White Paper

API Implementation WP-1

Subject: CRYPTO-BOX implementation into the source code with Smarx API
Version: Smarx OS PPK 8.4 and higher
Last Update: 15 May 2017
Target Operating Systems: Windows 32/64, Linux, macOS, iOS, Android
Target Processor Platforms: Intel x86, ARM
Access to source code needed (of protection application): ☑ Yes ☐ No
Supported Programming Tools: see chapter 5.1 in this document
Applicable for Product: CRYPTO-BOX® SC / XS / Versa

Implementation with API

The integration directly into the application source code using the Smarx® OS API unleashes the full power of the CRYPTO-BOX® and provides the most flexible way to protect your software and data against piracy and unauthorized usage.

This document describes how to get started with the API integration, and where to find the required libraries and tools in our Protection Kit.

Even if you are working with the API integration already, or if you are planning to revise your API implementations soon, this document will provide you with valuable information about the most recent changes and give you useful tips and references.

CRYPTO-BOX® Key Features

- Quick and easy protection of Windows and Linux applications with AutoCrypt
- Individual implementations with API for all common programming languages, incl. .NET
- The CRYPTO-BOX system can be customized according to individual requirements
- Multi-platform support: Windows, Linux, macOS, iOS and Android
- Unique and stable metal case, optional with customer-specific color and labeling
- Internal secure memory of 4-64 kB
- Network and remote update capability
- AES/Rijndael encryption on-chip
- RSA support on-chip (CRYPTO-BOX SC) or on driver level (CRYPTO-BOX XS/Versa)

Table of Contents

1. The meaning of “Implementation with API”

1.1. 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 (completely or partially, depending on your protection strategy). If the CRYPTO-BOX is attached, the program will communicate with it, performing more detailed verification of information stored in the CRYPTO-BOX, such as:

- Verification of serial number (BoxName) or Developer ID
- Using the hardware-based encryption engine to decrypt information during application run-time
- Querying license information from the internal memory of the CRYPTO-BOX during application run-time

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 *RUMS Application Notes* for more details).

Implementation with API means that all this functionality mentioned above can be added directly into the source code of the application by using predefined API calls (see chapter 3.2 for details).

1.2. Automatic Protection and Implementation with API

When protecting your software with the CRYPTO-BOX, you have two basic choices:

- Automatic protection of your compiled executable - see separate *AutoCrypt Application Notes* for further details.
- Implementation into the source code of your application through API.

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

It is possible to combine both AutoCrypt and API implementation, for instance if you want to take advantage of the encryption options offered by AutoCrypt. Or you can consider AC API which combines the simplicity of AutoCypt with the flexibility of API implementation. This is especially helpful if your type of application is not compatible with the AutoCrypt Wrapper. See chapter 2 for more details.

2. Recommended Steps for Protecting Applications with API

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

1. Make yourself familiar with our API (see chapter 3 and 4 in this document, and *Smarx Compendium* chapter 10). Select your preferred API and check out the sample code for your environment (see chapter 5.2) Now choose your own protection strategy.
2. Check our hints for secure implementation in the *Smarx Compendium* chapter 17.
3. The easiest way to configure the CRYPTO-BOX with the protection and licensing setting required by your protected application is the usage of the Smarx Application Framework (SxAF, see *Smarx Compendium* chapter 4.5):
   - Start SxAF on your computer: Choose “SxAF Client” in Start Menü → MARX CryptoTech → MARX PPK → Application Framework).
   - Create a new SxAF project and specify *Implementation with API* as project type.
   - Choose the project-specific values for the CRYPTO-BOX, such as label and AES keys.
   - 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 *Smarx Compendium* chapter 4.5.5 for further details).
   - Use the “CB Format” option in SxAF to format your CRYPTO-BOX units with the project settings.
   - Optionally, you can export your project settings into an XML file to use with command line based tools for automated protection and CRYPTO-BOX formatting (see *Smarx Compendium* chapter 4.9 for further details).
4. 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 *Smarx Compendium* chapter 4.10.3 for more information).
5. Test all licensing options carefully.
6. Ship your protected application, along with the CRYPTO-BOX and supplemental files (drivers, network server for network licensing if applicable). MARX provides an easy-to-use redistribution setup. See our *Application Notes “Driver Installation”* for further instructions.

