** library. See chapter *12* for details.

Sample code on SmarxCpp and CBIOS implementation can be found in PPK Control Center under: Implementation with API → Libraries/Samples

10.13.2. Microsoft .NET Platform

The .NET Framework environment, intensively developed by Microsoft, provides developers with extended flexibility and portability of their projects.

The .NET Framework consists of two main parts: the common language runtime (CLR) and the class library (FCL). CLR provides common services for applications developed for the .NET platform. Applications can be written in any language supporting CLR (C++, C#, VB). CLR simplifies writing code by providing support for memory management, security management, error handling. .NET Framework class library includes a standard set of classes to help a developer with common tasks.

Managed code is code that has its execution managed by the .NET Framework CLR rather than directly by the operating system. An application written as managed code gains CLR common services.

With CBIOS4NET and Smarx4NET, MARX provides an object oriented component based API for .NET platform which combines multiple Smarx OS programming interfaces (see chapter 10.1):

Smarx4NET combines CBIOS Networking, DO and RFP programming interfaces under one roof for .NET and .NET Core developers.

CBIOS4NET combines all SmarxOS programming interfaces (including local CBIOS and DP API) under one roof for .NET developers.

Based on CBIOS4NET, MARX offers with **Smarx API** a highly abstracted layer which makes CRYPTO-BOX implementation very easy – without dealing with API function calls! Refer to chapter *11* for more details on Smarx API.

APIs for .NET – Overview:

IDE	.NET	Local & Network Mode	Platform	Requ. Redistributable **	PPK Assembly	PPK Path***	MSI / MSM (Redistributable)
MS VS 2013 +	4.5.1+	*	Any CPU	-	Smarx4Net.dll	\dotNET4.5\ Any CPU	\SMARX4NET\ SMARX4NET.msi, SMARX4NETMergeModule.msm
	.NET for Windows Store				SmarxRuntime.winmd	\WRC	-
	4.x	Yes	x86, x64 (platform specific loader	VC Redist 2013	CBIOS4NET.dll	\dotNET 4\x86\ signed, \dotNET 4\x64\ signed	\CBIOS4NET\ CBIOS4NET_x86.msi, CBIOS4NET_x86_x64.msi, CBIOS4NetMergeModule.msm

			CBIOSLoader.cs for .NET 2.0-3.5)	VC Redist 2010		\dotNET 4\Obsolete\x86, \dotNET 4\Obsolete\x64	\Obsolete\CBIOS4NET\CBIOS4NET_x86.msi, CBIOS4NET_x86_x64.msi, CBIOS4NetMergeModule.msm
	2.0 - 3.5			VC Redist 2005	CBIOS4NET.dll, CBIOS4NET64.dll	\dotNET 2\asm signed	
MS VS 2010 - 2012	4.x			VC Redist 2010	CBIOS4NET.dll	\dotNET 4\Obsolete\x86, \dotNET 4\Obsolete\x64	
MS VS 2005 - 2008	2.0-3.5			VC Redist 2005	CBIOS4NET.dll, CBIOS4NET64.dll	\dotNET 2\asm signed	

* Smarx4Net requires CBIOS Network Server for local mode
** Included to MSI/MSM
*** See [PPK root folder]\SmarxOS\API\Win\SDK

For detailed information on Smarx4NET/CBIOS4NET (including class reference), please refer to the *CBIOS4NET Developer's Guide* which is available at www.marx.com → Support → Documents → White Papers.

Differences between Smarx4NET and CBIOS4NET:

1. Both interfaces are close to each other. Some methods and type names are different.
2. Smarx4NET supports network mode only: if using on a single computer, the CBIOS Network Server must be running on localhost or on a remote computer (see chapter 5). CBIOS4NET supports both local and network modes.
3. Smarx4NET is a managed library and does not need VC Redistributables. It only requires Framework.NET 4.5 or later. CBIOS4NET supports .NET framework version 2.0 and up, too, but requires VC redistributable of the corresponding MS Visual Studio environment.
4. Smarx4NET does not requires a platform specific (x64 or x32) assembly as CBIOS4NET does
5. Smarx4NET is based on interfaces, not on classes like CBIOS4NET. Class instance creation is performed through class factory (Smarx class)
6. Smarx4NET supports quick search of servers & attached hardware for net broadcasting (see SmarxNetworkLicensing sample in PPK). Direct connect won't hang up even when server is switched off (happens on CBIOS4NET network mode)

Conclusion:

The biggest advantage of Smarx4NET compared to the older CBIOS4NET interface: it is fully managed code which makes implementation more flexible and requires no VC redistributables to be preinstalled. It supports standard C# applications as well as .NET Core applications, and is ideal for protecting multi-device applications for multi-platform desktop and mobile usage. Smarx4NET runs in network mode only, which requires an installation of the CBIOS Server (either on the same computer or in the network).

However, if you already use CBIOS4NET, support only local licensing scenario in your .NET application and you do not need multi-platform support, it makes sense to stay with CBIOS4NET.

10.13.3. Microsoft Visual Basic 6.x

Microsoft Visual Basic support is implemented as a special wrapper for the VC++ static CBIOS library. The resulting dynamic library is distributed as **CBIOSVB6.DLL**. Its interface section contains the same set of functions as described in the CBIOS API part of this Compendium (chapter 11). All functions have the same name as their C counterparts. Parameter types, however, are specific to Visual Basic.

The differences between the C and Visual Basic CBIOS APIs are as follows:

- All defined directives without parameters (error codes, etc.) are replaced by Visual Basic constants with the same names.
- All structures are replaced by VBasic types.
- VBasic byte array types are used instead of C array arguments. To submit pointer on array or structure the proper VBasic type reference is submitted (ByRef keyword)
- CBIOS_BYTEARRAY16 type is defined as byte massive(16). CBIOS_BYTEARRAY16 type is used for all password and serial number arguments and also in CBIOS_SetKeySession, CBIOS_SetIVSession, CBIOS_SetKeyPrivate and CBIOS_SetIVPrivate, CBIOS API calls for the AES/Rijndael key and initialization vector
- CBIOS_BYTEARRAY_LABEL type is defined as byte massive(CBIOS_LABEL_LEN). CBIOS_BYTEARRAY_LABEL type is used for all CRYPTO-BOX label arguments
- THandle type is defined as Pointer

The sample (available in Smarx OS PPK) is written entirely in Visual Basic and performs the same tasks as its C counterpart.

10.13.4. Borland C/C++ CBuilder 5,6, BDS 2006, RAD Studio 2007 and up

The CBIOS library for Borland C/C++ is built from the same C sources as those used for the Microsoft Visual C/C++ build. So CBIOS API for CBuilder is the same as for Microsoft Visual C/C++ but the library (OMF format) can be used in projects of corresponding CBuilder C/C++ environment. The sample included is also identical to the Microsoft Visual C/C++ version.

10.13.5. Embarcadero Delphi 5 and up

The Delphi CBIOS library is implemented as a wrapper for the CBuilder CBIOS library. It uses static linking with CBuilder-generated object files (via {$LINK} Delphi compiler directive) for improved security. The resulting library is distributed in the form of the Delphi compiled unit CBIOS.DCU. Its interface section contains the same set of functions (including extended

CBIOS API calls) as described in the CBIOS API Developer's Guide (can be found in the "Documentation" section of the Smarx OS Control Center). All functions have the same names as their C counterparts.

The parameter types, however, are specific to Delphi. The differences between the C and the Delphi CBIOS APIs is as follows:

- All defined directives without parameters (error codes, etc.) are replaced by Delphi constants with the same names.
- All defined directives with parameters are replaced by functions with the same names.
- All structures are replaced by packed records.
- Delphi pointer types are used instead of C array arguments. This is done because NULL is a valid argument value in those C functions and because array and pointer are different types in Delphi.
- TByteArr16 type is defined as array [0..15] of Byte and PTByteArr16 as pointer to TByteArr16. The PTByteArr16 type is used in CBIOS_SetKeySession, CBIOS_SetIVSession, CBIOS_SetKeyPrivate and CBIOS_SetIVPrivate, CBIOS API calls for the AES/Rijndael key and initialization vector.
- TPasswd type is defined as array [0..15] of Byte and PTPasswd as pointer to TPasswd. The PTPasswd type is used for all password arguments.
- TBoxLabel type is defined as array [0..CBIOS_LABEL_LEN-1] of Byte and PTBoxLabel as pointer to TBoxLabel. The PTBoxLabel type is used for all box label arguments.
- TSerNum type is defined as array [0..15] of Byte and PTSerNumas pointer to TSerNum. The PTSerNum type is used for all serial number arguments.
- THandle type is defined as Pointer.

Sample code is available in the Smarx OS Protection Kit (PPK). It is written entirely in Delphi and performs the same tasks as its C counterpart.

10.13.6. Java (Sun JDK 1.6 and up)

The Java CBIOS component is implemented as Java wrapper over static library CBIOS.LIB. It consists of:

- Dynamically linked modules (jnicbios.DLL),
- A collection of Java classes implementing the interface to access native methods of CBIOS.

Java classes are released as com.marx.jcbios package. This package is a wrapper using the Java Native Interface (JNI) to call native methods. Since the native programming interface for Java is part of the JDK, Java applets/applications can be developed using pure Java. If you want to make sure that your code is completely portable between Windows, Linux and OS X platforms, write your programs in Java using JCBIOS.

The Jcbios package consists of the following classes:

Jcbios.class	wrapper class containing native method calls;
Constants.class	class containing constants, error codes;
ErrorHandler	class used for error handling;
CBIOSAppInfo	auxiliary class used by Jcbios methods to pass parameters;
CBIOSBoxInfo	auxiliary class used by Jcbios methods to pass parameters;

CBIOSInteger	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPrivateKey	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPublicKey	auxiliary class used by Jcbios methods to pass parameters.

The Smarx OS solution for Java assumes that Java applet/application communicates with CRYPTO-BOX devices through JCBIOS. Using JNI allows developing Java applet/application using only pure Java.

For security reasons it is highly recommended to invoke jcbios methods via SSL.

10.13.7. Qt/MinGW

Qt is an Open Source application development framework which creates cross-platform applications under C++ (see http://qt.digia.com for more details). Access to the CRYPTO-BOX is provided via the static libCBIOS.a library which is based on the static Linux library for GCC (MinGW for Windows platform) and contains the same set of functions as described in CBIOS API part (see chapter 11). All functions have the same names as their C counterparts.

Standard sample code is available in the Smarx OS PPK.

10.14. Supported Environments: Linux

The “Smarx OS for Linux” package is available at www.marx.com/downloads (MyMARX login and valid Support Level Option required).

Smarx OS for Linux doesn't require a special USB kernel driver for the CRYPTO-BOX.

10.14.1. Installing CRYPTO-BOX Support Under Linux

Smarx OS support is provided for i386/amd64/armhf/arm64 Linux platform. It was tested on the following Linux distributions (as at December 2021, but should also work with recent Linux distributions – just contact us in case of any compatibility issues):

- Ubuntu 18.04 and 20.04, 64bit
- OpenSUSE Leap 15.2, 64bit
- Debian 10, 32bit
- Raspberry Pi OS 5.10, 32 and 64bit (tested on Raspberry Pi 3 and 4)

Accessing the CRYPTO-BOX on Linux systems requires Read-Write permissions trough libusb. By default only root has such permissions.

You can use udev rules to configure CRYPTO-BOX permissions. Just copy 10-cryptobox.rules file to /rules.d directory: (root rights required):

```
  $ sudo cp 10-cryptobox.rules /etc/udev/rules.d/
```

and reattach your CRYPTO-BOX for signalizing udev system to reload rules.

Alternatively, access rules can be set by installing packages containing them. The network_server package (containing the CBIOS Network Server for Linux) will automatically configure udev rules for CRYPTO-BOX access during installation.

The standard libusb (version 1.0) is trequired to build applications with CRYPTO-BOX support targeting Linux platforms. It can be either downloaded from its official site http://www.libusb.org and installed manually or installed using apt-get (Ubuntu), yum (Fedora) or any other package management system.

For example, in Ubuntu command line execute command:

```
$ sudo apt-get install libusb-1.0-0-dev
$ sudo apt-get install libudev-dev
```

In OpenSUSE Linux:

```
$ sudo zypper install libusb-1_0-devel
$ sudo zypper install libudev-devel
```

> If you get a "LOCK_TIMEOUT" error when accessing the CRYPTO-BOX, try this: Remove all CBIOS mutex files from /tmp. Their names start with "CBIOS_" or "SH_MEM_MUTEX". This error can occur, when CBIOS was started with root privileges and then terminated incorrectly resulting in the mutex files not being cleared.

10.14.2. GCC

The static library libcbios.a is available in /api/sdk/static folder of the Linux package.

Sample code for accessing the CRYPTO-BOX under Linux using GCC can be found in /api/samples folder.

10.14.3. Qt

Qt is an Open Source application development framework which creates cross-platform applications under C++ (see http://qt.io for details). Access to the CRYPTO-BOX is provided via the static libCBIOS.a library which is based on the static library for GCC (see 10.14.2) and contains the same set of functions as described in CBIOS API part (see chapter 11). All functions have the same names as their C counterparts.

Sample code is available in the /api/samples folder. It is written entirely in Qt and performs the same tasks as its C counterpart.

10.14.4. Java (Sun JDK 1.6)

The Java CBIOS component is implemented as a Java wrapper through the static libcbios.a library (see 10.14.2). It consists of:

- Dynamically linked module (libjnicbios.so)
- Collection of Java classes implementing an interface for accessing native CBIOS methods.

The Java classes are released as a com.marx.jcbios package. This package is a wrapper using the Java Native Interface (JNI) to call native methods. Since the native programming interface for Java is part of the JDK, developing Java applets is possible using only pure Java.

By writing programs in Java using JCBIOS you can make sure that your code is completely portable between the Windows, Linux and macOS platforms.

The Jcbios package consists of the following classes:

Jcbios.class	wrapper class containing native method calls;
Constants.class	class containing constants, error codes;
ErrorHandler	class used for error handling;
CBIOSAppInfo	auxiliary class used by Jcbios methods to pass parameters;
CBIOSBoxInfo	auxiliary class used by Jcbios methods to pass parameters;
CBIOSInteger	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPrivateKey	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPublicKey	auxiliary class used by Jcbios methods to pass parameters.

Set the LD_LIBRARY_PATH environment variable to the directory containing the libjnicbios.so library file – this way, the dynamic loader will be able to find it. If the file is in the current directory, the following command can be used:

```
export LD_LIBRARY_PATH=’.’:$LD_LIBRARY_PATH
```

10.15. Supported Environments: macOS

The “Smarx OS for Mac” package is available at www.marx.com/downloads (MyMARX login and valid Support Level Option required) or in the Protection Kit (PPK) and includes support for macOS (OS X) version 10.4 and up

CBIOS for macOS requires no special kernel driver. It uses a standard USB driver included into macOS.

10.15.1. macOS CBIOS Framework

An XCode framework sample is provided in /api/samples folder.

> The framework sample requires CBIOS framework module to be installed first. Use the unified binary module for Apple M and Intel 64 bit processors in /api/sdk/framework folder of the “Smarx OS 4 Mac” package and install it into /Library/Frameworks/ directory.

10.15.2. macOS CBIOS Static Library

The CBIOS static library is designed for developers who prefer to use static linkage for security reasons. Sample code is provided in /api/samples folder.

> The static library for Apple M and Intel 64bit processors can be found in /api/sdk/static folder. Static sample requires IOKit and CoreFoundation frameworks, which are standard for macOS.

10.15.3. Java (Sun JDK 1.6 and Higher)

The Java CBIOS component is implemented as a Java wrapper over the static libCBIOS.a library (see *10.15.2*). It consists of:

- Dynamically linked module (libjnicbios.jnilib)
- Collection of Java classes implementing an interface to access native methods of CBIOS.

The Java classes are released as a com.marx.jcbios package. This package is a wrapper using the Java Native Interface (JNI) to call native methods. Since the native programming interface for Java is part of the JDK, developing Java applets is possible using only pure Java.

By writing programs in Java using JCBIOS you can make sure that your code is completely portable between the Windows, Linux and macOS platforms.

The Jcbios package consists of the following classes:

Jcbios.class	wrapper class containing native method calls;
Constants.class	class containing constants, error codes;
ErrorHandler	class used for error handling;
CBIOSAppInfo	auxiliary class used by Jcbios methods to pass parameters;
CBIOSBoxInfo	auxiliary class used by Jcbios methods to pass parameters;
CBIOSInteger	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPrivateKey	auxiliary classused by Jcbios methods to pass parameters;
CBIOSRSAPublicKey	auxiliary class used by Jcbios methods to pass parameters.

Set the LD_LIBRARY_PATH environment variable to the directory containing the libjnicbios.so library file – this way, the dynamic loader will be able to find it. If the file is in the current directory, the following command can be used:

```
export LD_LIBRARY_PATH='.':$LD_LIBRARY_PATH
```

10.15.4. Qt

Qt is an Open Source application development framework which creates cross-platform applications under C++ (see http://qt.io for more details). Access to the CRYPTO-BOX is provided via the static library libCBIOS.a which is based on the GCC library for Xcode (see *10.15.2*) and contains the same set of functions as described in CBIOS API part (see chapter 11). All functions have the same names as their C counterparts.

Sample code is available in the /api/samples folder of the Smarx OS for Mac package (see *10.15*)

10.16. Supported Environments: iOS

The Smarx OS package for iOS contains the CBIOS Network Client for iOS. The sample code written in Xcode demonstrates interaction with a remote CBIOS Server over the network from iOS devices.

Please contact us If you need iOS support.

