

Software Protection System

CRYPTO-BOX®



Including professional anti-debugging features!

www.marx.com

Securing the Digital WorldSM

Professional Protection Kit MPI 2.62

Electronic Software Distribution
with the CRYPTO-BOX® –
the easy and efficient way to
control software licenses

MARX®
Software Security
March 1, 2004

Solutions based on the CRYPTO-BOX® System

- Software protection with automatic program wrapper
- Custom integration into source code via MARX API
- Powerful, fast and secure encryption of applications and data with officially approved AES/Rijndael Algorithm
- Versatile document and data protection: PDF, Audio, Video, ...
- Remote programming and updating via the Internet
- Secure Logon for networks and web portals
- License Management

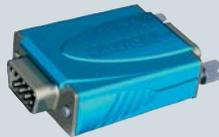
All CRYPTO-BOX® hardware types integrated via one API (MPI programmer's interface)



CRYPTO-BOX USB



CRYPTO-BOX
Parallel Port



CRYPTO-BOX Serial



CRYPTO-BOX Card
(PCI- and ISA-Bus)

Securing the Digital WorldSM

www.marx.com

MARX Software Security
2900 Chamblee Tucker Rd. N.E., Building 9
Atlanta, GA 30341 USA
Tel.: (+1) 770-986-8887
Fax: (+1) 770-986-8891
info@marx.com

MARX Software Security GmbH
Vohburger Str. 68
D-85104 Wackerstein
Tel. +49 (0) 8403-92 95-0
Fax +49 (0) 8403-15 00
contact-de@marx.com



Preface

From Philipp Marx, Managing Director MARX Software Security

Dear valued customer:

We are often asked: why bother with software protection? And why the CRYPTO-BOX?

These questions – frequently coming from software developers and distributors who have already suffered considerable revenue losses as a result of unprotected distribution – are justified. In many cases, however, they are founded on prior experience with less flexible software protection solutions.



testing of the full application, or customized software packages. And ordered and paid is only, what's actually needed.

Additional program options can be activated at any time, without delay, by means of an "on-site license manager". No need to deliver another CD or different protection key! This allows you to fully exploit the advantages of ESD

(Electronic Software Distribution) and achieve a decisive time advantage in the customer relationship path.

The CRYPTO-BOX changes all that: It's your "on-site license manager" – at the customer's place. Reliable and uncompromising, it ensures that your terms of use are adhered to and protect your copyright over intellectual property.

Today, intelligent software protection is not just the one and only effective means of license control, but also an indispensable sales, marketing and installation tool

A modern protection system is much more than just a line of defense. Your marketing department will profit from 100% customer registration.

Remote maintenance provides flexibility, and independence from customs barriers and postal delivery time. Software distribution needs digital logistics!

And not at least: The customer, too, profits greatly. Whether through the availability of obligation-free

Another interesting related fact:

At MARX, there is no additional hardware-related charge for network versions! The same affordable price applies to a standalone installation and a 250-seat network installation.

That makes the "Per Seat License" for networks unbelievably good value for money. And definitely more attractive than just "printed license agreements". Or would you rather be suing your customers?

This handbook will show you better alternatives.

I wish you much success with your software protection.

Yours sincerely,
Philipp Marx

**For your quick start - if you are already familiar with the CRYPTO-BOX: For Automatic protection of applications with AutoCrypt see chapter 5.1 (page 37).
The API reference for implementation into source code can be found in chapter 10 (page 116).**

Table of contents

1. Software protection and distribution strategies	3
1.1 Electronic Software Distribution (ESD) – your path to success	3
1.2 Now to specifics: ESD strategies	5
1.2.1 Flexible distribution using the License Activation Wizard (LAW)	6
1.2.2 "Classic" activation using the CRYPTO-BOX®	11
1.2.3 Site License Model	13
1.2.4 Pay-per-Use or software leasing model	15
1.3 No software protection = No control over licenses!	17
1.4 Conventional software protection is inflexible	18
2. If effective license management is your goal	19
2.1 Pirated copies and multiple users result in loss of sales	19
2.2. Professional software protection secures revenue	20
2.3 Overview: Two implementation paths	21
2.4 Software protection products tailored to your requirements	22
2.4.1 CRYPTO-BOX hardware protection	23
2.4.2 Software support, license management, remote upgrades	24
2.4.3 Expertise, consulting, OEM solutions	25
3. Overview: CRYPTO-BOX Systems	27
3.1 MARX Product Selector	27
3.2 CRYPTO-BOX USB	28
3.2.1 Differences and functions of the USB models	28
3.2.2 CRYPTO-BOX USB Implementation	31
3.3 The CrypToken®	32
3.3.1 Authentication and data transfer	32
3.4 Parallel keys	32
3.4.1 CRYPTO-BOX 560/Net	32
3.4.2 CRYPTO-BOX Versa	33
3.5 Serial keys	33
3.5.1 CRYPTO-BOX Serial	33

II Table of contents

4.	Support, information and more ...	35
4.1	Current product information and downloads on the Internet	35
4.2	Technical support - Hotline, FAQ	35
4.3	Customer-specific solutions, special models, OEM	36
4.4.	Just-in-time distribution	36
4.5	Customer-specific product labeling	36
5.	Fundamentals	37
5.1	The Professional Protection Kit	37
5.1.1	MPI and the TEOS Professional Protection Kit	37
5.1.2	Professional Protection Kit (PPK) Installation	38
5.1.3	Software protection with AUTO:CRYPT	41
5.1.4	AutoCrypt: Automatic software protection for EXEs/DLLs	41
5.1.5	Protecting applications using the API	56
5.1.6	Comparison of automatic vs manual implementation	57
5.2	Using MARX Data Objects	58
5.3	The MARX Programming Interface (MPI)	59
5.3.1	MPI - easy, secure and portable	59
5.3.2	MPI and the CRYPTO-BOX system	60
5.3.3	MPI in networks	60
5.4	Configuring CRYPTO-BOX	61
5.4.1	Configurable parameters of a CRYPTO-BOX	61
5.4.2	AutoCrypt Wizard and CBProg	61
5.4.3	Configuring a CRYPTO-BOX using the AutoCrypt Wizard	62
5.4.4	Configuring a CRYPTO-BOX using CBProg	63
5.5.	MarxProbe - the test and diagnostic tool	64
5.5.1	Functionality	65
5.5.2	Using MarxProbe	66
5.6	License Control System and Remote Field Programming	68
5.6.1	License Control System (LCS)	68
5.6.2	Remote Field Programming (RFP)	70
5.7	Multiple Application Authorization System (MAAS)	74

6. Components for custom-made solutions	77
6.1 Distribution of protected applications	77
6.2 Device drivers	77
6.3 CRYPTO-BOX DLL and static libraries	78
6.4 CRYPTO-BOX USB under Windows	78
6.5 CRYPTO-BOX 560/Net and Versa for the parallel port	81
6.6 CRYPTO-BOX Serial	83
6.7 CRYPTO-BOX Card	84
7. Secure distribution of digital data	85
7.1 Protection of Microsoft® Office documents	85
7.2 PDF Protection supports DRM (Digital Rights Management)	85
7.3 Security Extension for Macromedia Director	86
7.4 HTML Security Extension	86
7.5 AudioVideo RTE™: Protection for video and audio streams	86
8. Supported Operating Systems	87
8.1 Windows XP/2000 and Me/9x	88
8.2 Support for Microsoft .NET environment	89
8.3 LINUX/ UNIX/ Solaris/ QNX	89
8.4 Macintosh / MacOS, OS/X Jaguar/Panther	90
8.5 DOS	90
8.6 Embedded Systems	90
9. Secure Integration techniques for CRYPTO-BOX modules	91
9.1 Important rules for professional software protection	91
9.2 Tips for protection against debugging	96
9.3 Protection against Disassembling	100
10. The MPI- (API-) Reference	105
10.1 MPI – Makes Network Access a Snap	106
10.2 Using MARX Data Objects	107
10.3 Where to start using the API	108
10.4 Where to find the functions of the MPI	110

IV Table of contents

10.5	Available Function Calls	110
10.6	Return Codes of CRYPTO-BOX devices	113
10.7	Access Codes of a CRYPTO-BOX® Evaluation Kit	115
10.8	Type Declarations of MPI	115
10.9	The MARX API Reference	116
11.	Appendix	173
	Appendix A: Codes of a CRYPTO-BOX	173
	Codes of a CRYPTO-BOX USB - and Serial Evaluation Kit	173
	Codes of a CRYPTO-BOX 560/Net, Versa Evaluation Kit	174
	Appendix B: Converting to Hexadecimal Numbers	175
	Appendix C: Technical Data	176
	Features of CRYPTO-BOX USB models	177
	CRYPTO-BOX 560/Net and Versa	178
	CRYPTO-BOX Serial CBS3/9-Pin	179
	CRYPTO-BOX Technical Data and Overview	180
	Appendix D: Supported Compilers and Applications	181
	Supported Compilers (MPI based samples)	181
	Supported Compilers (legacy samples)	181
	Supported Applications	182
	Supported Standards and Interfaces	182
	Appendix E: Distributors	183
	Appendix F: The AES/Rijndael algorithm	184
	Appendix G: Glossary	186
	Appendix H: Trademarks	191
	Appendix I: Developer's Agreement	192
	Notice to users	193
12.	Index	195

1. Software protection and distribution strategies

If you are an advocate of Electronic Software Distribution, then you know that without reliable software protection even the smartest distribution strategy will be on a shaky foundation. We have the tools you need to implement your individual marketing and distribution strategy.

1.1 Electronic Software Distribution (ESD) – your path to success

It's finished! After endless meetings and quite a few system crashes, your new product is finally finished, and the odds are it will be a great success. Your target market is very diverse – you want to make a big launch on the international market. Your marketing strategy is, also, all set to go. Now all you need is a perfect distribution strategy so that your "new baby" can bring in some healthy returns.

But that's the snag. You'd like to use the Internet as a distribution tool and so gain a decisive competitive advantage. Retail business hours are not an issue for you because your product will be available to customers 24/7 on the Internet!

Intelligent software protection using a CRYPTO-BOX key is the great software protection solution in particular for modularly designed products where each customer obtains only those components of the application that they need. But how can efficient license management be achieved without having to constantly put together new program packages to satisfy your customer's needs?

Everybody gets what they pay for – no more, no less.

The CRYPTO-BOX system provides a solution to all these issues. Now all you need is a full-featured version of your software. The CRYPTO-BOX key "decides" which parts of the application can be used, so creating a tailored application bundle for each customer requires very little effort. Do you want to limit the number of licenses, i.e. the number of workstations in a network allowed to run the software concurrently? No problem. We will configure the CRYPTO-BOX key according to your needs. The customer will benefit from this sophisticated distribution approach. He pays only for what he really needs.

Note

We've got solutions prepared for any distribution scenario

There's more ...

But that's not all! What if your customer requires a new configuration, additional program functions or simply more licenses? The CRYPTO-BOX key can be updated over the Internet using Remote Field Programming.

The diversity of applications of the CRYPTO-BOX system are an invaluable asset to your marketing department because it enables promising and innovative distribution strategies to be implemented. Simply give your customers a full-featured version of your software to evaluate so that they can convince themselves of the benefits of your product. With the CRYPTO-BOX software protection system, you are the boss. You determine the number of program starts or limit the evaluation period to 30 or 60 days.

The quickest wins

Would you like to use the Internet as a sales tool and exploit the potential of this flexible and modern sales channel, while reducing distribution and administration costs at the same time? With the CRYPTO-BOX system, you pull all the strings. The customer can download the application directly from your server and evaluate it. You determine what's possible and what isn't!

Why even bother with software protection ...

... I can hear some of you asking. Now be honest – would you jump from a height of 5000 feet without a parachute? Presumably not. If you rely on the legal enforceability of your licensing rights when distributing your software, that amounts to a "free fall" .. In practice, a license agreement is often not worth the paper it's written on. Control what your customer can do with your software with a CRYPTO-BOX and not a piece of paper.

Note

MARX ESD systems are a perfect support tool for your marketing and Customer Relation Management (CRM) staff.

Therefore

The CRYPTO-BOX system opens up all manner of possibilities. For example, "classical" hardware protection where the key has to remain connected to the computer all the time for the program to run.

But the potential of the CRYPTO-BOX system doesn't end there. In contrast to conventional protection devices, it can be kept continuously "up-to-date" via remote programming. This can occur over the Internet, by sending a password, or by phone. That's flexible Electronic Software Distribution in its purest form, allowing you to react quickly and easily to your customer's needs. That will put you one step ahead of the rest.

But that's not all the advantages of direct distribution. It also reduces costs. Shipping costs and customs duty no longer apply. Another side benefit: As soon as a customer contacts you to order an update or additional products, you receive reliable data on their needs and requirements. That's indispensable "fodder" for your marketing department and Customer Relation Management team. Only those who keep their finger on the pulse of their market and know their customer's requirements can look with confidence into the future of their business.

1.2 Now to specifics: ESD strategies

The Internet, as a distribution tool, opens up revolutionary new possibilities for software vendors who wish to practice Electronic Software Distribution.

The benefits of online distribution are obvious:

- Your application is available for downloading 24 hours a day, 7 days a week, with no interruptions or delays in the path of your customer. Marketing experts know that that's the key to B2C and B2B!
- The cost aspects are positive too: Your workload will sink dramatically. Thanks to automation, streamlining of your ordering and shipping process can be achieved - even for updates.
- There's more: You will be able to react flexibly and quickly to customer needs. Tailored solutions and services can be delivered with ease. As a result, customer satisfaction will grow. That gives you a decisive competitive advantage.
- With a smart strategy in place, your after-sales service can achieve considerable follow-up sales with a minimum of effort.
- The customer benefits from direct distribution. Updates are always just one mouse click away, and queries are handled promptly.

Where to from here?

The facts speak for themselves. We can offer you a suitable solution for any distribution scenario; typical activation by means of a CRYPTO-BOX hardware key, a software-based protection solution for vendors who sell large quantities, and tailored options for niche markets. Thanks to MARX's new License Activation Technology and the innovative License Activation Wizard (LAW), all options are available to you. You can integrate the LAW into your application, combine it with the CRYPTO-BOX system or operate progressive license management over the Internet.

1.2.1 Flexible distribution using the License Activation Wizard (LAW)

The License Activation Wizard (LAW) is a professional and reliable distribution tool that offers unbeatable value for the money, and it is an ideal tool for vendors who wish to sell applications over the Internet.

But that by no means exhausts the LAW's potential. A hardware-based distribution strategy, that can react flexibly and quickly to customer requirements, can also be implemented easily using the LAW.

And the LAW is extremely easy to use. During the software installation on the end user's PC, the LAW creates a hardware-specific identification code (PC-ID). If, after an optional evaluation period, the prospect decides to purchase the software, an automated process transfers this "identifier" to the vendor. The vendor then transmits to the customer's PC an activation code that will allow the application to be activated. This ties the program to the end-user's PC so that it will only run on that PC.

Your big advantage in using LAW: A profitable approach is often to distribute a "base package" - usually low-priced and aimed at "novices" - via the LAW and downloading over the Internet. If a customer goes on to purchase additional program options sold on a modular basis, or a professional version of the program, that purchase will always be accompanied by a CRYPTO-BOX. The CRYPTO-BOX is the "on-site license manager" - remotely programmable and uncompromising. So every additional license will guarantee more revenue!

- Your product is available around the clock. With just a few mouse clicks, customers can conveniently download your software from the Internet for evaluation purposes.
- You can react with lightning speed to new market requirements.
- Flexible marketing strategies are easy to implement.
- Customers can make a no-obligation evaluation of the capabilities of your product.
- The probability of follow-on sales is much larger because the inhibition threshold towards carrying out a quick, low-cost update is considerably lower.
- Customer problems can be solved with little effort. Customer satisfaction grows.
- Control over all licenses. The customer needs to contact you; this provides you with reliable data about your customer base, which is essential for future marketing strategies to be a success.
- The CRYPTO-BOX system will round off your marketing options. It offers the optimum solution - whether it be for USB, parallel or serial interfaces.

Phase I (1st step): Download and installation of evaluation version

Up-Front-Version Download



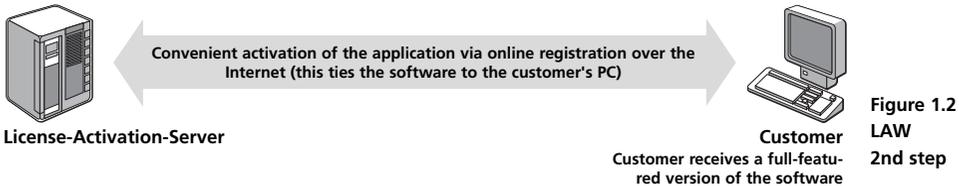
Note

The customer can try the software over an evaluation period (e.g. 30 days). During installation, the downloaded version is bound to the customer PC.

The diagram illustrates the diverse applications of the License Activation Wizard. If your preference is for software-based protection, then the following scenario is conceivable: Simply make available on your homepage an application that the customer can easily download and install ("Up-Front-Version Download").

Important: The software will not run until it is installed.

After successful installation, in which you can incorporate a registration step if your marketing strategy requires it, the software is ready to be evaluated. You determine for how long (e.g. 30 days) or how often, and with which functionality, the customer can run your application.

Phase I (2nd step): Activation after expiry of the evaluation period*Up-Front-Version Activated*

Once the evaluation period has expired, the customer must make a decision. If he wants to purchase the software, then he must activate it to be able to use the full functionality of the program (Up-Front-Version Activated). After successful (online) registration, the vendor supplies the customer with a special activation code, either online, by phone or by fax. This provides you with reliable data about your customer base, which is essential for future marketing strategies to be a success. After activation, the software is activated and tied to the PC, that is, the customer has access to the full functionality of your application but can only use it on this computer.

The licensing process could end at this point.

**Phase II (optional): Final activation using the CRYPTO-BOX key
After successful "Initial Activation"**

Up-Front-Version Final Activation



The License Activation Wizard has even more tricks up its sleeve. It also enables you to easily implement marketing strategies with an emphasis on hardware-based protection. After successful activation by the customer, it would be possible, for example, for him to have a "version with full functionality" but not be able to upgrade it. Alternatively, the customer might only be able to use limited functions of a modular application bundle.

Final activation of all functions occurs by sending a service release pack that includes a CRYPTO-BOX hardware key. The CRYPTO-BOX key overwrites the previous software activation.

1.2.2 "Classic" activation using the CRYPTO-BOX

The CRYPTO-BOX Software Protection System provides the answer to your urgent questions concerning software distribution and license management.

With the CRYPTO-BOX system, you have everything under control:

- You determine the number of workstations in the network on which your application can run.
- If desired, you can set an expiration date, an execution, or a time limit on your programs use.
- Remote programming simplifies the distribution of updates and follow-up business (e.g. additional licenses), and the costs involved are negligible!
- A single CRYPTO-BOX key can protect several applications. An entire network can be protected by a single CRYPTO-BOX key, and no hardware surcharge applies!
- Your marketing department benefits as well: You will obtain reliable customer data and be able to recognize market requirements early, thereby improving market information to assist your long term marketing strategies.

Note

Only achievable with hardware protection: Ship your software package on a single CD, regardless of how much functionality it contains and which functions the customer has ordered. The on-site CRYPTO-BOX, which you have preconfigured appropriately, determines which components your customer may use. You can change the programming at any time via the Internet to suit the changing requirements of your customer. The CRYPTO-BOX solution simplifies your distribution.



Using the CRYPTO-BOX system as a marketing tool allows you to design a flexible marketing strategy. You provide software that the customer can easily download from a website and install. You determine how long, how often, and with what functions, the customer runs your application.



Figure 1.4
A typical example of
a smart ESD strategy
- implemented with
the CRYPTO-BOX key

At the end of the evaluation period, the customer needs to make a decision. If he buys the application, you send him a CRYPTO-BOX hardware key to convert the software to a full-featured version.

Further advantage: This allows mobile use of the application, e.g. on a laptop, but only if the CRYPTO-BOX key is connected.

1.2.3 Site License Model

Ideal for companies, departments, organisations

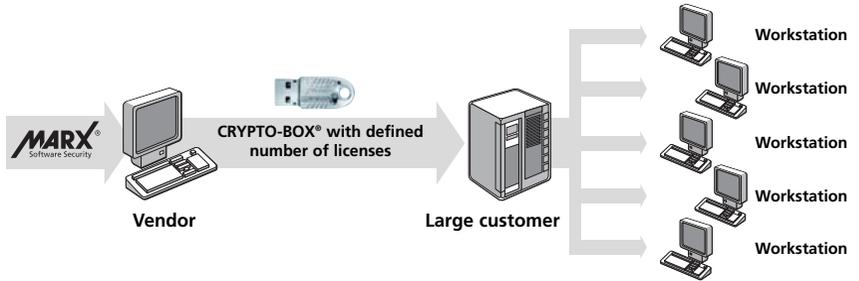
The CRYPTO-BOX system offers a perfect distribution solution for software vendors who supply modular programs to companies or company departments.

The CRYPTO-BOX acts as a license container, i.e., a fixed number of licenses are programmed into the CRYPTO-BOX memory. The customer can install the software on multiple workstations in the network, as required. There is also an option where the customer can activate as many licenses as he wants over a specified period - e.g. for one year.

The CRYPTO-BOX system provides absolute flexibility:

- You can react to specific customer requirements. You provide the customer with an agreed number of licenses. The customer only pays for what he really needs, and you can offer your product at a competitive price. An unbeatable argument for your marketing department.
- Lucrative follow-up business can be conducted with minimal effort via Remote Field Programming (RFP). Does the customer require additional licenses or an update? No problem. The CRYPTO-BOX's memory can easily be programmed remotely via the Internet. No more costly on-site "maintenance work".
- Expensive mailing campaigns become a thing of the past. Customs duty and shipping costs are reduced significantly. The customer is no longer required to return the hardware key
- The CRYPTO-BOX can also be used in networks (TCP/IP or Novell). This is done by connecting the CRYPTO-BOX key to a PC or server on the network and starting the CRYPTO-BOX Server. The number of licenses per network can be set using the optional License Control System tool (available on the CBU XS, CBS, and CBN keys). One CRYPTO-BOX per network is all that's needed.

Figure 1.5
This diagram shows a CRYPTO-BOX being used as a "License Container" in a "Site Licensing" scenario.



The diagram illustrates the applications of a CRYPTO-BOX system that is being used as a distribution tool by a vendor who supplies large customers or specific company departments. For example, if a company orders 50 single-seat licenses you send them a single CRYPTO-BOX. The memory of the hardware key acts as a "License Container"

The customer can now utilize the 50 licenses in a flexible manner. If he needs additional licenses - no problem! You can perform the update conveniently from your computer by means of Remote Field Programming. No more after-sales deliveries or maintenance is required.

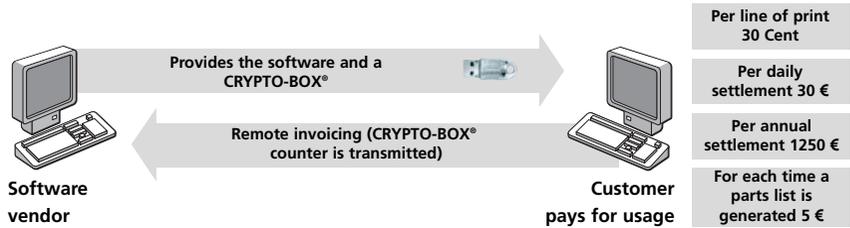
1.2.4 Pay-per-Use or software leasing model

In the TV broadcasting arena it's already commonplace: If a customer wants to see a particular program, he must pay to do so. The same can apply to software distribution. Rather than paying for the application, he pays for the number of program starts, for the amount of time he uses the software, or for the functionality that he works with. In short: Intensive users pay more.

A leasing model such as this places particularly high demands on the protection scheme is obvious. To guarantee fair invoicing, the data must be absolutely safe against tampering. Should the process ever be cracked, the product is as good as "dead". To prevent this from happening, install an execution counter in the memory of the CRYPTO-BOX.

This distribution model opens up many promising possibilities for the vendor:

- The application doesn't need to be purchased, so only usage charges arise, the purchasing barrier for the customer is very low.
- Discount schemes for frequent users increases the "purchase incentive".
- Ongoing application updates increase customer satisfaction and turnover.
- The "account" is easily transferable. Follow-up business is increased.



The diagram illustrates a possible Pay-per-Use distribution model. You send the customer your software and a CRYPTO-BOX. The application will only run if the CRYPTO-BOX is connected to the customer's computer.

The advantage for the customer: He pays only for what he uses, or the number of program starts. The "usage profile" is stored in the CRYPTO-BOX key's memory. At the end of the month the CRYPTO-BOX's counter value is transmitted to you - secure and fast via the Internet. This guarantees fair invoicing.

1.3 No software protection = No control over licenses!

Those vendors who sell large numbers of applications are faced with the same dilemma again and again: "How do I secure my returns without driving up the price of my software by implementing an expensive license protection solution?". Experience shows that business software, in particular, is often a popular "target" because of the difficulty of restricting the number of users on a network.

"Buy once, copy many times", as they say. Software pirating is regarded as a petty offense - a petty offense that will cost you dearly. Besides lost additional revenue from the purchase of additional licenses, your marketing department's work suffers as well. Registration is subject to the good will of the purchaser and so reliable customer data is not available. Your long-term marketing strategy stands on shaky ground from the outset, and so too the future of your company. This illustrates that the legal solution, i.e. a license agreement, is a blunt weapon in the fight against software piracy. A long-winded lawsuit with an uncertain outcome is very costly, and besides, who wants to sue their own customers?

The gaming industry in particular can tell you a thing or two about the other side of the coin. Many products when launched on the market are immediately outdated because the copy protection has already been cracked by "fan communities". But even the supposed beneficiaries, the end-users, are faced with disadvantages. As soon as they open the packaging, they are required to buy the product. A no-obligation evaluation is not possible. That can drive even honest people toward piracy.

1.4 Conventional software protection is inflexible

If you choose a conventional protection method, you could be backing yourself into a corner over the long term. You may well be protecting your software against piracy, but it won't support a state-of-the-art marketing strategy or allow control of the distribution of your product. The fact is: simple license protection is inflexible.

Thanks to the CRYPTO-BOX Software Protection System, we can offer you a ready-made solution for any scenario, from software-based protection schemes right through to a pure hardware implementation. All variants have one thing in common however: Licenses can be updated at any time via Remote Programming over the Internet. You can authorize additional seats in a network or activate further features of the program, and so react quickly and flexibly to your customer's needs while saving costs.

2. If effective license management is your goal

To put an effective license management plan into action, reliable software protection is essential. In this chapter we will present our wide range of products which offer a suitable solution for every protection and distribution strategy.

2.1 Pirated copies and multiple users result in loss of sales

Nobody knows it better than you. The distribution of unprotected software means:

- Reduced sales
- Lower profits
- Smaller budget for research and development
- Customers often buy just one copy, but use it umpteen times
- No control over circulation
- No control over usage in a network
- No protection in a number of countries
- Lower number of updates sold

Unauthorized copying of software is much more than just a "petty offense". It has a negative impact on research and development and sales because it significantly curtails your returns and profits.

What disadvantages result from the international distribution of unprotected products?

The fact is: Pirated copies lead to considerable loss of revenue when distributing internationally. In the worst case scenario, a single pirated copy of a program will be enough to render your new software product obsolete in a particular market segment.

Experienced international software vendors know that strong copy protection is an effective distribution tool for curtailing the damage caused by software piracy. Customers choose the CRYPTO-BOX whenever a truly reliable, hardware-based copy protection solution is required.

2.2. Professional software protection secures revenue

The CRYPTO-BOX – Your guide in insecure markets

Only by protecting your software you, can guarantee that every single user has paid for your software and your intellectual property. Not to mention follow-up sales resulting from license updates. Thanks to the Remote Programming capability of the CRYPTO-BOX solution, you can react quickly and flexibly to your customer's needs and keep shipping and administration costs to a minimum.

Furthermore, every purchaser is required to register, at least when it comes to obtaining updates, which is an invaluable advantage for your marketing department! This is the only way to obtain the data that is essential for a long-term company strategy. All these advantages are unimaginable without protection.

You know...

Written agreements and licenses are not sufficient to prevent unauthorized use of your product. Furthermore, to enforce them you would need to sue your own customers! Using the CRYPTO-BOX system, contract violations or license abuses simply won't occur.

The advantages for software manufacturers:

- All users pay for the software
- You know who your customers are
- Every customer is registered. An enormous marketing advantage!
- Your support service is only accessed by paying customers
- Secure demo versions; available 24 hours, 7 days a week via Internet download!
- Updates - only to authorized users - ensure a constant revenue stream
- Controlled usage in networks

- Every additional license means more revenue
- Secure (remote) upgrades and remote activation
- Not only programs, but also whole databases can be protected
- Fast protection using Auto:Crypt - no source code required
- Support for all compilers
- Virus and tampering recognition
- Professional anti-debug functions and protection against disassembly
- Checksum and hash functions

2.3 Overview: Two implementation paths

The PPK (Professional Protection Kit) features the **MARX Programming Interface (MPI)**. It incorporates all products "under one roof" and enables our customers to communicate with a variety of protection devices and products using the same functional calls and commands! Using MPI, expensive adaptation of different protection devices to different computer interfaces is no longer required. Furthermore, it supports all commonly used crypt-algorithms, including AES/Rijndael, RSA, MD4/5, Blowfish, IDEA and conventional hash algorithms.