3. Smarx®OS as Basis for the CRYPTO-BOX Integration

3.1. Overview

Smarx®OS is the basic input-output system of the CRYPTO-BOX system. It is used for communication with the CRYPTO-BOX within all components available in the *Professional Protection Kit (PPK)*, such as:

- Libraries provided by MARX for API implementation
- Smarx Application Framework (SxAF)
- Command Line Tools

Smarx OS supports all popular platforms:

- Windows 32 and 64 bit
- Linux 32 and 64 bit
- macOS (32 and 64 bit applications)
- iOS (version 4.3 or higher)
- Android (version 3.1 or higher)

Many programming environments (IDE's) for these platforms are supported, see chapter 5 for more details.

3.2. Smarx®OS API Subsets

Smarx OS consists of several APIs providing a different subset of functionality. Not all API subsets are available for each platform/compiler.

The following tables provides an overview about 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 8+, QT 5+
AC API Simple automatic protection API with SxAF/AutoCrypt Wizard projects			
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 the future.

See chapter 5.2 in this document for information on obtaining libraries and sample code for your preferred interface.

3.2.1. Smarx®API

This is a high level API layer for the CRYPTO-BOX SC, XS and Versa models which exposes a more simple and user friendly programming interface to developers than other Smarx OS based APIs (CBIOS, see below).

Please refer to the *Smarx Compendium* chapter 11 for more details about the Smarx API.

3.2.2. AC API

The AC API (Smarx AC) introduces a higher abstract layer allowing developer with only one function call implemented into the source code of his application to:

- Start periodic validation of the license information stored in the CRYPTO-BOX (e.g. expiration date, counters, etc.) for both local and network scenarios
- Add exit event notification (AppExitEvent) with exception argument

AC API support C/C++ and C# (Visual Studio 2013 and higher) under Windows. Further details can be found in the Readme file in AC API sample folder (see chapter 5.2)

3.2.3. SmarxCpp

This is an object oriented implementation of CBIOS (Networking)/DO APIs mentioned below for C++ developers (MS Visual Studio 2013 or higher). Refer to the *Smarx Compendium* chapter 10.13 for details.

3.2.4. CBIOS4NET/Smarx4NET

This is an object oriented, components based implementation of CBIOS (Networking)/DO/RFP APIs mentioned below for .NET developers (C#, VB.NET, etc.). Please refer to the *Smarx Compendium* chapter 10.13.2 for more details.

3.2.5. CBIOS API

This is the basic API for the CRYPTO-BOX SC, XS and Versa models. It includes functions for CRYPTO-BOX search and identification, access to its internal memory and encryption functions.
Please refer to the *Smarx Compendium* chapter 12 for more details about the CBIOS API.

3.2.6. CBIOS Networking

A 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 run in a network. Please refer to the *Smarx Compendium*, chapter 13 for more information about accessing the CRYPTO-BOX in networks.

3.2.7. DO API

The Data Objects (DO) API is a 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.
Please refer to the *Smarx Compendium*, chapter 14 for more details.

3.2.8. RFP API

The Remote Update API allows to update the CRYPTO-BOX directly on the end-user side. It is intended for customers who prefer API integration instead of using tools provided by MARX (RUMS component in SxAF or "RU_Tool.exe" command line tool, see separate *RUMS Application Notes* for details).
Please refer to the *Smarx Compendium*, chapter 15 for more information on the Remote Update technology.

3.2.9. 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).
Please refer to the *Smarx Compendium*, chapter 16 for more details.

3.2.10. Smarx Cloud Security (WEB API)

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 the *Smarx Cloud Security White Paper*.

4. Using the Smarx®OS API under Different Environments

4.1. Overview

Depending on the platform (OS) and programming environment used, the Smarx OS API libraries are provided in different formats, such as:

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

For an introduction into accessing the CRYPTO-BOX via API, we strongly recommend you to read the *Smarx Compendium* chapter 10.