10.17. Supported Environments: Android

The Smarx OS package for Android contains libraries and a sample application demonstrating how to access the CRYPTO-BOX under Android in network or local mode. In network mode, it allows to query a CRYPTO-BOX which is connected to a remote CBIOS Server. For local access, a customized implementation of the USB stack based on libusb library is used. Android SDK and Eclipse IDE are required.

Samples for both network and local mode are available on request. Please contact us for libraries and source code.

11. Smarx®API – High Level API for Developers

11.1. Overview – What is Smarx®API?

Smarx API is a high level API for software developers looking for manual integration of customer specific protection and licensing logic into their products:

- Without spending time on deep understanding of hardware specific CBIOS, DO and RFP APIs (see chapter *10.1* for an overview of all available APIs).
- Without increasing risks of adding new bugs to the product caused by potential technical issues related to incorrect usage of the above mentioned APIs
- Offering a better work-flow for integration into programming environments, like LabVIEW, VBA, etc.

Smarx API introduces a higher abstract layer allowing developer with one call to:

- Validate the whole license and/or this or that atomic licensing data object including both local and network scenarios (see chapter *10.11* for details)
- Add event notifications and customer specific processing related to MARX hardware events to the program (see *10.5)*
- Remote Update (RUMS) – generate request for updates license update / execute activation code

Smarx API exposes simple and user friendly programming interface to developers, significantly reducing (almost excluding) chances of bugs and logical errors caused by adding protection and licensing logic to the program. At the same time, it provides developer with more flexibility on protection and licensing logic comparing to AutoCrypt or AutoCrypt API based approach.

Starting with PPK 8.0, the Smarx Application Framework (SxAF) fully supports Smarx API for API based projects by generating necessary XML based files – parameters for Smarx API calls.

Currently the PPK includes Smarx API support for:

- MS C++ as static library (VS 2013+, SmarxCPP.lib)
- .NET (4.0+, CBIOS4NET.dll)

11.2. Smarx®API License File and License ID

License File is a key notion of Smarx API. It contains encrypted licensing data for its one call validation. Besides information on where and how to search for customer specific CRYPTO-BOX hardware the License File also includes description of atomic licenses included to the product license and their **License IDs**. This allows the application to access and examine values of individual atomic licenses if required.

Starting with version 8 the SxAF can generate Smarx API License File together with License Map File – list of License IDs to be used by developers to refer to this or that atomic license in

Smarx API calls if needed.

Definitions:
License ID - string license identifier, to be optionally used as a parameter of Smarx API calls, allowing to access the atomic licenses - Data Objects (DO from the DO Map).
License File – encrypted XML file to be used as a parameter for Smarx API calls. It contains complete information on product license.

Steps for generating Smarx API license file from SxAF API based project (see chapter *4.5* for details on creating API projects with SxAF):

1. Select “Project” menu, then “Create Smarx API license” option
2. In the “Create Smarx API license” dialog choose required partition(s)
3. Press "Export" button and specify Smarx API license file name
4. Smarx API license file and Smarx API license map file will be generated

11.3. SmarxLicense class and its common methods

The SmarxLicense class represents Smarx API with the following methods:

- **void LoadFile**(string licensePath) - load the license file
- **void LoadString**(string license) - Load the license content from string
- **void Validate**() - Validate the license (all atomic licenses)
- **void Validate**(string licenseID) - Validate atomic license with specified license ID
- object **GetValue**(string licenseID) - Get value of atomic license with specified license ID

If the method causes an error, then the **SmarxException** exception will be thrown

The **CryptoboxEvent** event allows to subscribe to CRYPTO-BOX attach/detach events.

11.3.1. C# Implementation

Typical scenario:

```
SmarxLicense smarxLicense; // creates SmarxLicense class
try
{
   smarxLicense.LoadFile('C:\Users\All Users\Documents\mylicense.xml');
   // load mylicense.xml license

   smarxLicense.Validate(); // Validate mylicense.xml license

   license1 = smarxLicense.GetValue("License01"); // Get license “License01”

   // Subscribe to CRYPTO-BOX attach/detach events
   smarxLicense.Notify += (CBIOSNotifyEventArgs e) =>
   {
   LicenseValidate();// Validate license file again in case of CRYPTO-BOX
                    // attach/detach events, please see our SDK samples
   }

}
catch(SmarxException e)
{
   //....
}
```

11.3.2. C++ Implementation

Please see description of Protection::ISmarxLicense class methods in the Smarx.hpp (see [Smarx OS PPK root]/SmarxOS-SDK/MSVC/Include).

A typical scenario:

```
auto smarxLicense = SmarxRuntime::Smarx::CreateSmarxLicense();
                                 // creates ISmarxLicense class
try
{
   smarxLicense->LoadFile("C:\\Users\\All Users\\Documents\\mylicense.xml");
                                 // load mylicense.xml license

   smarxLicense->Validate();     // Validate mylicense.xml license

auto license1 = smarxLicense->GetValue("License01");
                          // Get license "License01"

// Subscribe to CRYPTO-BOX attach/detach events
   *smarxLicense += [&](const SmarxRuntime::CBIOS::CryptoboxEventInfo& e)
   {
        LicenseValidate();
                    // Validate license file again in case of CRYPTO-BOX
                    // attach/detach events, please see our SDK samples
   }

}
catch (const SmarxRuntime::LicenseException& e)
{
// …
// auto exceptions = e.GetPriorityExceptions(); // The prioritized exceptions
}
catch(const SmarxRuntime::SmarxException& e)
{
// …
}
```

11.4. Smarx®API: Quick Evaluation Scenario

1. Make sure that you have Smarx OS PPK Version 8.1 or higher
2. Launch the SxAF and create a new API based project or use one of the existing API projects (see chapter *4.5* for details on creating API projects with SxAF)
3. Create some partition, add a couple of Crypto Data Objects and assign some valid values to them
4. Generate license file: Go to Project → Create Smarx API license

5. Export newly created partition to the list of export and press "Export" button
6. Two XML files will be exported: The license itself and a map file containing License IDs of the Crypto Data Objects
7. Format a CRYPTO-BOX for this project (see chapter *4.9* for details)
8. Launch the sample ([Smarx OS PPK root]/SmarxOS-Samples/SmarxAPI/):
   a) C# (C#/.../bin/Release/SmarxAcSample.exe), and select newly generated license file by pressing the "Validate" button
   b) C++ (C++/.../Release/SmarxCppACSample.exe <license file path>), run it using Command Prompt
9. Make sure that it works
10. Now do more testing by modifying the project (network-based scenario, other data objects, changing values, etc.). Don’t forget to generate the license file and program the CRYPTO-BOX every time before testing
11. Have a look at the source code of the sample to see how simple license validation is with Smarx APIs.

12. Smarx®OS CBIOS API

12.1. Overview

This chapter describes the basics of the CRYPTO-BOX API for developers: CBIOS API. It provides functions for CRYPTO-BOX identification and access to the internal memory and encryption functions of a connected CRYPTO-BOX. To realize an effective protection scheme with the CRYPTO-BOX, CBIOS is always required while all other Smarx OS APIs provide extended functionality on top of it (see chapter *10.1* for an overview of all available APIs).

Both the CRYPTO-BOX XS and Versa models (CBU XS/Versa) and the CRYPTO-BOX SC (CBU SC) are supported by CBIOS API. The CBU SC is 100% compatible with the CBU XS/Versa, so applications written for those models work with the CBU SC (for applications using a very old CBIOS library a recompilation with the recent CBIOS version may be required – CBU SC is supported starting with CBIOS version 1.5). CBIOS commands specific to the CRYPTO-BOX SC only have "CBU2" as a Prefix. These functions provide hardware-based AES/Rijndael encryption/decryption in CBC mode, hardware-based RSA encryption/decryption, and key management (see chapter *10.10* for more details).

For a detailed description of all CBIOS API calls, check out www.marx.com → Support → Documents → White Papers which provides Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET). For Python developers, API reference is available at [Smarx PPK root]/SmarxOS-SDK/Python

12.2. CBIOS API Main Calls (cbios.h)

12.2.1. Smarx OS System Brackets

To start CBIOS Engine the CBIOS_Startup() function is used. It initializes the CBIOS API for this application. Calling this function is not necessary, though, as it will be launched automatically by the first CBIOS_ScanBoxes() call.

So, the first function usually to be called by the application is CBIOS_ScanBoxes(). This function returns the number of attached CRYPTO-BOX units.

Although it suffices to call this function once (for one application), it should be called after each change to the hardware configuration (i.e. after removing or attaching a CRYPTO-BOX).

The last CBIOS function to be called by the application is CBIOS_Finish(). This function tells Smarx OS that the application does not need CBIOS API anymore and that all internal system objects and buffers can be released. Failing to call this function may result in memory leaks.

So, every application should start its Smarx OS communication with CBIOS_ScanBoxes(). And CBIOS_Finish() is considered to be the final “Good bye, Smarx OS!”

CBIOS_Startup() and CBIOS_Finish() can be called more than once in a program. However, it is necessary to have the number of CBIOS_Startup() calls - explicit or implicit, through CBIOS_ScanBoxes() - equal the number of CBIOS_Finish() calls. Only the first CBIOS_Startup() initializes a CBIOS instance and allocates the resources required, and only the last CBIOS_Finish() releases the CBIOS resources. This logic allows developers to use nested CBIOS_Startup/CBIOS_Finish brackets, which can be helpful for external libraries using CBIOS.

12.2.2. Using CBIOS from within DLL

If you need to use CBIOS from within a DLL it is strongly recommended to create two special functions in your DLL, like: Begin_CBIOS_Support() and End_CBIOS_Support() calling CBIOS_Startup() and CBIOS_Finish() correspondingly. The first one must be called before any other CBIOS related DLL function call, while the second must be the last one.

If for some reason it is not acceptable, then the following "special" DLL implementation of CBIOS system brackets must be used:

```
//***********************************
BOOL APIENTRY DllMain (
       HANDLE hModule
       , DWORD ul_reason_for_call
       , LPVOID lpReserved )
{
       switch (ul_reason_for_call) {
             case DLL_PROCESS_ATTACH:
             // Important!!! Pass hModule into CBIOS_StartupDLLEx.
             // If none then this call will be equivalent to CBIOS_Startup ().
             // i.e. memory leaks.
                    CBIOS_StartupDLLEx (hModule);
                    break;
             case DLL_PROCESS_DETACH:
                    CBIOS_FinishDLL();
                    break;
       }
       return TRUE;
}
//**********************************
```

Frankly speaking it is not a good idea to do it from within DLLMain, because thread and window creation is not supported at that point. That’s why special non-standard functions are provided, bypassing standard Windows limitations – CBIOS_StartupDLLEx() / CBIOS_FinishDLL() used in the above sample of DLLMain entry point.

12.3. CRYPTO-BOX Plug In/Out Notifications

As mentioned in chapter *10.5*, the new CBIOS (Smarx OS kernel) processes USB Plug & Play event notifications and automatically refreshes its internal hardware configuration table. Thus CBIOS will always have an up-to-date hardware table, providing information immediately on CBIOS_ScanBoxes() calls, instead of doing a time-consuming configuration scan on each request. This improvement also dramatically increases the data caching efficiency, because CBIOS internal cache is not released on each CBIOS_ScanBoxes() call.

CBIOS allows applications to be notified on CRYPTO-BOX hardware plug/unplug events by registering their callback function. There are also special calls allowing you to retrieve notifications as messages sent to specified windows. The application can register more than one notification handler. This is especially useful when developing static libraries or DLLs.

Apart from the status change notification, the handler also receives more detailed information:

- whether a CRYPTO-BOX was attached or detached;
- global notification ID;
- user specific information (defined during handler registration).

The CBIOS_RegisterNotificationCallback() is used to register such notifications. In order to get notifications as window messages use: CBIOS_RegisterNotificationMessage().

One application can register several notification handlers. This can be useful, if the application contains more than one component and requires hardware change status notifications.

12.4. Getting Information About Attached Hardware

If CBIOS_ScanBoxes() found one or more CRYPTO-BOX units attached, then it may make sense to get additional information about them. This information is important in order to decide which of the attached CRYPTO-BOX units will be opened.

That can be done through one of the following CBIOS calls:

CBIOS_GetBoxInfoAdvI()	Returns full information on the CRYPTO-BOX.
CBIOS_GetBoxInfoI()	Returns limited information on the CRYPTO-BOX (obsolete, use CBIOS_GetBoxInfoAdvI() instead).
CBIOS_GetDeveloperIDI()	Returns Developer ID of the CRYPTO-BOX (unique for every MARX customer, all CRYPTO-BOXes of this customer).
CBIOS_GetSerialNumI()	Returns CRYPTO-BOX Serialnumber.
CBIOS_GetAppInfoI()	Helps to figure out whether or not this application has a partition in this CRYPTO-BOX.

All these functions refer to CRYPTO-BOXes by Index: From 1 to <Box Quantity> value, returned by the preceding CBIOS_ScanBoxes() call.

The CBIOS calls mentioned above provide information about any one of the attached CRYPTO-BOXes. After one CRYPTO-BOX is open (see section *12.5*) use CBIOS calls: **CBIOS_GetBoxInfoAdv(), CBIOS_GetDeveloperID().** They provide the same information about currently open CRYPTO-BOX (Index is not submitted).

The following C sequence illustrates the typical startup of a CBIOS session:

```
DWORD dwRet;
INT32 iNumBoxes;
CBIOS_BOX_INFO_ADV BoxInfoAdv;
iNumBoxes = CBIOS_ScanBoxes();
```

```
if (iNumBoxes == 0) { /* MARX hardware not found */ }
else { // some MARX hardware found
// get extended info about hardware found (if required)
dwRet = CBIOS_GetBoxInfoAdvI(1, &BoxInfoAdv); // the first CRYPTO-BOX found
...
}
// open required CRYPTO-BOX
...
```

12.5. Opening the CRYPTO-BOX®

Smarx®OS provides more than one way to open a CRYPTO-BOX® in order to establish a special session between the application (process-thread) and the selected CRYPTO-BOX.

CBIOS_OpenByIndex() opens a CRYPTO-BOX by using the Index value (from 1 to <Box Quantity> value) that was returned by the preceding CBIOS_ScanBoxes call.

The Open by Index approach is recommended for obtaining information on currently attached boxes when trying to find the one that is needed. Using this function only makes sense, if application logic assumes only one CRYPTO-BOX to be attached at a time. However, application logic should not rely on the returned Index number because this number may change after hardware is added or removed. If you know the name of the CRYPTO-BOX (dwBoxName) you want to open, use the CBIOS_OpenByName() instead.

While opening the CRYPTO-BOX, CBIOS does not perform any calls, it only saves the information that the particular CRYPTO-BOX was opened to internal tables (global). To check if the CRYPTO-BOX is really attached, call CBIOS_CheckBox().

To identify the established session, CBIOS uses ThreadID and ProcessID. This makes sure that only one CRYPTO-BOX can be opened in one thread at the same time. To open another CRYPTO-BOX you need to close the first one using CBIOS_Close().

It is important to mention here, that - when opening by Name, Label, App (Application) - the first CRYPTO-BOX found that fills the criteria specified is addressed. Here are some typical CRYPTO-BOX open call samples (C Syntax):

```
   ...
   dwRet = CBIOS_OpenByIndex(1, 999);  // will open sample partition of the 1st
                                       // CRYPTO-BOX found
   if (dwRet == 0) {
        BoxInfo.dwStructSize = sizeof(BoxInfo);
        dwRet       = CBIOS_GetBoxInfoAdvI(1, &BoxInfo);
        boxname     = BoxInfo.dwBoxName;
        boxDevID    = BoxInfo.dwDeveloperID;  // Developer ID
        ...
   }
   dwRet = CBIOS_Close();
   ...
   dwRet = CBIOS_OpenByLabel(DEMO_LABEL, 0);  // will open CRYPTO-BOX labeled
                                              // “DEMO_LABEL”
   // 0 partition (Smarx OS system) will be used
   ...
   dwRet = CBIOS_Close();
   ...
   dwRet = CBIOS_OpenByApp(999);       // will open the 1st found CRYPTO-BOX
                                       // containing 999 partition
   ...
   dwRet = CBIOS_Close();
```

```
...
dwRet = CBIOS_OpenByName(boxname,0);// will open CRYPTO-BOX with
                                    // submitted boxname
                                    // partition#0 (Smarx OS system) will be
                                    //used
...
dwRet = CBIOS_Close();
...
// BoxInfo.dwDeveloperID can be examined for some predefined value
// to determine if this box belongs to distributor's boxes or not
```

12.6. Accessing CRYPTO-BOX® Partitions

All applications, including predefined applications, solutions and services (provided by MARX), as well as customer specific ones, are associated with one or more partitions allocated in CRYPTO-BOX memory.

> Refer to chapter *10.2* for an introduction into CRYPTO-BOX partitions – it is important to understand this concept!

See chapter *12.9.2* for information on UPW/APW login which is required before accessing or writing to restricted RAM areas (RAM1/2) in partitions. Details on reading/writing to partitions/RAM areas with the CBIOS API can be found chapter *12.9.4*. Furthermore, the DO (DataObjects) API provides a very flexible concept of managing data objects with licensing information in the CRYPTO-BOX memory. See chapter *14* for further details.

12.7. Sharing CRYPTO-BOX Between Different Applications, Lock/Unlock Logic

When accessing a CRYPTO-BOX from within more than one application/process/thread concurrently, it is important to understand the underlying CBIOS logic.

CBIOS API supports concurrent sessions. When used properly, it can help block unwanted access and allow parallel processing where it is required.