Executable files can be protected in a snap using the **AutoCrypt Wizard**, which is intuitive to use, even for beginners. You install the PPK, start the AutoCrypt Wizard and then select the desired protection options: With just a few mouse clicks you can limit the number of program starts by means of an execution counter, set an expiration date, or prompt the user to enter an additional password at program startup.

The large - programmable on-the-fly - memory of each CRYPTO-BOX or CryptToken key gives the option of storing a large variety of data, thereby increasing the level of security even more:

Benefits of the large CRYPTO-BOX memory - ideal for storing:

- Whole program segments
- Pointers and vectors - without which the program will not run
- Installation information
- Configuration of the customer's hardware
- Serial numbers and customer-specific data

Note

MARX products are always state-of-the-art. Our R&D departments participate in the software beta testing programs of Microsoft and many other vendors!

- Checksums: Tampering recognition!
- Recognition of virus attacks on EXE or DLL files
- MAAS protection system for multiple applications per individual device
- Important configuration data for your application
- Keys and X.509 certificates

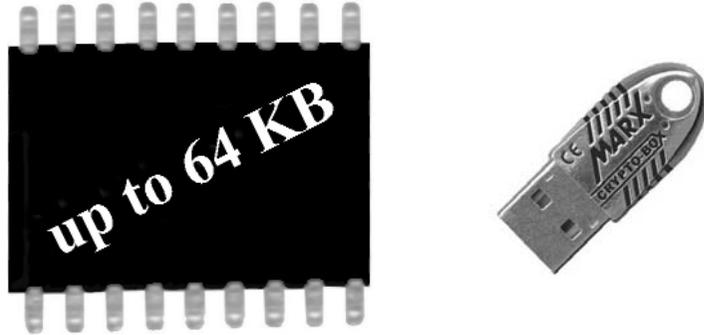


Figure 2.1
CRYPTO-BOX with up to
64Kbyte of memory

Extensive compiler and operating system support

With several hundred libraries, ActiveX objects, DLLs, etc. available, our R&D department ensures that you receive the required support for your favorite compiler.

2.4 Software protection products tailored to your requirements

Note

For current prices and descriptions, see our product catalogs at www.marx.com.

Hardware-based keys – external

The CRYPTO-BOX system is comprised of several hardware variations:

For a USB interface:

- CRYPTO-BOX USB XS - the shortest USB key on the market. But with the AES/Rijndael algorithm implemented fully in hardware.
- CRYPTO-BOX USB XL - with True White Noise Generator for random numbers

The CRYPTO-BOX supports Windows XP / 2000, NT4, Me/98 as well as MacOS and Linux / Unix / Solaris.

For a parallel interface

- CRYPTO-BOX 560/Net - key for stand-alone or networked PCs; remote programming, 560 Bytes of memory
- CRYPTO-BOX Versa - Ideal for stand-alone PCs; remote programming, 64 Bytes of memory

For an RS232 serial interface

- Ideal for Linux/Unix, Windows and special environments, 4KBytes of memory, compatible with the CRYPTO-BOX

Note

For information on the CRYPTO-BOX extension card, see the CB-Card documentation which is available separately or visit us at www.marx.com.



Figure 2.2
Parallel CRYPTO-BOX
Figure 2.3
Serial CRYPTO-BOX

2.4.1 CRYPTO-BOX hardware protection**Hardware-based keys - internal**

- Crypto-Card PCI: With or without integrated CRYPTO-BOX
- Crypto-Card ISA: Similar to the "PCI" model, but for older PCs

Typical applications for the CRYPTO-BOX Card models:

- Where there is a risk of external keys being stolen (schools, department stores, computers in public places),
- To accommodate several keys inside the PC (including those of other manufacturers).

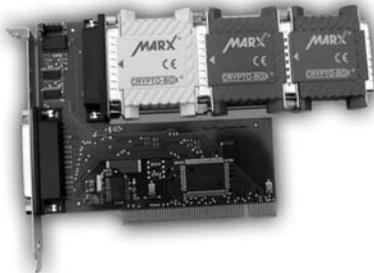


Figure 2.4
Crypto-Card

2.4.2 Software support, license management, remote upgrades

LCS® License Control System (only available with CRYPTO-BOX USB XS/XL and 560/Net parallel)

- The License Control System (LCS) can be used to set the maximum number of computers in a network on which a certain application is allowed to run concurrently.
- Can be used in conjunction with the CRYPTO-BOX USB and 560/Net models.

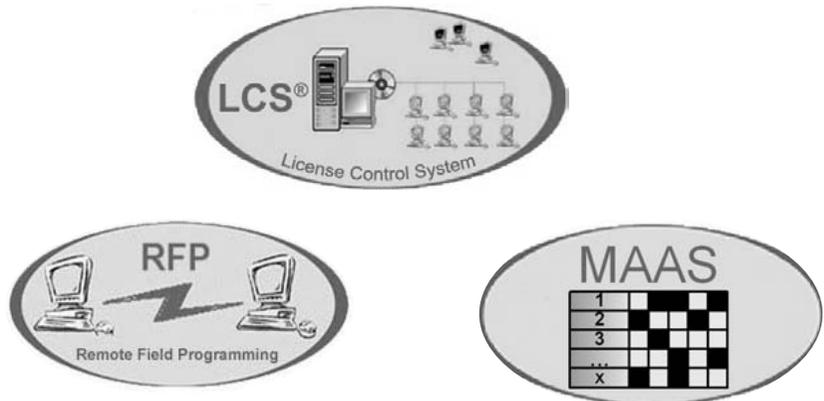


Figure 2.5

RFP - Remote Field Programming

- Enables secure, remote programming of the memory contents and the freely programmable ID codes (passwords) of a CRYPTO-BOX.
- An ideal tool for activating program components, installing upgrades or updating arbitrary components of the CRYPTO-BOX at the customer site.

MAAS – Multiple Application Authorization System

- Allows multiple applications to be protected by a single key.
- Can be used to individually restrict the usage of a certain program. You can specify an expiration date, or a counter-based limit on the number of program starts.
- Ideal for use in conjunction with **LCS**.

2.4.3 Expertise, consulting, OEM solutions

On request, we can put together for you a comprehensive package – if necessary tailored to your individual requirements:

- Customer-specific development for non-standard environments
- OEM solutions, casing in company colors or with embossed logo
- Even development of complex hardware solutions: Our developers make "the impossible possible"!

Take advantage of our expertise and experience. Why not give us a call!

3. Overview: CRYPTO-BOX Systems

On the following pages we will describe our products and provide you with a decision making tool to enable you to select a suitable CRYPTO-BOX solution for your marketing strategy.

3.1 MARX Product Selector

The **MARX Product Selector** offers a quick overview of our products. Detailed descriptions of the products, and examples, are provided in later sections of this user manual.

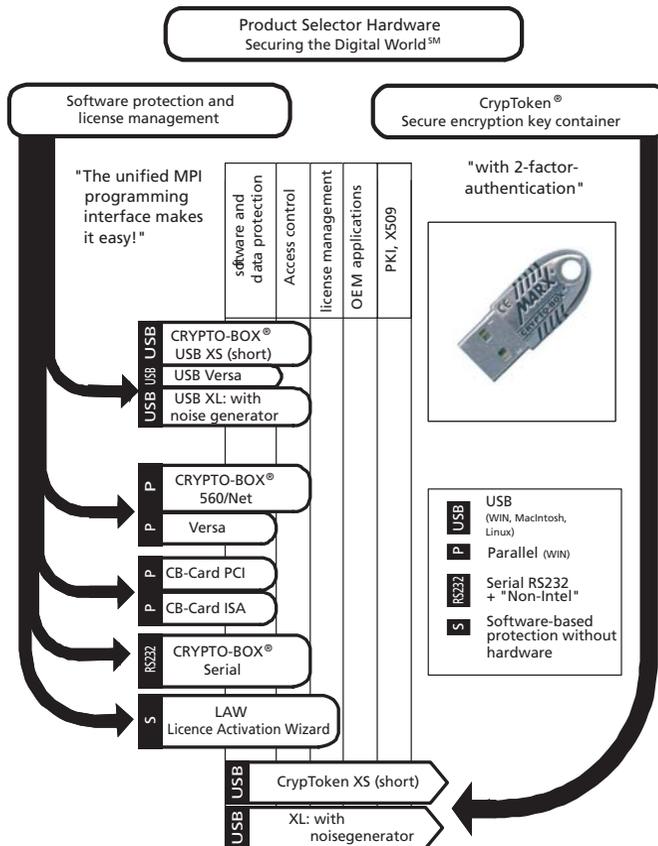


Figure 3.1 MARX Product Selector

3.2 CRYPTO-BOX USB

This hardware key offers all the functions you need for reliable plug-n-play software protection on an USB interface. It can be integrated seamlessly by means of the MARX Programming Interface (MPI) libraries.

Note

Now available: The first hardware key implementation of the AES/Rijndael algorithm, "True Physical Noise Generator" and RSA support (key length: 2048 Bit)! And up to 64 KByte of memory. The CRYPTO-BOX USB supports Windows XP, Windows 2000, NT4 and ME/98, as well as MacOS and Linux.

3.2.1 Differences and functions of the USB models

The CRYPTO-BOX USB is available in three different versions:

- CRYPTO-BOX USB Versa (CBU/VS): The model with all the important software protection functions but with more than just the base features. The table below provides an overview of the features of the CRYPTO-BOX. It is the shortest USB key in the world - so small that it fits easily into a laptop carry bag.
- CRYPTO-BOX USB XS (CBU/XS). The model with the same short casing, but with more functions for software and data protection. Optionally available with up to 64 KByte of memory.
- CRYPTO-BOX USB XL (CBU/XL). This model offers the maximum number of security functions. The true white noise generator (hardware implementation) guarantees additional security. The USB XS and XL models contain a software implementation of the RSA algorithm.

Product examples (also available with OEM casing):



Figure 3.2
CBU XS and
Figure 3.3,
CBU XL

Type	CRYPTO-BOX® USB Versa	CRYPTO-BOX® USB XS	CRYPTO-BOX® USB XL
Short description	U/VS	U/XS	U/XL
special feature	Shortest USB key on the market! Ideal for notebook- and laptop users.		True White Noise Generator
Microprocessor	+	+	+
Unique serial number		+	+
True white noise generator			+
Length	approx. 17 mm	approx. 17 mm	approx. 27 mm

Table 3.4.
Overview

The most important features of the CRYPTO-BOX USB models:

- On-board encryption of data by means of the Rijndael algorithm (Advanced Encryption Standard (AES), official successor to the DES algorithm) with a key of length 16 Bytes that never leaves the hardware platform, in OFB bit-stream cipher mode (Output Feedback Mode)
- The software-based authentication method supports the RSA standard (key length: 2048 Bits)
- Access control (PIN-based)
- Every CRYPTO-BOX USB (except USB Versa) is assigned its own unique serial number
- CRYPTO-BOX USB XL with integrated True Physical Noise Generator (random number generator, used for the purpose of key generation)
- Encrypted EEPROM with 4 KByte of on-board memory (up to 64 KByte available on request)
- Reliable communication and key identification ("on the fly" when plugged in/unplugged)

Extra features:

- Metal casing - effective protection against incoming and outgoing radiation
- Unique serial number (XS and XL models)

Applications of the CRYPTO-BOX USB:

- Software and data protection

When you purchase the CRYPTO-BOX USB XL, you are getting a product which offers the highest level of protection available. The integrated **True Physical Noise Generator** fulfills a frequently voiced request of software developers - the generation of non-computable and non-sequential random numbers.

The CRYPTO-BOX models support officially certified/recommended public algorithms, in particular AES/Rijndael and RSA (XS and XL models). The algorithms are generally regarded as undecryptable by means of realistic computing power (10^{40}).

The user can decide whether he wishes to use the integrated **True Physical Noise Generator** to generate keys and passwords, or to define his own sequences.

Thanks to the high data throughput of the stream cipher mode (the most effective implementation of the Rijndael algorithm) and the high transfer rates of the USB interface, this hardware key gives users the ability to encrypt large data packets.

On account of its large internal memory (up to 64 KBytes), the CRYPTO-BOX USB can be utilized as a secure, portable storage medium for sensitive data.

MARX customers can either use the pre-programmed keys and passwords (16 Byte each and preset by MARX) or generate his own codes. When own codes are used, only the customer can access encrypted data.

Secure memory - TOKEY™ Secure Password Manager

The secure memory of the CRYPTO-BOX USB key makes it highly versatile. You can store customer names and/or copyright information as text in the key, which is then displayed within your application.

Safeguard important constants used in calculations performed by your program.

TOKEY Secure Password Manager provides a good example of how the secure memory of the CRYPTO-BOX USB/CrypToken USB key can be used. This small tool

manages your passwords and PIN numbers by storing them in the CBU/CT memory and safeguarding them against unauthorized access or viruses! You can find the "Secure Password Manager" in the \SPO directory of the MARX PPK CD-ROM.

3.2.2 CRYPTO-BOX USB Implementation

The CRYPTO-BOX USB provides a set of passwords that are either predefined (allocated by MARX) or programmable on-the-fly by MARX customers. Fixed passwords guarantee that the hardware key is unique for each MARX customer. User-defined passwords ensure that only the software vendor/distributor has full knowledge about accessing the CRYPTO-BOX. The validation of all passwords occurs within the firmware in the form of parameters to API functions.

All keys, initialization vectors and passwords are 16 Bytes long and are implemented within the hardware, which means they never leave the CRYPTO-BOX hardware after they have been set. There is no firmware-based means of extracting these elements from the key. The generation of preprogrammed passwords and keys is assisted by the integrated True **White Noise Generator** which guarantees that only non-sequential number sequences are produced. The user can decide whether he wishes to use these components or define his own sequences.

3.3 The CrypToken®

Note

For more information about the CrypToken, see the CrypToken user manual, which is available separately, or visit www.marx.com.

The functions integrated into the CrypToken

- Secure storage of passwords, certificates, digital signatures and lots more
- A hash function
- Digital signature
- Virus and tampering recognition
- Two-factor authentication

3.3.1 Authentication and data transfer

For more information on this subject, see the CrypToken user manual, which is available separately, or visit us at www.marx.com.

3.4 Parallel keys

3.4.1 CRYPTO-BOX 560/Net

The **CRYPTO-BOX 560/Net** key is designed to be used on either a stand-alone computer or a Novell, NetBIOS or TCP/IP network. If your application is running in a LAN, you can configure the number of concurrent users and only one CRYPTO-BOX 560/Net is needed on the server.

Features of the CRYPTO-BOX 560/Net

- Equipped with an 8-bit secure microprocessor
- Supports Windows XP, Windows 2000/NT, Me/9x, 3.1x, DOS and OS/2
- 576 Bytes of user memory, 483 bytes of which are programmable on-the-fly
- AES and IDEA crypt-algorithm

- Two separate memory areas with access codes that can be programmed on-the-fly
- Eight ID codes, each 5 bytes long, five of which are programmable on-the-fly

3.4.2 CRYPTO-BOX Versa

The CRYPTO-BOX Versa key is the most popular key among programmers who need high level security, but not the network capabilities of a product like the CRYPTO-BOX 560/Net.

Features of the CRYPTO-BOX Versa

- Equipped with an 8-bit secure microprocessor
- Supports Windows XP, Windows 2000/NT, Me/9x, 3x, DOS and OS/2
- 64 Bytes of user memory, 50 bytes of which are programmable on-the-fly
- IDEA crypt-algorithm
- Two separate memory areas with access codes that can be programmed on-the-fly
- Four ID codes, each 5 bytes long, two of which are programmable on-the-fly

3.5 Serial keys

3.5.1 CRYPTO-BOX Serial

The CRYPTO-BOX Serial key is an ideal distribution tool for software applications running on PC or UNIX platforms because it provides full control over the license management of your software.

This model supports similar features to the CRYPTO-BOX USB and can be accessed via the MPI.



Figure 3.3
CRYPTO-BOX Serial

Features of the CRYPTO-BOX Serial

- 9 Pin/9 Pin connector
- Adaptable to any RS232 and RS423 compatible system
- Supports Windows XP, Windows 2000/NT, Me/98 and Linux/UNIX
- Support for other platforms available on request
- 4 KBytes of user memory, up to 64KB if required
- **AES/Rijndael crypt-algorithm**

4. Support, information and more ...

Whether it's a missing driver CD, a hardware problem, or the need for customized barcodes printed on your CRYPTO-BOX: This is your "first port of call" - and we respond promptly.

We know that hardware is not everything! That's why we offer a comprehensive service that includes, amongst other things, the following: Internet downloads of new software versions, hotline support for programmers and fast delivery. On request, we can also supply customized models with individualized company logos, adhesive labels or overprinting, etc. So your Corporate Design will get a showing too!

4.1 Current product information and downloads on the Internet

On our website www.marx.com, you'll find online price lists for our entire product range and up-to-date information on protecting software using the CRYPTO-BOX system. We also provide free software upgrades for the CRYPTO-BOX. Product information and complete user manuals are also available for downloading.

4.2 Technical support - Hotline, FAQ

MARX offers prompt technical support

In North America:

- Phone support is available by calling **1-770-986-8887**
- Fax us your question to **1-770-986-8891**
- You can reach us by e-mail at **support@marx.com**

In Europe and worldwide:

- Phone support is available by calling **+49(0) 8403-92950**
- Fax us your question to **+49(0) 8403-1500**
- You can reach us by e-mail at **support-de@marx.com**

4.3 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.

4.4. Just-in-time distribution

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).

4.5 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 4.1) or serial numbers, laser printing, right through to adhesive labels or overprinting. Just ask us - we would be happy to provide a quote.



Figure 4.1
and Figure 4.2
Data Matrix Code (l.)
and Serial number (r.)

5. Fundamentals

On the following pages we will show you just how simple and easy it is to implement our products.

5.1 The Professional Protection Kit

5.1.1 MPI and the TEOS Professional Protection Kit

MPI enables you to communicate with any CRYPTO-BOX model via a common programming interface. This makes it quick and easy to switch from one CRYPTO-BOX type to another, or to implement multiple CRYPTO-BOX models at a time. MPI supports all features of the CRYPTO-BOX system you need to produce secure **software and data protection solutions**:

- License control by means of a CRYPTO-BOX installed locally or on the network,
- Choice of automatic (no programming knowledge required) or manual implementation of the CRYPTO-BOX protection system,
- Utilization of the internal memory and the hardware-based encryption functions of the CRYPTO-BOX,
- Easy creation of data objects for licensing options such as expiration days, expiration date or execution counter,
- Remote maintenance.

TEOS (Token Embedded Operating System), a new "operating system" specially tailored to the features of the CRYPTO-BOX USB, is also available. TEOS allows you to run several applications concurrently and independently on a single security token. This is achieved through intelligent file management and a sophisticated programming interface.

A variety of applications can profit from TEOS **with no programming at all**. Besides having an "encrypted partition" on the hard drive, you can also run a Single Sign-On system for Windows and a password manager simultaneously.

Note

The PPK CD is suitable for use under Windows XP, Windows 2000/NT4 and Me/98. You will need administrator rights to install the Software Developer's Kit under Windows XP, Windows 2000 or NT4. Support for MacOS is also included on the PPK CD (for more information, see the file 'mac user.txt' in the top directory of the CD). Support is also available for DOS and Linux: Our technical support (see Page 35) will be happy to help you further.

Multiple applications can access the CRYPTO-BOX internal memory concurrently without conflicts arising. Each application is assigned its own partition and cannot access memory areas that are already reserved for another application. The memory can be partitioned as needed by the developer; the relevant tool is supplied with the TEOS Developer Kit.

Furthermore, TEOS allows different access hierarchies to be defined, like free access, user access or administrator access.

The generation of **AES/Rijndael private keys** within the CRYPTO-BOX USB can occur directly via API functions. A convenience for software developers is that the same function calls are used under Windows, Linux and Mac OS, which means minimal porting costs to other platforms. This is also the option to create RSA keys via a software driver.

TEOS, when used in conjunction with a CRYPTO-BOX as a security token, offers a modern, cost-effective and powerful alternative to SmartCards. The key feature of TEOS is its versatility in supporting a large variety of applications.

For more information about TEOS, visit us at www.marx.com, or refer to the TEOS Developer's Kit, which is available separately.

5.1.2 Professional Protection Kit (PPK) Installation

The CRYPTO-BOX Developer's Kit contains all the necessary hardware and software to implement a software protection system based on the CRYPTO-BOX:

- A CRYPTO-BOX programmed with demo codes
- A user manual with a complete reference of all available functions (also available in PDF format on the CD)
- An installation CD containing protection and configuration tools and libraries for Windows, MacOS and Novell NetWare

To install the software (Important: Do not attach the CRYPTO-BOX to your PC at this time!):

The installation program comes with a user-friendly graphical interface. Insert the PPK CD into your CD-ROM drive. On Windows, the CD will autorun and the menu shown below will appear.

If the MARX setup program is not started automatically from the CD, click **RUN** in the Windows **START** menu and enter: **x:\setup.exe** (where "x" is the drive letter of your CD-ROM-drive).

On Windows, a device driver must be installed before any communication with the CRYPTO-BOX can be established. Click "**Evaluate CRYPTO-BOX Software Protection System**" to install the **Professional Protection Kit (PPK)** for your type of CRYPTO-BOX - this will automatically install the required driver. For more information about the driver installation, and how to install the device driver on your customer's PC, see Section 6.1 "Distributing protected applications" (page 77).

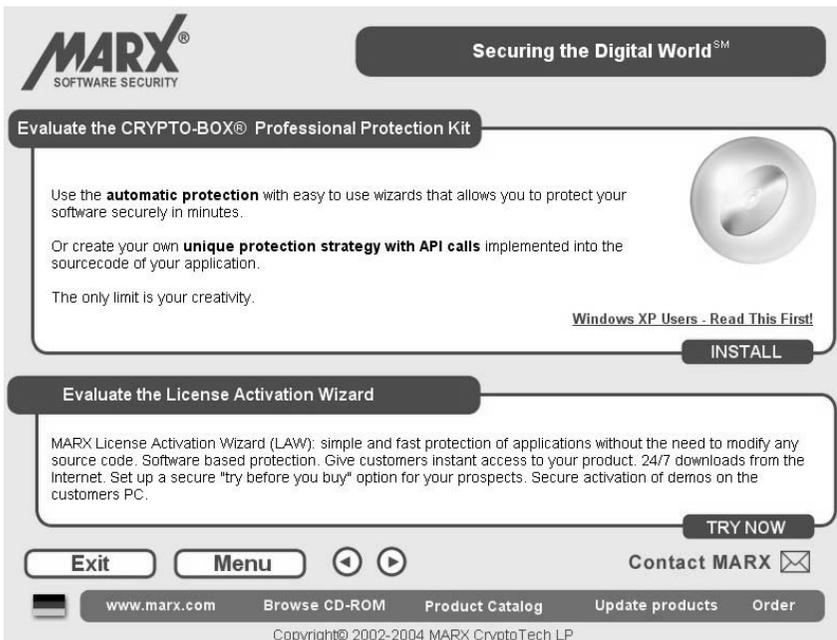


Figure 5.2
PPK-Installation

After installing the PPK, we recommend that you start the Control Center which provides an overview of the PPK components and their function. Now plug the CRYPTO-BOX that delivered with your Evaluation Kit into the appropriate port (USB or parallel or serial).

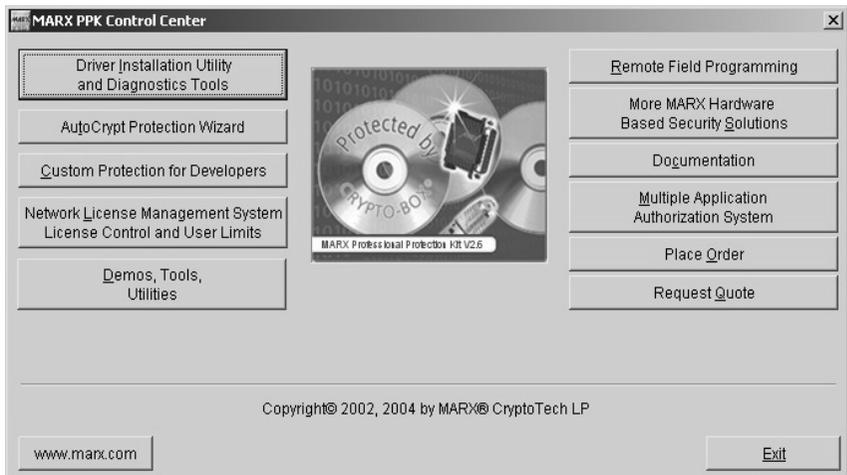


Figure 5.3.
PPK Control Center

Parallel port (CRYPTO-BOX 560/Net or Versa):

- Plug the CRYPTO-BOX into the parallel port of your PC - you can start using it immediately.

USB port (CRYPTO-BOX USB):

- Plug the key into the USB port of your PC
- The Windows Hardware Wizard appears
- Click **Next** to start the automatic driver installation
- Windows locates the driver and installs it automatically
- On Windows XP/ 2000/ NT4, you won't need to restart the computer after installation of the driver

Important note for Windows XP users:

In the Windows XP Hardware Wizard, Microsoft have added a new dialog box that may be confusing (see next page). Please disregard this message. The CRYPTO-BOX USB/CrypToken driver has been tested and is compatible with Windows XP. Click "Continue anyway" to proceed with the driver installation.



Click here

Figure 5.4.
Hardware Installation

5.1.3 Software protection with AUTO:CRYPT

The CRYPTO-BOX provides two methods for protecting your software:

- The **AUTO:CRYPT** module (a component of the AutoCrypt Wizard) is used for automatic protection of 32-bit Windows applications.
- A **high level API** (Application Programming Interface) allows you to manually implement the protection system by means of 25 easy-to-call CRYPTO-BOX functions.

Note

16-Bit Windows applications can be protected using the CRYPTO-WIZARD Rel. 2.19 (CRYPTO-BOX 560/Net and Versa only), which is available on request.

5.1.4 AutoCrypt: Automatic software protection for EXEs/DLLs

The **AutoCrypt Wizard** is a user-friendly software protection tool. In the following sections we will describe how to use the AutoCrypt Wizard with the CRYPTO-BOX USB.

The **AutoCrypt Wizard** can be used with all currently available MARX products.

Automatic protection of applications

The powerful AUTO:CRYPT tool, a component of the **AutoCrypt Wizard**, provides an intuitive-to-use environment for fast implementation of software protection systems. AUTO:CRYPT provides the easiest and fastest means to protect a 32-bit

Windows program without accessing or modifying its source code. AUTO:CRYPT wraps your program in a binary protection shell and injects code into the application for further security.

The **AUTO:CRYPT** method is particularly suitable for standard software, where the source code is not available. If no CRYPTO-BOX is connected to the computer, the program will not run and will display a user-defined message.

The AutoCrypt Wizard is installed so that all passwords are initialized correctly within the hardware configuration profile for Evaluation CRYPTO-BOX. To reinitialize the AutoCrypt Wizard for use with your customer-specific CRYPTO-BOX, import the hardware configuration profile supplied with the CRYPTO-BOX you received from MARX.

Selecting an AutoCrypt project

The AutoCrypt Wizard uses projects to manage your implementation of various automatic protection options or mechanisms for a CRYPTO-BOX. You can use an existing project or create a new one.

To create a new project

Start the AutoCrypt Wizard (Start/Programs/MARX Software Security/Professional Protection Kit MPI/Tools), and choose whether you want an animated assistant to be displayed (the assistant is only available under Windows XP/ 2000).

Click **Next** to proceed to the **Select project** screen. The following screen appears:

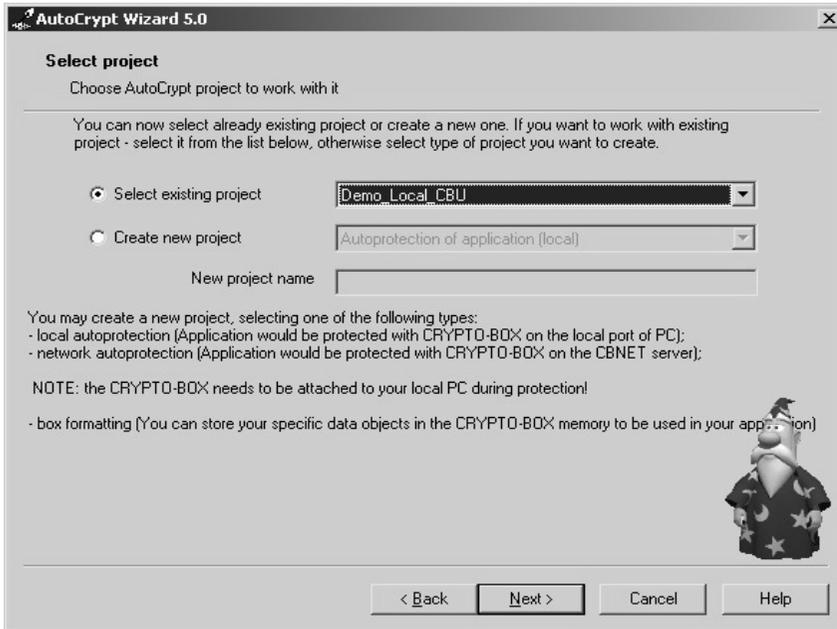


Figure 5.5.
Project selection screen

- Either select an existing AutoCrypt project or create a new one.
- If you chose to create a new project, you also need to select the project type (Autoprotection of application (local), Autoprotection of application (network) or Data objects programming).
- Click **Next** to continue.