4.2. Static Libraries (C/C++, Delphi)

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

If you are working with Visual Studio 2013 and later or Delphi 10.1 and later, you can consider using our high level Smarx API (see chapter 3.2.1 or SmarxCpp (see chapter 3.2.3).

See the *Smarx Compendium* chapter 11 for a description of the Smarx API.

If you are using other or older environments, or you don't want to use Smarx API: many environments are supported by the CBIOS (Networking)/DO/RFP API (see chapter 3.2.5 to 3.2.8).

- See *Smarx Compendium* chapter 12 for an introduction to the CBIOS API and implementation details.
- See *Smarx Compendium* chapter 13 for details on CBIOS Networking.
- See *Smarx Compendium* chapter 14 for details on the DO API.
- See *Smarx Compendium* chapter 15 for details on the RFP (Remote Update) API.

4.3. Dynamic Libraries (DLL)

Dynamic libraries (DLLs) allow easy, but less secure linkage. DLL based implementation should be considered only if for some reason 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 all environments of Windows (x86 and x64) platform, there is a special DLL (CBIOSVB6.DLL) for Visual Basic 6.0 environment.

The *Smarx OS CBIOS API Reference* contains a detailed description of the API calls within the CBIOS API and the CBIOS Network API (see chapter 3.2.5 and 3.2.6) for developers working with Visual Basic.

4.4. .NET

MARX provides .NET developers with an object oriented, component based approach, simplifying integration of protection and licensing to .NET applications. C# programming community got used to object oriented component based way of software development (main benefit of .NET).

If you are working with Visual Studio 2013 and later, you can consider using our high level Smarx API (see chapter 3.2.1 which is based on CBIOS4NET. See *Smarx Compendium* chapter 11 for a description of the Smarx API.

If using old environment (Visual Studio <2013) or customer specific protection and licensing logic is required, then consider using CBIOS+DO API, rather than Smarx API. CBIOS+DO API is available for .NET developers in Smarx4Net or CBIOS4NET assemblies. See the ".NET interfaces for developers" table on the next page for more details.

Both interfaces combine all Smarx programming interfaces under one roof for .NET platform:

- CBIOS (network mode)
- DO API (including CDO support)
- RU API (Remote Update API)

They cover the following types of MARX hardware:

- CRYPTO-BOX® XS and Versa (CBU)
- CRYPTO-BOX® SC (CBU SC)

The CBIOS4NET Developer's Guide contains an introduction into CBIOS4NET and a detailed description of its Classes.

This table provides an overview about .NET interfaces for developers:

IDE	.NET	Local & Net Mode	Plat form	Additional Redistribu table **	PPK Assembly	SDK Path	MSI / MSM (Redistributable)
MS VS 2013 - 2017	4.5.1+	*	Any CPU	-	Smarx4Net.dll	\dotNET 4.5\Any CPU	\SMARX4NET\ SMARX4NET.msi SMARX4NETMergeModule .msm
	.NET for Windows Store				SmarxRuntime. winmd	\WRC	-
	4.x	+	x86, x64 (platform specific loader CBIOSLoader .cs for .NET 2.0-3.5)	VC Redist 2013	CBIOS4NET.dll	\dotNET 4\x86\signed \dotNET 4\x64\signed	\CBIOS4NET\ CBIOS4NET_x86.msi, CBIOS4NET_x86_x64.msi CBIOS4NetMergeModule. msm
				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	
	2.0-3.5			VC Redist 2005	CBIOS4NET.dll CBIOS4NET64.dll	\dotNET 2\asm signed	
MS VS 2005 - 2008							

* Smarx4Net requires CBIOS Network Server for local mode

** Included to MSI/MSM

The following two chapters explain the differences between Smarx4NET and CBIOS4NET.