Let us start with basic statements:

1. Every CBIOS call addressed to a CRYPTO-BOX locks this CRYPTO-BOX for the time of its execution. Other requests received by the CRYPTO-BOX during this period, will wait in the queue.
2. Every queued CBIOS call is waiting for some predefined time interval (timeout) and will return a CBIOS_ERR_LOCK_TIMEOUT error code, if it does not get access within this interval.
3. By default the CBIOS timeout value is set to 20 seconds.
4. Read/write data calls sometimes may take longer (for CRYPTO-BOX XS with firmware <3.0 and 32/64 KB memory, CRYPTO-BOX units starting with Firmware 3.0 are much faster). See chapter 12.9.4 for important information on reading the CRYPTO-BOX memory.
5. Some CBIOS calls may lock the CRYPTO-BOX for arbitrary intervals.

Particular Calls and Situations:

CBIOS_OpenBy###() ... CBIOS_Close()

CBIOS_OpenBy###() calls define which Smarx OS partition will be used by the application. After a successful CBIOS_OpenBy###() neither the CRYPTO-BOX, nor the opened partition is locked by the calling application. This means that other applications may still access the box and the partition.

CBIOS_LockLicense() ... CBIOS_ReleaseLicense()

The CBIOS_LockLicense() call allows you to get exclusive access to the open partition, locking it for any other application. After this call no other application can open the partition using CBIOS_OpenBy###(), any attempts to open it will receive a CBIOS_ERR_NO_FREE_LICENCE error code. The exclusive access mode will be canceled only after the application that uses the partition exclusively issues CBIOS_ReleaseLicense() or closes the session through CBIOS_Close().

CBIOS_LockLicense()/CBIOS_ReleaseLicense() can be used to prevent several copies of the protected application being launched through Terminal Server (see *12.9* for more details).

CBIOS_LockLicense() locks only the currently opened partition. Other partitions on the CRYPTO-BOX are still available for other applications/processes; the same is true of hardware encryption functionality.

CBIOS_UPWLogin()/CBIOS_APW_Login() ... CBIOS_Logout()

When one application logs into the CRYPTO-BOX by submitting a UPW/APW password (CBIOS_UPWLogin or CBIOS_APWLogin), the CRYPTO-BOX is locked by this application. No other application can access this CRYPTO-BOX until CBIOS_Logout() is issued. This way all functionality of the CRYPTO-BOX is blocked for other applications: Not only access to partitions, but also hardware encryption, random generation, etc. Only the application that is logged in can perform calls, including memory read/write (for open partition), encryption, etc.

> ⚠ In local mode (CRYPTO-BOX is attached to the local USB port of the computer) it is important to put all API calls which require UPW/APW login into Login/Logout brackets to make sure that the CRYPTO-BOX is not blocked by other applications or threads. For the network mode, the network server itself takes care of adding login/logout brackets to every CRYPTO-BOX query. See chapter *13* for more details.

CBIOS_LockBox ... CBIOS_UnlockBox

CBIOS_LockBox() allows you to get exclusive access to the CRYPTO-BOX without login (requiring password submission). No other application can access the CRYPTO-BOX until CBIOS_UnlockBox() is issued. Not only access to the open partition is locked: the CRYPTO-BOX is blocked completely! This function allows you to exclude any possible influence of other processes/threads, which can be useful for critical transactions and for multi-threaded applications.

CBIOS_IsBoxLockedByOthers[I] allows you immediately to check if the CRYPTO-BOX is locked or not by other thread/process.

CBIOS_LockBoxEx is similar to CBIOS_LockBox, but it allows you to control returning time of this function in case when the CRYPTO-BOX was locked by another thread/process.

12.8. Attaching/Detaching CRYPTO-BOX

It is also important to understand CBIOS logic for asynchronous CRYPTO-BOX attachment/detachment (plug in/plug out) during communication with running applications.

When receiving a new notification from the OS that a CRYPTO-BOX was attached or detached, CBIOS refreshes its status information on currently attached CRYPTO-BOX units and sends notifications to all CBIOS applications which registered proper handlers. However, at the moment of notification some applications may be working with any of the attached CRYPTO-BOX units, while other applications may be waiting in the queue in case the required CRYPTO-BOX is busy.

In such a situation CBIOS will wait until all current (pending) CRYPTO-BOX related operations have completed. All new requests coming from the applications will be automatically put on hold. Only after all current operations concerning the CRYPTO-BOX(es) have completed, CBIOS will renew hardware-related info in its internal tables, send notifications to the applications and resume processing for all pending requests.

Thus, in case of intensive communications between applications and CRYPTO-BOX, or if one application locked one CRYPTO-BOX for a long period of time, preceding attach/detach events can result in noticeable delay. In this case, notifications to those CBIOS applications, which registered their handlers for plug-in/plug-out events, will also be delayed.

12.9. Working With the Open CRYPTO-BOX®

12.9.1. Overview

After the CRYPTO-BOX is open and the session has been established, it is possible to:

- check CRYPTO-BOX presence using CBIOS_CheckBox().
- retrieve more info on the open CRYPTO-BOX with CBIOS_GetBoxInfoAdv(), CBIOS_GetDeveloperID().
- lock/unlock the open CRYPTO-BOX using CBIOS_LockBox(), CBIOS_UnlockBox().
- login to the open CRYPTO-BOX in User/Admin modes through CBIOS_LoginUPW(), CBIOS_LoginAPW(), CBIOS_Logout().
- lock/unlock opened partition (protection against terminal sessions): CBIOS_LockLicense()/CBIOS_ReleaseLicense().
- use internal RSA keys for encryption: CBIOS_EncryptInternalRSA(), CBIOS_DecryptInternalRSA().
- set the CRYPTO-BOX Volume Label using CBIOS_SetLabel().
- communicate with the CRYPTO-BOX using a general set of API calls
- set of calls for both CRYPTO-BOX XS/Versa and CRYPTO-BOX SC:

CBIOS_GetDriverLastError()	get last driver error
CBIOS_GetHWRand()	retrieve hardware random bit stream
CBIOS_ReadRAM1()	read data from RAM1
CBIOS_WriteRAM1()	write data to RAM1
CBIOS_ReadRAM2()	read data from RAM2
CBIOS_WriteRAM2()	write data to RAM2
CBIOS_ReadRAM3()	read data from RAM3

CBIOS_WriteRAM3()	write data to RAM3
CBIOS_SetUPW()	set User Password
CBIOS_SetKeySession()	set Key for session hardware encryption/decryption
CBIOS_SetIVSession()	set Initialization Vector for session hardware encryption/decryption
CBIOS_SetKeyPrivate()	set private Key for hardware encryption/decryption
CBIOS_SetIVPrivate()	set private Initialization Vector for hardware encryption/decryption
CBIOS_CryptFixed()	encrypt/decrypt data with internal hardware algorithm using fixed Key and fixed Initialization Vector
CBIOS_CryptPrivate()	encrypt/decrypt data with internal hardware algorithm using private Key and private Initialization Vector
CBIOS_CryptSession()	encrypt/decrypt data with internal hardware algorithm using session Key and session Initialization Vector
CBIOS_EncryptRSA()	encrypt data with RSA algorithm
CBIOS_DecryptRSA()	decrypt data with RSA algorithm
CBIOS_GetKeyRSA()	retrieve private RSA Key stored in RAM for software RSA encryption
CBIOS_SetKeyRSA()	set private RSA Key stored in RAM for software RSA encryption
CBIOS_GenerateKeyPairRSA()	generate RSA Key pair and store into corresponding RAM area
CBIOS_PrepareRSAKey()	Assembles RSA key from modulus and exponent into internal format

- The following API calls cover CRYPTO-BOX SC specific AES functionality (see chapter *10.10* for more details):

CBIOS_CBU2_SetKeyAES()	write AES keys to special cells in CBU SC dedicated memory (RAM4)
CBIOS_CBU2_SetKeyInfoAES()	set access rights for AES keys in CBU SC
CBIOS_CBU2_GetKeyInfoAES()	get information about access rights for AES keys in CBU SC
CBIOS_CBU2_LockKeyAES()	lock specified AES key in CBU SC

- The following API calls cover CRYPTO-BOX SC specific RSA functionality (see chapter *10.10* for more details):

CBIOS_GenerateKeyPairRSAEx()	generate CBU SC RSA keypair in computer memory
CBIOS_CBU2_SetKeyRSA()	write RSA keypairs to special cells in CBU SC dedicated memory (RAM4)
CBIOS_CBU2_EncryptRSA()	encrypt data with RSA algorithm using CBU SC hardware-based RSA encryption/decryption
CBIOS_CBU2_DecryptRSA()	decrypt data with RSA algorithm using CBU2 hardware-based RSA encryption/decryption
CBIOS_EncryptRSAEx()	encrypt data with software based RSA encryption/decryption (CBU SC compatible)
CBIOS_DecryptRSAEx()	encrypt data with software based RSA

encryption/decryption (CBU SC compatible)

The next subsections discuss and illustrate the most typical and most important parts of the CBIOS API.

12.9.2. Logging Into a CRYPTO-BOX

CRYPTO-BOX hardware supports password protection. This means that a password must be provided to get access to protected functionality: read/write protected memory areas, perform hardware based encryption, etc. Two levels of password protection are provided: user password (UPW) and administrator password (APW).

CBIOS also allows applications to perform UPW or APW level login to a currently opened CRYPTO-BOX using CBIOS_UPWLogin()/CBIOS_APWLogin(). All subsequent CBIOS calls requiring the password can omit its value, specifying NULL instead. This logic will work until CBIOS_Logout() is issued (see the example in section "Read/write CRYPTO-BOX memory" below).

For security purposes it is recommended to use Admin level login (APW) only at distributor's production facility, limiting protected application with User (level login (UPW). Thus, even if UPW is compromised for some reason, the intruder will not be able to change RAM2 contents and other APW protected codes.

12.9.3. Protection Against Terminal Sessions

Windows Terminal Server environments (provided by Microsoft Windows Server platforms) permit more than one instance of a protected application to be launched on the server in different client sessions. In some cases, this can be used for breaking license limitations: Terminal Server environments may allow several clients to access the protected application, which was originally licensed for one user only.

Although the problem can be solved through the CBIOS network license management model or by locking a required partition indirectly, it makes sense to provide a direct solution in the CBIOS API. For this purpose, CBIOS local mode implementation contains special calls similar to those included in the network mode:

CBIOS_LockLicense()/CBIOS_ReleaseLicense(). CBIOS_LockLicense() can be called after opening the required partition with one of the CBIOS_OpenBy functions. CBIOS_LockLicense() blocks other processes (applications) or threads running on the Terminal Server from opening this partition until CBIOS_ReleaseLicense() is received.

12.9.4. Read/Write CRYPTO-BOX Memory

The CBIOS API includes read/write partition memory operations for all memory areas: RAM1/RAM2/RAM3.

> Please refer to chapter *10.2* for more detailed information on Smarx OS partitions and RAM areas – it is important to understand this concept!

The following C code fragment demonstrates CBIOS read/write calls (assumes that CRYPTO-BOX is already open – see corresponding code in chapter *12.5*):

```
// in assumption that some CRYPTO-BOX partition is already open:
dwRet = CBIOS_UPWLogin(bUPW); // UPW Login to the open partition
if (dwRet == 0) {
   ...
   dwRet = CBIOS_WriteRAM1(0, sizeof(szDemo1), &szDemo1, NULL);
   // last parameter (UPW) may not be submitted (NULL) because of UPW Login
   dwRet = CBIOS_ReadRAM1 (0, sizeof(szDemo2), &szDemo2, NULL);
   ...
   dwRet = CBIOS_WriteRAM2(0, sizeof(szDemo1), &szDemo1, bAPW);
   // last parameter (APW) must be provided to write to RAM2, not logged with APW
   dwRet = CBIOS_ReadRAM2(0, sizeof(szDemo2), &szDemo2, NULL);
   // last parameter (UPW) may not be submitted (NULL) because of UPW Login
}
CBIOS_Logout(); // end of UPW Login
…
```

⚠ For older CRYPTO-BOX XS units with firmware <3.0 and 32/64 KB memory only, read/write data calls sometimes may take longer (some seconds). The CRYPTO-BOX SC and CRYPTO-BOX XS units starting with Firmware 3.0 are much faster. However, your application should only read the data from the CRYPTO-BOX which are actually needed instead of reading one big data block. Use CB Format/SmrxProg to create multiple memory objects. Working with DataObjects Map files can help here (see chapter *14.5* and *4.5.6* for more details).

12.9.5. Using Symmetric Encryption

ℹ For an introduction into AES encryption with the CRYPTO-BOX, see chapter *10.8*.

The following C code fragment demonstrates CBIOS AES (Rijndael) encryption:

```
char bTest1[] = “DEMO STRING DEMO STRING DEMO STRING”;
char bTest2[] = “ “;
strcpy(bTest2,bTest1);
dwLen = sizeof(bTest2);
dwRet = CBIOS_CryptSession(dwLen,bTest2); //encrypt data with
Rijndael Session Key
dwLen = sizeof(bTest2);
dwRet = CBIOS_CryptSession(dwLen,bTest2); // decrypt data
...
dwRet = CBIOS_SetKeyPrivate(bAPW,bNewKEY); // change Private Key value
dwRet = CBIOS_SetIVPrivate(bAPW,bNewIV); // change its IV
dwRet = CBIOS_UPWLogin(bUPW); // Private Key requires UPW or APW login
dwRet = CBIOS_CryptPrivate(dwLen,bTest2); // encrypt data with
Rijndael Private Key
...
```

12.9.6. Asymmetric RSA Encryption

ℹ For an introduction into RSA encryption with the CRYPTO-BOX, see chapter *10.9*.

For details on RSA implementation and corresponding API calls, click the "Browse Documentation" button in the PPK Control Center which provides Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET).

Apart from pre-created RSA key pairs, MARX customers (developers) can also create their own key pairs using CBIOS_GenerateKeyPairRSA(). They may store them in the CRYPTO-BOX memory (associated partitions with required protection levels) and use them for encryption through the CBIOS_EncryptRSA(), CBIOS_DecryptRSA() functions.

Please, refer to the cbios_sample.c sample code provided (look for *RSA()* function) to figure out how to create RSA key pairs and how to use them as well as predefined values for encryption.

12.9.7. MD5 Hash Encryption

CBIOS_MD5Hash() – this extra function calculates MD5 hashes from any input sequence of data. The MD5 hash algorithm is software-implemented, so it may be called without an CRYPTO-BOX attached.

12.10. CBIOS API Description

For a detailed description of all CBIOS API calls, check out www.marx.com → Support → Documents → White Papers which provides Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET).

13. Smarx®OS Networking: CBIOS on the Network

13.1. General Issues

Smarx®OS Networking allows Smarx OS based applications to access a CRYPTO-BOX unit attached to one computer within a network.

A special program, called Smarx OS Network Server (or CBIOS Network Server), monitors remote connections to all local CRYPTO-BOX units on this computer.

Please refer to our White Paper "Network Licensing" for:

- An introduction to network licensing with the CRYPTO-BOX
- CBIOS Network Server Administration
- Typical network session scenario
- How to start with network implementation via API

13.2. Network CBIOS API Calls

For a detailed description of all CBIOS API calls, check out check out www.marx.com → Support → Documents → White Papers which provides Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET).

14. Smarx®OS DataObjects API

14.1. Concept: What is Smarx®OS DO API? Why DataObjects?

Smarx OS DataObjects API is based on top of the low-level CBIOS API (described above) and is targeted at the software, data copy protection and licensing market.

Smarx OS DataObjects API provides convenient access to various objects, like an expiration date, usage counter, password, self-defined objects, etc., stored in the CRYPTO-BOX memory partitions.

Please refer to chapter *10.2* for more detailed information on Smarx OS partition handling - it is important to understand this concept!

The Smarx OS DO API is one of the foundations for Smarx OS-based automatic protection (AutoCrypt). For API implementation, MARX provides tools for configuring the CRYPTO-BOX with pre-defined DataObject settings which can be queried via API later. Partitions (and their content) can be added to the CRYPTO-BOX creating an “Implementation with API” project in the Smarx OS Application Framework (see chapter *4.5*) or using the SmrxProg command line tool (see chapter *4.8*).

Because Smarx OS DO API calls and Data Objects are implemented on top of the CBIOS API, they are named with the special prefix “**TEOS**”, like: **TEOSDO_EXPIRATION_DATE** or **TEOS_DoCreateReference(...);**

14.2. Smarx®OS DataObject Types

For convenience, generic data object types occupy 4 (or more) bytes of memory in one of the application's partitions. The premise of Smarx OS is that every application operates exclusively with memory inside its own partition(s) to prevent data loss.

CDO (Crypto Data Objects) – a recently introduced new generation of Data Objects – preventing intruders from changing licensing data and/or from emulating the CRYPTO-BOX functionality even if the UPW value was exposed (e.g. by analyzing/disassembling the code of protected applications).

Crypto Data Objects are encrypted with the System AES key. CDO adds reliable protection against memory transparency: it is impossible to read/write CDO value from the CRYPTO-BOX memory directly. Moreover, data which were read from one CRYPTO-BOX will become invalid if being written to another CRYPTO-BOX.

You can see auxiliary functions for CDO (distr_cdo_create_signed_data, etc) in DO sample code: [Smarx OS PPK root]/SmarxOS/API/{Platform}/Samples/DO/.

Each data object has the following “physical” properties:

- offset (address) in partition memory
- memory area (RAM1 / RAM2 / RAM3)
- size

- value.