Note

If you still do not have a hardware configuration profile (.MRX file) for your customer-specific CRYPTO-BOX, please call or email us and we will send it to you. The advantage of the hardware profile is that it prevents you from entering the required ID codes manually.

Tip

If you need help for a particular option, simply move the mouse pointer over the option to display some explanatory text.

Selecting a hardware configuration profile

To select or import a hardware configuration profile:

- The hardware configuration profile (.mrx file) contains the fixed passwords and ID codes (encrypted), which are needed to access your CRYPTO-BOX. MARX will supply a suitable hardware configuration profile for your customer-specific CRYPTO-BOX. Import a hardware configuration profile by choosing "Import" or drag and drop it on the screen using the mouse.
- If you have a CRYPTO-BOX Evaluation Kit, select a suitable profile for your CRYPTO-BOX from the list. For example, for the CRYPTO-BOX USB you would select the profile **CBU_Demo**. All passwords will now be initialized with the values of a standard CRYPTO-BOX USB (CBU) Evaluation Kit.

for the CRYPTO-BOX USB	select CBU_Demo
for the CRYPTO-BOX 560/Net	select CBN_Demo
for the CRYPTO-BOX Serial	select CBS_Demo
for the Smartcard (SX7)	select SX7_Demo
click Next	

Table 5.1
Hardware configuration profile.

- All passwords will be initialized with suitable values for the particular type of CRYPTO-BOX included in your evaluation kit.

The following figure provides an overview of which MARX applications are able to use the .MRX hardware configuration profile.

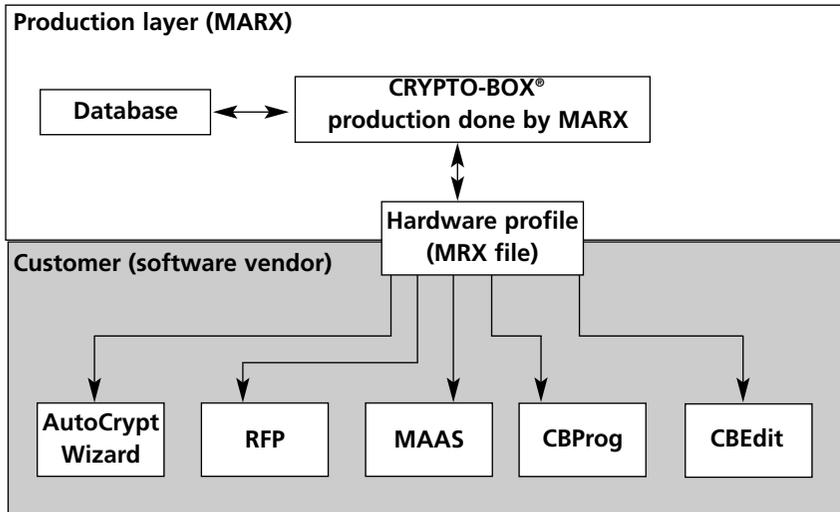


Figure 5.6
Applications

Selecting a hardware configuration profile in the AutoCrypt Wizard:

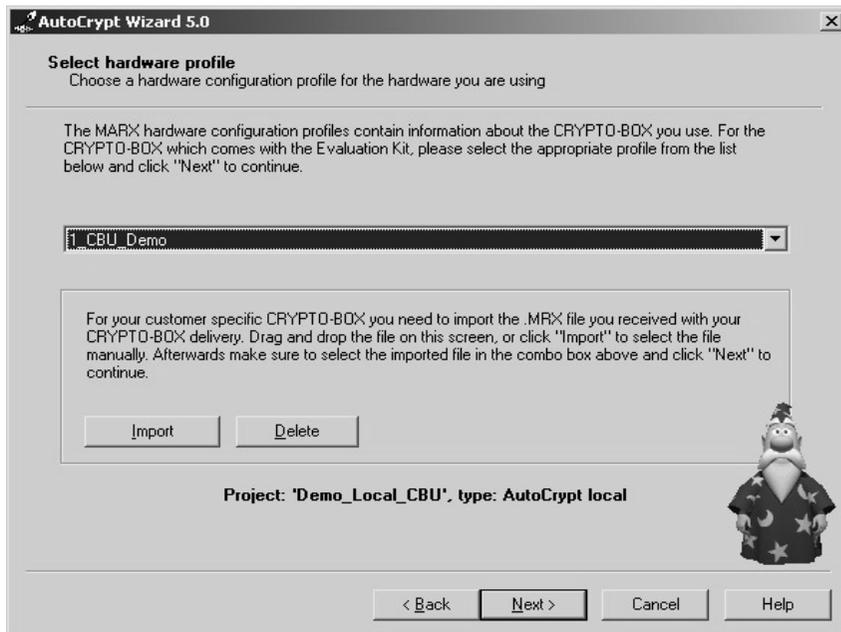


Figure 5.7
Hardware profile
selection

Click **Next** to continue.

Configuring the CBNET Server

This screen appears if you have selected the project type **Autoprotection of application (network)**.

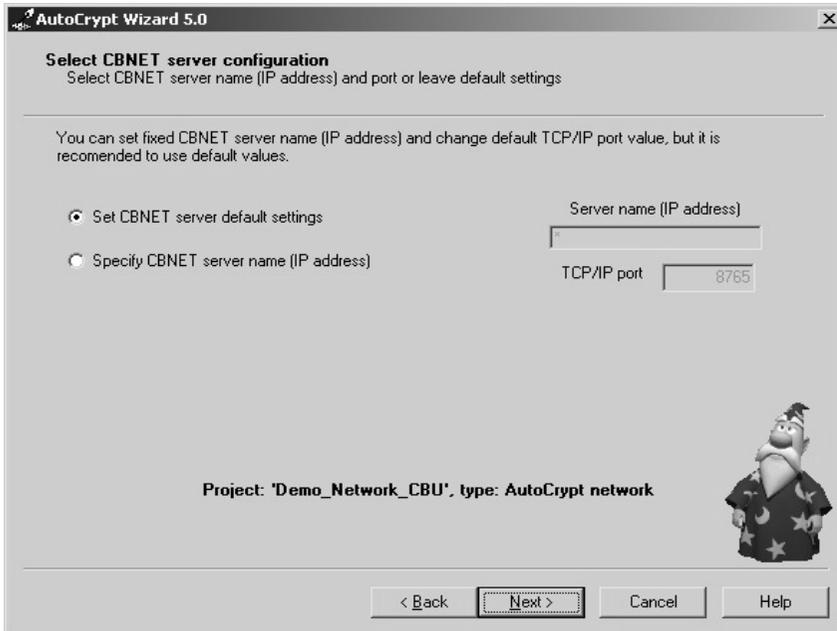


Figure 5.8
CBNET server
configuration

Either accept the default settings (adequate in most cases) or specify your own custom settings (CBNET server name/IP address and TCP/IP port being used).

For instructions on using the CBNet Server, see "Network license management and limiting users" in the PPK Control Center.

Click **Next** to continue.

Note

If you want to protect applications in a network, please make sure that your CRYPTO-BOX contains enough network licenses! For more information, see "Configurable parameters of a CRYPTO-BOX" in Section 5.4.1

Tip

Activate the "Compress protected application" option to take advantage of the powerful compression and encryption functions of the AutoCrypt Wizard.

Selecting the application to be protected

This screen appears if you have selected the project type Autoprotection of application (local or network).

The **Select file** screen is used to specify which program is to be protected, and a name for the protected program. To prevent confusion, you should ensure that you choose different names for the original and protected application. You can also select an application number. This number specifies a location in the CRYPTO-BOX memory and is required if you want to protect more than one application using the CRYPTO-BOX. The default value is 1. If you do want to protect more than one application, please also enter the total number of applications to be protected.

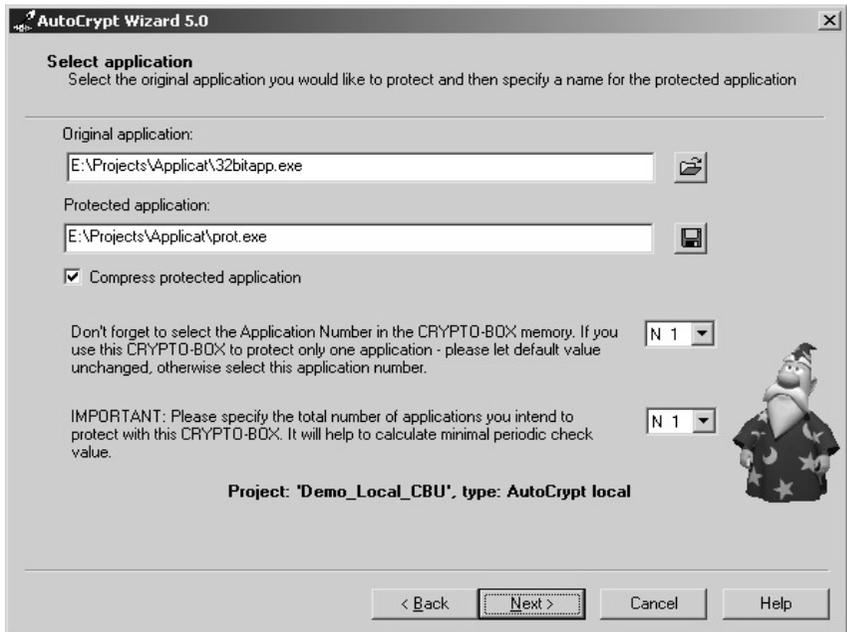


Figure 5.9
File selection

Click **Next**.

Selecting the protection options (data objects)

The **Dataobjects** option in the tree structure on the left hand side of the screen is used to configure your chosen protection scheme. The data objects and protection options displayed depend on the project type you have selected.

"Data objects programming" project

The following data objects will be available: **Memory object**, **Execution counter**, **Expiration date**, **Expiration days** and **Expiration time**. They can be stored in the CRYPTO-BOX memory at a user-specified offset value.

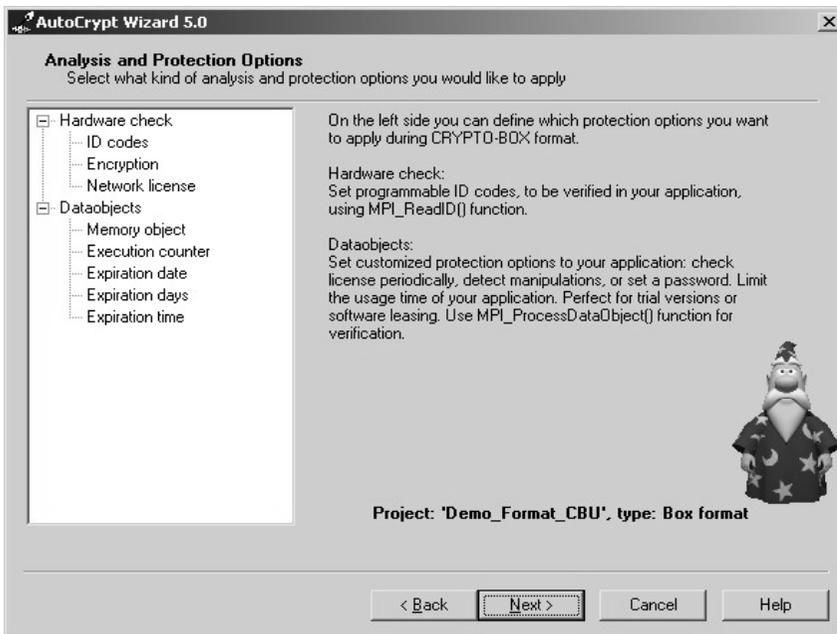


Figure 5.10
"Data objects programming" project

Note

The "Expiration time option" is only available if you have also selected the "Periodic check option".

"Autoprotection of application (local)" project

The following data objects will be available: Execution counter, Expiration date, File authenticity and Password check. You can also use the Periodic check option to check for the presence of the CRYPTO-BOX at regular intervals.

Note

The Encrypt protected application using AES/Rijndael option is only available if you have previously selected the Compress protected application option (see page 48).

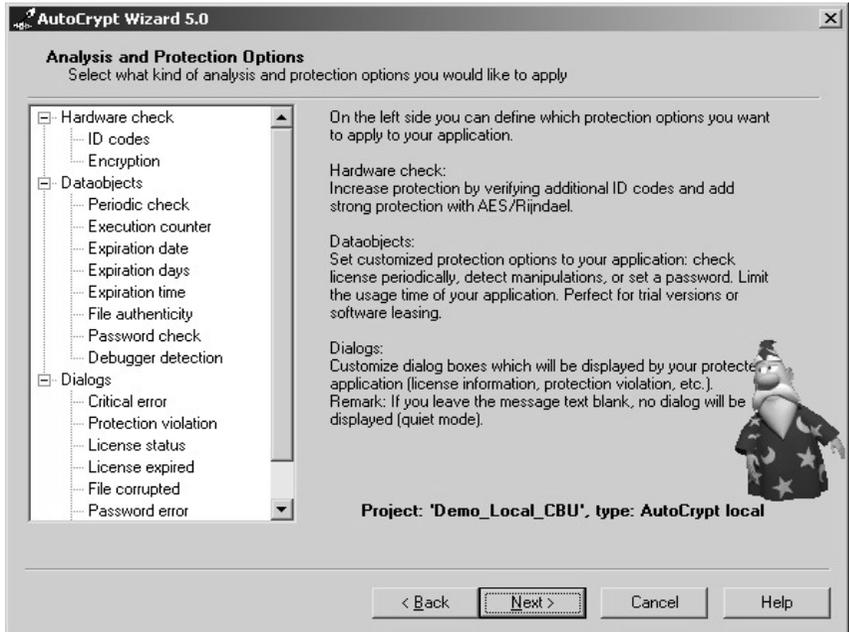


Figure 5.11
Autoprotection of application (local)

Tip

The "Encryption", "File authenticity" and "Debugger recognition" options provide comprehensive protection against hacker or virus attacks on your application.

"Autoprotection of application (network)" project

The following data objects will be available: **File authenticity and Password check**. You can also use the Periodic check option to check for the presence of the CRYPTO-BOX at regular intervals.

Tip

The "Encryption", "File authenticity" and "Debugger detection" options provide comprehensive protection against hacker or virus attacks on your application.

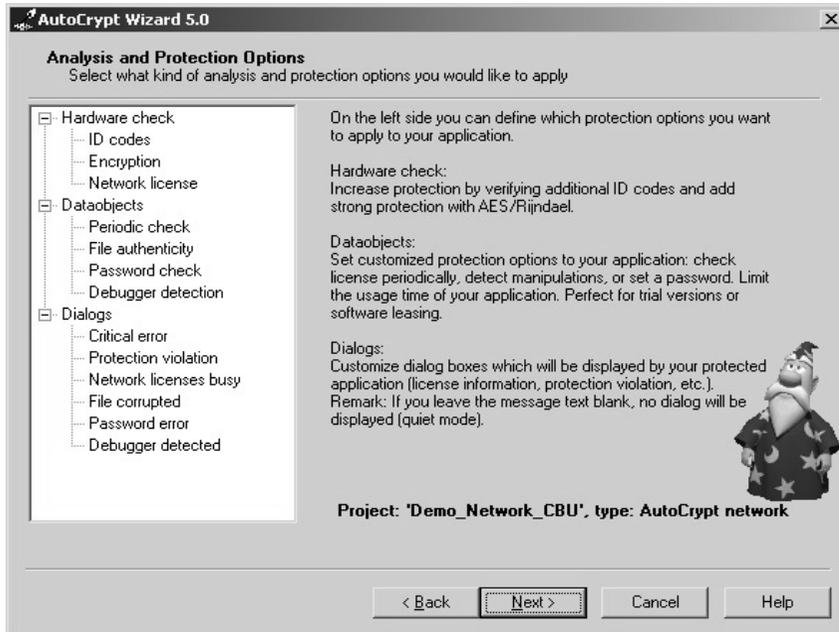


Figure 5.12
Autoprotection of application (network)

AUTO:CRYPT features a powerful compression method and a "Total FILE:CRYPT Feature" with SECURE COMPRESS!

The compression rate of up to 50% dramatically reduces the size of your exe files - the advantages: easier storage, delivery and installation.

Protection against tampering, object code patching and unauthorized modification of your software.

Figure 5.13
Secure Compress

```

r Serial No. 0344-8995
Copyright 2000 / 2001
SuperSoft Inc. Las Vega
s, NV«p²+License agreem
ent Royalties apply
; ; ° ≈ flt > / | r r ≈ * † ≈ * •
    
```

```

t º [ º | i > = z t 7 º = r a i | » ü
ü σ √ ( º ≡ : > s J º + r μ s º + i º ∞ r ≥
≈ Ω θ φ º • μ 9 Σ - ½ á i ü √ p t ¼ √ δ Σ μ
| º . ≡ 1 º | Ω γ : + ( Ω ∞ Ω γ º ∞ δ Γ μ φ
= [ º º θ i ü º £ f ñ r « | f g θ ≈
    
```

Original file (extract)

After SECURE COMPRESS

Original application:

E:\Projects\Applicat\32bitapp.exe

Protected application:

E:\Projects\Applicat\prot.exe

Compress protected application

Don't forget to select the Application Number in the CRYPTO-BOX memory. I use this CRYPTO-BOX to protect only one application - please let default value unchanged, otherwise select this application number.

Figure 5.14
Screen shows how to
select the
SecureCompress
feature.

Defining your passwords

The **Hardware check** option initializes the passwords of the **CRYPTO-BOX** system. If you select the **ID2** checkbox, the specified values will be programmed into the CRYPTO-BOX. **AUTO:CRYPT** checks these value, configures the CRYPTO-BOX and injects all the selected ID codes into the protected program.

Tip

This option can be used in a "Data objects programming" project to modify the freely programmable ID codes of a CRYPTO-BOX! For more information about programming CRYPTO-BOX, see Section 5.4.1 "Configurable parameters of a CRYPTO-BOX".

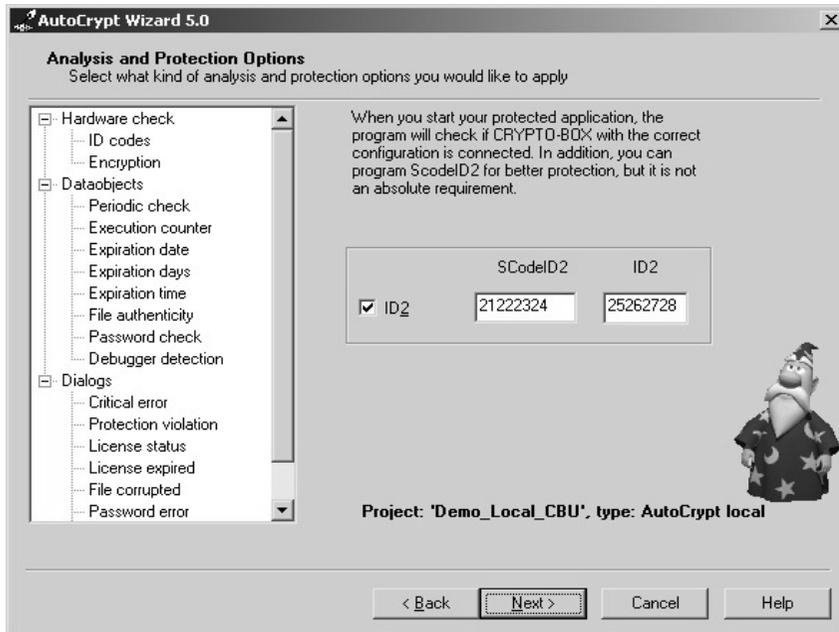


Figure 5.15
Defining passwords

Defining the protection dialogs

The **Dialogs** option is used to configure the error and status dialog boxes to your needs. The messages displayed when a protected application is running can be easily edited by means of the fields on the right hand side.

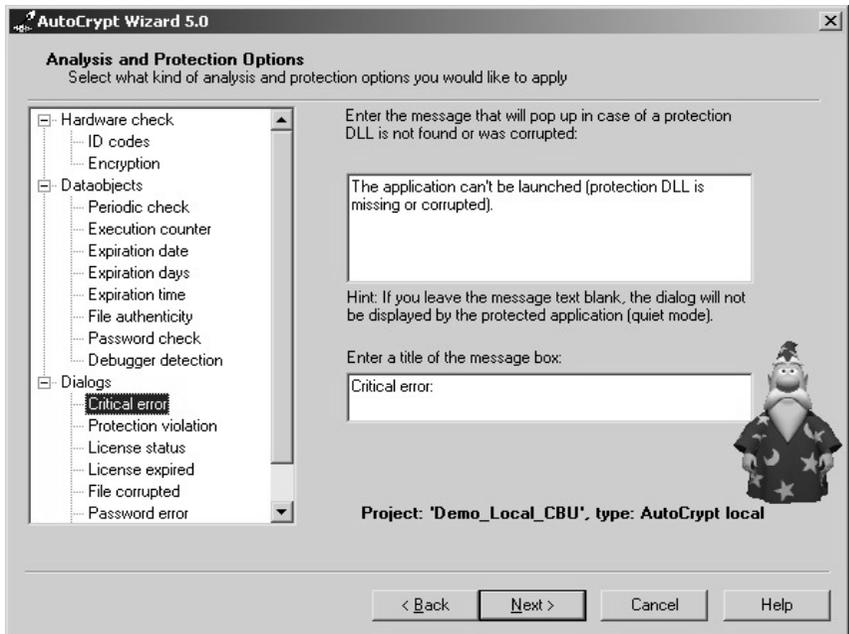


Figure 5.16
Defining the protection
dialogs

Click **Next** to see a summary of the selected protection options, then click **Next** again to proceed.

Configuring the hardware key and protecting the application

Click **Configure hardware** to store your selected values in the CRYPTO-BOX.

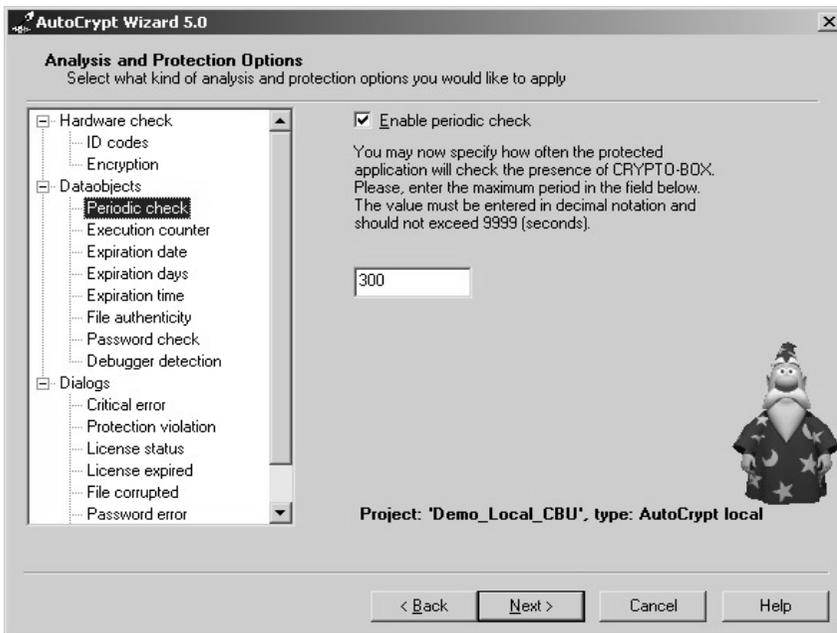
For **Autoprotection** (local or network) projects, choose **Protect application** to activate the software protection for the application. The protected application and a CRYPTO-BOX library are copied into the destination directory. Next click **Save signature** to save an application signature in the CRYPTO-BOX memory.

You can now use this program together with the CRYPTO-BOX on another computer by copying the protected application and the CRYPTO-BOX library "**acwmpi32.dll**" to the target computer. By default, AutoCrypt saves this library to the same directory as the protected application.

In order for the protected program to run correctly, the CRYPTO-BOX device drivers need to be installed also. The easiest way to install the necessary drivers is to use the CRYPTO-BOX installation program (**cbsetup.exe**). For further information, see the readme file of the CRYPTO-BOX PPK or the "**Driver installation and diagnostic tools**" entry in the PPK Control Center.

Note

Keep in mind the restrictions imposed by the File authenticity option - for example, if you have set the option to check the file path.



Note

The AutoCrypt Wizard is continuously being extended with additional functions and data objects. Visit our website (www.marx.com) regularly to check for updates.

Figure 5.17
Selecting the protection options and dialogs

5.1.5 Protecting applications using the API

Note

Never distribute a program which is protected with an Evaluation CRYPTO-BOX, because all demo keys are programmed the same (see Appendix A "Codes of a CRYPTO-BOX").

General overview

This implementation method via a programming interface is used when you want links directly into the source code. If you only want to get acquainted with the functionality of a CRYPTO-BOX, choose **MPI DEMO** which shows you all the features of the key. You will find this program under "**Demos, Tools, Utilities**" in the PPK Control Center. The MPI demonstration program, which only works if an evaluation key is connected to the computer, will assist you in learning about the ID codes, the memory, or the integrated algorithmic functions of a CRYPTO-BOX. We recommend that you familiarize yourself with the various options of the CRYPTO-BOX system before integrating it into your program.

Note

You will receive from us a printout of the ID codes and passwords for the CRYPTO-BOX we have configured for you. You will find the access codes for our CRYPTO-BOX evaluation kit in the Appendix.

In-depth evaluation

Install the CRYPTO-BOX API for your favorite compiler. During the installation, a subdirectory is created containing an object file or library and a demo source code written in your compiler's programming language. This source code contains calls to all available functions the microprocessor can execute. A **batch file** or **make file** is also included to compile and link the demo program. You will see just how easy it is to implement an effective copy protection scheme based on a CRYPTO-BOX.

Comprehensive copy protection

Having familiarized yourself with the functioning of the CRYPTO-BOX, you are now ready to implement a CRYPTO-BOX to your software. You can do this using the demo key in the evaluation kit. Please read Chapter 9 for some tips you should consider when you develop your individual protection scheme. When you place your first order, you will obtain a unique, customer-specific code that we have allocated to you. After replacing the demo code with your individual code, the program will be ready to ship to your customers.

Note

For further information on configuring the CRYPTO-BOX, see Section 5.4.1 "Configurable parameters of a CRYPTO-BOX" beginning on page 61.

Programming a CRYPTO-BOX with specific codes

Every CRYPTO-BOX uses several ID codes, some of which are preset by MARX (customer-specific) and some of which are programmable on-the-fly. You can use either the **AutoCrypt Wizard** or **CBEdit/CBProg** to do this programming.

5.1.6 Comparison of automatic vs manual implementation

Note

The AutoCrypt Wizard can program the ID codes ID2 (for CBU), ID4-8 (for CRYPTO-BOX 560/Net) or ID4 and 5 (CRYPTO-BOX Versa) and also the memory (size varies depending on the type of CRYPTO-BOX - see also Section 5.4.3).

The CRYPTO-BOX provides two methods for protecting your software:

- The **AUTO:CRYPT** module (a component of the AutoCrypt Wizard) is used for automatic protection of 32-bit Windows applications.
- **A high level API** (Application Programming Interface) allows you to manually implement the protection system by means of easy-to-use function calls.

You can choose either single method, or combine the two, depending on your requirements. Use the table below to determine which method will best meet your needs.

Automatic implementation

No source code needed
Quick and easy protection
Least effort required
You define an expiration date and/or the number of allowed program starts.
The AUTO:CRYPT module of the Auto Crypt Wizard automatically injects license code into the unprotected program.

Manual implementation

Source code must be available
Maximum security and flexibility
Customized applications
The protection scheme is implemented by integrating function calls into the source code.
All functions are available through a high level API.

Table 5.2
Requirements

Automatic implementation

The program stops.

Manual implementation

You define how the protected program reacts. You can stop the program, switch to demo mode or limit the program functionally.

Table 5.3
If no CRYPTO-BOX is attached

5.2 Using MARX Data Objects

To protect your application against tampering or hacker attacks, MARX software protection utilities (for example AUTO:CRYPT) provide various types of data objects that are stored in the CRYPTO-BOX memory.

The values of these data objects are calculated and encrypted by means of special algorithms.