4.4.1. Smarx4NET

Smarx4NET is the most recent interface developed by MARX. Compared to the legacy CBIOS4NET interface (the corner stone of Smarx4NET, see chapter 4.4.2), the new Smarx4NET is based on fully managed code which makes the implementation more flexible. There is no need for VC Redistributables anymore.
It supports standard C# applications as well as Windows Store applications, and is ideal for protecting multi-device applications for desktop and mobile usage, including support for Windows Phone devices.
Smarx4NET runs in network mode, which requires an installation of the CBIOS Server (either on the same computer or in the network). See our White Paper *"Network Licensing"* and the *CBIOS Server readme file* for installation instructions.
A Smarx4NET package which contains documentation, libraries and sample code can be found in our *Download section*. Please refer to the included readme file for the latest information and updates.

4.4.2. CBIOS4NET

CBIOS4NET was the first interface for C# developers. The CBIOS4NET assemblies are based on unmanaged

code which requires installation of corresponding Visual C/C++ Redistributable components. Furthermore, a Loader is required to load platform specific CBIOS4NET components (32 or 64Bit).

CBIOS4NET allows direct access to the CRYPTO-BOX on the local USB port (Smarx4NET too, but the CBIOS Network Server needs to be installed and running on the same computer).

If you are starting a new project, we recommend to use Smarx4NET because it offers more flexibility such as AnyCPU support and pure managed code.
If you are already using CBIOS4NET or need direct access to local USB port, you can stick with CBIOS4NET. If you want to switch to Smarx4NET, slightly refactoring of your code is required, plus using network mode instead of local mode. More details can be found in chapter 4 of the readme file in the *Smarx4NET package*.

4.5. COM/ActiveX

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 installed and properly registered by our driver setup utility (CBUSetup.exe, see separate Application Notes *"Driver Installation"*).

Native DLL/native SO are specific to Java environment (Windows and Linux correspondingly).

5. How to Find the Corresponding Library and Sample Code for Your Environment

5.1. Overview about Supported Environments

The following table contains programming environments currently supported by Smarx®OS APIs:

Smarx®OS library	Target audience	Smarx OS Interfaces	Platform	Language	Environment
SmarxCPP static library	If you develop apps in C++ 11, you can use: 1. AC API – implement protection with only one function call 2. Smarx API - validate license with only one call using higher abstract layer (see ch. 4.2) 3. SmarxCPP - develop your licensing model with enhanced C++ classes	Smarx API, AC API, RUMS*, CBIOS, DO API	Win, Linux*, macOS*, Android*, iOS*	C++ 11	MSVS 2013+, gcc 4.8+, Xcode 8+, QT 5+
CBIOS static library	For C/C++, Delphi, Swift, COBOL, MATLAB, IVFortran developers	CBIOS, DO, RUMS API	Win, Linux, macOS, Android, iOS	C/C++, Delphi, Swift, COBOL, MATLAB, IVFortran	MSVS 6+, Builder 6+, Delphi 5+, gcc 4+, Xcode 4+ 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 *4.4* 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. To be considered for VBA development only	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 *3.2*) it provides maximum flexibility. Available for C/C++ and Delphi only.	RFP API	Win, Linux	C/C++	MSVS 6+, gcc 4+
RFP dynamic library			Win	Delphi	Delphi 6+
SmarxRuntime component	For .NET/Windows Store developers	CBIOS, DO API	.NET for Windows Store	C#, C++.NET	MSVS 2012+
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 data files used by your app.	DP API	Win	C#, Delphi	MSVS 2005+, Delphi 7+

+ - and higher

* - Can be implemented upon request

5.2. Obtaining the required Library/Samplecode from the Protection Kit

All libraries and samples for the supported environments can be found in the Smarx OS Protection Kit (PPK), which will be delivered together with the CRYPTO-BOX Evaluation Kit or with the first CRYPTO-BOX order you received from MARX.

5.2.1. Windows

First, you need to install the latest Smarx OS Protection Kit (PPK) from the CD-ROM which was shipped with your CRYPTO-BOX delivery, or download it from our web page (MyMARX login and valid *Support Option* required). After the installation has finished, click on the “PPK Control Center” shortcut on your desktop. The Control Center provides an overview of the installed PPK components, including a brief introduction and links to the components.