Supported DataObject Types:

API Constant	Memory Storage	Methods
TEOSDO_CDO_EXPIRATION_DATE_AND_TIME **TEOSDO_EXPIRATION_DATE_AND_TIME** **(Fixed Value)** Fixed expiration date and time, submitted in the following format: 1) as a string including date only:“19 DEC 2014” 2) as a string including date and time (24 hour format): “19 DEC 2014 23:59:59” 3) as a string including date and time stamp (as date only): “19 DEC 2014, 20 DEC 2012” 4) as a string including date and time (24 hour format) and time stamp (as date only): “19 DEC 2014 23:59:59, 20 DEC 2012” 5) as a SYSTEMTIME structure 6) as a SYSTEMTIME structure with time stamp specified as a second SYSTEMTIME structure. Current time stamp is written to the CB memory during verification to prevent time tampering by clock time reset. Alternatively time stamp can be specified along with expiration date (see options 3,4, and 6 above). Increment/Decrement methods add/subtract number of days to/from the expiration date & time value (the same way as it is done for expiration date).	8 (+12 for CDO) bytes in the application partition memory	**SET,** **GET,** **INC,** **DEC,** **VERIFY,** **UNLIMITED**
TEOSDO_EXPIRATION_DATE (Fixed Value) Fixed expiration date, submitted in the following format: “19 DEC 2014”. This data object type is obsolete and only preserved for compatibility purposes; it is recommended to use “Expiration Date & Time” instead (see above). Current time stamp is written to the memory during verification to prevent time tampering by clock time reset. Alternatively, time stamp can be specified along with expiration date in the following format: “20 DEC 2020, 20 DEC 2012”.	4 bytes in the application partition memory	**SET,** **GET,** **INC,** **DEC,** **VERIFY,** **UNLIMITED**
Expiration Date Relative (TEOSDO_NUMBER_OF_DAYS with dwParameter = TEOSDO_NUMBER_OF_DAYS_ AS_EXPIRATION_DATE 0x02**)** “Floating” expiration date – submitted as the number of days the application may be used from the first launch (after first **VERIFY** operation). Current timestamp is written in the memory during verification to prevent clock time reset.	4 (+12 for CDO) bytes in the application partition memory	**GET,** **INC,** **DEC,** **VERIFY,** **CLEAR,** **UNLIMITED**
TEOSDO_CDO_NUMBER_OF_DAYS **TEOSDO_NUMBER_OF_DAYS** Flexible expiration date, submitted as the number of	4 (+12 for CDO) bytes in the application partition.	**SET,** **GET,** **INC,**

days the application may be used. Only the days the application is actually used will count. Current time stamp is written to the CB memory when days counter is decremented to prevent clock time reset (time tampering).		**DEC,** **VERIFY,** **CLEAR,** **UNLIMITED**
TEOSDO_CDO_TIME_ALLOWED **TEOSDO_TIME_ALLOWED** Real-time expiration, submitted as a period of time (in seconds) that application usage is permitted. The counter of seconds is decremented periodically during application execution.	4 (+12 for CDO) bytes in the application partition.	**SET,** **GET,** **INC, DEC,** **VERIFY,** **CLEAR** **UNLIMITED**
TEOSDO_CDO_COUNTER **TEOSDO_COUNTER** Number of allowed application executions (runs). Run counter is decremented each time the application is launched. NOTE: automatic decrement is performed only for AutoCrypt based protection. For protection using API, run counter must be decremented each time the application is launched (using **DEC** method).	4 (+12 for CDO) bytes in the application partition.	**SET,** **GET,** **INC,** **DEC,** **CLEAR,** **UNLIMITED**
TEOSDO_CDO_MEMORY **TEOSDO_MEMORY** Customer-specific array of licensing data in CB memory supporting application specific licensing logic.	N (+12 for CDO) bytes in the application partition.	**SET,** **GET**
TEOSDO_DOUBLE_WORD Customer-specific data DWORD value – can be used for any application specific licensing data (should be considered as a subtype of TEOSDO_MEMORY).	4 bytes in the application partition.	**SET,** **GET**
TEOSDO_CDO_PSW_HASH **TEOSDO_PSW_HASH** Hash value, calculated from password string.	4 (+12 for CDO) bytes in the application partition.	**SET,** **VERIFY**
TEOSDO_CDO_APP_CS **TEOSDO_APP_CS** CRC value, calculated from the application file.	4 (+12 for CDO) bytes in the application partition.	**SET,** **VERIFY**
TEOSDO_CDO_APP_NAME_HASH **TEOSDO_APP_NAME_HASH** Hash value, calculated from the application name.	4 (+12 for CDO) bytes in the application partition.	**SET,** **VERIFY**
TEOSDO_NET_License Application Network License value. Needs Admin Password (APW) to be set/changed.	4 bytes in LCS partition	**SET,** **GET,** **INC,** **CLEAR,** **UNLIMITED**

TEOSDO_NET_License_Ex
Application Network License value along with License Sharing Rules. Needs Admin Password (APW) to be set/changed.
Data format (for *SET* and *GET* methods):
DO_NET_LICENCE_EX_DATA{
BYTE bNetLic;
BYTE bRuleId;
WORD wReserved;}
where
	4 bytes in LCS partition	**SET,**
		GET,
		INC,
		CLEAR,
		UNLIMITED

bNetLic - Network License value; bRuleId - License Sharing Rule Id.		
TEOSDO_RSA RSA encryption key implemented as Data Object. Provides DO API based access to CBU SC RSA keys. *SET* method takes pointer to *CBIOS_RSA_KEY* structure (as *pData* parameter). TEOSDO_RSA specific methods are: TEOS_DoSetKey, TEOS_DoClearKey, TEOS_DoGenerateEx, TEOS_DoEncryptRSA, TEOS_DoDecryptRSA*.	RSA key in dedicated RAM4 memory zone of CBU SC	**SET, CLEAR**
TEOSDO_RSA_DISTRIBUTOR, **TEOSDO_RSA_CLIENT** Internal RSA encryption keys (SmarxOS system objects) implemented as Data Object. Provides DO API based access to Client/Distributor RSA keys. Specific methods are: TEOS_DoEncryptRSA, TEOS_DoDecryptRSA*.	RSA keys in dedicated RAM4 memory zone for CBU SC or reserved memory in case of CBU XS/Versa	

TEOSDO_RSA_EX
Descriptor for RSA encryption key implemented as Data Object. Allows programming of key type (CBU SC RSA or Internal RSA) as well as RSA algorithm specific data (like *padding*).
Data format (for *SET* method): *DO_RSA_EX_DATA* structure or *DO_RSA_EX_DATA* + *CBIOS_RSA_KEY*.
DO_RSA_EX_DATA{
DWORD dwKeyIndex;
DWORD dwKeyMode;
DWORD dwUpdateDescr;
DWORD dwReserved;}
where
dwKeyIndex – CBU SC RSA key index, or -2 for Distributor public, or -3 for Client private;
dwKeyMode – CBIOS_RSA_MARX_PADDING (0x00) or CBIOS_RSA_RSAREF_PADDING (0x10).
TEOSDO_RSA_EX specific methods are: TEOS_DoSetKey, TEOS_DoClearKey, TEOS_DoGenerateEx, TEOS_DoEncryptPublic, TEOS_DoEncryptPrivate, TEOS_DoDecryptPublic, TEOS_DoDecryptPrivate (TEOS_DoEncryptRSA, TEOS_DoDecryptRSA are also supported)*.
	4 bytes in the application partition.	**SET, CLEAR**

TEOSDO_AES AES encryption key implemented as Data Object. Provides direct DO API based access to CBU SC AES keys. *SET* method takes pointer to *CBIOS_AES_KEY* structure (as *pData* parameter). TEOSDO_AES specific methods are: TEOS_DoSetKey, TEOS_DoClearKey, TEOS_DoCryptAES*.	AES key in dedicated RAM5 memory zone of CBU SC	**SET, CLEAR**
TEOSDO_AES_FIXED (read-only), **TEOSDO_AES_PRIVATE,**	AES keys in the RAM5 memory zone (CBU	**SET, CLEAR**

TEOSDO_AES_SESSION Internal AES encryption keys (SmarxOS system objects) implemented as Data Object type; provides direct DO API based access to system AES keys . *SET* method takes BYTE [0x20] - AES key & IV (as pData parameter). TEOSDO_AES specific methods are: TEOS_DoSetKey, TEOS_DoClearKey, TEOS_DoCryptAES*.	SC) or reserved memory (CBU XS/Versa)	

TEOSDO_AES_EX
Descriptor for AES encryption key implemented as Data Object. Allows programming of key type (CBU SC AES or Internal AES) as well as AES algorithm specific data.
Data format (for *SET* method): *DO_AES_EX_DATA* structure or *DO_AES_EX_DATA* + AES key (*CBIOS_AES_KEY* or BYTE [0x20] accordingly).
DO_AES_EX_DATA{
DWORD dwKeyIndex;
DWORD dwKeyMode;
DWORD dwUpdateDescr;
DWORD dwReserved;}
where
dwKeyIndex – CBU SC AES key index, or -2 for Fixed, or -3 for Private, or -4 for Session;
dwKeyMode – OFB (1) or CBC (2).
TEOSDO_AES_EX specific methods are: TEOS_DoSetKey, TEOS_DoClearKey, TEOS_DoEncryptAES, TEOS_DoDecryptAES (TEOS_DoCryptAES is also supported)*.
	4 bytes in the application partition.	**SET, CLEAR**

TEOSDO_SIGNATURE Combination of two RSA encryption key descriptors (“A” and “B”) used for signing routine implemented as Data Object. Data format (for *SET* method): *DO_SIGNATURE_DATA* structure or *DO_SIGNATURE_DATA* + *CBIOS_RSA_KEY* or *DO_SIGNATURE_DATA* + 2x *CBIOS_RSA_KEY*. *DO_SIGNATURE_DATA*{ DO_RSA_EX_DATA keyA; DO_RSA_EX_DATA keyB; DWORD dwHashType; DWORD dwTimeStampSize;} where keyA, keyB - RSA encryption key descriptors (keyB is optional and can be absent – for single key signing; in this case dwKeyIndex for keyB must be -1); dwHashType – hash algorithm type (0 stands for “MD5”); dwTimeStampSize – size of time stamp, used for additional signature validation (0 stands for none, default - 4). TEOSDO_SIGNATURE specific methods are:	4 bytes in the application partition.	**SET, CLEAR**

TEOS_DoSetKeys, TEOS_DoGenerateA, TEOS_DoGenerateB, TEOS_DoCalculateDigitalSignature(F), TEOS_DoValidateDigitalSignature(F)*.		
TEOSDO_CDO_BINDING **TEOSDO_BINDING** "Locking" the software to the computer on which it is running or to the network server on which the CRYPTO-BOX is attached. All hardware parameters are validated and the total vote count (sum of changes multiplied by weight – see table below) is calculated to compare it with a threshold value (default value is used for now, will be specified by developers in future). If the total vote count is greater than or equal to the threshold value then the validation is considered as failed (the local/network hardware was changed, thus violating the license). TEOSDO_BIND_GENERATE_ACTIVATION_REQUEST – generates activation request TEOSDO_BIND_ACTIVATE – activates bound system with obtained distributor's signature TEOSDO_LOCAL_IN_NET_MODE – see "Network binding support" paragraph below **The table with hardware parameters and additional information is provided below. *Note for CDO_BINDING:* **TEOS_DoClear** doesn't clear the bound status, use **TEOS_DoSet** to reset the DO status.	115 (+12 for CDO) bytes in the application partition	**SET,** **GET,** **BIND (=INC),** **VERIFY,** **CLEAR**
TEOSDO_CDO_GEOLICENSE **TEOSDO_GEOLICENSE** "Locking" the software to some location and/or civic address or its part (City/ZIP code/Post code/Country) of end-user's computer on which the protected software is running. The CRYPTO-BOX has to be attached to this computer or available on the network through CBIOS Network Server. This DO is currently based on Microsoft Location API for Windows (supported for Win8). It is possible to define all or only some geo parameters for their further validation (see DO_GEO_DATA in TEOSDO.h). In particular the software can be bound to some Geo-location + error radius and/or complete or partial civic address specifying Country, State/province, ZIP/Postal code, Street address. The exact value of each Geo-parameter included to the license can be set by distributor as static part of the license (say, this product can be used in "GA, USA" only) or the value can be obtained on end-user's side on the first program run using TEOSDO_GEO_BIND_*. It will bind protected application to location of its first run (in terms of the above example it means: this product can be used	120 (+12 for CDO) bytes in the application partition	**SET,** **GET,** **BIND (=INC),** **VERIFY,** **CLEAR**

only in <State, Country> of its first run). **NOTE:** There is no need to call **BIND** if values of required geo-parameters are set by distributor. The TEOSDO_GEO_DOUBLE_UNINIT flag for errorRadius and for altitudeError ignores GPS position. A Civic Address provider is currently not supported with Windows 8. **TEOSDO_GEO_LOCATION** – get current location For network based scenario **TEOSDO_LOCAL_IN_NET_MODE** – allows binding to the client computer which issued the request.		

***** - these methods are not supported by TEOS_DoOperation. See section *14.6* for more information on API functions.

** - Vote Weights Table for supported platforms:

Win, Linux(15+Id), Mac OS X(30+Id)		
Id	**Parameter**	**Default Vote Weights**
1	Processor	10
2	MAC address	
3	Motherboard	
4	Hard Disk Drive	
5	Video Controller	5
6	OS Product ID	
7	Sound Card	
8	User Name	
9	BIOS	4
10	Physical memory	
11	File Path	2
12	Computer Name	
13	IP	
14	Reserved	0
15		
Threshold		**20**
iOS		
46	UUID	10
47	MAC	
Threshold		**10**

Android		
48	MAC	10
49	Reserved	
Threshold		**10**

14.2.1. Network Binding Support

Two different scenarios of network binding are supported for the "Binding to local/network computer" DO. For the first one, protected software will be bound to the server computer where the CRYPTO-BOX is attached, while the second scenario (flag TEOSDO_LOCAL_IN_NET_MODE for TEOS_DoCreateReference) allows binding to the client computer which issued the request. For the second scenario, it is important to keep in mind that each client will require a separate DO for binding.

The object will be bound with the first TEOSD_DO_BIND call issued by one of the network clients running on this computer.

The first scenario should be considered for network seat based licensing model, when protected software bound to the server computer can be launched simultaneously from several clients (seats).

The second scenario is best suited for times when the CRYPTO- BOX cannot be attached directly to client's computer (for example, tablet/smart-phone) using network licensing as a way to connect to the CRYPTO-BOX remotely. In some cases, this approach can be used for several clients, but a separate Data Object has to be reserved for each client.

14.2.2. File with Hardware Binding Data

This file is created at the time of binding. The time-stamp of operation (stored in the DO) is used as its name. The file is created in *\All Users\MARX\Binding\* folder of local or server computer. Activation file is created at the same path.

14.3. Set of Data Objects

Smarx OS DO API encapsulates a dynamic set of data objects (application license). So you can define the application license as a collection of one or more data objects (set of data objects). Later the application can perform different actions on this set and data objects included to it.

A typical scenario:

```
TEOS_DoCreateReference(…);       // creates reference for DataObject #1
TEOS_DoCreateReference(…) ;      // creates reference for DataObject #2

……                  // proceed data objects

TEOS_DoDeleteReference(…);       // delete reference for DataObject #1
TEOS_DoDeleteReference(…);       // delete reference for DataObject #2
```

```
// or

TEOS_ClearReferences(…);         // clear all references
```

DataObject references can be loaded into memory and released - either before or after partition is opened and login to the CRYPTO-BOX is made.

See section *14.6* for further information on Smarx OS Data Objects API calls.

14.4. Accessing DataObjects from Applications

Smarx OS DO API is based on CBIOS API, so you need to open the proper partition and login to the CRYPTO-BOX first in order to access data objects located in CRYPTO-BOX memory. You may define data object references before or after that.

A typical scenario (all TEOS_Do calls should be considered as optional depending on application specific licensing logic):

```
CBIOS_OpenByApp(...);      // open partition
CBIOS_UPWLogin(...);       // login to CRYPTO-BOX

TEOS_DoSet(…);      // DO value initialization
….
TEOS_DoGet(…);      // Retrieving DO value
….
TEOS_DoVerify(…);   // DO verification (if implemented), checks DO value validity
…
TEOS_DoInc(…);      // DO value increase (if implemented)
….
TEOS_DoVerify(…);
….
TEOS_DoDec(…);      // DO value decrease (if implemented)
….
TEOS_DoVerify(…);
…..
TEOS_DoClear(…);    // Reverting DO value to unitialized state
…
TEOS_DoVerify(…);
….
TEOS_DoUnlimited(…);…
….
TEOS_DoVerify(…);

CBIOS_Logout(...);  // CRYPTO-BOX logout
CBIOS_Close(…);     // close partition
```

See section *14.6* for further information on Smarx OS DataObjects API calls.

14.5. Creating DataObjects Map: Import/Export

A set of Data Objects representing application license can be saved to a binary map file. This file can be later imported by another application. It makes DO programming more flexible and convenient; it can be also used for data exchange. Data Object information is exported and imported as a Data Object Map (contents of binary map file). Data Object Map manipulations are relevant for those who use API directly. Data Object Map is used, for example, in Remote Update Activation Sequence creation.

A typical scenario:

```
TEOS_DoCreateReference(…);        // create reference for DataObject #1
TEOS_DoCreateReference(…) ;       // create reference for DataObject #2

TEOS_DoSaveMap(…);                // export data objects map into memory buffer
// or
TEOS_DoSaveMapFile(…);            // export data objects map into file

TEOS_ClearReferences(…);          // clear all references

TEOS_DoLoadMap(…);                // import data objects map from memory buffer
// or
TEOS_DoLoadMapFile(…);            // import data objects map from file
……            // process data objects
TEOS_ClearReferences(…);          // clear all references
```

Data Object information can be also created in SxAF, including export of Data Object map file. See chapter *4.5.6* for more information.

14.6. Smarx®OS Data Object API Calls

For a detailed description of Data Object API calls, check out www.marx.com → Support → Documents → White Papers where Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET) are provided.

The PPK Control Center section "Implementation with API" provides more information on available DO API sample code (and RFP API sample code with DO API) for different compilers. We strongly recommend that you have a look at them before you start writing your code.