Example values:

- an application file checksum that is calculated when the original application is protected (to prevent file modifications)
- hash values calculated from the application's file name/path (to prevent renaming or copying of the application file to another subdirectory)
- the size or last modified time of the application file (additional/optional to the checksum check)

You can also store a number of application specific parameters in the memory of the CRYPTO-BOX, for example various handles or passwords that are to be encrypted or for which a hash value is to be calculated. To check these parameters, the program should first encrypt them (e.g. using a hash function) then compare the result with the values stored in the key's memory. Then, even if an unauthorized person knows the memory contents, he will not be able to restore the original parameters.

You can also store data objects such as **Expiration date** or **Execution counter** in the CRYPTO-BOX memory. The first one is used to restrict the period in which the program can be launched. The second one restricts the number of times the program can be launched. Both solutions are very effective if you wish to restrict your application licenses to a certain period of use or for demo versions.

AUTO:CRYPT also allows you to set a password that the application will prompt for every time it is launched. The password value (up to 12 bytes) is encrypted and stored in the CRYPTO-BOX. This enables you to protect your applications against unauthorized access. The "annoying" password prompt can also be used effectively in demo versions.

5.3 The MARX Programming Interface (MPI)

5.3.1 MPI - easy, secure and portable

The **MPI (MARX Programming Interface)** was developed to provide a flexible means of integrating CRYPTO-BOX into your application. The **MPI** interface enables the development of easily portable software. All CRYPTO-BOX can be accessed through identical function calls. This allows you to concentrate on the creative aspects of your product development work rather than having to rewrite program code every time you want to port the application to another platform or use a different CRYPTO-BOX model.

MPI provides the following basic functionality:

- Access control
- Authentication
- Event counting
- Encryption of data

The desired result can be achieved with just a few lines of code. You will be amazed how easy it is to work with MPI.

5.3.2 MPI and the CRYPTO-BOX system

CRYPTO-BOX systems have been on the market since 1985. Back then, DOS was the main target operating system for developers. These legacy APIs are still available for the parallel CRYPTO-BOX. These interfaces are, however, too cumbersome for today's 32-bit and 64-bit operating systems. To prevent you from working with several different libraries and CRYPTO-BOX-specific APIs, MARX developed MPI - an easily portable, cross-platform programming interface that can be used with all CRYPTO-BOX.

The following operating systems are currently supported via MPI:

- Windows XP/2000/NT4/Me/9x
- MacOS including OS-X

The following operating systems are supported via the "TEOS" CRYPTO-BOX API:

- Windows XP/2000/NT4/Me/9x
- Linux
- UNIX
- QNX
- Solaris

5.3.3 MPI in networks

MPI supports access to CRYPTO-BOX over a network. This is one of the main benefits of MPI. The following protocols are supported for Win32 networks:

- TCP/IP
- NetBIOS
- IPX/SPX
- CITRIX and Windows Terminal Server

and TCP/IP, SPX/IPX for Novell NetWare Server and Win32 clients.

5.4 Configuring CRYPTO-BOX

5.4.1 Configurable parameters of a CRYPTO-BOX

Freely programmable ID codes

Every CRYPTO-BOX has several ID codes, some of which are fixed, customer-specific codes programmed by MARX. Another area of memory is programmable on-the-fly. The number of ID codes depends on the CRYPTO-BOX model you are using.

Network license counter

The CRYPTO-BOX USB Versa, XS and XL, and the 560/Net and Serial models also feature a network license counter that can be used to limit the number of workstations in the network that can execute your program. The counter can be set to 0 (no network functionality) or 255 (unlimited network licenses) using the CBProg utility (see section 5.4.4, "Configuring a CRYPTO-BOX using CBProg"). If you want to limit the network licenses to a specific value, you can do so using **LCS** (License Control System), an option available for the CRYPTO-BOX USB XS and XL, CrypToken XS and XL, and the 560/Net models (see also Section 5.6.1 **License Control System**).

5.4.2 AutoCrypt Wizard and CBProg

You can use either **AutoCrypt Wizard** or **CBProg** to configure the programmable ID codes, memory and network licenses of a CRYPTO-BOX. The AutoCrypt Wizard features a graphical user interface that enables you to configure your CRYPTO-BOX with just a few mouse clicks. **CBProg** is a command line based program that is ideally suited, for example, for automated programming of a large number of CRYPTO-BOXes.

Note

To configure your CRYPTO-BOX using AutoCrypt Wizard or CBProg, you will need a suitable hardware configuration profile (.mrx file) for your type of CRYPTO-BOX. This file contains the fixed passwords or ID codes (encrypted) needed to access your CRYPTO-BOX. You will receive it with your customer-specific CRYPTO-BOX, or you can request it from us without charge. For further information on hardware configuration profiles, see page 44: "Selecting the hardware configuration profile".

Note

For further information on CRYPTO-BOX ID codes, see Appendix A "Codes of a CRYPTO-BOX".

5.4.3 Configuring a CRYPTO-BOX using the AutoCrypt Wizard

For further information on CRYPTO-BOX ID codes, see Appendix A: Codes of a CRYPTO-BOX. There you will find not only information on the passwords/ID codes of the CBU in the Evaluation Kit, but also which ID codes are fixed and which are programmable for a particular CRYPTO-BOX model. The AutoCrypt Wizard recognizes automatically which parameters are configurable for a particular key type and displays only those parameters.

Programming of the freely programmable ID codes (example for CRYPTO-BOX USB):

Note

For more information about using the AutoCrypt Wizard, see pages 41-55 or refer to the AutoCrypt Wizard help.

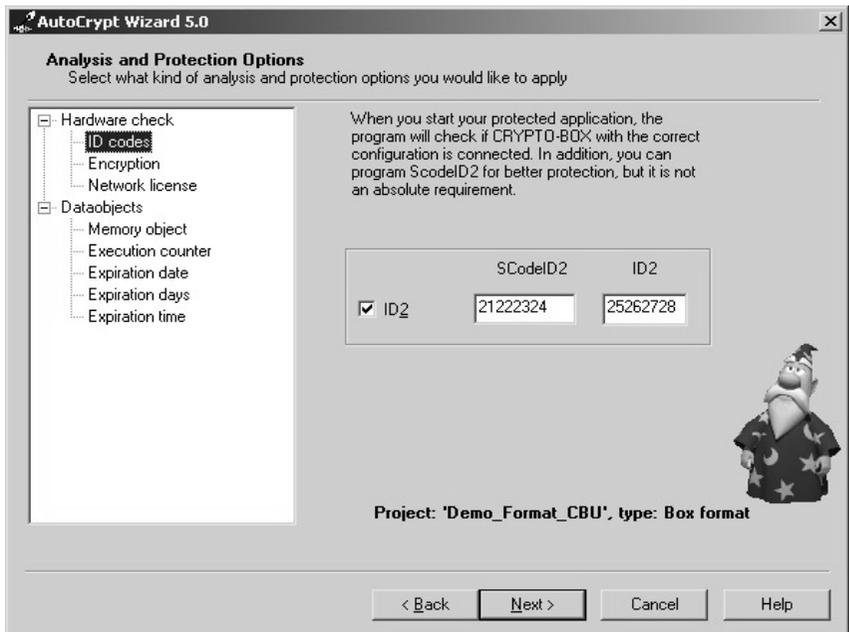


Figure 5.18
Programming the ID
codes

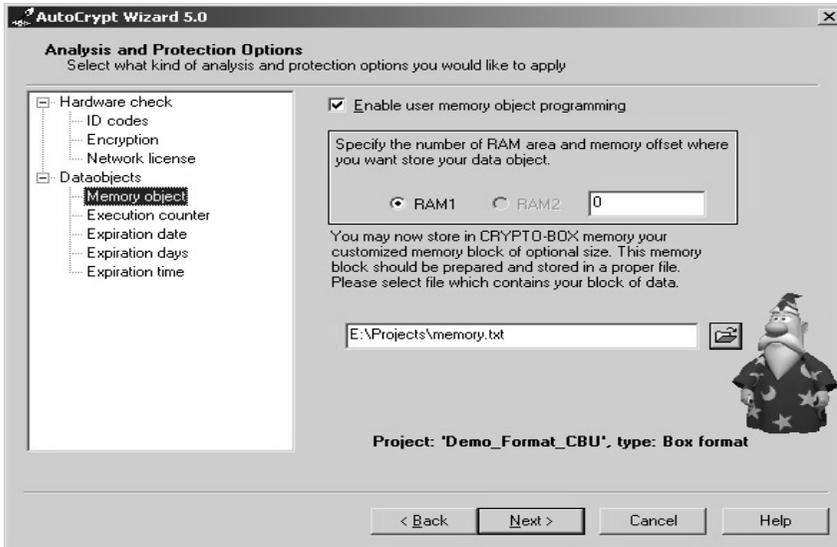


Figure 5.19
Programming a memory
object

Note

By default, the CBProg program allows the network license counter to be set to 0 (no network functionality) or 255 (unlimited network licenses). If you want to limit the network licenses to a specific value, you need to purchase the License Control System (LCS) first.

5.4.4 Configuring a CRYPTO-BOX using CBProg

CBProg is a command line based programming tool for the CRYPTO-BOX USB, 560/Net, Versa and Serial models. It is located in the `\tools\cbprog` folder of the CRYPTO-BOX PPK.

Syntax for calling CBProg: `CBPROG.EXE file1.mrx file2.ini [-q]`

file1.mrx is the hardware profile for your particular type of CRYPTO-BOX (for further information, see Section 5.4.2 "AutoCrypt Wizard and CBProg").

file2.ini contains information about the values (ID codes, memory content, number of network licenses) to be programmed into your CRYPTO-BOX. You can edit the file using any text editor, for example Windows Notepad.

In the folder `\tools\cbprog` you will find several sample .ini files for the various CRYPTO-BOX models. You can modify the appropriate .ini file for your CRYPTO-BOX model to suit your requirements.

-q (optional) activates "**Quiet Mode**" (no screen output)

Note

Programming of the CRYPTO-BOX memory contents is supported from CBProg version 1.0.1.1129 onwards.

5.5. MarxProbe - the test and diagnostic tool

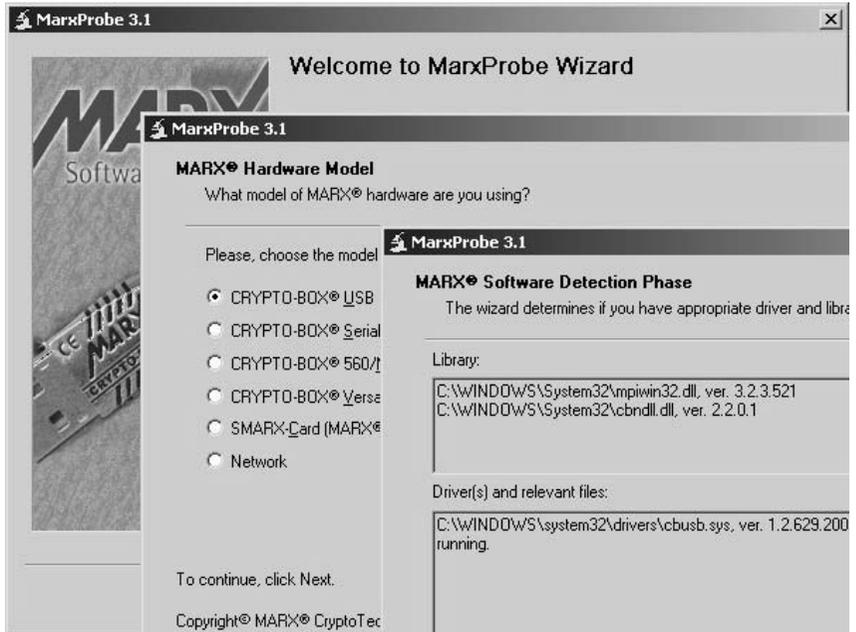


Figure 5.20
MarxProbe

MarxProbe is a powerful tool which greatly simplifies the troubleshooting of CRYPTO-BOX. MarxProbe performs a number of different tasks: it performs extensive diagnostics of your operating system, detects which MARX DLLs and device drivers are installed and running, automatically downloads missing files from the Internet and then installs them on the local computer.

5.5.1 Functionality

- Analysis of installed MARX DLLs and device drivers. On Windows XP/2000/NT4 systems, if a device driver is installed but has not been run, the program attempts to start it immediately without the need to reboot the computer.
- Detection of MARX hardware.
- Download (via the Internet) and installation of missing MARX components. On Windows XP/2000/NT4 systems, you won't need to restart the computer after the installation of device drivers!
- **On Windows XP/2000/NT4 systems, an error message is displayed if the user does not have administrator rights.**
- Display of troubleshooting information. When a problem is detected, the program suggests suitable measures for resolving it.
- Report generation. A report can be saved to a file or e-mailed directly from within the program.

Supported operating systems

MarxProbe supports the Windows XP, Windows 2000/NT4, Windows Me/98 and Windows 95 operating systems. Internet Explorer Version 4 or higher must be installed on Windows NT4 and Windows 95 systems.

Documentation

The MarxProbe tool features context-sensitive online help that is accessible from anywhere within the program.

5.5.2 Using MarxProbe

MarxProbe works like a Wizard and guides you step-by-step through the troubleshooting process.

CRYPTO-BOX model

This screen is used to select the CRYPTO-BOX model you are using. If you are not sure which CRYPTO-BOX model you have, refer to Chapter 3 "Main features of the CRYPTO-BOX models", or click Help to access the online help. After selecting an entry, click **Next**.

Detection of MARX Software

The **MarxProbe** Wizard helps you identify whether any components required for interacting with the CRYPTO-BOX are missing. If any DLL or device driver is missing, an entry appears in the corresponding form field. Click Next to proceed to the next screen.

Detection of CRYPTO-BOX models

At this point the program checks whether the selected CRYPTO-BOX model is connected to the computer, and whether it is responding. A message is displayed indicating whether or not the detection was successful.

This screen appears only if all required software components have been installed, otherwise no hardware detection is necessary.

Note

You need to be connected to the Internet to download files because MarxProbe doesn't perform automatic dialup (e.g. by a Modem).

Troubleshooting

This screen will appear if one or more of the required MARX software components are missing, not installed, not registered or not running properly (applies to Windows XP/2000/NT4 device drivers). The program suggests possible measures to resolve the problem and, after prompting for confirmation, automatically downloads and installs missing DLLs, device drivers etc..

Report

MarxProbe generates a status report indicating whether your system is ready to interact with the MARX hardware. If the problem was not able to be resolved, you can send this report to MARX support.

Send via e-mail

Click the Send via e-mail button to launch the standard e-mail client. MarxProbe attaches the report to the body of the e-mail and enters the MARX support e-mail address in the To: field.

Save to file

Click the Save to file button to export the report to a text file. This file can be used as an e-mail attachment.

On Windows Me/9x systems, don't forget to restart the computer after driver installation.

5.6 License Control System and Remote Field Programming

5.6.1 License Control System (LCS)

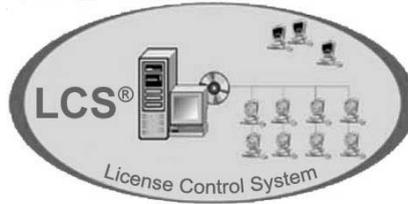


Figure 5.21

The **License Control System (LCS)** lets you control the licensing of your application in network environments by restricting the number of program instances that can be running concurrently on the network.

The CRYPTO-BOX USB Versa, XS and XL, and 560/Net and Serial models feature a hardware-based license counter. You can protect your applications by programming this counter and connecting the CRYPTO-BOX to any computer in the network - only the predefined number of programs will be able to be run concurrently on the computers in the LAN.

The **MARX Programming Interface (MPI)** offers developers a convenient means of managing licenses. Using **MPI**, you can search for a CRYPTO-BOX not only on the local computer but also on the network from within your protected application. To be able to do this, the CNetServer (an integral component of the MARX network solution) must be running on those LAN computers to which a CRYPTO-BOX with a pre-programmed license counter has been connected. If the right hardware key is found, the application (client) can attempt to open it with appropriate MPI calls. The CNetServer only allows the key to be opened if the number of currently connected clients does not exceed the number of licenses programmed into the key.

The CBNetsServer provides a convenient means for network administrators to monitor the LAN status, connected devices, connected clients etc. It supports the **MARX Administrative Interface (MAI)** and the **Administrative Console**, a special application for remote administration of the CBNetsServer. Developers can use MAI to implement custom solutions.

The CBNetsServer also features an HTTP interface, thereby allowing it to be configured using over any standard Internet browser.

Initial programming and remote reprogramming of the license counter

The AutoCrypt Wizard or CBProg utilities can be used for initial programming of the license counter (see Section 5.4.2 "AutoCrypt Wizard and CBProg"). You can use the AutoCrypt Wizard to protect your applications and/or program CRYPTO-BOX. License counter programming is only one of the many features this program provides.

If you want to change (e.g. increase) the number of licenses directly at the customer site, it is best to use **Remote Field Programming (RFP)**.

LCS (License Control System) for configuring user limits

The **License Control System (LCS)** is only supported by the CRYPTO-BOX USB XS and XL, CryptToken XS and XL, and 560/Net and Serial models. This feature for license management in a network allows you to limit the number of users that can concurrently access an application that has been protected with a CRYPTO-BOX.

To use a CRYPTO-BOX in a network, you need to configure it with a valid user limit between 0 and 254. The value 255 is reserved for unlimited use ("Floating License"); it ensures that there is at least ONE CRYPTO-BOX present in the network.

The network license counter can be configured using the **AutoCrypt Wizard** or **CBProg**. If you have purchased LCS, you will receive from MARX a hardware profile for your CRYPTO-BOX which will activate the feature allowing you to program the network licenses of your CRYPTO-BOX.

Note

To configure the user limit on Windows Me/9x systems, you need to install version 2.60 or higher of the CRYPTO-BOX 560/Net and Versa device driver. LCS is not supported under Windows 95.

If you do not have LCS, CBProg allows you to set the network license counter either to **0** (no network functionality) or **255** (unlimited network licenses). All other values are ignored.

5.6.2 Remote Field Programming (RFP)

The abbreviation **RFP** stands for **Remote Field Programming**. This function allows you to update the memory contents or the parameters of a CRYPTO-BOX at the customer site without having to ship the CRYPTO-BOX back and forth. The

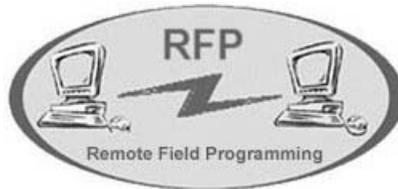


Figure 5.22
Remote Field
Programming

RFP standard solution consists of two components: a **client** and a **server**.

Your customers use the **RFP client component, RUpdate** (Remote Update Utility) whereas you, the software manufacturer or vendor, use the **RFP server component, RFPDistr** (Distributor Utility).

Using the RFP server component, you generate an encrypted, secure batch of commands (the so-called **activation key**). The purpose of this key, which is processed on the client side, is to reprogram the parameters, and update the memory contents of the CRYPTO-BOX.

The **RFP** standard solution saves an activation key to a file which you is sent afterwards to the user via E-mail/Internet.

Benefits of RFP

The **RFP** utility allows you to access all CRYPTO-BOX functions even if you don't have the key at hand. The **RFP Distributor Utility** can be used to program any CRYPTO-BOX model, including the CRYPTO-BOX USB, CRYPTO-BOX Versa, CRYPTO-BOX 560/Net and CRYPTO-BOX Serial.

Further benefits:

- RFP can be custom configured for your hardware
- Saves time and money because hardware keys no longer need to be shipped back and forth to be reprogrammed
- Fast remote provision of updates and upgrades to end-users
- Remote activation of software functions and remote access to your software
- Remote extension of usage and software leasing expiration dates
- Remote execution of maintenance work and remote exchange of data
- No license fees
- Can be used with all CRYPTO-BOX models

Note

Updates of RFP are available at www.marx.com.

How does Remote Field Programming work?

RFP enables you to update the content of your customer's CRYPTO-BOX by means of a remote update. The advantage of this method is that there is no need to ship the key back and forth to have it reprogrammed.

When an end-user wants to request an upgrade, he runs the Remote Update Utility to generate a unique transaction key which is sent to the software vendor.



The vendor processes the transaction key using the RFP Distributor Utility which generates a unique, customer-specific activation code which he then sends to the end-user.



After receiving the activation key, the end-user processes it using the Remote Update Utility which updates the content of the hardware key.



Figure 5.23
RFP schematic

What is a transaction key?

When an end-user generates a request to change the memory contents of his CRYPTO-BOX (by initiating a so-called transaction), an encrypted sequence is produced which is called a transaction key. This transaction key is sent to you (the vendor). With the aid of this key, you then generate for the end-user an encrypted sequence, the so-called activation code.

What is an activation code?

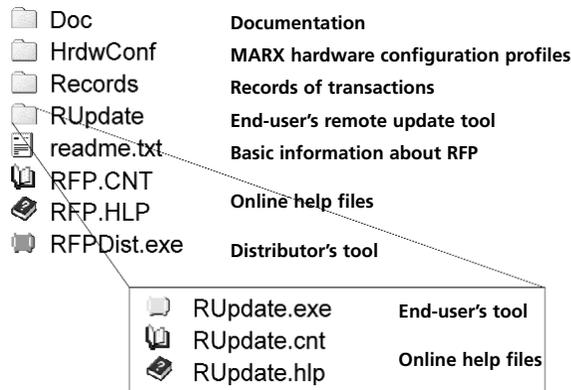
Having received a transaction key from an end-user, you then create an encrypted sequence called an activation code. You need to send this activation code back to the end-user. The end-user processes this activation code using the Remote Update Utility, thereby updating the content of the CRYPTO-BOX.

Activation codes can be understood as encrypted commands that are sent to a CRYPTO-BOX. These commands can only be executed **ONCE** and are only understood by the CRYPTO-BOX that originally generated the unique transaction key.

RFP Components

The following directory tree shows a number of the RFP files:

The Distributor's Tool (**RFPDist.exe**) is your primary remote update tool. Your end-



Note

Make sure you read the online help or the RFP user manual to understand how Remote Field Programming works and to ensure that you don't inadvertently send sensitive data to your end-users.

Figure 5.24
Directory-tree

users receive the file **Rupdate.exe**. See the online help for information on how to create the files that need to be sent to end-users.

5.7 Multiple Application Authorization System (MAAS)

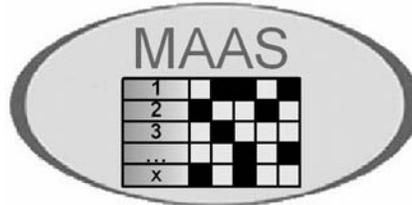


Figure 5.25
MAAS



Every one of your customers receives the same software (e.g. on CD or via the Internet) along with a customer-specific programmed CRYPTO-BOX. During the software installation on the customer's system the CRYPTO-BOX "decides" which applications are to be installed and with which options.



MAAS supports Electronic Software Distribution (ESD)!

Note

Updates of MAAS are available at www.marx.com.

MAAS (Multiple Application Authorization System) is a tool which allows you to protect several applications with just a single CRYPTO-BOX. It also enables to you individually configure the protection options for each customer and application.

All options are stored in the CRYPTO-BOX internal memory and can be called within your application. The calls are integrated into your application's source code by means of our **MPI** programming interface which supports a large number of development environments. The options specified within **MAAS** are passed to the application and evaluated there.

Specify each protection option conveniently using **MAAS**! You can choose from the following options - customizable for each application:

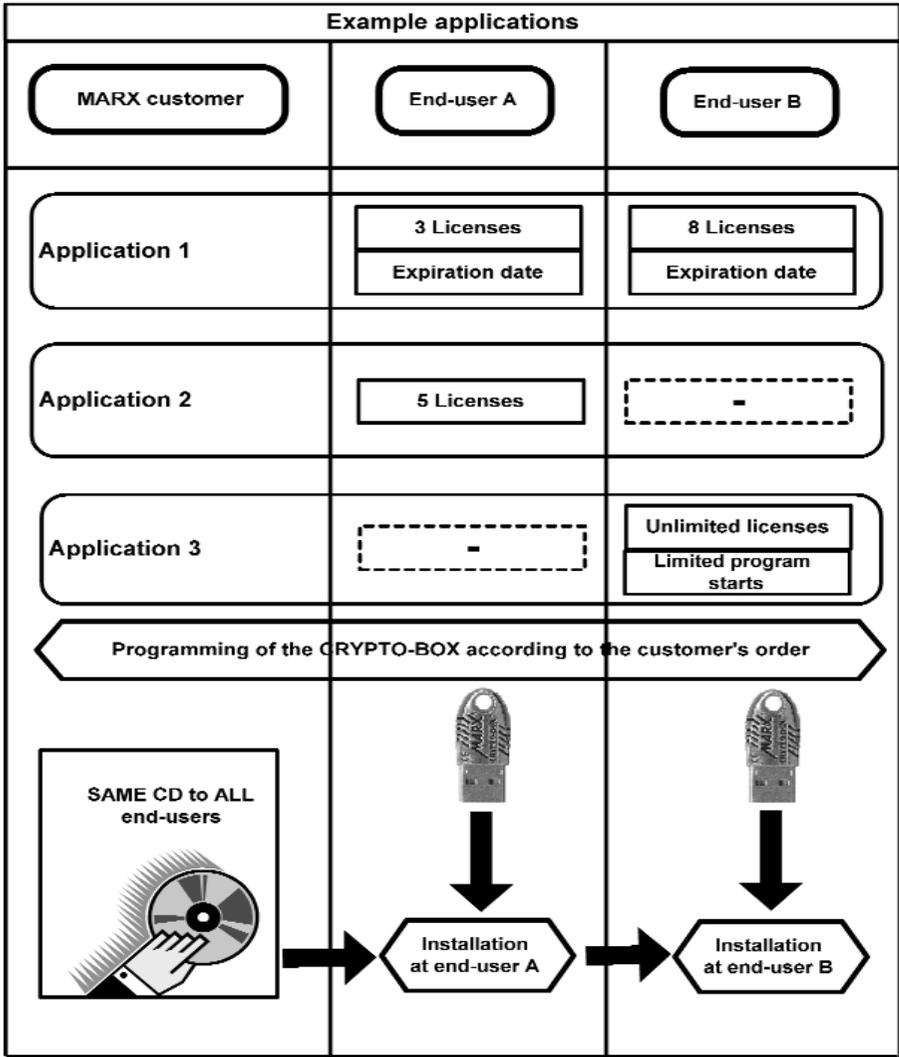
- Specify an expiration date after which the application can no longer be started
- Limit the number of program executions
- Limit the number of instances of an application that can be started in a network
- Specify the workstation(s) on which the application is allowed to be installed
- Additional, custom settings (e.g. language)

The advantages of this method are obvious:

- You deliver the same software to all customers, which cuts costs and administration work
- It is still possible however to customize the software for each customer
- Absolute flexibility thanks to integration into your application's source code. For example, you can decide during installation which applications are to be installed and which aren't.

MARX - Securing the Digital WorldSM

MAAS[®]
Multiple Application Authorization System



6. Components for custom-made solutions

MARX provides a wide range of services for customers with special needs. We are able to develop special types of the CRYPTO-BOX Serial for your preferred platform. For example, our services could include solutions for SPARC workstations with inverted male/female connectors, or for implementations in the QNX real-time operating system.

6.1 Distribution of protected applications

You have reached an important milestone. Your application is finished. The program is working properly on your development system and you now want to bundle it and the CRYPTO-BOX Software Protection System into a package and ship it to your customers. What components does the software package need to contain, and what do you have to do to ensure that the CRYPTO-BOX runs on your customers different systems? The purpose of this chapter is to help you create reliable distributions.

Besides your software and the CRYPTO-BOX, your distribution will need to contain a number of additional components. The relevant components will be described in the following sections. You can skip these sections if you are already familiar with device drivers, DLLs and registry settings.

6.2 Device drivers

A device driver is a software component that enables a computer system and the CRYPTO-BOX to communicate with one another. Peripheral devices won't work correctly if the correct device drivers are missing in the system. To put it simply, a device driver ensures that the operating system can "talk" to the hardware.

For a CRYPTO-BOX to work correctly, the proper device drivers need to be installed on the system.

Note

Linux and UNIX users can skip this entire chapter, because these platforms do not require either drivers nor DLLs to be installed.

Note

It is not sufficient to just copy device drivers into their required directories. The drivers also need to be registered in the system registry so that they can be located by the operating system.

In the following sections we will explain which device drivers are required for each type of CRYPTO-BOX. Furthermore, you will find information about how to simplify, for yourself and your end-users, the distribution of these files.

6.3 CRYPTO-BOX DLL and static libraries

A **Dynamic Link Library** or **DLL** ensures that executable code modules are loaded as required and linked during run time. This allows library code fields to be automatically updated (transparent to applications) and subsequently unloaded when they are no longer needed. DLLs contain some functions that are available to other applications also.

Communicating with device drivers is not always easy. This is where DLLs come into play - they take care of this complex task for you. All you need to do is insert simple DLL calls in your applications. DLLs are used in Windows-based operating systems. UNIX and OS/2 operate on a different principle. Alternatively, you can use static libraries that you link directly into your program code. Static libraries for Microsoft Visual C++ are available on request.

We will now explain which DLLs need to be shipped with the protected application.