Click on the "Implementation with API" button, then on “Libraries/Samples”. For Windows there are two options:

a) Windows Libraries

Here you can select the required library for your platform. This library needs to be implemented into your project.

b) Windows Samples

Here you will find the sample code for your compiler which demonstrates the available API calls. These samples are a good starting point to get familiar with the Smarx API. For different API subsets (e.g. CBIOS and DO, see chapter *3.2* for more details) there may be different samples available.

Select the corresponding libraries for your compiler from the "Libraries (SDK)" section, and take the sample code from the "Samples" section to make sure to have the correct library version for your compiler version! Refer to the included readme files for detailed information and implementation hints!
Please *contact us* if you need libraries or sample code for environments which are not listed in the Protection Kit.

5.2.2. Linux

The Linux package which includes libraries and sample code for the supported Linux based environments (see chapter 5.1) can be found in the /linux subfolder on the Protection Kit CD-ROM. Please refer to the included readme file for further details.

The *Smarx Compendium*, chapter 10.6 provides an introduction on Linux support.

5.2.3. Mac (macOS/OS X)

The package for Mac includes libraries and sample code for the supported Mac environments (see chapter 5.1). It can be found in the /macosx subfolder on the Protection Kit CD-ROM. Please refer to the included readme file for further details.

The *Smarx Compendium*, chapter 10.7 provides an introduction on Mac support.

5.2.4. iOS

The Smarx OS package for iOS contains the CBIOS Network Client for iOS. The sample code written in Xcode

(version 4.2.1 or newer required) demonstrates interaction with a remote CBIOS Server over the network from iOS devices.

For more information on using the CRYPTO-BOX in networks, please read chapter 6 in the *Smarx Compendium*.

All devices running iOS 4.3 or later are supported: iPhone, iPad or iPod Touch. Please *contact us* to receive the iOS sample code.

5.2.5. 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. This requires root access on the Android device. Android SDK and Eclipse IDE are required.

The Smarx OS package for Android supports tablets or smartphones with Android 3.1 or higher.

Samples for both network and local mode are available on the Protection Kit CD-ROM in the /Android subfolder. Please *contact us* for libraries and source code.

White Paper

API Implementation WP-1

CRYPTO-BOX® Data Sheet

	CRYPTO-BOX SC (CBU SC)	CRYPTO-BOX XS/Versa (CBU XS/Versa)
		
Controller chip	RISC Smart Card Processor	RISC Smart Card Processor
Chip certification	EAL4+	EAL4+
Supported operating systems	Windows, Linux, macOS, iOS, Android	Windows, Linux, macOS, iOS, Android
In hardware implemented algorithms	AES 128 bit, RSA (up to 2048 bit key length), others (for example: ECC) on request	AES 128 Bit in hardware, RSA up to 2048 Bit key length on driver level
Memory size (complete)	72KByte, approx. 30KByte free	4, 32 or 64 KByte
Internal memory read/write performance	ca. 80kByte/s	ca. 12kByte/s
Password (PIN/PUK)	up to 16 Byte length	
Case & LED	Designer metal housing, cast zinc, with LED display of operating status, eye for key ring/lanyard	
Connector	USB Type A	
Memory programming	minimum 100,000 write cycles	
Data retention time	minimum 10 years	
Conformity & Certifications	FCC, CE, RoHS, WEEE, USB logo	
Dimensions	14 x 7 x 32,5 mm / 0.55” x 0.28” 1.28”	14 x 7 x 32,5 mm / 0.55” x 0.28” 1.28”
Weight	7,5g	7,5g
Temperature	-10°C to +70°C / 14°F to 158°F	
Humidity	0% to 95% relative humidity	

CRYPTO-BOX Certifications

Evaluation Kit

www.marx.com/eval

MARX Software Security GmbH
Vohburger Strasse 68
85104 Wackerstein, Germany
Phone: +49 (0) 8403 / 9295-0
Fax: +49 (0) 8403 / 9295-40
contact-de@marx.com

MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518 U.S.A.
Phone: (+1) 770 904 0369
Fax: (+1) 678 730 1804
contact@marx.com

www.marx.com