15. Smarx®OS Remote Update Technology

15.1. What is Smarx®OS Remote Update API? How Can It Be Used?

Smarx OS Remote Update API provides a convenient way of remotely updating Data Objects, stored in CRYPTO-BOX partitions. The Remote Update basic API is mainly an addendum on top of the Smarx OS DO API (see chapter *14*). Moreover it provides MARX customers/distributors with a way to change the CRYPTO-BOX geometry remotely, for example to create new partitions, delete or re-size existing partitions, and more.

Smarx OS DataObjects API provides convenient access to various objects, like expiration date, usage counter, password, memory objects (customer defined), etc., stored in CRYPTO-BOX memory partitions. Smarx OS DO API is one of the foundations for software and document protection and is used in Smarx OS Application Framework (SxAF) components like AutoCrypt (for automatic software protection), Implementation with API, and Document/ Media Protection.

Smarx OS Remote Update API is designed for updating any set of DataObjects, programmed into a CRYPTO-BOX after automatic protection or implementation with the API. Using Smarx OS Remote Update API, you may extend the expiration period of evaluation or demo versions, change limitations, and turn features on/off and even add new partition(s) with additional licensing options. In fact, there are no limitations on how to use Remote Update API you utilize it for protecting application with API: you can update everything you need in any manner you want.

Smarx OS Remote Update API can be divided into **End-user side functions** and **Software vendor side functions**. *End-user side functions* are to be executed on end-user's side (where a CRYPTO-BOX with licensing data is attached). *Software vendor side functions* are to be executed on software vendor's side (where remote update operator resides).

> Alternatively to implementing Smarx OS Remote Update API functions directly into your application SxAF (Smarx Application Framework), or Remote Update command line utility can be used (see chapter *6.1* for more details). This approach requires no programming efforts. It uses RUpdate utility on end-user side (which can be generated using SxAF or with RU_Tool command-line utility) while Software vendor side functions are provided by SxAF or RU_Tool directly.

See chapter *14* for information on Data Objects API.

15.2. Brief Description of Remote Update API

Types, RFP_Free, RFP_GetVersion functions:

RFP types:

RFP_ErrorCode	error code (DWORD)
RFP_TrMark	Remote Update transaction mark (DWORD)

RFP_PASSW	Remote Update password (16 bytes)
RFP_Request	structure, used for specifying needed request for remote update

RFP_Request structure members:

Field	Type	Description
dwSize	DWORD	structure size
dwMask	DWORD	structure mask, verifies which structure members are valid for request (Structure mask: see table below)
dwTrMark	DWORD	transaction mark (is generated inside **RFP_CreateRequest** function)
dwUserId	DWORD	end user/clientID (In some cases information about end-user is optional)
dwProjId	WORD	project ID
dwParam	DWORD	customer parameter

Project ID and User ID are used on your side to determine end user and project details; they are required for remote updates.

Structure mask may have the following values:

RFP_REQUEST_MASK_TRANS_MARK	parameter dwTrMark is valid ()
RFP_REQUEST_MASK_USER_ID	parameter dwUserId is valid
RFP_REQUEST_MASK_PROJ_ID	parameter is valid
RFP_REQUEST_MASK_PARAM	parameter is valid
RFP_REQUEST_MASK_ALL	all above parameters are valid

RFP_Free function is a common function for end-user and distributor side. It is used for releasing the memory allocated by the Remote Update library. See detailed description RFP API description (available in the "Browse Documentation" section of the PPK Control Center).

RFP_GetVersion function returns the number of the current RFP API version to pdwMajor and pdwMinor OUT parameters.

End-user side functions:

RFP_CreateRequest function generates a Transaction Mark that is written into the CRYPTO-BOX system area. The proper request is stored in memory buffer, and pointed to using **ppRequestData**. The memory buffer is allocated dynamically – the **RFP_Free** function must be called to release the allocated memory. The **dwMaskNotEncrypted** mask parameter specifies a list of request parameters that should not be encrypted.

RFP_ClearRequest clears the Transaction Mark in the CRYPTO-BOX system area.

RFP_ProceedActCode executes the Activation Code.

Advanced functions (working with parameters):

RFP_PrepareRequest creates a **Request object** and returns its pointer in phRequest out parameter.

RFP_AddRequestParam adds a parameter to the Request object. Call this function for each parameter you want to add.

RFP_MakeRequest repeats **RFP_CreateRequest** for Request object previously prepared by **RFP_PrepareRequest** and **RFP_AddRequestParam**.

RFP_RequestFree removes Request object and releases occupied memory. It must be called when the Request object is not needed anymore (after **RFP_MakeRequest**).

Software vendor side functions:

RFP_TranslateRequest translates the **RFP_Request** structure from a customer request. The **pbRSADistribPrivate** and **pbRSAUserPublic** parameters may be NULL, in this case, only those request fields not encrypted will be initialized. The proper bits will be set in **pRequest**->dwMask.

RFP_ActSeqCreate creates an Activation Sequence object and returns its pointer in the **phActSeq** parameter. Activation Sequence consists of a set of records, called Activation Sequence Records. Each record corresponds to a sequence executed for one partition in the CRYPTO-BOX memory.

RFP_ActSeqRecCreate creates one record in activation sequence.

RFP_ActSeqRecCreateEx creates one extended record in activation sequence.

RFP_ActSeqAddRec adds a record to the Activation Sequence.

RFP_ActSeqFree removes an Activation Sequence object and releases the memory occupied. It must be called when the Activation Sequence is not needed anymore (after Activation Code generation).

RFP_ActSeqRecFree must be called, if the proper record was created but not added to the Activation Sequence. If it has already been added to the Sequence, there is no need to call it as the record will be released automatically when **RFP_ActSeqAddRec** is called.

RFP_ActSeqAddStep function adds one step to the update sequence.

RFP_GenerateActCode generates an Activation Code that is stored in memory buffer. Later on, the memory buffer must be released using **RFP_Free**.

Advanced functionality (working with parameters):

RFP_LoadRequest loads a request from a binary array to the Request object

RFP_TranslateRequestFromHandle repeats **RFP_TranslateRequest** but gets a Request object instead of a binary array. If this function is supplied with RSA keys, it decrypts the data in the Request object. As the Request object is in “decrypted” state, **RFP_GetRequestParam** calls for encrypted parameters will be successful.

RFP_GetRequestParam returns the parameters of the Request to pParamData memory buffer.

RFP_RequestFree function removes the Request object and releases the memory occupied. It must be called when the Request object is not needed anymore.

15.3. How to Initiate Remote Update Request on the End-user Side?

Smarx OS Remote Update API is based on the CBIOS API. Therefore, initiating remote updates on the end-user side requires that the proper CRYPTO-BOX be found first.

A typical scenario:

```
CBIOS_ScanBoxes();          // scans for CRYPTO-BOX
CBIOS_GetBoxInfoI(…);       // obtains CRYPTO-BOX info

RFP_ClearRequest (…);       // clears previous request
RFP_CreateRequest (…);      // generates request

….                          // exports request to external file

RFP_Free(…);                // releases memory
CBIOS_Finish();

Advanced scenario:

CBIOS_ScanBoxes();          // scans for CRYPTO-BOX
CBIOS_GetBoxInfoI(…);       // obtains CRYPTO-BOX info

RFP_PrepareRequest (…);     // creates Request object
for (…)
   RFP_AddRequestParam(…); // adds parameters

RFP_ClearRequest (…);       // clears previous request
RFP_MakeRequest (…);        // generates request
RFP_RequestFree (…);        // releases request memory
….                          // exports request to external file

RFP_Free(…);                // releases memory
CBIOS_Finish();
```

After the request is generated it needs to be exported to an external file and sent to you as the software vendor/distributor. Project and (optionally) end-user information must be present in the request and may not be encrypted. Otherwise, you will fail to decrypt the request information.

For security reasons, remote update data traffic is encrypted and decrypted using two RSA key pairs (yours and the end-user's). Your public key and the end-user's private key are programmed into the CRYPTO-BOX (pre-configured by MARX). These two keys are used for encryption/decryption on the end-user side (CRYPTO-BOX must be attached). Your private key and end-user's public key are exclusively known to you. They are used for encryption/ decryption on your side.

For more details see section *15.6* and <rfp.h>.

15.4. How to Generate Remote Update Code on Software Vendor Side

After you have received the request, the information on the project (end-user) is translated, the corresponding RSA keys are obtained and the request is decrypted. Then, you create a list of updates for several data objects. Here, knowledge of the project data objects map is required. The DataObjects map contains information on several data objects inside the

CRYPTO-BOX memory partition. The DataObjects map support is provided with Smarx OS DO API (see chapter *Error: Reference source not found* for more information).

Next, you prepare a list of update records (one record for one application or partition). Each update record may contain several update steps (one step = an update action for one data object).

A typical scenario:

```
…                          // import request
{
RFP_TranslateRequest(…);   // obtain non encrypted information from request

CBIOS_PrepareRSAKey(…);    // obtain distributor private RSA key
CBIOS_PrepareRSAKey(…);    // obtain end-user public RSA key

RFP_TranslateRequest(…);   // obtain encrypted information from request
}
// or (if parameters are passed)
{
RFP_LoadRequest(…);
RFP_TranslateRequestFromHandle (…);     // obtain non encrypted information from
                                        // request for (…)
RFP_GetRequestParam(…);    // get non encrypted parameters
CBIOS_PrepareRSAKey(…);    // obtain distributor private RSA key
CBIOS_PrepareRSAKey(…);    // obtain client public RSA key

RFP_TranslateRequestFromHandle (…);     // obtain encrypted information from
                                        // request for (…)
RFP_GetRequestParam(…);    // get encrypted parameters
}

RFP_ActSeqCreate(…);       // create update sequence
RFP_ActSeqRecCreate(…);    // create update record
// or
// RFP_ActSeqRecCreateEx(…); // create advanced update record
//(update/create/upgrade/delete partition command)
RFP_ActSeqAddStep(…);      // add update step to the record
   …
RFP_ActSeqAddStep(…);      // add update step to the record
RFP_ActSeqAddRec(…);       // add update record to the sequence
// or
// RFP_ActSeqRecFree(…);   // release update record, if not included into
                           // update sequence

   RFP_GenerateActCode(…);// generate activation code
   RFP_ActSeqFree(…);      // release memory, used for update sequence

   …    // export activation code into external file

   RFP_Free(…);     // release memory
```

When the activation code is generated, it can be sent to the end-user. The update information is encrypted with RSA keys. This way it can be decrypted only, if the proper CRYPTO-BOX is present.

For more details see section *15.6* and <rfp.h>.

15.5. How to Activate Remote Update Code on End-User Side

After the end user receives the Activation Code, a Remote Update can be performed. The unique transaction mark (generated for each request) guarantees it will be a one time update and prevents any additional unauthorized activations. One Remote Update may include information for several data objects and even partition updates. It is up to you to decide how many update actions should be performed during one update sequence.

A typical scenario:

```
…                                 // import activation code
CBIOS_ScanBoxes();                // scan for CRYPTO-BOX
CBIOS_OpenByIndex(...);           // open CRYPTO-BOX
CBIOS_GetBoxInfo(…);              // obtain BoxName

RFP_ProceedActCode(…);            // perform remote update activation
```

For more details, see section *15.6* and <rfp.h>.

15.6. Remote Update API Calls

For a detailed description of RFP API calls, check out www.marx.com → Support → Documents → White Papers where Developer's Guides with API references for different development environments: C/C++/Delphi/VB and C# (Smarx4NET/CBIOS4NET) are provided.

The Control Center section “Implementation with API” → “API Components” → “4 c) Remote Update (RFP API” provides more information on available RFP API sample code for different compilers.

16. Extended Smarx®OS API Calls – CRYPTO-BOX® Reconfiguration

16.1. General Issues

Extended Smarx OS API is intended for CRYPTO-BOX initial programming / reconfiguration. It allows an initial formatting of a CRYPTO-BOX by offering functionality such as adding new partitions, resizing RAM zones as well as deletion of existing partitions.

> As an alternative to the Extended API calls, the Smarx OS Application Framework (SxAF, see chapter *4.5*) or the SmrxProg.exe command line tool (see chapter *7.4*) can be used for CRYPTO-BOX programming and configuration. In fact, the SmrxProg utility provides even more options comparing to the Extended API.
> For Linux and macOS platforms, the SmrxProg utility is the only option currently available.

> **IMPORTANT NOTE:**
> Because of security reasons, the Extended Smarx OS API as well as the other tools mentioned above for CRYPTO-BOX formatting are intended to be used for CRYPTO-BOX formatting/configuration on the software manufacturer/ distributor side only. They are not intended to be used on the end-user side!

Extended Smarx OS API is provided for those customers who are looking to use the CRYPTO-BOX reconfiguration functions in their applications. It is implemented as a ActiveX/COM object for the Windows platform. Implementation as ActiveX provides a convenient interface for customers and brings the following benefits:

- ActiveX COM interfaces are standard for all Win64/32 programming environments.
- Each Smarx OS API revision will affect only COM; there will be no need to rebuild applications based on this component.
- Adding support for new environments (upon customer requests) will take less development efforts.
- ActiveX COM component is signed by MARX (as its publisher), to be considered as a “trusted” module/component .

Here is a list of extended Smarx OS API functions, implemented in **XSMRXCOM** ActiveX:

XSMRX::Clear	clears all data in virtual CRYPTO-BOX image
XSMRX::ReadBox	opens CRYPTO-BOX, reads information and saves all extended data in virtual image
XSMRX::GetBoxInfo	gets BoxInfo structure data obtained from CRYPTO-BOX and stored in virtual image
XSMRX::GetBoxLabel	gets CRYPTO-BOX label stored in virtual CRYPTO-BOX image
XSMRX::GetAppRec	returns application(partition) data obtained from CRYPTO-BOX and stored in virtual image (by App Id)
XSMRX::GetAppRecByIndex	returns application(partition) data obtained from

	CRYPTO-BOX and stored in virtual image (by Index)
XSMRX::GetAvailableRAMSize	returns size of free memory, available in virtual image
XSMRX::SetBoxLabel	sets CRYPTO-BOX® label stored in virtual CRYPTO-BOX image
XSMRX::AddApp	adds application (partition) to the virtual image list
XSMRX::RemoveApp	removes application (partition) from the virtual image list
XSMRX::FormatBox	formats CRYPTO-BOX with data, taken from virtual image
XSMRX::SetUPW	changes User password for attached CRYPTO-BOX
XSMRX::SetAPW	changes Admin password for attached CRYPTO-BOX
XSMRX::ErrorToText	returns text description of error

16.2. CRYPTO-BOX Configuration Scenario

The main idea of extended Smarx OS functionality is to encapsulate all required CRYPTO-BOX function calls into one transaction, which will internally execute a set of steps, prepared in advance. The rest of the function calls will work with a virtual CRYPTO-BOX image, stored in memory.

The first function **XSMRX::ReadBox()** has to be called before any other calls. It will read the attached CRYPTO-BOX to obtain all data from it, create and fill up virtual CRYPTO-BOX images and place them in memory. After that, the other **XSMRX** functions can be called to get data from the virtual image and make changes.

Function **XSMRX::FormatBox()** will take all needed information from the virtual image, format the CRYPTO-BOX and create all required partitions. It will also clear the virtual image and release all occupied memory.

Function **XSMRX::Clear()** will clear the virtual image and release all memory.

Functions **XSMRX::SetAPW(), XSMRX::SetUPW()** provides the ability to change administrator (APW) and user (UPW) passwords and are similar to those used in the standard CBIOS API.

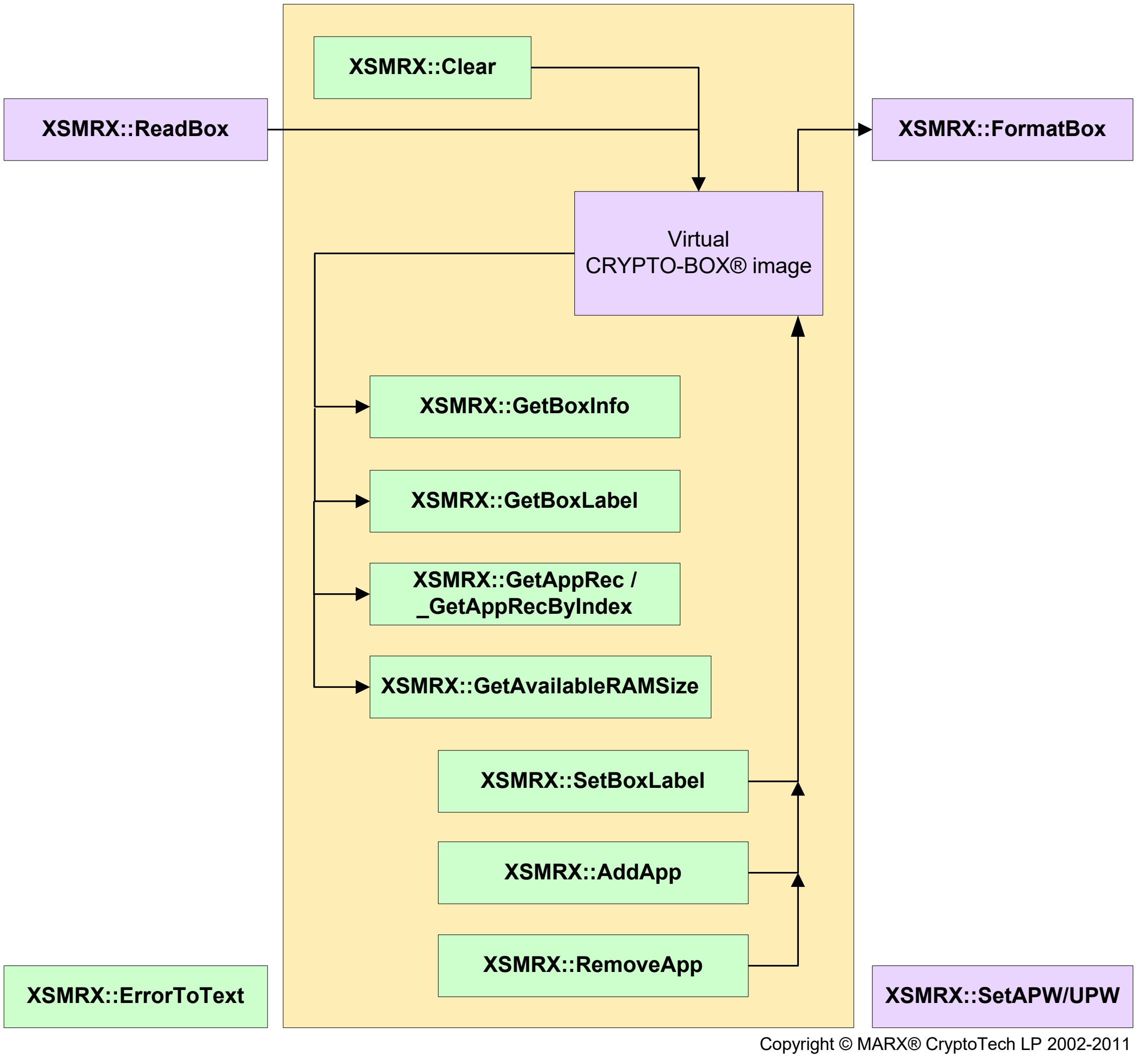


Fig. 13.1:
Extended Smarx®OS API

16.3. Extended Smarx®OS API Calls (In Detail)

public void XSMRX::Clear ()

Description: Clears all data in virtual CRYPTO-BOX image

Parameters: none

Return: none

public int XSMRX::ReadBox (int *iBoxIndex*, System.Array *bAPW)*

Description: Opens the CRYPTO-BOX, reads information and saves all extended data in a virtual image. This function has to be called before other functions, working with the CRYPTO-BOX. It encapsulates CBIOS_EX calls which enter extended mode, read the CRYPTO-BOX and leave extended mode.