6.4 CRYPTO-BOX USB under Windows

Files and drivers for the CRYPTO-BOX USB (CBU)

The **MARX Programming Interface (MPI)** supports applications running on Windows XP, 2000/NT4.0, Windows Me/98 and Windows 95 (please note that the CBU does not support Windows 95).

Linux/UNIX support is also available. For more information, see Section 8.3. "LINUX/UNIX /Solaris /QNX".

To use the CRYPTO-BOX on a Windows platform, you will need the following files:

Operating system	Windows XP/2000/NT4	Windows ME/98
Libraries	mpiwin32.dll	mpiwin32.dll
Device driver	cbusb.sys	cbusb.sys

Table 6.1
Libraries and drivers for
CRYPTO-BOX USB

The following figure illustrates the tasks performed by each of the files in the distribution:

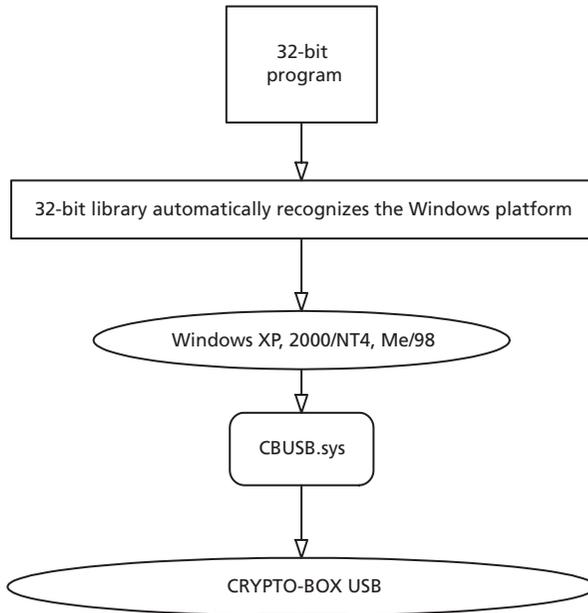


Figure 6.1
Task of the files in the
distribution

Using CBSetup with the CRYPTO-BOX USB

The easiest way to install the device drivers and DLLs required for the CRYPTO-BOX USB is to use MARX's **Automatic Installer (CBSetup)**. The installer is a small self-extracting program. It identifies the operating system, installs and registers the device drives and copies the DLLs. You will find the installer under "Driver installation and diagnostic tools" in the PPK Control Center. Simply execute the file **cbsetup.exe** in that directory and then follow the instructions on the screen.

You may also use the installer in your distribution. We have implemented a "silent" installation mode that ensures trouble-free running of the installer from proprietary setup programs. An overview of the available command line switches is provided in the **readme.txt** file. For further help, run **cbsetup.exe /h**.

Note

In the Windows XP Hardware Installation Wizard, Microsoft has added a new dialog box that may be confusing. Your customer can disregard this message. The CRYPTO-BOX USB/CrypToken driver has been tested and is compatible with Windows XP. Click "Continue anyway" to proceed with the driver installation.

Driver installation on the end-user's system

- **cbsetup.exe** must be run on the PC (in administrator mode on Windows XP/2000/NT4) **before** plugging the CRYPTO-BOX into the USB port
- Next plug the key into the USB port
- The Windows Hardware Assistant appears
- The customer must now click Next to start the automatic installation of the driver
- Windows locates the driver and installs it automatically
- On Windows XP/2000/NT4, it will not be necessary to restart the computer after the installation.

6.5 CRYPTO-BOX 560/Net and Versa for the parallel port

In the following sections we describe which software components you need to provide, along with the CRYPTO-BOX 560/Versa, to customers who are using Windows.

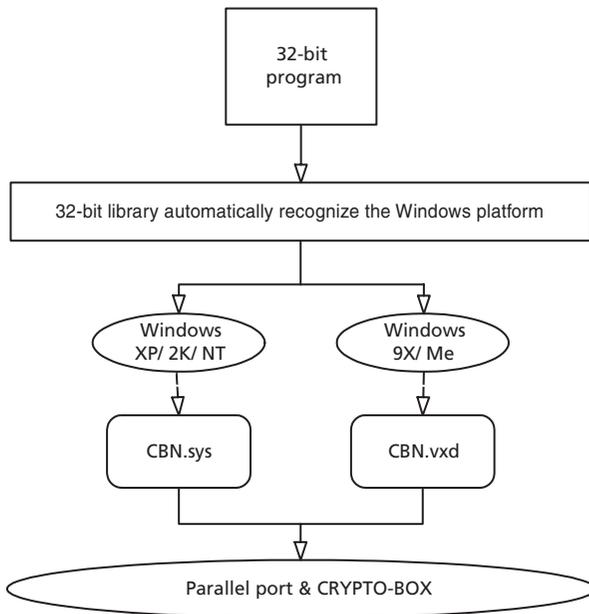
Files and drivers for the CRYPTO-BOX 560/Net and Versa

The MARX Programming Interface (MPI) supports applications running on Windows XP, 2000/NT, Windows Me/98 and Windows 95. To use the CRYPTO-BOX, you will need the following files:

Operating system	Windows XP/2000/NT4	Windows Me/98
Libraries	mpiwin32.dll	mpiwin32.dll
Device driver	cbn.sys	cbn.vxd

Table 6.2
Libraries and drivers for
CRYPTO-BOX Parallel

The following figure illustrates the tasks performed by each of the files in the distribution:



Note

MPI does not support 16-bit applications. An older MARX library does however offer support for DOS and Win3.x. Please contact our Technical Support - we are glad to send you the required libraries.

Figure 6.2
File functions

MARX device drivers support SPP, ECP and EPP modes thereby ensuring error-free communication via the printer interface.

Using CBSetup with the CRYPTO-BOX 560/Net and Versa

The easiest way to install the device drivers and DLLs required for the CRYPTO-BOX 560/Net and Versa models is to use MARX's **Automatic Installer**. The installer is a small self-extracting program. It identifies the operating system, installs and registers the device drives and copies the DLLs. You will find the installer under "Driver installation and diagnostic tools" in the PPK Control Center. Simply execute the file **cbsetup.exe** in that directory and then follow the instructions on the screen.

You may also use the installer in your distribution. We have implemented a "silent" installation mode that ensures trouble-free running of the installer from proprietary setup programs. An overview of the available command line switches is provided in the **readme.txt** file. For further help, run **cbsetup.exe /h**.

Driver installation on the end-user's system

- First run **cbsetup.exe** on the PC (in administrator mode on Windows XP/2000/NT4)
- Next plug the CRYPTO-BOX 560/Net or Versa into the parallel port of the PC - it will now be operational.

6.6 CRYPTO-BOX Serial

When accessing the CRYPTO-BOX Serial (CBS) using our library, this occurs directly via a serial interface of your computer. Developers who implement the CBS via static libraries will not need to add anything to their distribution.

The CRYPTO-BOX Serial is a platform-independent model which can be used under Windows, DOS, Linux and UNIX as well as many embedded systems. For more information, see Section 8.3. "LINUX/UNIX /Solaris /QNX" of this user manual.

Windows DLL for CRYPTO-BOX Serial

No special device driver is required for the CRYPTO-BOX Serial. All you need to supply to your customers is the file **mpiwin32.dll**. You will find this file in the PPK installation folder: **mpi\library**.

Note

MPI does not support 16-bit applications. Please contact us for more information and examples for DOS and other 16-bit environments.

Operating system	Windows XP/2000/NT4	Windows Me/98
Libraries	mpiwin32.dll	mpiwin32.dll
Device driver	none	none

Table 6.3
Libraries for
CRYPTO-BOX Serial

Using CSetup with the CRYPTO-BOX Serial

The easiest way to install the MPI library required for the CRYPTO-BOX Serial is to use MARX's **Automatic Installer**. The installer is a small self-extracting program. It identifies the operating system and copies the DLL(s). You will find the installer under the point "Driver installation and diagnostic tools" in the PPK Control Center. Simply execute the file **cbsetup.exe** in that directory and then follow the instructions on the screen.

You may also use the installer in your distribution. We have implemented a "silent" installation mode that ensures trouble-free running of the installer from proprietary setup programs. An overview of the available command line switches is provided in the **readme.txt** file. For further help, run **cbsetup.exe /h**.

Manual installation of the library required for CBS

You can also copy the MPI library file **mpiwin32.dll** by hand. The library does not need to be registered and can be used immediately by applications. On Windows 95/98/Me, copy the file to the path **x:\Windows\System**. On Windows XP/2000/NT4, copy the file to **x:\WinNT\System32**, where "x" is the letter of the drive on which Windows has been installed.

The following table summarizes this information again:

Table 6.4
Location of the
CRYPTO-BOX
library files

Operating system	Windows XP/2000/NT4	Windows Me/98
Libraries	mpiwin32.dll	mpiwin32.dll
Location	x:\WinNT\System32	x:\Windows\System

6.7 CRYPTO-BOX Card

CRYPTO-BOX Card PCI

Instructions for installing the card and the associated device driver on Windows are included with the CB-Card PCI you have received. You will also find them as a PDF file (Acrobat Reader) in the **X:\TOOLS\CBCARD** directory on the PPK CD (X represents the drive letter of your CD-ROM drive).

CRYPTO-BOX Card ISA

Instructions for installing the card are included with the CB-Card you have received. No driver installation is required; the card will be detected automatically by the PC's BIOS.

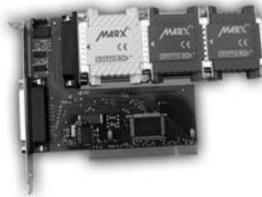


Figure 6.3
CB-Card

7. Secure distribution of digital data

We offer comprehensive solutions for the distribution and protection of digital data.

7.1 Protection of Microsoft® Office documents

The CRYPTO-BOX can provide effective protection for all MS-Office documents (Word, Excel, Access files). Implementation occurs by means of macros written in the Visual Basic for Applications (VBA) programming language provided in MS-Office. Protection functionality, such as encryption of an entire Word document or sections thereof, or encryption of Excel tables or Access databases, can be realized via the **MARX Programming Interface (MPI)**. Examples are provided in the Professional Protection Kit (PPK) MPI.

Note

For more information, refer to the PPK Control Center or visit www.marx.com

7.2 PDF Protection supports DRM (Digital Rights Management)

MARX PDF Protection allows secure distribution of digital documents in PDF format. It provides information rights management (IRM) functionality as part of a DRM strategy to protect and have control over digital information. Only the user who has the appropriate CRYPTO-BOX can open, edit or print the PDF document. Multiple authorization levels and implementation of an expiration date are available. This provides real security compared to just a password based solution - a password can be known by many persons or may have been already compromised. Documents are protected with 128 bit encryption; the encryption key is stored safely inside the CRYPTO-BOX.

MARX PDF Protection is ideal for paid eBooks, Internet delivery, CDR distribution, company internal documents, subscription services and regular updates. It consists of two packages.

The **Distributor's package** includes a Virtual printer driver (PDF Converter), allowing customers to convert documents to PDF format. Furthermore the MARX PDF Protector - an interactive application to encrypt/protect PDF documents, program the CRYPTO-BOX and process remote update requests.

The **End-user's package** includes the MARX PDF Viewer - to view the protected

Tip

Encrypt sections of a document, individual table cells or records using the hardware-based AES/Rijndael algorithm in the CRYPTO-BOX USB! Or store individual records and important mathematical formulas in the CRYPTO-BOX's memory. This makes it impossible to circumvent the protection because the CRYPTO-BOX needs always to be present to perform the decryption.

PDF files, a properly formatted CRYPTO-BOX and the Remote update utility (optional).

Availability: March 18, 2004. Evaluation Kit upon request.

7.3 Security Extension for Macromedia Director

MARX Security Extension (Scripting Xtra) provides an ideal solution for electronic distribution of copy-protected films and presentations created using Macromedia Director 8.

7.4 HTML Security Extension

The purpose of the HTML protection solution is to encrypt/decrypt information of any kind that has been stored in HTML format. You can use this solution to encrypt HTML-based information and then supply these encrypted files to your end-users.

7.5 AudioVideo RTE™: Protection for video and audio streams

AudioVideo RTE enables you to securely distribute video/audio files or streams. Possible applications are Video-On-Demand or subscription services (Pay-TV via PC), or as a reliable defense against industrial espionage in the case of video conferencing and meetings.

Facts about AudioVideo RTE:

- Protection of video/audio files and video/audio streams (ASF-WMV, WMA formats), support for other formats on request
- Supports Windows Media Services
- Uses AES/Rijndael for encryption and decryption (integrated into the CRYPTO-BOX USB, the encryption key never leaves the hardware)

A demo version is available on the PPK CD.

8. Supported Operating Systems

The main focus of this manual is on 32-bit Windows operating systems like Windows XP/2000/NT or Windows 9x. However, we also support many other operating systems. As a matter of fact MARX provides hundreds of libraries and provides solutions for the most exotic operating systems.

This chapter will help you:

- find the right MARX security device for your operating system,
- locate sample code for your programming environment.

For a quick overview we use the following numbers to represent a certain operating system in the overview table.

1 DOS	11 SCO Unix
2 Windows 3.1x	12 QNX
3 Windows 98/95	13 AIX
4 Windows NT 4.0	14 HP-UX
5 Windows XP/2000	15 Novell
6 Windows Me	16 Citrix Metaframe /
7 Linux	Windows 2000/2003 Terminal Server
8 Solaris	17 Apple MacOS 8/9/X
9 UNIX	18 OS/2
10 Embedded systems with proprietary OS	

Table 8.1
Operating system

The following table gives an overview of operating systems supported by the CRYPTO-BOX system.

CRYPTO-BOX system		Port	Operating Systems
CRYPTO-BOX USB	CBU	USB-Bus	3, 4, 5, 6, 7, 16, 17
CRYPTO-BOX 560/Net	CBN	LPT	1, 2, 3, 4, 5, 6, 15, 16, 18
CRYPTO-BOX Versa	CBV	LPT	1, 2, 3, 4, 5, 6, 18
CRYPTO-BOX Serial	CBS	RS232	1, 2, 3, 4, 5, 6, 7, 9 (8, 10, 11, 12, 13, 14 on request)

Table 8.2
Overview
supported systems

8.1 Windows XP/2000 and Me/9x

All CRYPTO-BOX devices support the Windows operating system. You can choose between CRYPTO-BOX for USB, parallel and serial ports.

The following Windows versions are supported:

- Windows XP/2000
- Windows NT4
- Windows Me/98/95
- Windows 3.1x

(USB without Windows 95 and 3.1x)

Programming Languages and Compilers for Windows

We are constantly developing sample source code that shows you how to integrate the CRYPTO-BOX using YOUR development environment. Source code examples for Visual Basic, Visual C, Delphi and many other Windows programming environments are located in the following directory:

MPI\examples\YourCompiler

The Protection Kit readme file contains the latest documentation about the available samples and where to find them. All major compilers are supported and updated, see APPENDIX D, page 181.

8.2 Support for Microsoft .NET environment

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

.NET Common Language Runtime (CLR) can handle partially or completely code of .NET projects. Those code modules which are developed with a language compiler that uses the CLR are called managed code or managed components. They benefit from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

MARX released a special version of the **MARX Programming Interface** (MPI), implemented as C++ Managed Extension. This .NET specific target includes meta-data (storing MPI calls syntax and parameters) and it also allows developers to call MPI from within different programming languages supporting .NET environment (C++, C#, VB) without any extra function declarations, headers or include files.

Sample source code demonstrating MPI calls for all popular environments like C#, C++ or VB, is now included in the MARX PPK.

8.3 LINUX/ UNIX/ Solaris/ QNX

CRYPTO-BOX Serial (CBS) is especially designed for cross-platform portability. MARX provides a grown number of libraries for many systems running LINUX/UNIX. At the moment Linux (Kernel 2.4 or higher) is supported by the CRYPTO-BOX USB and CRYPTO-BOX Serial.

If you need support for other (UNIX-) systems (such as Solaris, QNX etc.), please contact us. We most likely can port our code to your system. In the majority of cases only a re-compilation of our library sources on the target platform is needed. For customers that work with their own proprietary hardware and/or operating system we make the sources available after signing a **Non-Disclosure Agreement** (NDA).

Programming Languages & Compilers for LINUX/ UNIX

Unix samples are compiled with the GNU compiler. They contain makefiles to build the demo program. Source code examples are located on the PPK CD (see "linux.txt" in the PPK CD root directory).

8.4 Macintosh / MacOS, OS/X Jaguar/Panther

All CRYPTO-BOX USB and CrypToken models support MacOS. More information under "mac user.txt" in PPK CD main folder.

8.5 DOS

MARX develops software protection systems since 1983. Therefore, samples for nearly all DOS legacy compilers are available. They can be found in a separate directory on the PPK CD-ROM, or can be obtained upon request.

8.6 Embedded Systems

More information is available on request.

9. Secure Integration techniques for CRYPTO-BOX modules

In this chapter we provide some suggestions and innovative ideas and techniques for a secure integration of MARX hardware devices into your software. You should consider these hints when you develop applications that access CRYPTO-BOX modules using the MARX Programming Interface (MPI).

9.1 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 strong encryption algorithms of the CRYPTO-BOX for important variables and licensing options.

The encryption engine is the most important part of the CRYPTO-BOX because it makes the module indispensable and prevents emulation attacks.

The various CRYPTO-BOX hardware types currently support four encryption algorithms. Symmetric: **AES/Rijndael**, **IDEA**, **Blowfish (MARX algorithm)** and asymmetric: **RSA**. Random sequence generation and MD4/5 hash calculation are also supported.

MARX Programming Interface (MPI) provides developers with universal support of encryption algorithms, their portability and cross compatibility for all CRYPTO-BOX types. It is possible to implement protection/authentication/e-commerce solutions that benefit from the newest symmetric/asymmetric encryption techniques - on local PCs or networks (Intranet/Internet).

Example: you can generate **RSA keys** 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 users.

Note

The encryption engine is the most powerful part of the CRYPTO-BOX devices. Use it whenever you can.

It is possible to create/change encryption keys dynamically with the use of **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 ID codes. 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. It is recommended to not show error messages immediately 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 one 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 when he discovers one routine.

8. Change the applications behavior during runtime.

Make your protected application as dynamic as possible. Merge dummy queries with real queries. You will discourage the cracker since he/she will never be completely sure if he is stepping into 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. Do NOT Make Simple Yes/No Decisions

When you send an access code to read an ID code, 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:

You want an ID Code to equal 144 (you can program the ID Codes). 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 ID

Note

Even a "high end" security device is weakened if you integrate only a single YES/NO decision into your program.

Tip

Spread protection routines, queries and program parts in different places: in the CRYPTO-BOX and in a DLL. This makes it very difficult to remove the protection.

Codes or with values read from the memory of a CRYPTO-BOX. This way you create checking procedures that are spread throughout the program. For a cracker this means he must first study which parts belong together.

Example:

Store a returned ID Code 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 files, **system.dat** and **bootlog.txt**. 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.

Tip

Use MARX data objects to realize counters and other licensing options.

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 that 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 he buys the full version the customer receives the CRYPTO-BOX 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 chapter 9.2 and 9.3. Frequent updates to your program makes an older, cracked version inattractive, 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's protection for all operating systems under which are supported by your application. Often, protection that works with Windows 9x doesn't work correctly with Windows XP, 2000, or NT.

9.2 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 more easy 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 tricks and examples for a good 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 to have 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 Windows 9X will not work under Windows XP, 2000 or NT. 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.

At first your application should perform a simple test if a debugger is present in memory at startup. Now you can stop your application and display a warning message to remove the debugger. The cracker will probably find this simple test easily and removes 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.

Find below an example 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 samples below is also used quite often. It checks 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:

Tip

This is only a small choice of tricks. There are many more different tricks and possibilities. A good recommendation is to combine different anti-debugging tricks for obfuscation: redirect the cracker down the wrong path and do not warn him that you are making attempts to detect their debugging tools. This makes the crackers job difficult and time consuming and many of them will give up.

This example checks via API-function "**CreateFile**" if the **SoftICE VxD** (driver) is loaded. This example works under Windows 9X only.

```

{
    HANDLE hFile;

; "\\.\SICE" without esc sequences
    hFile = CreateFile( "\\.\SICE",
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL);

; If a valid handle is returned, SoftIce is loaded
    if( hFile != INVALID_HANDLE_VALUE )
    {
        CloseHandle(hFile) ; closes the handle
        return TRUE ; and returns TRUE
    }
    return FALSE ; SoftICE not detected
}