Parameters: iBoxIndex – number of attached CRYPTO-BOX
bAPW – administrative password

Return: XSMRX_SUCCESS – if success, otherwise CBIOS_EX level error (call **ErrorToText()** for description).

public int XSMRX::GetBoxInfo (out XSMRXW32.BoxInfo *lBoxInfo)*

Description: Returns BoxInfo structure data obtained from the CRYPTO-BOX and stored in the virtual CRYPTO-BOX image

Parameters: (Out) lBoxInfo – structure where all CRYPTO-BOX-related information stored
returns: XSMRX_SUCCESS – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty

Return: *XSMRX_SUCCESS* – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty.

public int XSMRX::GetBoxLabel (out System.Array *saLabel*)

Description: Returns CRYPTO-BOX label obtained from the CRYPTO-BOX and stored in virtual CRYPTO-BOX image

Parameters: saLabel – CRYPTO-BOX Label

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_APP_NOT_FOUND – application not found.

public int XSMRX::GetAppRec (ushort *wAppId*, out XSMRXW32.AppRec *lAppRec*)

Description: Returns application(partition) data obtained from the CRYPTO-BOX and stored in virtual CRYPTO-BOX image

Parameters: wAppId – application(partition) number
lAppRec – application(partition) record

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_APP_NOT_FOUND – application not found,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty.

public int XSMRX::GetAppRecByIndex (ushort *wAppIndex*, out XSMRXW32.AppRec *lAppRec*)

Description: Returns application(partition) data obtained from the CRYPTO-BOX and stored in a virtual CRYPTO-BOX image

Parameters: wAppINDEX – index of application(partition) in CRYPTO-BOX
lAppRec – application(partition) record

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_APP_NOT_FOUND – application not found,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty.

public int XSMRX::GetAvailableRAMSize (out int *iRAMSize*)

Description: Returns size of free memory, available in virtual CRYPTO-BOX image

Parameters: iRAMSize – free memory available in CRYPTO-BOX

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty.

public int XSMRX::SetBoxLabel (System.Array *saLabel*)

Description: Sets the label stored in a virtual CRYPTO-BOX image to be programmed later to the CRYPTO-BOX

Parameters: saLabel – CRYPTO-BOX Label

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty.

public int XSMRX::AddApp (XSMRXW32.AppRec *lAppRec*)

Description: Adds a new application (partition) to the list, stored in a virtual CRYPTO-BOX image to be used when programming the CRYPTO-BOX

Parameters: lAppRec – application(partition) record

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty,
XSMRX_ERROR_APP_ALREADY_EXISTS – application already exists,
XSMRX_ERROR_APP_NO_FREE_RAM – not enough free memory.

public int XSMRX::RemoveApp (ushort *wAppId*)

Description: Removes an application (partition) from the list, stored in a virtual CRYPTO-BOX image to be used when programming the CRYPTO-BOX

Parameters: wAppId – application(partition) number

Return: XSMRX_SUCCESS – if success,

XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty
XSMRX_ERROR_APP_NOT_FOUND – application not found.

public int XSMRX::FormatBox (int *iBoxIndex*, System.Array *bAPW*)

Description: Formats the CRYPTO-BOX with data taken from a virtual CRYPTO-BOX image. This function has to be called after XSMRX::ReadBox and other functions, changing the CRYPTO-BOX geometry. It encapsulates CBIOS_EX calls, which enter extended mode, format the device, write extended data, create partitions and leave extended mode.

Parameters: iBoxIndex – number of attached CRYPTO-BOX
bAPW – administrative password

Return: XSMRX_SUCCESS – if success,
XSMRX_ERROR_EMPTY_IMAGE – virtual image is empty,
XSMRX_ERROR_BOX_SIZE_TOO_SMALL – CRYPTO-BOX size is smaller than virtual image,otherwise CBIOS_EX level error (call **ErrorToText()** for description).

public int XSMRX::SetAPW (int *iBoxIndex*, System.Array *bAPW*, System.Array *bNewAPW*)

Description: Changes administrator password (APW) for the attached CRYPTO-BOX. This function encapsulates CBIOS_EX calls, which enter extended mode, change APW and leave extended mode.

Parameters: iBoxIndex – number of attached CRYPTO-BOX
bAPW – administrative password
bNewAPW – new administrative password

Return: XSMRX_SUCCESS – if success, otherwise CBIOS_EX level error (call **ErrorToText()** for description).

public int XSMRX::SetUPW (int *iBoxIndex*, System.Array *bAPW*, System.Array *bUPW*)

Description: Changes user password (UPW) for the attached CRYPTO-BOX. This function encapsulates CBIOS_EX calls, which enter extended mode, change UPW and leave extended mode.

Parameters: iBoxIndex – number of attached CRYPTO-BOX
bAPW – administrative password
bUPW – new user password

Return: XSMRX_SUCCESS – if success, otherwise CBIOS_EX level error (call **ErrorToText()** for description).

public void XSMRX::ErrorToText (int *iError*, out string *sText*)

Description: Returns text description of error.

Parameters: iError – error code
sText – text string, which contains error description

Return: none

17. Professional Software Protection

In this chapter we provide some suggestions and innovative ideas and techniques for a secure integration of the CRYPTO-BOX into your software. You should consider these hints when you develop applications that access the CRYPTO-BOX using the Smarx OS Programming Interface.

17.1. Important Rules for Professional Software Protection

1. Never give a testing routine a self-describing name:
For example, if you use the function name CheckProtection in a DLL library, the cracker will immediately know where in the code to focus his effort.

2. Avoid unnecessary error messages:
If the program gives an error message after a negative check routine, the cracker can simply search the program code for the error message to track down the procedure that called it. When error messages are unavoidable, hide them as much as possible and create them (dynamically) in real time rather than use resources for them. It's also a good idea to encrypt the data and the procedure that creates an error message, which makes it much more difficult for the cracker to find in the disassembled code.

3. Use the power of the CRYPTO-BOX encryption algorithms for important variables and licensing options:
The encryption engine is the most important and powerful part of the CRYPTO-BOX because it makes the module indispensable and prevents emulation attacks. The current CRYPTO-BOX models support two encryption algorithms. Symmetric: AES/Rijndael, and asymmetric: RSA (only the CRYPTO-BOX SC provides hardware-based RSA, the CRYPTO-BOX XS contains a software implementation of RSA). Random sequence generation and MD5 hash calculation are also supported.

Smarx OS provides developers with universal support of encryption algorithms. It is possible to implement protection/authentication/e-commerce solutions that benefit from the newest symmetric/asymmetric encryption techniques – on local computers or networks (Intranet/Internet).

Example#1: Securing User Password (UPW) with hardware based encryption

One possibility is to use hardware based encryption (not requiring User login to the CRYPTO-BOX) to secure UPW. In other words, the program can store UPW encrypted with CRYPTO-BOX Fixed AES or some extra AES keys of the CRYPTO-BOX SC model, or even CRYPTO-BOX SC RSA key. It means that there is no other way to obtain decrypted UPW value than to attach a valid CRYPTO-BOX.

Possible modifications of this approach to consider:

- keep hardware encrypted and digitally signed UPW in RAM3 zone;

- extend the above scenario with RSA encryption, where the program itself keeps public key of some RSA key pair A, while its private key is stored in the CRYPTO-BOX and validation involves this key pair.

Also, for some protection strategies, it makes sense to consider using hardware encryption even without using further functions of the CRYPTO-BOX.

This recursive protection (where the access password to the CRYPTO-BOX is encrypted by its internal cryptography) can be used to achieve a high level of security when combined with additional techniques such as:

- real-time multi-threading;
- anti-debug protection;
- hardware based digital signatures (the program code is digitally signed with the hardware based cryptography)

Example#2: Using asymmetric encryption (RSA) for secure delivery of symmetric (AES) key

You can generate RSA key pairs and use them for secure delivery of the Rijndael session keys via the Internet to organize secure sessions. You can use hash calculation and random sequence generation to verify passwords (without storing real values) or to authenticate end-users.
It is possible to change encryption keys dynamically with the use of a random sequence generator. If the key is dependent on the date, the year or the length of the path of the working directory, it will lead to pseudo static encryption keys. A cracker may bypass an encryption sequence but his result will be worthless with changing running conditions. Use the checking responses to initiate the key for the encryption algorithm.

4. Vary the CRYPTO-BOX function calls every time the software is run:
This protection strategy will create very strong security, especially if you use different queries and other function calls when the application is started next time (see also point 7). Check different data in the CRYPTO-BOX. Do function calls only once in a while, maybe every other day or week. The main idea is: what's queried at one run does not necessary allow conclusions on the results of the next run.

5. Wait a while before you show error messages when the CRYPTO-BOX is not found:
Wait a while before displaying an error message and put the error routine into another part of the program. This makes it more difficult for the cracker to locate the check routine.

6. Use checksums in your application:
If you test your EXE and DLL files, and even other files, for changes, crackers will be unable to modify them with patches. However, they will still be able to modify the code directly in memory, so it is a good idea to test for changes to the application code in memory. You can improve protection by performing checksums for smaller sections of your program. When you perform checksums on smaller sections of the program, you make it much more difficult for crackers.

7. Use more than one protection routine:

Any protection routine should test only a part of the protection and should not contain all protection options. This prevents the cracker from understanding the complete protection scheme if one routine is discovered.

8. Change the application's behavior during runtime:
Make your protected application as dynamic as possible. Merge dummy queries with real queries. You will discourage the cracker if every step could be a trap. Insert dummy calls to the CRYPTO-BOX. The dummy calls could initialize variables that are only partly needed in other program parts.

9. Avoid simple Yes/No decisions:
When you read information from the CRYPTO-BOX, use complicated mathematical expressions, which use floating-point operations instead of simple integer operations. This will lead to very complicated structures on a machine code level and hackers will have more work deducing what is really happening.

Example:

Instead of incorporating a statement like "if ID_Code=144 then", apply the following statement:

```
if(SQRT{IDCode}+8={IDCode+16}OVER 8)then ...
```

10. Store program parameters and variables in the CRYPTO-BOX:
If you keep the registration information in the Windows registry, it can be discovered. The CRYPTO-BOX contains memory that you can program on the fly. Use this feature to store parameters that are essential for the program to run. This way you can check the presence of a security device indirectly with a delayed reaction and it will be very difficult for a hacker to understand.

Example:

"Calculating the surface of a circle"

Store the string "3.1415" in the CRYPTO-BOX memory before delivering your program. At runtime read this string on the fly and transform it into a number that is used to initialize a temporary variable, let's say "PiFromMARX". Then calculate:

```
Surface=PiFromMARX*Radius^2
```

11. Spread protection routines and separate Queries and Logical Divisions:
It is a good idea to separate queries and logical decisions. Place the function calls in different sections of your program. Furthermore, use the returned values to control the program flow. You may initialize program control parameters with values read from the memory of a CRYPTO-BOX. This way you create checking procedures that are spread throughout the program. So, the cracker must first study which parts belong together.

Example:

Store a returned value in a global variable. Then check its value in a different program module. Depending on the result, you set a new variable "NewVar". A third subroutine tests "NewVar" and makes a decision to stop the program to display an error message. The original returned ID Code is hidden well and the relation to this value is no longer obvious.

12. Use long registration information:
The longer the registration file or number, the longer it takes to understand it. If the registration number is sufficiently long, it may contain essential information that the program needs. If such a program is patched, it will not run correctly.

13. Test several current bits of data when using time-limit protection:
Check the time of system log files. If the current date or time is the same or smaller than when the program was run previously, it will be clear that the time was adjusted. Also, you can save the date and time after each launch of the program and then test the current date and time during a new launch.

14. Use long testing routines:
Routines that take only a few seconds when a program is running may take a longer time to run while disassembling or debugging: especially when it is not obvious if these routines are important for the protection or if it is just useless code.

15. If you distribute an application with certain features and functions disabled, you should not include the full features and functions in this distribution:
Many developers make the mistake of including the code for a function that will be executable only after registration (the ability to save files, for example). In those cases, the cracker can modify the code so that the function will work.

A better approach is to include parts of the code (a sufficiently long bit) together with the full version. With such a protection scheme, it's virtually impossible for the cracker to remove the protection. You can also encrypt the code for the limited function. When the customer buys the full version, the proper CRYPTO-BOX is sent out which is used to decrypt the code for the limited functionality.

16. If your program has been cracked, release a new version:
Maybe the effort for implementing protection was not sufficient. Also pay attention to the Anti-Debugging and Anti-Disassembling tricks in the following sections 9.2 and 9.3. Frequent updates to your program make an older, cracked version unattractive and the new version comes with an improved protection. The cracker needs to start from the very beginning.

17. Use the best, current compression or encoding programs to encode your software:
Keep your compression or encoding program up to date. A good compressor will be difficult for a cracker to remove.

18. If your application uses a registration number, that number should never be visible in memory:
This means that it should be impossible to find your program's registration number when the program is being debugged. When programming with a method that checks to see whether the correct registration number was entered, do something other than just comparing two strings. The best way is to encode the entered registration number and the correct registration in the same way. In this way, the two numbers can be compared without risk of the cracker discovering the code. You can also use the CRYPTO-BOX memory to store the registration information. You might also compare a checksum of the entered registration

number with the checksum of the correct registration number, though if you do so, you will have to use more checking methods to really make sure that the correct registration number was entered, rather than modified in accordance with the checksum that the cracker had seen in his debugger.

19. Use the Internet and require online registration:
When a program is registered online, its registration data is sent to a particular server. In its most basic form, this server then sends back information to the program telling it whether the registration was successful or not. However, the server can also be used to send data that the program needs in order to launch the registered application. This data may range from important parts of the code to a key needed to decode parts of the program.

20. Do not forget to test your software's protection thoroughly:
Test your software protection for all operation systems which are supported by your application. Often, protection that worked on older Windows versions doesn't work correctly with Windows 8 or higher.

17.2. Tips for Protection Against Debugging

Anti-debugging and anti-disassembling actions
Protecting your application against debugging and disassembling is very important. Without any debugging protection it is much easier for a cracker to understand the protection mechanism you are using. Even simple anti-debugging tricks can complicate debugging and anti-disassembling tricks make it hard to understand the debugged code. A good combination of both makes it much more difficult for a cracker to understand and remove even a simple protection.

Many strategies and options for an effective protection against debugging and disassembling with SoftICE, IDA Pro, TRW2000 or Turbo Debugger (you can find a lot of them in the Internet) are written in assembler. This allows you to develop small and effective routines. Many high-level programming languages allow you to insert assembler code.

Pay attention: Many anti-debugging tricks which worked well under older Windows versions may not work under the recent Windows versions. Therefore it is very important to test your software thoroughly under all environments you want to support. Anti-disassembling tricks are mostly independent from the operating system, so you should use them as much as possible.

First your application should perform a simple test to determine if a debugger is present in the memory at startup and display a warning message to remove the debugger. The cracker will probably find this simple test easily and remove it. Therefore, you should perform at least one more test at a later time – but without displaying any warning or error messages. Instead, let your program "freeze" or do something unexpected (wrong calculations, just exit without any error message) which makes it difficult to understand for the cracker.

Below is an example of how to detect SoftICE by calling INT 3. This is one of the most well known anti-debugging tricks and works in all versions of Windows.

INT 3h is called with the following registers: EAX=04h und EBP=4243484Bh ("BCHK" string). If SoftICE is active in memory , the EAX register will contain a value other than 4.

```
   mov  ebp, 04243484Bh ; 'BCHK'
   mov  ax, 04h
   int  3 ; Trap debugger.
   cmp  al,4
   jnz  SoftICE_Detected

   SoftICE_detected:
```

The following example illustrates how to detect via INT 41 if a debugger is present. This interrupt is used by Windows debugging interface.

```
   mov  eax,0x4f    ; AX = 004Fh
   int  0x41        ; INT 41 CPU - MS Windows debugging kernel -
          ; check for debugger installation
   cmp  ax,0xF386   ; AX = F386h if a debugger is present
   jz   SoftICE_detected
   xor  eax,eax

   SoftICE_detected:
```

This example checks via the API-function "CreateFile" if the SoftICE driver is loaded.

```
 {
    HANDLE hFile;

    ; "\\.\NTICE" without esc sequences
    hFile = CreateFile( "\\\\.\\NTICE",
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL);

    if( hFile != INVALID_HANDLE_VALUE )
    {
        CloseHandle(hFile);
        return TRUE;
    }

    return FALSE;
 }
```

This example checks if a potential cracker has set breakpoints on key API functions in a DLL (in this sample GetDlgItemTextA). Breakpoints on DLL functions of the operating system are often used to understand what is done inside the application:

```
LEA     ESI, GetDlgItemTextA
CALL CheckForSoftICEBP
CMP     EAX, "xxxx" <-- Substitute for your own identifier.
JE SoftICEBPIsSet <-- Send bad cracker to some really
<-- horrid routine.
CALL    ESI

CheckForSoftICEBP:
PUSH    ESI
PUSH DS
PUSH CS
```

```
POP     DS
MOV     ESI, [ESI+2] <-- Get dll function jmp address.
MOV     ESI, [ESI] <-- Get dll function real address.
MOV     EAX, ESI <-- Get first dword of dll function.
AND     EAX, 0FFh <-- Use only first byte.
CMP     AL, 0CCh <-- INT 3 ?.
MOV     EAX, 'xxxx' <-- Your identifier.
JE BPXSet
XOR     EAX, EAX <-- No BPX.

BPXSet:
POP     DS
POP     ESI
RET
```

Another possibility to detect debuggers is to set a timer which controls the execution time of a program routine. The routine runs much slower during analysis with a debugger.

17.3. Protection against Disassembling

Ways of making software analysis even more difficult.
In addition to anti-debug protection, professional countermeasures against disassemblers are a must to protect the program from crackers.

The following are examples of two different approaches.
The first approach to consider is "self modifying code". This technique, applied correctly, does not represent any threat to the safety of the system and can be successfully called irrespective to the level of the user privileges.

A simple example of using the WriteProcessMemory function to create the self-modifying code is given in the program listing below. It replaces the instruction of the infinite loop JMP short $-2 with a conditional jump JZ $-2, which continues normal execution of the program. This is a good way of complicating the analysis of the program for the cracker, especially if the call of "WriteMe" is not located in the vicinity of changeable code, but in a separate thread.
It is even better if the modified code looks natural and doesn't arouse any suspicions. In such a case, the cracker may waste a lot of time wandering along the branch of code that never gains control during program execution.

```
int WriteMe(void *addr, int wb)
{
   HANDLE h=OpenProcess(PROCESS_VM_OPERATION|PROCESS_VM_WRITE,
   true, GetCurrentProcessId());
   return WriteProcessMemory(h, addr, &wb, 1, NULL);
}

int main(int argc, char* argv[])
{
   _asm {
        push 0x74   ; JMP --> > JZ
        push offset Here
        call WriteMe
        add esp, 8
   Here:            JMP short here
   }
   printf("#JMP SHORT $-2 was changed to JZ $-2\n");
   return 0;
}
```

There are limitations to consider with this approach. Using WriteProcessMemory is only reasonable in compilers that compile into memory or in unpackers of executable files.

Another limitation of WriteProcessMemory is its inability to create new pages. Only the pages already existing are accessible to it. But what can be done, for example, if another amount of memory must be allocated for the code dynamically generated "on the fly"? Calling the heap control functions, such as malloc, will not be helpful, since executing the code in the heap is not permitted. But the possibility of executing code in the stack is helpful.

Executing code in the stack is permitted because many programs and the operating system need an executable stack to perform certain system functions. This makes it easier for compilers and compiling interpreters to generate code.

Therefore, using the stack to execute self-modifying code is admissible and independent of the system (i.e, it is universal). Besides, such a solution eliminates the following drawbacks of the WriteProcessMemory function.

First – It is extremely difficult to reveal and trace the instructions that modify an unknown memory location. The cracker will have to laboriously analyze the protection code without any hope of quick success (provided that the protective mechanism is implemented without serious bugs that facilitate the cracker's task).

Most attackers will stop their efforts at this point.

Second – At any moment, the application may allocate as much memory for the stack as it sees fit and then, when it becomes unnecessary, free that space. Fortunately, John von Neumann's principle is fair: Program code can be considered data at one moment and executable code at another. This is just what is needed for normal functioning of all unpackers and decryptors of executable code!

Third – Repeated application of such technology in various (many) different program parts, invoked later, at unpredictable time or situation, will exhaust crackers. A supposed cracked program will fail to execute after some time – maybe after days and weeks and drive crackers crazy.

However, programming code that will be executed in the stack involves several specific issues that sometimes are beyond the scope of this manual.

In many cases, such an approach requires knowledge of support for inline assembler inserts by the compiler, which may be not very pleasant for application programmers uninterested in instructions and the structure of the microprocessor. To solve this using a high-level language exclusively, the stack function must pass the pointers (as arguments) to the functions called by it. This is a little inconvenient, but a shorter way doesn't seem to exist.

A simple program that shows how functions are copied to and executed in the stack is given in the listing found below.

```
void Demo(int (*_printf) (const char *,...) )
{
   _printf("Hello, World!\n");
   return;
}

int main(int argc, char* argv])
{
   char buff[1000];
   int (*_printf) (const char *,...);
   int (*_main) (int, char **);
   void (*_Demo) (int (*) (const char *,...));
   _printf=printf;

   int func_len = (unsigned int) _main - (unsigned int) _Demo;
   for (int a=0; a<func_len; a++)
   buff[a]= ((char *) _Demo)[a];
   _Demo = (void (*) (int (*) (const char *,...))) &buff[0];

   _Demo(_printf);
   return 0;
}
```

The question arises: What is the benefit of running a function in the stack? The answer is: The most significant advantage of such a procedure is the ability to change the code of a function running in the stack "on the fly". For example, it can be decrypted. The encrypted code severely complicates disassembling and strengthens protection. Certainly, encrypting just the code is not a serious obstacle for a skilled cracker equipped with a debugger or an advanced disassembler like IDA Pro.

However, if used in combination with the encryption functions of the CRYPTO-BOX, an "external crypto co-processor" is added to your protected application. This part is not accessible to even the most advanced debugger or disassembler!

Without that essential program part, the application will not run. By doing it this way, maximum security is achieved. The certified AES/Rijndael-Algorithm runs on the external CRYPTO-BOX and only the results of encryption/decryption tasks are exchanged with the application – never the keys. Furthermore, the new CRYPTO-BOX SC contains a secure Smart Card chip which offers hardware-based RSA encryption.

There are endless possibilities to take advantage of the various encryption and memory options of the different CRYPTO-BOX models:

The CRYPTO-BOX is additionally equipped with a hardware implementation of a "White Noise Random Generator". Best suited for random key generation, or just to "create noise" on data lines...

Debugging, disassembling, tracing: The CRYPTO-BOX in combination with sophisticated programming techniques provides the answer against crackers and their tools.

17.4. .NET Specific Protection

There is a high risk of reverse engineering for .NET applications and assemblies, because of complete transparency of MSIL (Microsoft Intermediate Language) internally used by this platform. The most efficient protection against reverse engineering for .NET is extensive obfuscation of the managed code, its control flow and related structures: Namespaces,

Types, Methods, Events, Properties and Fields. The obfuscation should be combined with resources encryption, merging and embedding assemblies, and protection against direct disassembling.
Therefore we strongly recommend to obfuscate your project before releasing it.
There are many different .NET obfuscators available. We can recommend the open source obfuscator Confuser https://mkaring.github.io/ConfuserEx/
An example which demonstrates usage of Confuser obfuscator can be found in the PPK:
[Smarx OS PPK root]/SmarxOS-Samples/Security/.NetProtection

Professional level of obfuscation can be combined with AutoCrypt based protection of the application itself (EXE.NET) and CRYPTO-BOX based license management for the product.

17.5. Sample Code

The Smarx OS Protection Kit (PPK) contains sample code for (for C++, Delphi and C#) which demonstrates how to make the integration of the CRYPTO-BOX into your code more secure by:

- Encrypting the UPW of the CRYPTO-BOX in the code
- Using obfuscation (for C# and C++ projects)

You can find these samples in PPK Control Center at Implementation with API → Demo Code under points 3. and 4.

18. Appendix A: Technical Data

	CRYPTO-BOX® SC (CBU SC or CBUCSC)	**CRYPTO-BOX® XS/Versa (CBU XS/Versa or CBUCXS)**
Controller Chip	Secure Smartcard based micro controller with USB interface	Secure Smartcard based micro controller with USB interface
Controller Chip security certifications	EAL4+	EAL4+
Firmware	Proprietary MARX	Proprietary MARX
Supported Operating Systems	Windows, Linux, macOS, iOS*, Android*	Windows, Linux, macOS, iOS*, Android*
Algorithms implemented in hardware	AES 128 bit (CBC/OFB mode), RSA up to 2048 bit key length, others (i.e. ECC) on request	AES 128 bit (OFB mode), RSA up to 2048 bit key length (on driver level)
Memory size (total)	72Kbytes, ca. 30 KBytes free	4, 32 or 64Kbytes
Read/write rate of the internal memory	Read/write: ca. 80 Kbytes/s	Read/write: ca. 12 KBytes/s
User/Administrator Password (PIN/PUK)	Up to 16 bytes sequence	
Casing & LED	Metal Designer Case, LED displays the operating mode, eye for key ring/lanyard	
Connector	USB Type A or USB Type C	
Programming memory	Typically more than 1 million cycles; 100 000 guaranteed	
Data retention min.	10 years	
Electrical certifications	CE, FCC, RoHS, WEEE, USB Logo	
Dimensions	USB A variant: 14 x 7 x 32,5 mm / 0.55” x 0.28” 1.28” USB-C variant: 22,5 x 11 x 4 mm / 0.89" x 0.43" x 0.16"	
Weight	USB A variant: 7,5 g / 0.265 oz USB-C variant: 3g / 0.11oz	
Temperature range	-10°C to +70°C / 14°F to 158°F	
Humidity	0% to 95% relative humidity	

* network mode

19. Appendix B: Support & Collaboration with Customers

Technical Support

- In North America:
  - Phone: **1-770-904-0369**
  - Email: **support@marx.com**

- In Europe and worldwide:
  - Phone: +**49 (0) 8403/9295-0**
  - Email: **support-de@marx.com**

MARX offers several outstanding services:

End-user Support

Even end-users can get direct support from MARX, e.g. downloads of the newest CRYPTO-BOX drivers. This saves your support staff a lot of effort and keeps your cost low! Of course, we will never disclose confidential material to your end-users.

Current CRYPTO-BOX device drivers, redistributable components or the MARX Analyzer diagnostic tool are published at www.marx.com.

Training for developers

On request we offer training courses to give you an introduction to secure implementation of the CRYPTO-BOX into your applications.

Beta Test Program

Customers having a valid support level option (see www.marx.com/en/support/support-level-options and would like to participate may enroll for free. They will receive the latest software updates and other benefits before new products hit the market.

Please contact support@marx.com for detailed information.

Customer-specific solutions, special models, OEM

Our hardware and software components can, in many cases, be customized to your particular requirements. Options, and terms and conditions, are available on request.

Just-in-time delivery

Just-in-time distribution allows you to profit from low storage costs. In addition to our fast delivery service, we can also fill orders the same day we receive them. Orders received before 10 a.m. (CET) can usually be shipped on the same day (with the exception of custom-made products and during holidays).

Customer-specific product labeling

Custom labeling provides you with more flexibility when distributing your product together with the CRYPTO-BOX. It allows quick identification in your distribution chain (e.g. by a scanner) or at the customer site. MARX offers a wide range of possibilities: from labeling with a data matrix code (2D-barcode); see Figure B.1) or serial numbers, laser printing, right through to adhesive labels or overprinting. Just ask us - we would be happy to provide a quote.

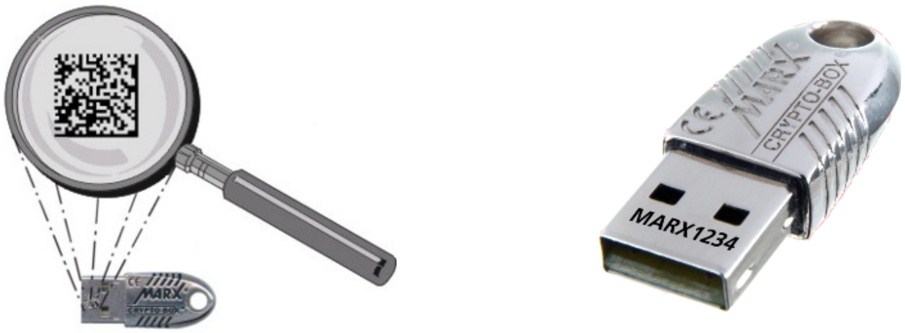


Fig. B.1: Data Matrix Code (l.) and Serial number (r.)

Colored cases for the CRYPTO-BOX

Additional to "shiny chromium" you may choose between two additional color variants: Black and Titanium. Using different colors can be useful if you sell different software products and want to distinguish the CRYPTO-BOX modules you ship with them.
Furthermore, customer specific colors are possible upon request - just ask us.

20. Appendix C: Distributors

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

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

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

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

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

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

Poland
INLOGICA Sp. z o.o. Sp. K.
ul. Św. Michała 43
61-119 Poznań
www.inlogica.com

Contact:	Grzegorz Bigos
Phone:	+48 61 2785830
E-Mail:	office@inlogica.com

21. Appendix D: Glossary

Access Codes
A set of codes (passwords) required to access a CRYPTO-BOX. MARX assigns these codes. They are unique for every company we ship to. The Access Codes are included in the CRYPTO-BOX profile (.trx file).

AES/Rijndael
Advanced Encryption Standard. The Rijndael data encryption formula was named by the U.S. National Institute of Standards and Technology as the winner of a three-year competition involving some of the world's leading cryptographers. This encryption algorithm is used in the CRYPTO-BOX.

API
Application Programming Interface is a set of routines a program can call in a function library or in the operating system. MARX provides a set of APIs (→CBIOS API , →DO API, →RFP API, →Smarx®API, →CBIOS4NET,→ Smarx4NET) to access the CRYPTO-BOX within the →Source Code of your application. Libraries for many platforms and development environments are provided in the →PPK.

Arguments
Parameters passed to →Function Calls. The arguments contain data required by functions to execute correctly.

Authentication
The establishing of identity of a person or process. Authentication helps to verify that a request came from a genuine source.

AutoCrypt
Contains a →Wrapper which provides automatic protection for programs. Makes applications tamper-proof and reduces size. No source code required. See chapter *4.4* for more information.

Binary shell
→Wrapper.

CBIOS API
The core component of the →Smarx®OS interface. It provides access to CRYPTO-BOX hardware on local computer or in the network: returns information like serial number, developer-ID or firmware version and provides access to the CRYPTO-BOX internal memory and encryptions functions. See chapter *12* more information.

CBIOS4NET
Object oriented component based →API interface for .NET developers which combines multiple →Smarx®OS interfaces for the CRYPTO-BOX: →CBIOS API (local and network mode), →DO API and →RFP API. In contrast to →Smarx4NET it supports both local and network access to the CRYPTO-BOX, contains unmanaged code and requires VC redistributables and platform specific assembly (32 or 64bit). See chapter *10.13.2* for more details.

CBUSetup
Tool for installing CRYPTO-BOX →Device Drivers and supplemental files for Windows on the end-user's computer. See chapter *8* for more information.

Compression
→AutoCrypt provides compression (similar to Zip) to make applications tamper-proof and reduces size.

Cloud Security (WEB API)
Online authentication (only users with a valid CRYPTO-BOX have access to the content) with for many fields of application: support only for paying customers, access to content only for authorized service personnel. Also ideal for subscription services and remote activation of software (including updating

licenses in the CRYPTO-BOX). See www.marx.com → Products → Cloud Security for more details.

Counter
A value used for monitoring the total number of program execution, access times, etc. A counter might be either incrementing or decrementing.

CRYPTO-BOX®
Security hardware attached to a computer through →USB, parallel or serial port. Mostly used for software protection, but may serve for access control purposes as well. Every CRYPTO-BOX has a microcontroller, which provides secure access to the contents of its memory and has the ability to encrypt programs and data. It can also hold any kind of sensitive information, like access codes, expiration dates, or license info.

CrypToken®
The CrypToken is a USB token from MARX which can be used as secure storage of certificates used as part of two-factor authentication systems. It offers integration into PKCS#11 and MS CAPI based applications. The CrypToken M2048 is based on MULTOS Operating System, the CrypToken MX2048 on JavaCard.

Device Driver
Software component that extends the operating system of a computer so that devices like the CRYPTO-BOX can be accessed.

Digital Signature
A digital signature is an electronic equivalent of an individual's signature. It authenticates the message to which it is attached and validates the authenticity of the sender. In addition, it provides confirmation that the contents of the message to which it is attached have not been tampered with.

DO API (DataObjects API)
This API is implemented on top of the →CBIOS API and provides convenient access to licensing data stored in the CRYPTO-BOX memory partitions for the purpose of license control. Such data objects can be: Expiration date or -time, counters, customer specific objects in memory, such as licensing information or encryption keys for AES and RSA encryption (CRYPTO-BOX SC only). See chapter *14* for more information.