```

The same example, but for Windows XP/2000/NT:

```

{
    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. Because during analysis with a debugger the routine runs much slower.

Also a good trick is to send a command to the debugger which switches off breakpoints and activates them again when the program is finished. But this requires to switch into **ring0** which is only possible under Windows 9X. A method to use this trick under Windows XP/2000 or NT is to place the routine into a .sys driver running in ring0.

Note

Use the internal memory of the CRYPTO-BOX to store essential parts of your application! The CRYPTO-BOX USB with their large memory (4-64KByte) is ideal for that. You can even write encrypted values to the CRYPTO-BOX memory to initialize variables or constants that are important for the program. Create a short program in your programming language to encrypt values and use those values in your program. A cracker may disassemble your program. But this will not help him because only the correct CRYPTO-BOX will lead to the original value using the decrypt function.

9.3 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 hackers.

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 hacker, 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 hacker 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)
{
    H           A           N           D           L           E
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 hacker 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 hacker's task). Several 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 hackers. A supposed cracked program will fail to execute after some time - maybe after days and weeks, and drive hackers crazy.

However, programming code that will be executed in the stack involves several

specific issues that sometimes are beyond the scope of this manual (for more details please consider the excellent book "**Hacker Disassembling Uncovered - Powerful Techniques to Safeguard Your Programming, Kris Kaspersky, 2003**", which can be ordered from MARX.

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 hacker equipped with a debugger or an advanced disassembler like IDA Pro.

However, if used in combination with the encryption functions of the CRYPTO-BOX (e.g. using AES/Rijndael unbreakable algorithm implemented in hardware) an "external crypto coprocessor" is added to the protected application - and 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. Certified **AES/Rijndael-Algorithm** runs on external CRYPTO-BOX hardware platform - and only results of encryption/decryption tasks are exchanged with the application, and never the keys.

There are endless possibilities to take advantage of the various encryption- and memory options of the different CRYPTO-BOX models. Hint: Model "XL" is additionally equipped with a true hardware implementation of a "White Noise 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 hackers and their tools.



Figure 9.1
Even this hardware attack was not successful. The inner components of the CBU are inaccessible. And water/dust resistant, too.

10. The MPI- (API-) Reference

In this chapter we describe the MARX Programming Interface, which will be referred to as MPI.

All CRYPTO-BOXes can be accessed through one common 32-bit interface, the MARX Programming Interface (MPI). Therefore you can access any CRYPTO-BOX type using the same source code.

The MARX Programming Interface (MPI) has important advantages:

- Easy-to-use function calls.
- Local access through USB, LPT and RS232 serial ports.
- Remote access via TCP/IP, IPX/SPX and NetBIOS.
- MARX Data Objects greatly simplify most typical protection solutions.
- Support of all CRYPTO-BOX models.
- Hardware access via device drivers for Windows XP/2000/Me/NT4/9x, MacOS starting from 8.6

Note

Even though this manual focuses on the Windows 32-bit operating system, libraries for MacOS, Linux/UNIX and Solaris are available.

10.1 MPI – Makes Network Access a Snap

Tip

On this page you will find an overview about all MPI functions. They are displayed in order of their application. Page links for every function are in chapter 10.5 at page 111

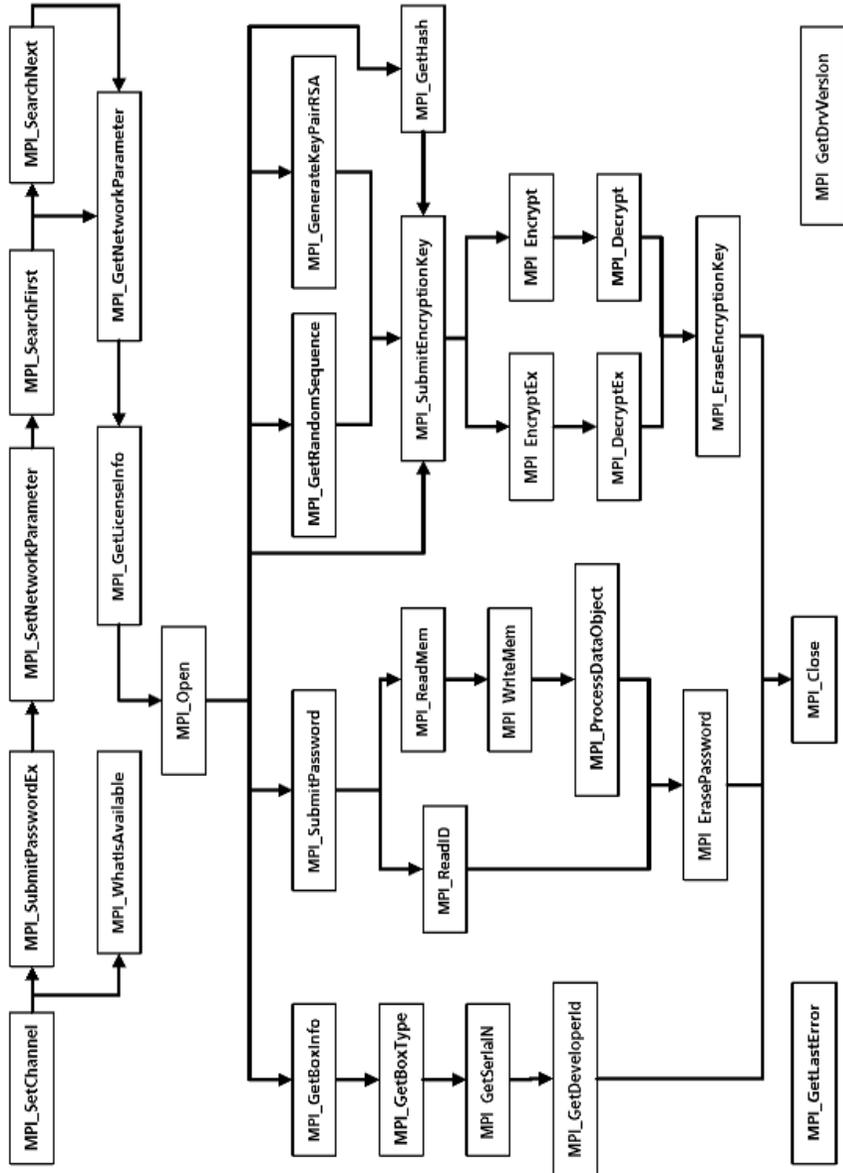


Figure 10.1 MPI Function calls

The purpose of the **MPI** is to provide an easy-to-use way to integrate the CRYPTO-BOX into your application. So far, we only talked about local access to CRYPTO-BOX modules.

To access a CRYPTO-BOX in a network there is really not that much to change in your code. For the MPI network access just means another channel that you submit through the **MPI_SetChannel** function. You have full control over the search order, e.g. the order of network protocols that are used to find a CRYPTO-BOX Server (CBNetServer) in the network.

Additional parameters can be submitted via the **MPI_SetNetworkParameter** function. If no Network Parameters are set MPI, will try to find the CBNet Server automatically using default values.

Where to start with network access

Only one function call decides whether or not a CRYPTO-BOX can be accessed over the network. **MPI_SetChannel** does the whole job. All other function calls can be used the same way as they are used for local access.

In addition to the functions that are already available for the local interface, the MPI provides functions to find out how many clients are accessing the CBNet Server at a certain time.

10.2 Using MARX Data Objects

MARX Data Objects supported by the **MPI** greatly simplify typical protection scenarios, like protecting programs with expiration date, number of executions (usage counter), etc.

MPI environment currently supports the following "ready to use" predefined types of Data Objects:

- expiration date;
- number of days allowed;
- time allowed;
- usage counter.

Note

The CRYPTO-BOX USB, CRYPTO-BOX Serial and CRYPTO-BOX 560/Net can be accessed over network. With the License Control System LCS (available as an option) you can define the number of users accessing the CRYPTO-BOX via network.

Note

Please refer to the sample programs in the \mpi\examples directory, which contains source code that demonstrates network access for Visual C, Visual Basic, Delphi and many more Windows programming environments.

Such objects can be placed in device memory, initialized, checked, verified, etc. Besides predefined type User Defined object (or memory object) is also supported, allowing to store structured secure data in the CRYPTO-BOX memory.

Where to start with Data Objects

Only one function call (`MPI_ProcessDataObject`) provides you with universal interface to MARX Data Objects - allows to create (SET), read (GET) Data Objects and perform other operations depending on object type.

10.3 Where to start using the API

Only a few function calls are necessary to do a very basic check of the CRYPTO-BOX presence and verify whether it is a device that is issued for your company.

Please refer to the sample programs in the `\mpilexamples` directory, which contains source code for Visual C, Visual Basic, Delphi and many more Windows programming environments.

Initializing Channels with SetChannel

Before accessing CRYPTO-BOX devices the channel needs to be selected. The expression "channel" refers to a certain hardware type that is accessed through a particular local or network port. This allows you to deal with the CRYPTO-BOX type that you chose for your security needs.

Submitting Access Codes with SetPassword

CRYPTO-BOX devices are protected by access codes. MARX assigns these codes. They are unique for every company we ship to. Most of the MPI functions work only if the correct access codes are submitted.

Note

Both local and remote channels can be set to access CRYPTO-BOX devices over local ports or networks. Once you have set the channels there is no difference in accessing the CRYPTO-BOX locally or over networks. All other function calls stay the same.

Identifying a CRYPTO-BOX with SearchFirst

In most cases you will deal with one CRYPTO-BOX device and define it through the channel you choose. However, other developers might use a CRYPTO-BOX, too. Maybe even on the same channel (=same hardware type, same local/remote port). That's why MPI provides a function called MPI_SearchFirst(). The only purpose is finding exactly YOUR hardware. This function returns a MARX hardware identifier so that you can reference to that particular CRYPTO-BOX directly without ever scanning the whole system again.

Managing Resources using Open and Close

CRYPTO-BOX hardware devices, connected to a USB, parallel or serial are acquired (=Open) and released (=Close). Every Open function requires a Close. The same applies for access to devices that are available over a network.

Typically there is one Open call in the beginning of a program and a Close call at the end. MARX pass-through devices, such as the CRYPTO-BOX 560/Net or Versa parallel, transparent on the hardware level. These devices are usually still available for other peripheral devices and even for simultaneous access by other applications.

MPI_Open () accesses the CRYPTO-BOX if the submitted hardware identifier is valid.

MPI_Close () releases resources and destroys passwords.

Doing a Simple Check using SearchFirst

Yes, you are right. SearchFirst was already explained earlier. We just wanted to point out that a successfully executed MPI_SearchFirst function call is already a simple check of the CRYPTO-BOX. Only with a valid access code (submitted by MPI_SetPassword) SearchFirst returns a hardware identifier.

The shortest MPI program consists of three functions for local access and 4 function calls for remote access:

1. MPI_SetNetworkParameter() (for remote access only)
2. MPI_SetChannel()
3. MPI_SubmitPassword()
4. MPI_SearchFirst()

Note

Since mpiwin32.dll Version 3.0.2.1009 it is possible to speed up significantly the MPI search & open sequence. If you want to open the first CRYPTO-BOX of predefined type with specified ScodeID1 password it is not needed to call MPI_SearchFirst/Next iteration any more. Simply use handle (0) for MPI_Open and MPI_Close in this case.

10.4 Where to find the functions of the MPI

All functions of the MPI reside in libraries that can be integrated into an application. The libraries support all current CRYPTO-BOXes.

The standard library file for combined local and remote access under Windows 9x/Me/NT/2000 is the **MPIwin32.dll**. If you need local access only you can use the **MPIloc32.dll** which does not include network functions. The **MPInet32.dll** is for network access only.

10.5 Available Function Calls

The following tables list the function calls that are available in the **MPI**. All function calls start with the **MPI_** prefix which stands for MARX Programming Interface.

Table 10.1
Functions that manage
CRYPTO-BOX access
Passwords

Function	Description	Page
ErasePassword	Remove password from password pool	132
SubmitPasswordEx/ SubmitPassword	Submit password to password pool	167/165

Access codes are submitted to a password pool for future access. Typical functions that require passwords are ReadMem, WriteMem, ReadID.

Table 10.2
Functions to set up a
communication channel

Function	Description	Page
SetApplication	Sets the application identifier (not yet implemented)	N/V
SetChannel	Sets the local/remote port and hardware type	157
SetNetworkParameter	Sets network specific parameters	160

A communication channel is the combination of local/remote port type and the CRYPTO-BOX model type.

Function	Description	Page
Close	Closes a MARX hardware resource	117
Open	Opens a MARX hardware resource	147
SearchFirst	Looks for the first CRYPTO-BOX with specific access password	153
SearchNext	Looks for the next CRYPTO-BOX with specific access password	155
WhatsAvailable	Looks for any available CRYPTO-BOX	169

CRYPTO-BOX USB

select CBU as hardware type and AUTO or USB (local access) or AUTONET (remote access) as port.

CRYPTO-BOX 560/Net (local access)

select CBN as hardware type and AUTO or LPT (local access) or AUTONET (remote access) as port.

CRYPTO-BOX Versa

select CBV as hardware type and AUTO or LPT as (local) port.

CRYPTO-BOX Serial

select CBS as hardware type and AUTO or COM as (local) port.

Table 3
Functions to establish a communication channel

Function	Description	Page
EraseEncryptionKey	Remove a key from keys pool	130
SubmitEncryptionKey	Submit an encryption key to keys pool	162
DecryptEx/Decrypt	Decrypts data	118/122
EncryptEx/Encrypt	Encrypts data	124/128
GenerateKeyPairRSA	Generates unique pair of RSA keys (private and public)	134
GetBoxType	Get the box type	137
GetBoxInfo	Get box model and memory size	
GetDeveloperId	Get the unique developer ID	136
GetDrvVersion	Get version of driver	138
GetFirmVersion	Get version of firmware	N/V
GetHash	Calculates hash value for submitted source buffer	140
GetLastError	Get error code and message of the latest function call	142
GetLibVersion	Get version of MPI library	N/V
GetSerialNr	Get the box specific serial number	146
ReadID	Read the ID code of a CRYPTO-BOX	150
ReadMem	Reads data from CRYPTO-BOX memory	151
WriteMem	Writes data to CRYPTO-BOX memory	171
ProcessDataObject	MARX Data Objects support	148
GetRandomSequence	Obtains generated random sequence	145

Table 10.4
Functions to query a
CRYPTO-BOX device

10.6 Return Codes of CRYPTO-BOX devices

All functions of the MPI return 0 (zero) in case of success. Values other than zero are indicating an error.

The following table gives a description of the available error codes. Error codes are sometimes referred to as return codes of a function. This is sometimes confusing, because functions that pass parameters "by reference" return values too. Therefore we prefer to call the return code of a function itself an error code.

In the sample programs we usually pre-define error codes in header files or program modules that can be used to deal with error codes.

Error Codes		Description
0000 (dez)	0000 (hex)	Function call was successful
4097 (dez)	1001 (hex)	The function is not implemented or not supported for this type of token hardware
4098 (dez)	1002 (hex)	Operating system is unknown or not supported
4099 (dez)	1003 (hex)	Incorrect library version
4100 (dez)	1004 (hex)	Device driver is missing
4101 (dez)	1005 (hex)	Wrong device driver version
4102 (dez)	1006 (hex)	Library file is missing
4112 (dez)	1010 (hex)	The selected system port failed
4113 (dez)	1011 (hex)	No CRYPTO-BOX attached
4114 (dez)	1012 (hex)	No CRYPTO-BOX of the selected Typ attached
4115 (dez)	1013 (hex)	CRYPTO-BOX failed (internal error)
4116 (dez)	1014 (hex)	CRYPTO-BOX-time out
4117 (dez)	1015 (hex)	Wrong firmware version
4118 (dez)	1016 (hex)	CRYPTO-BOX is busy
4119 (dez)	1017 (hex)	Destination buffer size is not enough (encryption/decryption)
4129 (dez)	1021 (hex)	Invalid or missing parameter
4130 (dez)	1022 (hex)	Invalid or missing password
4131 (dez)	1023 (hex)	Invalid or missing authentication
4145 (dez)	1031 (hex)	Remote access via IPX/SPX failed

Table 10.5
MPI error codes

Error coed		Description
4146 (dez)	1032 (hex)	Remote access via NetBIOS failed
4147 (dez)	1033 (hex)	Remote access via TCP/IP failed
4132 (dez)	1024 (hex)	Not logged into server correctly
4133 (dez)	1025 (hex)	Too many users connected
4134 (dez)	1026 (hex)	Server internal table is full
4135 (dez)	1027 (hex)	Password must be submitted before
4136 (dez)	1028 (hex)	Invalid request from client
4149 (dez)	1035 (hex)	Connection to server failed
4150 (dez)	1036 (hex)	Client is already connected
4151 (dez)	1037 (hex)	Client is already disconnected
4152 (dez)	1038 (hex)	Remote server not found
4153 (dez)	1039 (hex)	No compatible transport layer found
4160 (dez)	1040 (hex)	System out of memory
4161 (dez)	1041 (hex)	Network communication time out
4167 (dez)	1047 (hex)	Invalid response from server received
4162 (dez)	1042 (hex)	License counter is damaged
4163 (dez)	1043 (hex)	Out of range CRYPTO-BOX memory access
4164 (dez)	1044 (hex)	License counter read error: unable to read ID1
4165 (dez)	1045 (hex)	License counter read error: unable to read RAM
4166 (dez)	1046 (hex)	License counter read error: unable to decrypt
4176 (dez)	1050 (hex)	Encryption key must be submitted before
4177 (dez)	1051 (hex)	RSA key structure is invalid
4178 (dez)	1052 (hex)	The submitted destination buffer is too short
4353 (dez)	1101 (hex)	Data Object not found
4355 (dez)	1103 (hex)	End of usage counter
4356 (dez)	1104 (hex)	Date was modified (PC BIOS date modification was detected)
4357 (dez)	1105 (hex)	Date expired
6553 (dez)	1999 (hex)	Unlimited users quantity license
8192 (dez)	2000 (hex)	Close handle low-level error
8193 (dez)	2001 (hex)	Get handle low-level error
8194 (dez)	2002 (hex)	IOCTL low-level error
8195 (dez)	2003 (hex)	Driver exception
8196 (dez)	2004 (hex)	Driver System error

Table 10.6
MPI error codes
(Continuation)

10.7 Access Codes of a CRYPTO-BOX® Evaluation Kit

The CRYPTO-BOX USB (CBU) and CRYPTO-BOX Serial (CBS) devices are MPI enabled and therefore accessible through the **MARX Programming Interface**.

The passwords for accessing the CRYPTO-BOX included in the Evaluation kit, are shown in Appendix A, page 173.

The access passwords for your customer specific CRYPTO-BOX are displayed at the production sheet is enclosed to CRYPTO-BOX delivery.

Note

Variables can be passed "by value" or "by reference". In "C" this is indicated by a * after the variable type.

10.8 Type Declarations of MPI

The MARX interface can be ported to various platforms and therefore variable types are used that are system independent.

Variablentyp	Beschreibung
WORD	16 bit unsigned number
DWORD	32 bit unsigned number
CHAR	8 bit character
STRING	char
STRING10	10 character string
STRING80	80 character string
LPVOID	pointer to void

Table 10.8
MPI variable types
Overview

10.9 The MARX API Reference

How to read the MARX API Reference

On the following pages you will find a detailed description of all available MPI functions. Every function call is described on a single page. Here is a separate function description:

MPI_SampleFunction

| Supported CRYPTO-BOX type

The function name is always on the top left side of the description. The supported CRYPTO-BOX device is listed on the right side of the function name bar.

This is followed by an **Argument List** which lists all arguments that are passed during a function call, the type of the variables used and whether they are passed by value or by reference.

The **Usage** section explains briefly what the function is good for.

This is followed by an **Argument Description**. This section describes detail about the arguments (parameters) passed to the API. Many times parameter ranges are listed here, too.

The next section shows the **Results** of a function call. Every argument that is changed by the function is listed here. Arguments that are modified by the MPI are always passed by reference with one exception: the return code of the function itself.

In the **Example** section, one or more examples are written in pseudo-code to explain how an MPI function is typically called.

MPI function calls**MPI_Close****CBU, CB560/Versa, CBS**

Argument List:	Type	Passed by	Description
	DWORD	Value	Identifier

Usage:

To close a communication channel previously opened via MPI_Open. This function frees up all resources allocated via MPI_Open when a communication channel was established, and deletes all submitted access codes from memory.

Results:

The Return_code is zero for successful operation. For a Return_code other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

Identifier is a valid MARX hardware identifier obtained from MPI_SearchFirst or MPI_SearchNext functions and used by MPI_Open to establish a communication channel.

Example:

```

...
{ Submit correct access codes via MPI_SubmitPassword; then
search for security devices via MPI_SearchFirst which returns an
Identifier for a communication channel to be opened }

MPI_Open (Identifier)
...
...
{ Free up resources allocated for the active channel during the
session and delete all access codes from the memory }

MPI_Close (Identifier)

```

MPI_DecryptEx

| CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING*	Reference	AlgorithmType
	STRING*	Reference	EncryptionKeyType
	DWORD	Value	LenOfSourceBuffer
	LPVOID	Reference	SourceBuffer
	DWORD*	Reference	pLenOfDestBuffer
	LPVOID	Reference	DestinationBuffer

Note

MARX encryption algorithms are hardware specific. Be careful when you switch from one type of hardware to another, because encryption algorithm incompatibility may be encountered.

Usage:

The string of bytes in **SourceBuffer** is passed to the API and then decrypted either by MARX hardware or within the MPI library depending on the selected algorithm, the encryption key type and the CRYPTO-BOX type. The algorithm depends on the encryption key value (submitted by the **MPI_SubmitEncryptionKey()** function or hardcoded in the CRYPTO-BOX hardware) and/or system parameters that are configured by MARX. Make sure to use identical encryption key values for encryption and decryption.

Two parameters describing resulting (destination) buffer will take care of the facts that RSA is a block oriented algorithm. It's impossible to encrypt, say, 117 bytes string to the same size buffer. If you use a 64 bytes RSA key, the result will have length divisible by a 64. If 256 bytes key is used then the result's size should be divisible by 256.

If **DestinationBuffer** is NULL or **LenOfDestBuffer** is NULL, it means that **SourceBuffer** should be used to store decrypted data.

LenOfDestBuffer parameter should be passed "by reference" not "by value". If its value is not enough to keep decrypted data, the function will return **Return_Code = 0x1052** and this parameter will contain corrected value (number of bytes required to store decrypted data).

Argument Description:

AlgorithmType defines the algorithm used to decrypt the SourceBuffer. The following algorithm types are available (supported by the MPI):

IDEA_ALGORITHM	IDEA™ algorithm
MARX_ALGORITHM	MARX proprietary algorithm
RIJNDAEL_ALGORITHM	Rijndael algorithm
RSA_ALGORITHM	RSA algorithm

EncryptionKeyType defines the type of the encryption key used to decrypt the **SourceBuffer**. The following key types are available:

For the IDEA_ALGORITHM:

IDEA_KEY
IDEA_EXTERN_KEY

For the MARX_ALGORITHM:

MARX_KEY
MARX_EXTERN_KEY

If being called with "extern" key, it means that software emulated version of the algorithm should be used even if active MPI device (currently opened) supports hardware implementation of the required algorithm.

For the RIJNDAEL_ALGORITHM:

RIJNDAEL_FIXED_KEY
RIJNDAEL_PRIVATE_KEY
RIJNDAEL_SESSION_KEY
RIJNDAEL_EXTERN_KEY

First three names assume hardware implemented Rijndael + corresponding key slot usage: fixed/private/session (CBU box should be open). The last one can be used for any type of CRYPTO-BOX device, it assumes that an encryption key should be submitted to the MPI prior to encryption/decryption call.

Note

The CRYPTO-BOX USB devices and the CRYPTO-BOX Serial own a hardware implemented AES/Rijndael algorithm and support RSA (except CBU Versa). RSA keys will be stored inside the CRYPTO-BOX hardware, encryption and decryption is processed in the CBU device driver. IDEA and MARX (Blowfish) algorithms are software implemented (inside MPI library). The CRYPTO-BOX types 560/Net and Versa parallel are supporting AES/Rijndael and RSA via software implementation, IDEA and MARX algorithms are implemented in hardware.

For the RSA_ALGORITHM:

RSA_FIXED_PRIVATE_KEY
 RSA_FIXED_PUBLIC_KEY
 RSA_CBU_PRIVATE_KEY
 RSA_CBU_PUBLIC_KEY
 RSA_EXTERN_PRIVATE_KEY
 RSA_EXTERN_PUBLIC_KEY

First four key names assume hardware implemented RSA. For RSA_FIXED fixed key will be used, for RSA_CBU a proper key should be stored somewhere in the CBU box memory (RAM1) before encryption. The key offset and length should be told to the **MPI** prior to encryption/decryption call.

The RSA_EXTERN keys assume software emulated RSA algorithm, it can be used for any MPI box, the corresponding RSA key should be submitted to the **MPI** prior to encryption/decryption call.

DestinationBuffer must contain **LenOfDestBuffer** bytes (the same for **SourceBuffer** and **LenOfSourceBuffer**) so that the code works correctly.

If **DestinationBuffer** is NULL, it means that **SourceBuffer** should be used to store decrypted data.

Selected **AlgorithmType** and **EncryptionKeyType** must be the same as for the encryption to receive correctly decrypted data.

Results:

The **Return_code** is zero for successful operation. If **Return_code = 0x1052**, it means that **DestinationBuffer** size is not enough. In this case **LenOfDestBuffer** contains required size. For a **Return_code** other than zero or 0x1052, please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success (**Return_code = 0**), the **DestinationBuffer** contains **LenOfDestBuffer** bytes that have been decrypted using **AlgorithmType** and **EncryptionKeyType**.

Note

All CRYPTO-BOX USB devices for one customer are compatible during encryption and decryption with RIJNDAEL_PRIVATE_KEY or RSA algorithm (except CBU Versa) as long as the encryption keys were not reprogrammed with CBProg.

Example:

```
{ DecryptionBuffer should hold a string encrypted with the same  
seed }
```

```
AlgorithmType = "IDEA_ALGORITHM"
```

```
KeyType = "IDEA_EXTERN_KEY"
```

```
SecretSeed = hex 11117777
```

```
SubmitEncryptionKey(KeyType,length of(SecretSeed),  
pointer to (SecretSeed), NULL)
```

```
NumberOfBytes = length of (DecryptionBuffer)
```

```
DecryptEx (AlgorithmType, KeyType, NumberOfBytes,  
DecryptionBuffer, NULL, NULL)
```

```
{ The same buffer was used as a source and a target of decryp-  
tion }
```

MPI_Decrypt		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	STRING	Value	AlgorithmType
	WORD	Value	SecretSeed
	WORD	Value	NumberOfBytes
	CHAR*	Reference	DecryptionBuffer

Usage:

Obsolete function, use **MPI_DecryptEx** instead.

The string in **DecryptionBuffer** is passed to the API and then decrypted either by the CRYPTO-BOX or within the MPI library depending on the selected algorithm and the CRYPTO-BOX type. The algorithm depends on the **SecretSeed** and/or system parameters that are configured by MARX. Make sure to use identical seeds for encryption and decryption.

Note

MARX encryption algorithms are hardware specific. Be careful when you switch from one type of hardware to another, because encryption algorithm incompatibility may be encountered.

Argument Description:

AlgorithmType defines the algorithm used to decrypt the **DecryptionBuffer**. The following algorithm types are available:

IDEA_ALGORITHM	IDEA™ algorithm
MARX_ALGORITHM	MARX proprietary algorithm

DecryptionBuffer must contain **NumberOfBytes** so that the code works correctly.

SecretSeed and the selected **AlgorithmType** must be the same as for the encryption to receive correctly decrypted data.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success (**Return_code = 0**) the **DecryptionBuffer** contains **NumberOfBytes** bytes that have been decrypted using **AlgorithmType**.

Example:

```
{ DecryptionBuffer should hold a string encrypted with the same  
seed }
```

```
AlgorithmType = "IDEA_ALGORITHM"  
SecretSeed = hex 11117777  
NumberOfBytes = length of (DecryptionBuffer)
```

```
Decrypt (AlgorithmType, SecretSeed, NumberOfBytes,  
        DecryptionBuffer)
```

MPI_EncryptEx		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	STRING*	Reference	AlgorithmType
	STRING*	Reference	EncryptionKeyType
	DWORD	Value	LenOfSourceBuffer
	LPVOID	Reference	SourceBuffer
	DWORD*	Reference	pLenOfDestBuffer
	LPVOID	Reference	DestinationBuffer

Usage:

The string of bytes in **SourceBuffer** is passed to the API and then encrypted either by CRYPTO-BOX or within the MPI library depending on the selected algorithm, the encryption key type and the CRYPTO-BOX type. The algorithm depends on the encryption key value (submitted by the **MPI_SubmitEncryptionKey()** function or hardcoded in the CRYPTO-BOX) and/or system parameters that are configured by MARX. Make sure to use identical encryption key values for encryption and decryption.

Two parameters describing resulting (destination) buffer will take care of the facts that RSA is a block oriented algorithm. It's impossible to encrypt, e.g. a 117 bytes string to the same size buffer. If you use a 64 bytes RSA key, the result will have length divisible by 64. If 256 bytes key is used then the result's size should be divisible by 256.

If **DestinationBuffer** is NULL or **LenOfDestBuffer** is NULL, it means that **SourceBuffer** should be used to store encrypted data.

LenOfDestBuffer parameter should be passed "by reference" not "by value". If its value is not enough to keep encrypted data the function will return `Return_Code = 0x1052` and this parameter will contain corrected value (number of bytes required to store encrypted data).

Argument Description:

AlgorithmType defines the algorithm used to encrypt the SourceBuffer. The following algorithm types are available (supported by the MPI):

IDEA_ALGORITHM	IDEA™ algorithm
MARX_ALGORITHM	MARX proprietary algorithm
RIJNDAEL_ALGORITHM	Rijndael algorithm
RSA_ALGORITHM	RSA algorithm

EncryptionKeyType defines the type of the encryption key used to decrypt the SourceBuffer. The following key types are available:

For the IDEA_ALGORITHM:

IDEA_KEY
IDEA_EXTERN_KEY

For the MARX_ALGORITHM:

MARX_KEY
MARX_EXTERN_KEY

If being called with "extern" key, it means that software emulated version of the algorithm should be used even if active MPI device (currently opened) supports hardware implementation of the required algorithm.

For the RIJNDAEL_ALGORITHM:

RIJNDAEL_FIXED_KEY
RIJNDAEL_PRIVATE_KEY
RIJNDAEL_SESSION_KEY
RIJNDAEL_EXTERN_KEY

First three names assume hardware implemented Rijndael + corresponding key slot usage: fixed/private/session (CBU or CBS should be open). The last one can be used for any type of CRYPTO-BOX device. It assumes that an encryption key should be submitted to the MPI prior to encryption/decryption call.

Note

The CRYPTO-BOX USB devices and the CRYPTO-BOX Serial own a hardware implemented AES/Rijndael algorithm and support RSA (except CBU Versa). RSA keys will be stored inside the CRYPTO-BOX hardware, encryption and decryption is processed in the CBU device driver. IDEA and MARX (Blowfish) algorithms are software implemented (inside MPI library). The CRYPTO-BOX types 560/Net and Versa parallel are supporting AES/Rijndael and RSA via software implementation, IDEA and MARX algorithms are implemented in hardware.

Note

MARX encryption algorithms are hardware specific. Be careful when you switch from one CRYPTO-BOX type to another, because encryption algorithm incompatibility may be encountered.

For the RSA_ALGORITHM:

RSA_FIXED_PRIVATE_KEY
 RSA_FIXED_PUBLIC_KEY
 RSA_CBU_PRIVATE_KEY
 RSA_CBU_PUBLIC_KEY
 RSA_EXTERN_PRIVATE_KEY
 RSA_EXTERN_PUBLIC_KEY

Note

All CRYPTO-BOX USB devices for one customer are compatible during encryption and decryption with RIJNDAE_PRIVATE_KEY or RSA algorithm (except CBU Versa) as long as the encryption keys were not reprogrammed with CBProg. For the CRYPTO-BOX types 560/Net and Versa parallel all FIXED_KEY, IDEA_KEY and MARX_KEY are depending from the customer specific codes (defined by MARX for every customer).

First four key names assume hardware implemented RSA. For RSA_FIXED fixed key will be used, for RSA_CBU a proper key should be stored somewhere in the CBU box memory (RAM1) before encryption. The key offset and length should be told to the **MPI** prior to encryption/decryption call.

The RSA_EXTERN keys assume software emulated RSA algorithm. It can be used for any MPI box. The corresponding RSA key should be submitted to the **MPI** prior to encryption/decryption call.

DestinationBuffer must contain **LenOfDestBuffer** bytes (the same for **SourceBuffer** and **LenOfSourceBuffer**), so that the code works correctly.

If **DestinationBuffer** is **NULL**, the **SourceBuffer** will be used to store encrypted data.

Selected **AlgorithmType** and **EncryptionKeyType** must be the same as for the encryption to receive correctly encrypted data.

Results:

The `Return_code` is zero for successful operation. If `Return_code = 0x1052`, it means that `DestinationBuffer` size is not enough. In this case `LenOfDestBuffer` contains required size. For a `Return_code` other than zero and `0x1052`, please, refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113".

In case of success (**`Return_code = 0`**), the **`DestinationBuffer`** contains **`LenOfDestBuffer`** bytes that have been decrypted using **`AlgorithmType`** and **`EncryptionKeyType`**.

Example:

```
AlgorithmType = "IDEA_ALGORITHM"
KeyType = "IDEA_EXTERN_KEY"
SecretSeed = hex 11117777
SubmitEncryptionKey(KeyType,length of(SecretSeed),
pointer to (SecretSeed), NULL)
EncryptionBuffer = "A sample string to encrypt"
NumberOfBytes = length of (EncryptionBuffer)

EncryptEx (AlgorithmType, KeyType, NumberOfBytes,
           EncryptionBuffer, NULL, NULL)
{ The same buffer was used as a source and a target of encryption }
```

MPI_Encrypt		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	STRING	Value	AlgorithmType
	WORD	Value	SecretSeed
	WORD	Value	NumberOfBytes
	CHAR*	Reference	DecryptionBuffer

Usage:

Obsolete function, use **MPI_EncryptEx** instead

The string in EncryptionBuffer is passed to the API and then encrypted either by hardware or within the MPI library depending on the selected algorithm and the MARX hardware type. The algorithm depends on the **SecretSeed** and/or system parameters that are configured by MARX. Make sure to use identical seeds for encryption and decryption.

Note

MARX encryption algorithms are hardware specific. Be careful when you switch from one type of hardware to another, because encryption algorithm incompatibility may be encountered.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

AlgorithmType defines the algorithm used to encrypt the **EncryptionBuffer**. The following algorithm types are available:

IDEA_ALGORITHM	IDEA™ algorithm
MARX_ALGORITHM	MARX proprietary algorithm

EncryptionBuffer must contain **NumberOfBytes** so that the code works correctly.

SecretSeed and the selected **AlgorithmType** must be the same as for the decryption to receive correctly decrypted data.

Example:

```
AlgorithmType = "IDEA_ALGORITHM"
```

```
SecretSeed = hex 11117777
```

```
EncryptionBuffer = "A sample string to encrypt"
```

```
NumberOfBytes = length of (EncryptionBuffer)
```

```
Encrypt (AlgorithmType, SecretSeed, NumberOfBytes,  
EncryptionBuffer)
```

MPI_EraseEncryptionKey

| CBU, CB560/Versa, CBS

Argument List:**Type****Passed by****Description**

STRING*

Reference

EncryptionKey

Usage:

This function can be used whenever an encryption key should be removed from the pool. It is also useful to delete all previously submitted encryption keys as hacker defense.

Results:

The Return_code is zero for a successful operation. For a Return_code other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113. Encryption/Decryption functions that require the erased encryption keys won't perform encryption/decryption based on these keys anymore and can only be used after re-submission using the MPI_SubmitEncryptionKey function.

Argument Description:

EncryptionKey defines the key type. Possible values:

For the IDEA_ALGORITHM:

IDEA_EXTERN_KEY

For the MARX_ALGORITHM:

MARX_KEY

MARX_EXTERN_KEY

For the RIJNDael_ALGORITHM:

RIJNDael_EXTERN_KEY

For the RSA_ALGORITHM:

RSA_CBU_PRIVATE_KEY

RSA_CBU_PUBLIC_KEY

RSA_EXTERN_PRIVATE_KEY

RSA_EXTERN_PUBLIC_KEY

To erase all keys submit **ERASE_ALL**.

Example:

The following pseudo code shows how to erase all encryption keys.

```
PasswordType = "ERASE_ALL"  
MPI_EraseEncryptionKey (PasswordType)
```

Note

For more detailed information concerning encryption keys see **MPI_SubmitEncryptionKey** function description.

MPI_ErasePassword

CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING	Value	PasswordType

Usage:

This function can be used whenever a password should be removed from the password pool. It is also useful to delete all passwords as hacker defense.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Functions that require the erased password won't be accessible anymore and can only be accessed after re-submission using the **MPI_SubmitPassword** function.

Argument Description:

PasswordType defines the password type. Possible values:

For CBU and CBS:

PASSWORD_ID1 Read ID1
 PASSWORD_ID2 Read ID2

For CBN/CBV:

PASSWORD_ID1 Read ID1
 PASSWORD_ID2 Read ID2
 PASSWORD_ID3 Read ID3 (CBN)
 PASSWORD_ID4 Read ID4
 PASSWORD_ID5 Read ID5
 PASSWORD_ID6 Read ID6 (CBN)
 PASSWORD_ID7 Read ID7 (CBN)
 PASSWORD_ID8 Read ID8 (CBN)

Memory passwords:

PASSWORD_MEM1 To access RAM1

PASSWORD_MEM2 To access RAM2 (CBN)

To erase all passwords submit ERASE_ALL.

Example:

The following pseudo code shows how to erase all access passwords.

```
PasswordType = "ERASE_ALL"  
MPI_ErasePassword (PasswordType)
```

MPI_GenerateKeyPairRSA

| CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	DWORD	Value	RSAKeyLength
	LPVOID	Reference	PrivateKeyBuffer
	LPVOID	Reference	PublicKeyBuffer
	DWORD*	Reference	KeyBufferLength

Usage:

This function generates unique pair of RSA keys (private and public). RSA keys of this pair can be used further for RSA encryption/decryption (see **MPI_EncryptEx/MPI_DecryptEx: RSA_ALGORITHM**). You can store generated keys of this pair somewhere in the program memory (loaded from external memory) or they can be stored in the CRYPTO-BOX memory (MPI_WriteMem).

If the CRYPTO-BOX USB is currently open and RSA keys were written to its memory they should be referred as `RSA_CBU_PRIVATE_KEY` and/or `RSA_CBU_PUBLIC_KEY` for encryption.

If keys are stored in the program memory they should be referred as: `RSA_EXTERN_PRIVATE_KEY` and/or `RSA_EXTERN_PUBLIC_KEY`.

Argument Description:

`RSAKeyLength` - number of bits for RSA modulus (RSA Key length).

Only one of the following values can be used (defined in MPI.H):

2048 bit - `MPI_RSA_KEY_MODULUS_2048`

1024 bit - `MPI_RSA_KEY_MODULUS_1024`

512 bit - `MPI_RSA_KEY_MODULUS_512`

PrivateKeyBuffer - buffer to store generated RSA private key: its size must be not less than: $(2 + ((\text{RSA_KeyBits} / 8) * 2))$ bytes. You can use `MPI_RSA_KEY_BUF_LEN(x)` (see MPI.H) to define its size.

PublicKeyBuffer - buffer to store generated RSA public key: its size also must be not less than: $(2 + ((\text{RSA_KeyBits} / 8) * 2))$ bytes. The same macro could be used.

KeyBufferLength - length (in bytes) of allocated buffer for public or private key

(if these buffers have different size - smallest of sizes should be specified).
After execution contains actual key size. The same expression can be used for its calculation: $(2 + ((RSA_KeyBits / 8) * 2))$ bytes.

Results:

The Return_code is zero for successful operation. If Return_code = 0x1052, the size of one or both buffers is not enough. For a Return_code other than zero and 0x1052, please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success (**Return_code = 0**), the **PrivateKeyBuffer** contains generated private key and **PublicKeyBuffer** - public key. The **KeyBufferLength** contains RSA key length in bytes.

MPI_GetBoxInfo		CBU,CB560/Versa,CBS	
Argument List:	Type	Passed by	Description
	DWORD	Value	Identifier
	WORD*	Reference	Model
	DWORD*	Reference	MemorySize

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

Identifier - device Handle obtained by MPI_SearchFirst()/SearchNext() calls.

Model - returns model info for CRYPTO-BOX USB:

- 1 - CBU VERSA
- 2 - CBU XS
- 3 - CBU XL
- 4 - CrypToken OEM XS
- 5 - CrypToken OEM XL

MemorySize - returns available memory size in bytes

MPI_GetBoxType

CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING10	Reference	BoxType

Usage:

This function retrieves the CRYPTO-BOX type that is using the active channel.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

BoxType returns:

CBU for CRYPTO-BOX USB
 CBV for a CRYPTO-BOX Versa
 CBN for a CRYPTO-BOX 560
 CBS for a CRYPTO-BOX Serial

Example:

The following sample in pseudo code shows how to find out what kind of security device is using the active channel. Remember to establish a communication channel first.

```
ErrorCode = MPI_GetBoxType (BoxType)
```

```
If ErrorCode == MPI_SUCCESS then
  Print "BoxType = ", BoxType
Else print "Error = ", ErrorCode
```

MPI_GetDeveloperId		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	DWORD	Reference	DeveloperID

Usage:

Returns a unique ID assigned by MARX to each customer. Do not confuse it with a serial number. The developer ID is the same for **ALL** CRYPTO-BOX devices of a batch shipped to a MARX customer.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

DeveloperID uniquely identifies each MARX customer.

MPI_GetDrvVersion | CB560/Versa

Argument List:	Type	Passed by	Description
	DWORD	Value	Identifier
	DWORD	Reference	DrvVersion

Usage:

To find out the driver version of the CRYPTO-BOX that is using the active channel. It is not necessary to open a channel via MPI_Open to call this function. However, MPI_SearchFirst or MPI_SearchNext should be called to get an Identifier.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

Identifier A valid Identifier obtained from MPI_SearchFirst or MPI_SearchNext.

DrvVersion will hold the driver version.

MPI_GetHash		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	STRING*	Reference	AlgorithmType
	WORD	Value	SourceBufLength
	LPVOID	Reference	SourceBuffer
	WORD*	Reference	HashBufLength
	LPVOID	Reference	HashBuffer

Usage:

This function calculates hash value for submitted source buffer. Can be used in different password/user authorization/verification solutions. Typical scenarios are:

- UNIX password management: system stores not real passwords but hash values calculated from passwords;
- Challenge response authentication approach:
 - a random number is sent by a server to a client;
 - the client concatenates this number to the password, calculates hash and sends this hash value back to the server;
 - server performs the same operations and compares hashes.

Argument Description:

AlgorithmType defines the algorithm used. Only one algorithm is currently supported: **MD4_ALGORITHM**.

SourceBufLength defines the size of source buffer.

SourceBuffer must contain this number of bytes to make the algorithm works correctly.

HashBufLength must be at least 32 bytes (MPI_MD4_HASH_BUF_LEN).

HashBuffer addresses buffer to contain calculated hash string.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success (**Return_code = 0**), the **HashBuffer** contains hash value - string of 32 bytes calculated for the contents of **SourceBuffer**.

MPI_GetLastError		CBU, CB560/Versa, CBS	
Argument List:	Typ	Passed by	Description
	DWORD*	Reference	LastError
	STRING80	Reference	ErrorMessage

Usage:

To retrieve the error code of the latest function call and a corresponding error message.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

LastError returns the error code of the latest function call.

ErrorMessage returns an error message as a NULL-terminated string.

Example:

```
ErrorCode = 0; ErrorMessage = ""
```

```
MPI_Open (Identifier)
MPI_GetLastError (ErrorCode, ErrorMessage)
Print "Error=", ErrorCode, ": ", ErrorMessage
```

MPI_GetLicenseInfo

| CBU, CB560, CBS – NETWORK ONLY

Argument List:	Type	Passed by	Description
	DWORD	Value	Identifier
	DWORD	Reference	MaximumLicenses
	DWORD	Reference	UsedLicenses

Usage:

This function allows you to find out about the number of available user licenses. It can be used any time a remote channel is open.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success, the remote access to a CRYPTO-BOX device will include your parameter.

Argument Description:

Identifier is a valid Identifier obtained from **MPI_SearchFirst** or **MPI_SearchNext** functions.

MaximumLicenses returns the max number of user licenses.

UsedLicenses returns the number of licenses currently used.

Example:

```
Identifier = 0
MaximumLicenses = 0
UsedLicenses = 0
MPI_GetLicenseInfo(Identifier, MaximumLicenses, UsedLicenses)
print UsedLicenses + "of " + MaximumLicenses+"are used"
```

MPI_GetNetworkParameter

| CBU, CB560, CBS – NETWORK ONLY

Argument List:	Type	Passed by	Description
	STRING	Value	NetworkParameter
	STRING	Reference	ParameterValue

Usage:

This function allows you to query a specified network parameter. The string format allows easy display of parameters as server name, timeout and UDP port.

Please refer to **MPI_SetNetworkParameter** for further explanations.

Results:

The **Return_code** is zero for successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success, the remote access to a CRYPTO-BOX device will include your parameter.

Argument Description:

The following **NetworkParameters** can be queried:

SERVER, PROTOCOL, TIMEOUT, CLIENT_UDP_PORT, SERVER_UDP_PORT

"**SERVER**" Specifies server name(s) or address(es).

"**PROTOCOL**" Specifies the network protocol.

"**TIMEOUT**" Specifies Network timeout in milliseconds.

"**CLIENT_UDP_PORT**" Client UDP port for receiving.

"**SERVER_UDP_PORT**" Server UDP port for listening.

MPI_GetRandomSequence | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	DWORD	Value	Len
	LPVOID	Reference	Buffer

Usage:

This function allows you to obtain a random sequence of bytes desired length. The hardware implemented internal random generator will be used if CBU device is open, in other case random sequence will be generated by the MPI.

Please refer to **MPI_SetNetworkParameter** for further explanations.

Results:

The Return_code is zero for a successful operation. For a Return_code other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113

In case of success the Buffer will be filled by a random sequence.

MPI_GetSerialNr		CBU,CB560, CBS	
Argument List:	Type	Passed by	Description
	DWORD	Reference	SerialNumber

Note

For CBU Versa the Serial number is the same for all modules of one batch.

Usage:

Returns a unique serial number assigned by MARX to each CRYPTO-BOX USB XS or XL.. Do not confuse it with a Developer ID. The Developer ID is the same for **ALL** CRYPTO-BOX devices of a batch shipped to a MARX customer.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

SerialNumber uniquely identifies each CRYPTO-BOX device (if supported by the chosen CRYPTO-BOX type).

MPI_Open | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	DWORD	Value	Identifier

Usage:

Opens a communication channel with a CRYPTO-BOX. Before you call **MPI_Open** you need to

- submit access codes via **MPI_SubmitPassword**;
- set a channel via **MPI_SetChannel** and
- search for MARX hardware via **MPI_SearchFirst**.
- For remote access **MPI_SetNetworkParameter()** needs to be called before **MPI_Open()**.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

Identifier is a valid Identifier obtained from **MPI_SearchFirst** or **MPI_SearchNext** functions.

Example:

The following pseudo code sample illustrates how to open a communication channel for a CRYPTO-BOX device.

```
{ MPI_Open takes one parameter. This parameter is an Identifier
returned by MPI_SearchFirst or MPI_SearchNext. Use "0" as
Identifier if you do not want to use MPI_SearchFirst}
ErrorCode = MPI_Open(Identifier)
...
{ close the active channel }
MPI_Close (Identifier)
```

MPI_ProcessDataObject		CBU,CB560/Versa,CBS	
Argument List:	Type	Passed by	Description
	DWORD	Value	ObjectType
	DWORD	Value	MemoryBank
	DWORD	Value	MemoryAddress
	DWORD	Value	Operation
	LPVOID	Reference	pObjectData
	DWORD *	Reference	pDataLen
	LPVOID	Reference	pDOBuffer

Usage:

MPI interface to Data Objects. Allows to create (SET) required object, read its current value (GET), check/verify it, increment/decrement value.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Note

Please refer to the sample programs in the \samples subdirectory of the MARX PPK, which contains source code for a sample program "datatobj" that demonstrates the usage of Data Objects for Visual C, Visual Basic, Delphi and many more Windows programming environments.

Argument Description:

ObjectType is a valid Data Object type. Currently the following types are supported:

- **Expiration Date** (MPI_DO_EXPIRATION_DATE) - allows to SET date after which protected program/module (or feature) will be expired;
- **Number of Days** (MPI_DO_NUMBER_OF_DAYS) - number of days, after which protected program/module (or feature) will be expired - unlike Expiration Date is a relative value - allowed period starts after activation;
- **Time Allowed** (MPI_DO_TIME_ALLOWED) - helps to limit program/feature by usage time (say, it is allowed to use a program for 10 hours);
- **Usage Counter** (MPI_DO_USAGE_COUNTER) - allows to control number of execution/launches for a program/feature;
- **Memory Object** (or User Defined Object) (MPI_DO_MEMORY) - convenient container for secure storage of any structured data - like personal info, financial info, etc.

MemoryBank - number of memory bank (1 or 2) - actual for CBN devices only.

MemoryAddress - object's address in the device memory.

Operation - required operation/method:

- **SET** (MPI_DO_OP_SET) is implemented for all object types. Allows to set value - initialize/update object;
- **GET** (MPI_DO_OP_GET) also valid method for all types of objects. Allows to read current value;
- **INCREMENT** (MPI_DO_OP_INC) increments current value of the object: <days> for Expiration Date and Number of Days, <secs> for Time Allowed, number of runs for Usage Counter;
- **DECREMENT** (MPI_DO_OP_DEC) decrements current value of the object: <days> for Expiration Date and Number of Days, <secs> for Time Allowed, number of runs for Usage Counter;
- **VERIFY** (MPI_DO_OP_VERIFY) - checks current object's value - for all predefined types;
- **CLEAR** (MPI_DO_OP_CLEAR) - clears current value - for all predefined types;
- **UNLIMITED** (MPI_DO_OP_UNLIMITED) - sets object's value to "unlimited".

MPI_ReadID		CBU, CB560/Versa, CBS	
Argument List:	Type	Passed by	Description
	DWORD	Value	IdNr
	DWORD	Reference	IdCode

Usage:

To retrieve the value of an ID code. To get a successful response, you need to submit a correct access code that is required to read the specified ID. The number of available ID codes is different for the several CRYPTO-BOX types.

You can find an overview for the CRYPTO-BOX shipped with your evaluation kit in "Appendix A: Codes of a CRYPTO-BOX® Evaluation Key" at page 173. For your customer specific CRYPTO-BOX you will get additionally a production sheet with your customer specific ID codes.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

IdNr number of an ID. Remember to submit a correct access code to retrieve the value of the ID.

IdCode returns the value of the requested ID.

Example:

In the following sample we will read the value of ID1. Remember: a communication channel should be opened by now.

```

IdNr           = 1           { read ID1 }
IdCode        = 0           { IdCode will return the value of ID1 }

ErrorCode     = MPI_ReadID (IdNr, IdCode)
if ErrorCode  = MPI_SUCCESS then
print "ID1   = ", IdCode
else
    print "Error = ", ErrorCode

```

MPI_ReadMem | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	DWORD	Value	MemoryNr
	DWORD	Value	MemoryAddress
	DWORD	Value	BufferLength
	CHAR*	Reference	DataBuffer

Usage:

To read an area of the CRYPTO-BOX memory (RAM). The call will not be completed successfully unless you submit correct access codes via MPI_SubmitPassword. You need to submit both the ID1 and the RAM passwords to be able to work with the CRYPTO-BOX memory.

The size of the CRYPTO-BOX memory depends from the model. You will find an overview under "Appendix: Technical Data" beginning at page 178.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113

Argument Description:

MemoryNr can be one of the following:

- MPI_MEMORY_NR1 or/and MPI_MEMORY_NR2 for CBN
- MPI_MEMORY_NR1 only for CBV

MemoryAddress is a 0-based value. The total amount of available memory depends on the type of the security device.

BufferLength is the length of the buffer to hold returned data.

DataBuffer returns the data that has been read from the memory of the security device.

Note

The CRYPTO-BOX memory content will remain unchanged also when the CRYPTO-BOX is removed from the computer or the PC is switched off.

Example:

The following sample demonstrates how to read a 20-byte buffer at physical offset 10 from RAM1 of a CRYPTO-BOX device.

```
MemoryNr      = MPI_MEMORY_NR1
{start reading at physical memory offset 10 }
MemoryAddress = 10
{read 40 bytes of data }
BufferLength  = 20
DataBuffer    = "
MPI_ReadMem (MemoryNr, MemoryAddress, BufferLength,
             DataBuffer)
```

MPI_SearchFirst | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING10	Reference	HardwareType
	STRING10	Reference	Port
	DWORD	Reference	Identifier

Usage:

To search for the first available MARX Hardware that is accessible through the 1st ID password. This password must be submitted via SubmitPassword before calling this function.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

HardwareType returns

- CBU (for a CRYPTO-BOX USB)
- CBV (for a CRYPTO-BOX Versa)
- CBN (for a CRYPTO-BOX 560)
- CBS (for a CRYPTO-BOX Serial)

Local Port values can be

- PCMCIA
- COM1, COM2, COM3, COM4
- LPT1, LPT2, LPT3, LPT4
- USB

Remote (=network) **Port** returns can be any server name or server address in a network.

Argument Description:

HardwareType describes the CRYPTO-BOX type attached.

Port describes the local port (USB, COM, LPT,) the CRYPTO-BOX is attached to or, in a network the server and protocol that responded to a network request.

Note

Using MPI_SearchFirst / MPI_SearchNext to identify CRYPTO-BOX modules with the same customer specific codes works only for the CBU XS and XL. For all other CRYPTO-BOX types always the FIRST device which was found by MPI will return an answer.

Identifier is needed to connect to exactly the CRYPTO-BOX that was found via `MPI_SearchFirst` or `MPI_SearchNext`.

Example:

The following sample shows how to look up specified CRYPTO-BOX devices. Before calling **`MPI_SearchFirst`** you need to set a channel via **`MPI_SetChannel`** and submit correct access code(s) via **`MPI_SubmitPassword`**.

```
HardwareType = ""
Port = ""
Identifier = 0
{ some functions you will be calling later on will need this
Identifier }
```

```
ErrorCode = MPI_SearchFirst (HardwareType, Port, Identifier)
If ErrorCode == MPI_SUCCESS
    print "Device type = ",HardwareType
    print "Port = ", Port
    print "Identifier = ", Identifier
```

MPI_SearchNext | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING10	Reference	HardwareType
	STRING10	Reference	Port
	DWORD	Reference	Identifier

Usage:

To search for the next available CRYPTO-BOX that is accessible through the 1st ID password. You need to call **MPI_SearchFirst** to start the look-up process before calling this function.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

HardwareType returns

CBU (for a CRYPTO-BOX USB)
 CBV (for a CRYPTO-BOX Versa)
 CBN (for a CRYPTO-BOX 560)
 CBS (for a CRYPTO-BOX Serial)

Local Port returns can be

USB
 COM1, COM2, COM3, COM4
 LPT1, LPT2, LPT3, LPT4

Remote (=network) Port returns can be any server name or server address in a network.

Argument Description:

HardwareType describes the type of CRYPTO-BOX type attached.

Port describes the local port (USB, COM, LPT) the CRYPTO-BOX is attached to or stands for the server and protocol that makes the CRYPTO-BOX available in a network.

Identifier is needed to connect to exactly the CRYPTO-BOX that was found via MPI_SearchFirst or MPI_SearchNext.

Example:

See sample for MPI_SearchFirst.

MPI_SetChannel | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING	Value	HardwareType
	STRING	Value	Port
	VOID	Reference	NotUsed

Usage:

All CRYPTO-BOX types can be accessed through the **MPI**. This function allows control over the various channels that are available. A channel is defined by the **HardwareType** and the port (local or remote) that the hardware is connected to. The latter is represented by the **Port** parameter.

This function is the key to the flexibility of the **MPI interface** and allows an easy switch between solutions just by changing the **MPI_Channel**. Channels can be dedicated or non-dedicated and the order of channel submission defines the search order. This way multiple CRYPTO-BOX devices can be accessed without a single line of code that needs to be changed.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success the system is initialized on the software level. No physical access occurs. The API is set up for further operation with **MPI_SearchFirst**.

Argument Description:

The **HardwareType** is a descriptor of the CRYPTO-BOX used with one of the following values:

AUTO	find ANY CRYPTO-BOX automatically (allow processing time!)
CBU	(CRYPTO-BOX USB)
CBV	(CRYPTO-BOX Versa)
CBN	CRYPTO-BOX 560)
CBS	(CRYPTO-BOX Serial)

The **Port** defines the physical or remote port that the CRYPTO-BOX is connected through. This parameter can have one of the following values:

Local ports:

AUTO	access any available LPT, COM, USB port.
USB	access any USB port
LPT	access any LPT port available
LPT1	access LPT 1
LPT2	access LPT 2
LPT3	access LPT 3
COM	access any COM port available
COM1	access COM1
COM2	access COM2
COM3	access COM3

Remote ports:

AUTONET	any server via any available network protocol
SPX_IPX	via IPX/SPX protocol
TCP_IP	via TCP/IP protocol
NETBIOS	via NetBIOS protocol

Example 1:

```
HardwareType = "CBN"
LocalPort = "AUTO"
Reserved = 0
MPI_SetChannel (HardwareType, LocalPort, Reserved)
```

This code snippet initializes a system to look for a CRYPTO-BOX 560 on any local

port that is available. This function goes hand-in-hand with MPI_SubmitPassword and MPI_SearchFirst.

Here we set the channel to access SMARX-CARDS via any available smart card reader channel.

Example 2:

```
HardwareType = "CBN"  
LocalPort = "AUTO"  
RemotePort = "AUTONET"  
Reserved = 0  
MPI_SetChannel (HardwareType, LocalPort, Reserved)  
RemotePort = "AUTONET"  
MPI_SetChannel (HardwareType, RemotePort, Reserved)
```

Here we initialize a system that checks first the presence of a CRYPTO-BOX 560 on all available local ports and then on the remote (=network) ports.

Example 3:

```
HardwareType = "CBN"  
RemotePort = "AUTONET"  
Reserved = 0  
MPI_SetChannel (HardwareType, RemotePort, Reserved)
```

Here we initialize a system that we look first for a CRYPTO-BOX 560 on all available remote ports (=network) ports. The default search order defined by AUTONET is TCP/IP, IPX/SPX and NetBIOS.

MPI_SetNetworkParameter

| CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING	Value	NetworkParameter
	STRING	Value	ParameterValue

Usage:

This function allows you to configure network specific parameters as server addresses, server names and protocol settings.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success, the remote access to a CRYPTO-BOX device will include your parameter.

Argument Description:

The following **NetworkParameters** are available and described below.
SERVER, PROTOCOL, TIMEOUT, CLIENT_UDP_PORT, SERVER_UDP_PORT

"SERVER"

Specifies server name(s) or address(es) of the CRYPTO-BOX Server (CBNet Server).

Example:

```
MPI_SetNetworkParameter("SERVER", "AURORA");
```

Possible server identification ways:

"*" - all servers;

"10.10.10.10" - specified by IP address;

"AURORA" - specified by name;

"10.10.10.10 267.11.13.197 AURORA" - list of servers separated by spaces.

"PROTOCOL" Specifies the network protocol

Example:

```
char NetProtocol[32];
```

```
MPI_GetNetworkParameter("PROTOCOL", NetProtocol);
```

"TIMEOUT" Specifies Network timeout in milliseconds.
 Maximum time that client is waiting for answer from server.
 Default value: 2 seconds.

Example (sets timeout to 5 seconds):

```
MPI_SetNetworkParameter("TIMEOUT", "5000")
```

"CLIENT_UDP_PORT" Client UDP port for receiving:
 Default value: 8765.

Example (sets client UDP port to 8000 port):

```
MPI_SetNetworkParameter("CLIENT_UDP_PORT", "8000")
```

"SERVER_UDP_PORT" Server UDP port for listening:
 Default value: 8766.

Example:

```
MPI_SetNetworkParameter("SERVER_UDP_PORT", "8001")
```

Sets the server UDP port to 8001 port.

The following NetworkValues are reserved

DELETE	deletes a parameter entry
DEFAULT	sets a parameter to default value

Example 1:

The following pseudo code sets the server IP address. Every TCP/IP access to a CBNet Svrer will use this address.

```
NetworkParameter = "SERVER"
ParameterValue = "10.10.10.10"
RetVal = 0
MPI_SetNetworkParameter(NetworkParameter, ParameterValue)
```

Example 2:

You can combine multiple addresses by separating IP addresses by spaces.

```
NetworkParameter = "SERVER"
ParameterValue = "10.10.10.10 267.11.13.197"
MPI_SetNetwork(NetworkParameter, ParameterValue)
```

MPI_SubmitEncryptionKey

| CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING*	Reference	EncryptionKeyType
	DWORD	Value	Len
	LPVOID	Reference	EncryptionKey
	DWORD	Value	Offset

Usage:

This function is similar to the **MPI_SubmitPasswordEx ()** function. It allows to submit encryption key for its further utilization by the proper encryption algorithm (see **MPI_EncryptEx()** and **MPI_DecryptEx()** functions).

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

EncryptionKeyType defines the type of the encryption key.

Possible values are:

```

IDEA_KEY
IDEA_EXTERN_KEY
MARX_KEY
MARX_EXTERN_KEY
RIJNDael_PRIVATE_KEY
RIJNDael_SESSION_KEY
RIJNDael_EXTERN_KEY
RSA_CBU_PRIVATE_KEY
RSA_CBU_PUBLIC_KEY
RSA_EXTERN_PRIVATE_KEY
RSA_EXTERN_PUBLIC_KEY

```

The function interpretation by the **MPI** strongly depends on:

- a) type of open CRYPTO-BOX device;
- b) related algorithm.

Encryption key submission for hardware implemented algorithm is allowed only if a proper CRYPTO-BOX (supporting this algorithm) is currently open.

All "**_EXTERN_**" keys assume usage of software emulated algorithms and can be used for any type of CRYPTO-BOX. It is necessary to submit a proper encryption key before calling one of these algorithms (**MPI_EncryptEx()/MPI_DecryptEx()**).

The **RIJNDael_FIXED** key (active CRYPTO-BOX USB is assumed) has one unchangeable value of the fixed key stored in the box system memory. So, encrypt/decrypt should be called without submitting a key first.

The **RIJNDael_SESSION** and **RIJNDael_PRIVATE** keys are hardware implemented for the CBU. Encryption/decryption can be used without submission of an encryption key. Although it is possible to call the **MPI_SubmitEncryptionKey()** to change the value of the key.

For the **RSA_CBU** keys (open CRYPTO-BOX USB is assumed) it is necessary to call the **MPI_SubmitEncryptionKey ()** before Encryption/Decryption itself. There are two possible scenarios depending on the EncryptionKey pointer value.

If **EncryptionKey** pointer is NULL, it means that application specifies offset and length for the already existing RSA key (stored in the CBU memory). If this pointer is not NULL, it means that the key should be written to the CBU memory. The **Offset** parameter value will be used for offset.

So, the **Offset** parameter value is actual only for RSA_CBU keys to define correct location of the submitted key in the CBU internal memory. For other types of keys it is meaningless and can be 0.

Example:

The following pseudo code shows how to submit an encryption key for its further usage for encryption/decryption by software emulated MARX algorithm.

```
KeyType    = "MARX_EXTERN_KEY"  
Key        = hex 2a2b2c2d  
MPI_SubmitEncryptionKey(KeyType,length of (Key), Key, 0)  
{Offset is required only for RSA_CBU keys to define correct  
location of the submitted key in the CBU internal memory}.
```

MPI_SubmitPassword | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING	Value	PasswordType
	STRING	Value	AccessMode
	DWORD	Value	Password

Usage:

To submit passwords so that a CRYPTO-BOX can be accessed, ID codes can be checked or memory accessed.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

PasswordType defines the password type. Possible values:

For CBU and CBS:

PASSWORD_ID1 Read ID1
PASSWORD_ID2 Read ID2

For CBN/CBV:

PASSWORD_ID1 Read ID1
PASSWORD_ID2 Read ID2
PASSWORD_ID3 Read ID3 (CBN)
PASSWORD_ID4 Read ID4
PASSWORD_ID5 Read ID5
PASSWORD_ID6 Read ID6 (CBN)
PASSWORD_ID7 Read ID7 (CBN)
PASSWORD_ID8 Read ID8 (CBN)

Memory passwords:

PASSWORD_MEM1 To access RAM1
PASSWORD_MEM2 To access RAM2 (CBN)

Note

This function allows to submit password only as DWORD type values. Because of this limitation, this function is not suitable for the CRYPTO-BOX USB! Please use the function MPI_SubmitPasswordEx () instead to submit password as a string of bytes!

CRYPTO-BOX devices have passwords to read an ID code and to access memory. Furthermore, functions such as expiration date checks and usage counters can be protected with access codes that need to be submitted BEFORE calling any function that requires passwords.

AccessMode can be one of the following:

READ Allows to read IDx.

WRITE Allows to read from and write to the memory.

Example 1:

The following pseudo code shows how to submit a password and accesses memory section 1 of an CBN/CBV with demo memory access password.

```

PasswordType = "PASSWORD_MEM1"
AccessMode   = "WRITE"
Password     = hex a1a2a3a4
MPI_SubmitPassword (PasswordType, AccessMode, Password)
MPI_WriteMem (we discuss these parameters elsewhere)

```

Example 2:

The following pseudo code shows how to submit a password to check ID 1 of a CRYPTO-BOX.