Dummy Call
A →Function Call which drives the intruder into believing that this is a task-critical call. Dummy calls are useful for misleading potential pirates and crackers.

Encryption
A way to represent data in a ciphered form to prevent unauthorized access to it by an intruder.

ESD
Electronic Software Distribution is supported by the CRYPTO-BOX system.

Expiration Date
Date after which a protected application cannot be launched anymore.

Function Call
A definition of an application programming interface (→API) call, its parameters and return value (if any). After the call is executed, the control is passed back to the calling process.

Hardware Key
See "CRYPTO-BOX".

Hash-function
A Hash-function compresses data in a special way, which is irreversible. It can be used to create an electronic counterpart of a password or fingerprint, which can be transmitted over an unsecured pathway without having fear of spying/hacking.

Hexadecimal values
A value based on the number 16. Example: the number 32 is represented by hex 20

Kernel
The essential part of an operating system, responsible for resource allocation, low-level hardware interfaces, security, etc.

Licensing
Granting permission to use intellectual property on a software product marketed by the licensee in exchange for payment.

MARX Analyzer
This diagnostic tool for troubleshooting CRYPTO-BOX hardware and software components at the end-user side. See chapter *9.1* for more information.

MPI
Legacy programmers interface for the CRYPTO-BOX, supports USB, parallel and serial devices. Replaced by →Smarx®OS.

Multitasking
The concurrent operation by one central processing unit of two or more processes.

Mutex
A mutex is a program object that allows multiple program threads to share the same resource, such as CRYPTO-BOX access.

.NET Technology
Software framework developed by Microsoft. It includes a large class library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (as contrasted to hardware environment), known as Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling. FCL and CLR together constitute .NET Framework.
MARX provides 2 interfaces designed especially for .NET developers: →Smarx4NET and →CBIOS4NET.

Network License Management
A system used to define and control the total number of concurrent users in a LAN. See chapter *5* for more information.

Obfuscation
Methods used to make analysis of the protected software difficult for hackers. Refer to chapter *17* in this manual for more information and hints about secure implementation of the CRYPTO-BOX.

Pay-per-Use
In the internet (Netflix, Amazon Prime) it's already commonplace: If customers want to see a particular movie or series, they have to pay for it. The same can apply to software distribution. Rather than paying for the application, the customer pays for the number of program starts, for the amount of time the software is used or for the engaged functionality. In short: Intensive end-users pay more.

Piracy
Unauthorized duplication, use and distribution of computer software and/or related material.

PKI
Public Key Infrastructure defines the rules and organizational background that enables deployment of encryption-based security services.

RFP API (Remote Update API)
Remote Update provides a convenient way of remotely updating →data objects with licensing information stored in the CRYPTO-BOX memory directly on the end-user side. The RFP API allows seamless

integration of remote update into the source code of your application. This allows to have full control over the update procedure.

PPK (Professional Protection Kit)
Development Kit for the CRYPTO-BOX, contains all necessary components and tools for CRYPTO-BOX implementation – either via →AutoCrypt or Implementation with →API. The →Smarx Application Framework (SxAF) is one of its main components.

RUMS (Remote Update Management System)
RUMS provides a convenient way of remotely updating →data objects with licensing information stored in the CRYPTO-BOX memory directly on the end-user side without programming efforts (in contrast to the →RFP API). See chapter *6.1* for more information.

RSA
Most widely deployed public-key algorithm. The CRYPTO-BOX SC supports RSA on hardware level, the CRYPTO-BOX XS and Versa models support RSA on driver level.

Smarx®API
High level →API for software developers for CRYPTO-BOX integration into the application →Source Code. In contrast to other →Smarx®OS interfaces, Smarx API exposes simple and user friendly programming interface, significantly reducing implementation efforts compared to other CRYPTO-BOX interfaces. See chapter *11* for details.

Smarx Application Framework (SxAF)
SxAF is the central management software for the CRYPTO-BOX. It automates software protection and licensing and offers the following options:

- →AutoCrypt – define projects for automatic protection Windows executables and DLLs
- Implementation with →API – manage licensing information in the CRYPTO-BOX
- Document Protection – protection and licensing of documents
- Media Protection – protection and licensing of various media file formats
- CB Format – configure CRYPTO-BOX units for the tasks mentioned above
- →RUMS – update licensing information in the end-user's CRYPTO-BOX
- End-User Management – assign CRYPTO-BOX units to end-users
- Export project settings for usage with →SmrxProg command line tool

See chapter *4* for more information on SxAF.

Smarx®OS
Smarx OS is an "operating system" specially tailored to the features of the CRYPTO-BOX SC/XS/Versa. It allows several applications to access concurrently and independently on a single CRYPTO-BOX. This is achieved through intelligent file management and a sophisticated programming interface (e.g. →CBIOS API , →Smarx®API, →CBIOS4NET).

Smarx4NET
Object oriented component based →API interface for .NET developers which combines multiple →Smarx®OS interfaces for the CRYPTO-BOX: →CBIOS API (network mode only), →DO API and →RFP API. In contrast to →CBIOS4NET it does not support direct (local) CRYPTO-BOX access, but it is fully managed code and does not require VC redistributables or platform specific assemblies (32 or 64bit). See chapter *10.13.2* for more details.

SmrxProg
Command line based tool for CRYPTO-BOX configuration (create/update delete partitions, programming →data objects with licensing information, programming encryption keys, setting the CRYPTO-BOX label). SmrxProg is available for Windows, Linux and macOS. See chapter *7.4* for more information.

Source Code
The form in which a computer program is written by the programmer. A compiler or an interpreter must translate the source code into the object code for a particular computer before the code can be executed.

USB Port
Universal Serial Bus, provides much higher data transfer rates than a serial or parallel port and allows up to 127 devices to be attached.

USB device server
USB device servers make USB devices available to computers in the network. For the client computer it looks like the USB device is connected directly to the computer. This solution is ideal in cases where the USB device cannot be connected directly to the client computer (for example virtualized environments and server-based computing). Recommendations for USB device servers supported by the CRYPTO-BOX can be obtained from MARX on request.

Wrapper
Code which is combined with another piece of code to determine how that code is executed. A wrapper can be used for compatibility or security reasons (e.g. to prevent the calling program from executing certain functions). Realized in →AutoCrypt. The Wrapper within AutoCrypt also provides compression (similar to Zip).

22. Appendix E: Trademarks

MARX®, CRYPTO-BOX®, CrypToken®, LCS®, CRYPTO-WIZARD®, CRYPT:ACCESS®, FILE:CRYPT®, CD-ROM VENDOR SECURITY® and TOKEY™ are trademarks or registered trademarks of MARX.

Microsoft®, Windows®, Windows Server™ and Visual Studio® are registered trademarks of Microsoft Corporation in the United States and other countries.

Java is a trademark or registered trademark of Oracle and/or its affiliates in the United States and other countries.

UNIX® is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

The term "Linux" is a registered trademark of Linus Torvalds.

Mac and macOS are trademarks of Apple Inc., registered in the U.S. and other countries.

Other names may be trademarks of their respective owners.

23. Appendix F: License Agreement

1. General. The software, firmware embedded in chips, documentation, evaluation kits and any media accompanying this License whether on disk, in read only memory, on any other media or in any other form (collectively the “PRODUCT”) are licensed, not sold, to you by MARX® CryptoTech LP (short: “MARX”) for use only under terms of this License, and MARX reserves all rights not expressly granted to you. The rights granted herein are limited to MARX's intellectual property rights in the MARX Software and do not include any other patents or intellectual property rights. You own the media on which the MARX software is recorded but MARX's licensor(s) retain ownership of the software and product itself.

2. Permitted License, Uses and Restrictions. This License allows you to install and use one (1) copy of the software on a single device or computer at a time. This License does not allow the software to exist on more than one such device or computer at a time, and you may not make the software available over a network where it could be used by multiple devices or multiple computers at the same time.
You may make one copy of the software in machine-readable form for backup purposes only; provided that the backup copy must include all copyright or other proprietary notices contained on the original.
If MARX **HARDWARE**, eg. CRYPTO-BOX® or CrypToken® security devices, is purchased, you, as a retailer or distributor, may NOT remove any product names or copyright marks, or alter casings in any way without written permission.
Except as and only to the extent expressly permitted in this License or by applicable law, you may not copy, decompile, reverse engineer, disassemble, attempt to derive the source code of, modify, or create derivative works of the software or product or any part thereof. Any attempt to do so is a violation of the rights of MARX and its licensors of the software or product. If you breach this restriction, you may be subject to prosecution and you are always liable for all damages, including consequential damages. The same applies if attempts are made to disassemble, analyze or reverse engineer any security hardware obtained from MARX.

MARX SOFTWARE AND ALL MARX PRODUCTS IN GENERAL ARE NOT INTENDED FOR USE IN WHICH THE FAILURE OF THE SOFTWARE OR PRODUCT COULD LEAD TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE.

3. Transfer. You may not rent, lease, lend or sublicense the software. You may, however, make a one-time permanent transfer of all of your individual license rights to the software to another party, provided that: (a) the transfer must include all of the software, including all its component parts, original media, printed materials and this License; (b) you do not retain any copies of the software, full or partial, including copies stored on a computer or other storage device; and (c) the party receiving the software reads and agrees to accept the terms and conditions of this License.

4. Termination. This License is effective until terminated. Your rights under this License will terminate automatically without notice from MARX if you fail to comply with any term(s) of this License. Upon the termination of this License, you shall cease all use of the MARX software and destroy all copies, full or partial, of the MARX software.

5. Limited Warranty. MARX warrants the media on which the software is recorded and delivered by MARX to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of original retail purchase. Your exclusive remedy under this section shall be, at MARX's option, either a refund of the purchase price of the product containing the software or replacement of the software or product which is returned to MARX. Any replaced products or parts shall become MARX's property.
THIS LIMITED WARRANTY AND ANY IMPLIED WARRANTIES ON THE MEDIA INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF ORIGINAL RETAIL PURCHASE. SOME JURISDICTIONS DO NOT ALLOW LIMITATION ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU. THE LIMITED WARRANTY SET FORTH HEREIN IS THE ONLY WARRANTY MADE TO YOU AND IS PROVIDED IN LIEU OF ANY OTHER WARRANTIES (IF ANY) CREATED BY ANY DOCUMENTATION OR PACKAGING, THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY BY JURISDICTION.
Warranty claims must be made in writing during the warranty period. The documentation must contain a description of the defect and include sufficient proof for the defect detected in a MARX Product.
MARX is NOT responsible for any delays in delivery.

6. Disclaimer of Warranties. YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT USE OF THE SOFTWARE IS AT YOUR OWN RISK AND THAT THE ENTIRE RISK AS TO SATISFACTORY QUALITY, PERFORMANCE, ACCURACY MEDIA SET FORTH ABOVE AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE SOFTWARE OR PRODUCT IS PROVIDED “AS IS”, WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND, AND DANGER AND DANGER'S LICENSORS (COLLECTIVELY REFERRED TO AS “DANGER” FOR THE PURPOSES OF SECTIONS 6 AND 7) HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH RESPECT TO THE SOFTWARE, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES AND/OR CONDITIONS OF MERCHANTABILITY, OF SATISFACTORY QUALITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY, OF QUIET ENJOYMENT, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. DANGER DOES NOT WARRANT AGAINST INTERFERENCE WITH YOUR ENJOYMENT OF THE SOFTWARE, THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. NO

ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY DANGER SHALL CREATE A WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES OR LIMITATIONS ON APPLICABLE STATUTORY RIGHTS OF A CONSUMER, SO THE ABOVE EXCLUSION AND LIMITATIONS MAY NOT APPLY TO YOU.

7. Limitation of Liability. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT SHALL DANGER BE LIABLE FOR PERSONAL INJURY, OR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, LOSS OF DATA, BUSINESS INTERRUPTION OR ANY OTHER COMMERCIAL DAMAGES OR LOSSES, ARISING OUT OF OR RELATED TO YOUR USE OR INABILITY TO USE THE SOFTWARE, HOWEVER CAUSED, REGARDLESS OF THE THEORY OF LIABILITY (CONTRACT, TORT OR OTHERWISE) AND EVEN IF DANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OF LIABILITY FOR PERSONAL INJURY, OR OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION MAY NOT APPLY TO YOU. In no event shall MARX's total liability to you for all damages (other than as may be required by applicable law in cases involving personal injury) exceed the amount of fifty dollars ($50.00). The foregoing limitations will apply even if the above stated remedy fails of its essential purpose.
In no event MARX will be liable for any claims by you based on any third-party claim.

8. Export Law Assurances. You may not use or otherwise export or reexport the software or product except as authorized by United States law and the laws of the jurisdiction in which the software and hardware was obtained. In particular, but without limitation, the software and hardware may not be exported or re-exported (a) into (or to a national or resident of) any U.S. embargoed countries (currently Cuba, Iran, Iraq, Libya, North Korea, Sudan and Syria) or (b) to anyone on the U.S. Treasure Department's list of Specially Designated Nationals or the U.S. Department of Commerce Denied Person's List or Entity List. By using the software and hardware, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

9. Government End Users. The software and related documentation are “Commercial Items”, as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation”, as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. government end users (a) only as Commercial Items and (b) with only those rights as the granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States.

10. Controlling Law and Severability and Choice of Forum. This License will be governed by and construed in accordance with the laws of the State of Georgia, as applied to agreements entered into and to be performed entirely within Georgia between Georgia residents, that is, without giving any effect to the choice of laws provisions of the State of Georgia. This License shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, the remainder of this License shall continue in full force and effect. You agree that the only courts in which You will bring lawsuits concerning the application or enforcement of this License are courts of competent jurisdiction located in the State of Georgia and you consent to the exercise of jurisdiction by any such court. This paragraph shall survive in full force and effect regardless of any termination of this License.

11. Complete Agreement; Governing Language, Place of Jurisdiction
This License constitutes the entire agreement between the parties with respect to the software licensed hereunder and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by MARX.
Place of jurisdiction is Atlanta, Georgia, U.S.A.
However, MARX may file any claims at other locations at MARX' sole discretion.

24. Appendix G: Notice to Users

24.1. General Information

All attempts have been made to make the information in this document complete and accurate. MARX is not responsible for any direct or indirect damage or loss of business resulting from inaccuracies and omissions. The specifications contained in this document are subject to change without notice.

All product and company names used in this document are trademarks or registered trademarks of their respective holders.

24.2. Electrostatic Discharge (ESD) Precautions

MARX hardware was designed to resist high levels of electro-static discharge (ESD); however, extreme levels of ESD may cause damage. For example, when walking on a synthetic carpet (especially on dry days), a person can generate enough static electricity to cause a spark while touching a metal object. Use the following precautions to protect MARX devices from ESD:

- Keep the MARX hardware in anti-static bags until ready for use.
- Touch a grounded metal object before touching the MARX device.
- Avoid touching the contact pins.

24.3. Further Handling Precautions for MARX Hardware

In order to avoid damage to MARX devices or to the information written in their internal memory, please observe the following precautions when handling them:

- Do not exceed the temperature range specified in the data sheet of the device model you are using.
- Do not leave the device in places that are extremely humid for a longer time
- Do not remove the MARX device from the USB port while an application is accessing it to avoid data loss or corruption of the internal memory. Exit the application or shut down the computer before removing it.

25. Appendix H: Declaration of Conformity Statements

All the Declaration of Conformity statements related to this product can be found at www.marx.com → About → Regulatory Compliance

MARX devices generate, use and can radiate radio frequency energy. If not installed and used in accordance with the instructions, they may cause interference to radio communications. MARX devices have been tested and found to be harmless for residential electrical installations. However, we cannot guarantee that interference with radio communications will not occur in a particular installation.

If you hear or see interference with radio or TV reception that you think may be caused by a MARX device, determine whether the interference comes from the device by connecting it to your computer and disconnecting it again. We encourage you to try to correct the interference using one of the following measures:

- Relocate or re-orient the radio/TV antenna.
- Locate the computer and the receiver in different rooms.
- Plug the computer and the receiver into different electrical outlets.
- Consult a radio/TV technician for help.

Operation with unshielded cables is likely to result in interference of radio and reception. The user is cautioned that modifications and changes made to a MARX device without the manufacturer's approval could void the authority to operate this device.

26. Alphabetical Index

0-30Aug024ks(Compendium).odm

Unique Benefits of the CRYPTO-BOX® SC

- Common Criteria EAL4+ certified SmartCard chip.
- Hardware-based AES and RSA encryption with support for multiple encryption keys in one CRYPTO-BOX.
- Very fast memory access, probably the fastest token on the market.
- Extremely short and robust metal case: shields electronic circuitry perfectly!
- Supports Windows, Linux, macOS and Android.
- Customer specific versions and customized casing available.

Software Security

Establish secure distribution channels and assure revenue for every license. Includes automatic protection for Windows (with support for Win64 and .NET applications) and implementation with API for Windows, Linux, macOS and Android.

Online License Management and Smarx Cloud Security

Automated online updates for the CRYPTO-BOX. Extend or renew licenses remotely 24/7. Restrict and allow login into your web site or subscription service or any kind of online business.

Media Protection

Protection of digital media (video and audio) from piracy and unauthorized duplication with the CRYPTO-BOX. Incorporate DRM features into your media files and secure your digital assets!

Document Protection

Secure distribution of documents. Protect your valuable digital content with the CRYPTO-BOX and limit access to authorized users. Ideal for distribution via Internet. Licenses can be updated remotely (Digital Rights Management).

MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518, U.S.A.
☎ +1-770-904-0369
support@marx.com
www.marx.com

MARX Software Security GmbH
Vohburger Strasse 68
D-85104 Wackerstein
☎ +49 (0) 8403-9295-0
support-de@marx.com
www.marx.com