```

PasswordType = "PASSWORD_ID1"
AccessMode   = "READ"
Password     = hex 111213
MPI_SubmitPassword (PasswordType, AccessMode, Password)
MPI_ReadID (we discuss these parameters elsewhere)

```

MPI_SubmitPasswordEx | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING	Value	PasswordType
	STRING	Value	AccessMode
	LPVOID	Value	Password
	DWORD	Reference	PasswordLen

Usage:

To submit passwords so that a CRYPTO-BOX can be accessed, ID codes can be checked or memory accessed. This function allows to submit password of a variable length as a string of bytes in contrast to MPI_SubmitPassword() function, which assumes password as a DWORD value only. Variable length PASSWORD_ID1 is very important for the CRYPTO-BOX USB device.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

PasswordType defines the password type. Possible values:

For CBU and CBS:

PASSWORD_ID1 Read ID1
 PASSWORD_ID2 Read ID2

For CBN/CBV:

PASSWORD_ID1 Read ID1
 PASSWORD_ID2 Read ID2
 PASSWORD_ID3 Read ID3 (CBN)
 PASSWORD_ID4 Read ID4
 PASSWORD_ID5 Read ID5
 PASSWORD_ID6 Read ID6 (CBN)
 PASSWORD_ID7 Read ID7 (CBN)
 PASSWORD_ID8 Read ID8 (CBN)

Memory passwords:

PASSWORD_MEM1 To access RAM1

PASSWORD_MEM2 To access RAM2 (CBN)

CRYPTO-BOX devices have passwords to read an ID code and to access memory. Furthermore, functions such as expiration date checks and usage counters can be protected with access codes that need to be submitted BEFORE calling any function that requires passwords.

AccessMode can be one of the following:

READ Allows to read IDx.**WRITE** Allows to read from and write to the memory.**PasswordLen** length of password string**Password** string of bytes, containing password**Example:**

The following pseudo code shows how to submit how to submit a password to check ID 1 of a CRYPTO-BOX USB with demo access password.

```

PasswordType   = "PASSWORD_ID1"
AccessMode     = "READ"
Password       = [ 0x64, 0x65, 0x6D, 0x6F]           { "demo" }
PasswordLen    = length of (Password)                { 4 }
{ Password -   reference to a byte sequence (array)
  PasswordLen - length of the array (number of bytes)
}
MPI_SubmitPasswordEx (PasswordType, AccessMode, Password,
                    PasswordLen)

```

MPI_ReadID (we discuss these parameters elsewhere).

It is also possible to use this function for CBN/CBV:

```

...
Password       = [ 0x11, 0x12, 0x13]
PasswordLen    = length of (Password)                { 3 }
...

```

MPI_WhatsAvailable | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	STRING10	Reference	HardwareType
	STRING10	Reference	PortType
	DWORD	Reference	NumberOfTypes

Usage:

To look for any CRYPTO-BOX device that is available - locally or remotely. If this function doesn't find anything it is not worth looking any further. No passwords are necessary. This is a presence check that doesn't make a difference between hardware with DEMO configuration or hardware that is issued for a specific customer. Every CRYPTO-BOX will respond and the number of attached hardware types can be determined too. Expect a little longer response time for this function when setting both ports and hardware type to AUTO or AUTONET in MPI_SetChannel, because of the large variety of solutions MARX provides.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

In case of success, **HardwareType** contains the MARX hardware type that was found last, **PortType** contains the name of the port where the hardware was detected, and **NumberOfTypes** contains the number of Marx Hardware types found.

Argument Description:

HardwareType contains the MARX hardware type that was found last:

CBU (CRYPTO_BOX USB)
 CBN (CRYPTO-BOX 560)
 CBS (CRYPTO-BOX Serial)
 CBV (CRYPTO-BOX Versa)

PortType contains a string representation of the port where CRYPTO-BOX was found.

NumberOfTypes contains the number of CRYPTO-BOX types found.

Example:

The following pseudo code initializes two parameters with zero and passes them to the API. As they are passed by reference, they contain values for the type of hardware and the number of hardware keys found.

```
MPI_SetChannel ("CBN", "AUTO", 0)
MarxHardwareType = "No Hardware found"
Found = 0
if MPI_WhatIsAvailable(MarxHardwareType, PortType, FoundSoFar) =
0
then
print "Found" + FoundSoFar + "MARX hardware keys on" + PortType
print "The last hardware found was of type" + MarxHardwareType
endif
```

MPI_WriteMem | CBU, CB560/Versa, CBS

Argument List:	Type	Passed by	Description
	DWORD	Value	MemoryNr
	DWORD	Value	MemoryAddress
	DWORD	Value	BufferLength
	CHAR*	Reference	DataBuffer

Usage:

To write an area of CRYPTO-BOX memory (RAM) . The call will not be completed successfully unless you submit correct access codes via **MPI_SubmitPassword**. You need to submit both the ID1 and the RAM passwords to be able to work with the CRYPTO-BOX memory.

Results:

The **Return_code** is zero for a successful operation. For a **Return_code** other than zero please refer to the section "10.6 Return Codes of CRYPTO-BOX devices" on page 113.

Argument Description:

MemoryNr can be one of the following:

- MPI_MEMORY_NR1 or/and MPI_MEMORY_NR2 for CBN
- MPI_MEMORY_NR1 only for CBV

MemoryAddress is a 0-based value. The total amount of available memory depends on the type of the security device.

BufferLength is the length of the buffer to hold returned data.

DataBuffer returned the data that has been read from the memory of the security device.

Example:

The following sample demonstrates how to write a 20-byte buffer at physical offset 10 from memory 1 of a CRYPTO-BOX.

```
MemoryNr = MPI_MEMORY_NR1
{ start reading at physical memory offset 10 }
MemoryAddress = 10
BufferLength = 20 { write 20 bytes of data }
DataBuffer = "12345678901234567890"
MPI_WriteMem (MemoryNr, MemoryAddress, BufferLength, DataBuffer)
```

11. Appendix

In this Appendix you will find technical data and specifications of the CRYPTO-BOX system

Appendix A: Codes of a CRYPTO-BOX**Codes of a CRYPTO-BOX USB - and Serial Evaluation Kit**

Reference	Value (hex)	Value (dez)	Description
PWM	„admin“ (as String)		Master Password
PW_RAM	2A2B2C2D	707472429	Password for read/write access to the CRYPTO-BOX memory
Scodel1	„demo“ (as String) or 64656D6F (hex)		Security code for ID1 (necessary to open CRYPTO-BOX)
IDCode1	11121314	286397204	ID-Code 1
Scodel2*	22222222	572662306	Security Code for ID2
IDCode2*	21222324	555885348	ID Code 2

Note

* Codes are programmable with AutoCrypt Wizard or CBProg.

Codes of a CRYPTO-BOX 560/Net, Versa Evaluation Kit

Reference	Value	Description
PWM	f1f2f3f4f5 hex	Master Password (fixed)
SCodeSER	4142 hex (identical for all keys!)	security code to get the serial number or batch code (fixed)
SerNum	434445 hex	serial number (fixed)
PW_RAM1	a1a2a3a4 hex	password for read/write access to the CB-memory RAM1 (fixed)
PW_RAM2*	a1a2a3a4 hex	password for read/write access to the CB-memory RAM2 (var.)
SCodeID1	111213 hex	security code for ID1 (fixed)
ID_Code1	1415 hex	ID1 (fixed)
SCodeID2	212223 hex	security code for ID2 (fixed)
ID_Code2	2425 hex	ID2 (fixed)
SCodeID3*	313233 hex	security code for ID3 (fixed)
ID_Code3*	3435 hex	ID3 (fixed)
SCodeID4	414243 hex	security code for ID4 (var.)
ID_Code4	4445 hex	ID4 (var.)
SCodeID5	515253 hex	security code for ID5 (var.)
ID_Code5	5455 hex	ID5 (var.)
SCodeID6*	616263 hex	security code for ID6 (var.)
ID_Code6*	6465 hex	ID6 (var.)
SCodeID7*	717273 hex	security code for ID7 (var.)
ID_Code7*	7475 hex	ID7 (var.)
SCodeID8*	818283 hex	security code for ID8 (var.)
ID_Code8*	8485 hex	ID8 (var.)

Note

* Codes are only available for the CRYPTO-BOX 560/Net, "fixed" means that codes are not programmable, "var" means that codes are programmable using the AutoCrypt Wizard or CBProg utility.

Appendix B: Converting to Hexadecimal Numbers

Number in dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number in hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

The easiest way to do conversions from hex to dec and vice versa is using the calculator delivered with Windows. To convert a number manually multiply each digit by 16 to the power of its position. Do this from the farthest right digit (position 0) to the farthest left digit (in the following example position 3). For example 058E in hexadecimal will be 1422 in decimal.

Example:

Conversion of a hex value to a dec value

	0	5	8	E	hex
= 0	x 16 ³	+ 5 x 16 ²	+ 8 x 16 ¹	+ E x 16 ⁰	
= 0	x 4096	+ 5 x 256	+ 8 x 16	+ 14 x 1	
= 1422 dec					

Example:

Conversion of a dec value to a hex value

1422 : 16 ³ = 0	=> 1st position	= 0 hex; carry 1422 dec
1422 : 16 ² = 5	=> 2nd position	= 5 hex; carry 142 dec
142 : 16 ¹ = 8	=> 3rd position	= 8 hex; carry 14 dec
14 : 16 ⁰ = 14	=> 4th position	= E hex; carry 0 dec

result 1422 dec = 058E hex

Appendix C: Technical Data

CRYPTO-BOX USB

Connectors	Type A Connector
Programming ROM	configured by MARX
Data retention	min. 10 years
Programming EEPROM	typical more than 1 Million cycles; 100 000 guaranteed
Board Layout	SMD
Microprocessor	8-Bit CMOS-micro-controller
Case	Zinc - Ni plated
Storage temperature	-40 ° F to +185 ° F
Working temperature	32 ° F to +158 ° F
Humidity	0 % bis 95 % rel. hum.
Dimensions	4/5" x 1/3" x 1", body: 2/3" (12 X 8 X 29 mm, body 17 mm)
Weight	Versa and XS 0.326 oz (9.12 g) XL 0.515 oz (14.6 g)
Memory	4-64kB
Admin password	2 ¹²⁸ possibilities
Access control	2 ¹²⁸ Security codes possible
User password	2 ¹²⁸ possibilities
Algorithms supported	AES/Rijndael, RSA, IDEA, MD4/5, Blowfish and more

Features of CRYPTO-BOX USB models

CRYPTO-BOX	USB Versa	USB XS	USB XL
	U/VS	U/XS	U/XL
AES/Rijndael algorithm, fully implemented in hardware	+	+	+
RSA (software implemented on driver level, RSA keys stored inside the CRYPTO-BOX)	-	+	+
Secure memory	4KByte	4-64KByte	4-64KByte
Unique serial number per device	-	+	+
True White Noise Generator for random numbers	-	-	+
Secure Key Container for passwords, certificates, digital signatures, PKI, etc.	-	+	+
Low Total Cost of Ownership (TCO)	+	+	+
Versatile, small size, LED indicator	+	+	+
Programmable (ID codes and memory)	+	+	+
Ideal for OEM usage	-	+	+
Perfectly shielded against radiation	+	+	+
Windows, Linux and Mac support	+	+	+
Works with Terminal Server (Windows and Citrix)	+(1)	+	+
License management in networks	(1)	LCS®	LCS®
Remote programming with RFP	+	+	+
Customer specific cases on request (e.g. with your embossed company logo)	+	+	+

(1) Network Floating License support

CRYPTO-BOX 560/Net and Versa

Connectors	DB 25 on both sides
Daisy chaining	3 modules and more
Programming ROM	ready to use (configured by MARX)
Data retention	10 years
Programming EEPROM	typical more than 1 million cycles; 100 000 guaranteed
Board layout	SMD
Microprocessor	8-Bit CMOS micro-controller with »Sleep-Modus« (low power consumption) and „turbo modus“
Case	Macrolon from BAYER, non inflammable; recycling class PE4
Storage temperature	-13° F to +176° F (-25° C to +80° C)
Working temperature	41° F to 167° F (+5° C to +75° C)
Humidity	10 % to 80 % rel. hum.
Dimensions	1-15/16" x 2-1/16" x 11/16" (49 x 53 x 17 mm)
Weight	1.286 oz (36 g)
Memory	
Net560	512 bytes user-memory, 483 bytes programmable
Versa	64 bytes user-memory, 50 bytes programmable
Master password	2 ⁴⁰ possibilities
Access control	2 ²⁴ security codes possible
ID-Code	2 ¹⁶ possibilities
Number of ID-Codes	
Net560	8
Versa	4
Developer-ID	2 ²⁴ possibilities
Algorithms supported	AES/Rijndael, RSA, IDEA, MD4/5, Blowfish, and more (opt.)

CRYPTO-BOX Serial CBS3/9-Pin

Connectors	DB9 on both sides
Programming ROM	configured by MARX
Data retention	min. 10 years
Programming EEPROM	typical more than 1 million cycles; 100 000 guaranteed
Board layout	SMD
Microprocessor	8-Bit CMOS micro-controller
Case	Plastic, ABS
Storage temperature	-40° F to 185° F (-40 °C to +85 °C)
Working temperature	-4° F to 158° F (-20 °C to +70 °C)
Humidity	0 % to 95 % rel. hum.
Dimensions	9/16" x 1 3/16" x 1 5/16" (16 x 30 x 50 mm)
Memory	4-64KByte
Weight	0.775 oz (21.7 g)
Admin password	2 ¹²⁸ possibilities
Access control	2 ¹²⁸ Security codes possible
User password	2 ¹²⁸ possibilities
Algorithms supported	AES/Rijndael, RSA, IDEA, MD4/5, Blowfish and more

CRYPTO-BOX Technical Data and Overview

CRYPTO-BOX	CBU (USB)		560/Net (parallel)	Versa (parallel)	Serial (9-pin)
	U/XS	U/VS	CBN	CBV	CBS
Windows XP/2000/NT4/Me/9x/3.x, DOS supported	+ + (WinXP/2K/Me/98 + NT4)		+	+	+
Automatic Windows/DOS Integration	+ + (Windows)		+	+	+ (Windows)
Professional Protection Kit (PPK)	+	+	+	+	+
Network support	1 CBU per network	(1)	1 CBN per network	1 CBV per customer	1 CBS per network (3)
Linux/UNIX/Solaris-support	+	+	-	-	+
Apple (MacOS 8/9/X)	+	-	-	-	-
AES-Rijndael crypto algorithm	+	+	+ (2)	+ (2)	+
Pre-programmed with customer codes	+	+	+	+	+
Variable crypto algorithm	+	+	+	+	+
Hash function	+	+	+	+	+
Unique serial number	+	-	-	-	+
LCS [®] – License Control System	+	-	+	-	+ (3)
RFP – Remote Field Programming	+	+	+	+	+
MAAS – Multiple Application System	+	+	+	+	+
RSA (software implementation)	+2)	-	+ (2)	+ (2)	+ (2)
Memory	4 – 64 kBytes	4 kBytes	560 Bytes	64 Bytes	4 – 64 kBytes
Programmable, also via Internet	+	+	+	+	+

(1) Floating License; (2) implemented in software; (3) in preparation.

Appendix D: Supported Compilers and Applications

Supported Compilers (MPI based samples)

These compilers are fully supported via MPI library, samples are available on the PPK-MPI CDROM.

Alaska Xbase++	Microsoft C# (.NET Environment)
Apple Project Builder C/C++	MS Visual Basic WIN
Borland Delphi	Visual Basic .NET
C++Builder (Borland)	Visual C/C++
Clipper	Visual Fortran
Java	Visual Foxpro
Kylix	Visual Java
Metrowerks Code Warrior (Win + Mac)	Watcom C/C++ 11.0

Supported Compilers (legacy samples)

These compilers are supported via our legacy API for the CRYPTO-BOX 560/Net and Versa (parallel). Samples are available on request, please contact our Technical Support.

Borland C/C++	Microsoft C/C++
Borland Pascal	Microsoft Cobol
dBASE IV/5/5.5/7.5/dB2K	Microsoft Assembler (MASM)
Foxpro 2.5/2.6/3.0	MS Basic 7.1 PDS
GNU C++	MS Fortran Power Station
IBM CSet 2++ (OS/2)	MS Pascal
Lahey Fortran	MS Quick Basic
Logitech MODULA	MS Visual Basic DOS
Macro-Assembler	NDP C
Metaware High C/C++	
MicroFocus Cobol	

Supported Applications

Asymetrix Toolbook	Microsoft Excel
AutoCAD	Microsoft Outlook
Citrix Metaframe /	Microsoft Word
Windows 2000 Terminal Server	PARADOX
CLARION	PDF and HTML format
LabVIEW	SQL Windows
Lotus Notes	Virtual PC 4 for MacOS
Macromedia Director	WINDEV (French)
Microsoft.NET	
Microsoft Access	

Dynamic Link Library (DLL) for Windows XP/2000/Me/9x/3.1x and OS/2
Novell Loadable Module (NLM) for Novell NetWare 3.x/4.x/5.x
DOS-Extender (Pharlap, DOS/4GW, Exospace etc.)

Please contact us if you need support for other applications! In most cases we will find a solution for you.

Supported Standards and Interfaces

MS CAPI (Microsoft Crypto-API)
MPI (MARX proprietary programming interface, including support for legacy ports)
TEOS (Token Embedded Operating System by MARX)

Appendix E: Distributors**USA**

MARX Software Security	Sales:	sales@marx.com
2900 Chamblee Tucker Rd. N.E.	Support:	support@marx.com
Building 9	Phone:	(+1) 770-986-8887
Atlanta, GA 30341	Fax:	(+1) 770-986-8891
USA	E-Mail:	info@marx.com
www.marx.com		

Germany

MARX Software Security GmbH	Sales:	sales-de@marx.com
Vohburger Str. 68	Support:	support-de@marx.com
D-85104 Wackerstein	Phone:	(+49) 8403 9295-0
Germany	Fax:	(+49) 8403 1500
www.marx.com		
	E-Mail:	contact-de@marx.com

Italy

CS Computers S.r.l.	Sales:	Giorgio del Bene
Via Indipendenza, 4-12	Phone:	(+39) 541/963.801
I-47033 Cattolica (FO)	Fax:	(+39) 541/953.847
Italia	E-Mail:	cscomp@cscomputers.it
www.csccomputers.it		

Poland

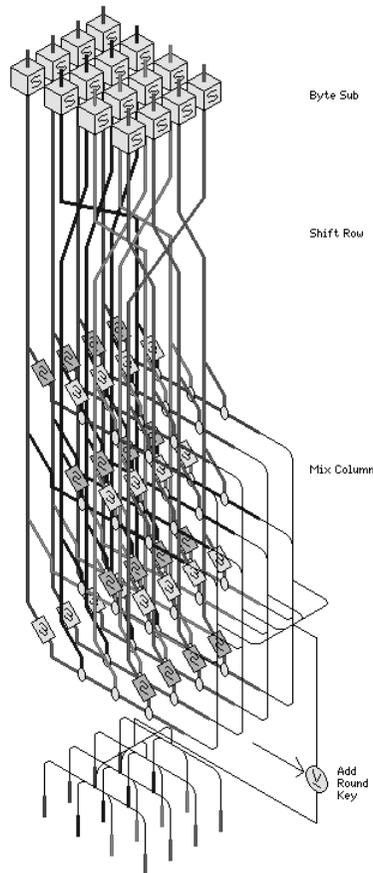
Microplan Polska Sp. z o.o.	Sales:	Gregor Bigos
Polwiejska 3	Phone:	(+48) 61 8518916
PL-61-885 Poznan	Fax:	(+48) 61 8518774
Polen	E-Mail:	big@microplan.pl
www.microplan.pl		

Appendix F: The AES/Rijndael algorithm

The communication between the CRYPTO-BOX USB, the driver, and the RAM data is internally encrypted.

The key supports Rijndael algorithm hardware encryption working in Output Feedback Bit stream mode (OFB). For the data encryption, the user can select the Fixed Key and Initialization Vector (Provided by MARX - KF, IVF) or he can define his own Session Key and Initialization Vector (KS, IVS).

Figure: F.1
RIJNDAEL ALGORITHM
 Winner of "Global
 Information
 Security Competition",
 implemented in the
 CRYPTO-BOX® USB and
 CrypToken®



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.

The effort to establish the **AES (Advanced Encryption Standard)** reflects the dramatic transformation that cryptography has undergone in recent years. Hundreds of encryption products currently employ **DES** or **Triple DES**, and such systems have become almost ubiquitous in the financial services industry. Consequently, the selection of the **AES** may affect millions of consumers and businesses.

Most valuable is Rijndael's combination of security, performance, efficiency, ease of implementation and flexibility make it an appropriate selection for the AES.

Specifically, Rijndael is consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key set-up time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against **power** and **timing attacks**.

The **block cipher Rijndael** is designed to use only simple whole-byte operations. Also, it provides extra flexibility over that required of an AES candidate, in that both the key size and the block size may be chosen to be any of 128, 192, or 256 bits.

Note

Detailed information about Rijndael can be obtained from:
<http://csrc.nist.gov/encryption/aes/>

Appendix G: Glossary

Access codes

A set of codes (passwords) required to access a CRYPTO-BOX.

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. Used in our products CRYPTO-BOX USB and CrypToken.

API

Application Programming Interface is a set of routines a program can call in a function library or in the operating system.

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 Wizard

automatic protection for programs, provides compression feature, similar to ZIP. Makes applications tamper-proof and reduces size. No source code required.

Binary shell

See "Wrapper".

Citrix

This system provides similar to Windows Terminal Server a server-based access to virtually any application, across any type of network connection to any type of client (platform independent). Citrix Metaframe is supported by the CRYPTO-BOX system.

Code injector

See "Wrapper"

Compression

AutoCrypt Wizard, our solution for automatic protection of exe-files provides compression (similar to Zip). Makes applications tamper-proof and reduces size. No source code required.

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®

A device attached to a user's computer through USB, parallel or serial ports. This hardware key is mostly used for software protection purposes. Every CRYPTO-BOX has a microprocessor, which provides secure access to the contents of its memory. CRYPTO-BOXes are used to hold any kind of sensitive information, like personal identification data, expiration dates, licenses, etc.

CrypToken®

Hardware key from MARX with USB-connector, AES/Rijndael algorithm, on-board True Physical Noise Generator (optional) and RSA support for encryption, authentication, PKI solutions, etc.

DES

Former Data Encryption Standard. A cryptographic algorithm; U.S. Government specification for encryption. Expired and frequently broken.

Device Driver

Software component that extends the operating system of a computer so that CRYPTO-BOX keys or other devices 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 also provides confirmation that the contents of the message to which it is attached, have not been tampered with.

DoD

Department of Defense, issuer of standards for data management in the domain of military (DoD 5015, CALS, etc.).

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 hackers.

Emulation

One system emulates another when it performs in exactly the same way. Emulation typically allows the developer to single step a piece of software and examine the execution flow.

Encryption

A way to represent data in a ciphered form to prevent unauthorized accesses to it by an intruder.

ESD

Electronic Software Distribution is supported by the CRYPTO-BOX system.

Expiration Date

Date after that 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 unsecure way without having fear of spying/hacking.

Hexadecimal values

A value based on the number 16. Example: the number 32 is represented by hex 20

IDEA algorithm

An encryption algorithm, considered to be among the best and most reliable ones. It operates on 64-bit blocks with a 128-bit key.

Kernel

The essential part of an operating system, responsible for resource allocation, low-level hardware interfaces, security, etc.

License management

A system used to define and control the total number of concurrent users in a LAN.

Licensing

Granting permission to use intellectual property on a software product marketed by the licensee in exchange for payment.

LPT port

See "Parallel port"

Microprocessor

An electronic chip which controls all interaction with the CRYPTO-BOX and processes data flow.

MPI

MARX Programming Interface. An easy-to-use API that provides one common interface for CRYPTO-BOXes.

Multitasking

The concurrent operation by one central processing unit of two or more processes.

.NET technology

Microsoft .NET is a set of Microsoft software technologies which enable users to interact with a broad range of smart devices via the Web, while ensuring that the user, rather than the application, controls the interaction.

Noise Generator

Generates random numbers from a thermal noise, used for encryption, digital signing and security protocols. A True White Noise Generator is used in our product CRYPTO-BOX USB XL.

Novell

Vendor of software for local area networking (Netware) and other network computing products. The CRYPTO-BOX Server (CBNetServer) supports Netware.

Parallel port

A connector on a computer where data is transmitted in or out in parallel (more than one wire). The most widespread type of parallel port is a printer port like Centronics, which transfers eight bits at a time.

Pay-per-Use

In the TV broadcasting arena it's already commonplace: If a customer wants to see a particular program, he must pay to do so. The same can apply to software distribution. Rather than paying for the application, he pays for the number of program starts, for the amount of time he uses the software, or for the functionality that he works with. In short: Intensive users pay more.

Piracy

Unauthorized duplication, use and distribution of computer software and/or related material.

PKCS#11

Cryptographic Token Interface Standard.

This standard specifies an API, called Cryptoki, to devices which hold cryptographic information and perform cryptographic functions.

PKI

Public Key Infrastructure defines the rules and organizational background that allows to deploy security services based on encryption.

Protective shell

See "Wrapper".

Rijndael

See "AES/Rijndael"

RSA

Most widely deployed public-key algorithm

S/MIME

Secure Multi-Purpose Internet Mail Extensions. SMIME is a secure version of the MIME protocol. By secure is meant that SMIME is used to encrypt and decrypt e-mail. The newest versions of MS Internet Explorer and Netscape include SMIME software.

Seed

A parameter that changes the result of an encryption or decryption process.

Serial port

A connector on a computer to which you can connect peripherals that communicate with the help of a bit-stream protocol. External devices are connected either through 9-pin or 25-pin connectors.

Smart card

An electronic device that is the same size as a regular plastic wallet card with a microchip which ensures secure access to the card and performs data processing. Smart cards can store a person's credentials, account information, network licenses, etc.

Disadvantage: requires card reader.

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.

TEOS

Token Embedded Operating System. TEOS is a new "operating system" specially tailored to the features of the CRYPTO-BOX USB and CrypToken. TEOS allows you to run several applications concurrently and independently on a single security token. This is achieved through intelligent file management and a sophisticated programming interface.

Token

Part of a two-factor authentication system to prove a user is who he is supposed to be. A token is a hardware (e.g. CrypToken from MARX) or software device that is used in conjunction with a password login.

Trialware

A software product used for a certain duration. As a rule, trialware has an expiration date after which the user has either to purchase the full version of the product or quit using it.

USB port

Universal Serial Bus, provides much higher data transfer rates than a serial or parallel port and allows to connect up to 127 devices. The USB port is supported by the models CRYPTO-BOX USB "Versa", "XS" and "XL".

White Noise Generator

See "Noise Generator"

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 Wizard. Our Wrapper also provides compression (similar to Zip).

X.509

Most widely deployed standard for digital certificates, agreed by CCIT and ISO.

Appendix H: Trademarks

MARX®, CRYPTO-BOX® USB, CRYPTO-BOX® 560, CRYPTO-BOX® Versa, CrypToken®, LCS®, CRYPTO-WIZARD®, CRYPT:ACCESS®, FILE:CRYPT®, CD-ROM VENDOR SECURITY®, TOKEY™ and AudioVideo RTE™ are trademarks or registered trademarks of MARX.

Microsoft®, Windows Server™ 2003 and Windows NT® are registered trademarks of Microsoft Corporation in the United States and other countries.

IBM® is a registered trademark of International Business Machines Corporation in the United States, other countries, or both.

Java™ or all Java-based trademarks and logos, and Solaris™ are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Mac OS® is a trademark of Apple Computer, Inc., registered in the United States and other countries.

Novell®, NetWare® and IPX/SPX® are registered trademarks of Novell, Inc. 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.

AutoCAD® and AutoLISP® are registered trademarks of Autodesk, Inc., in the United States and/or other countries.

Asymetrix® and ToolBook® are registered trademarks of Asymetrix Learning Systems, Inc. in the United States and other countries.

Macromedia® Director® is a trademark of Macromedia, Inc. in the United States and/or other countries.

Other company, product, or service names may be trademarks or service marks of their respective holders.

Appendix I: Developer's Agreement

All products that MARX® Software Security delivers to you, including documentation, software, hardware, diskettes, evaluation kits, etc. (referred to as "MARX products" below) are subject to the terms stated below. All future orders will be based on these terms as well. If you disagree with these terms, please return the MARX products to us, postage prepaid, within seven days of receipt, and we will provide you with a refund.

- 1.** MARX gives a warranty of 3 years from the date of delivery according to MARX general terms of business. This warranty is limited to significant defects in material and workmanship of the MARX products detected under normal use. 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.
- 2.** If you receive defective MARX products, MARX's sole obligation is to repair or replace, at MARX's choice, any MARX product free of charge. Any replaced parts shall become MARX's property.
- 3.** MARX is not responsible for any delays in delivery. MARX's entire liability for any damages to you or another party for any cause shall not exceed the price of the MARX product that caused the damage. MARX will in no event be liable for any damages caused by your failure to perform your obligations, or for any loss of data, profits, savings or any other consequential and incidental damages, even if MARX has been advised by you that such damages may be possible, or for any claims by you based on any third-party claim.
- 4.** You may not try to copy, reproduce, or reverse-engineer any part of the MARX products, except as allowed in item 5 below.
- 5.** You may make archive copies of the software. You may modify the demo programs supplied with the evaluation kit and link the libraries supplied to your software. If necessary, you are allowed to ship parts of the software (e.g., WINDOWS-DLL) to third party users with your software for the sole purpose of protecting your software.
- 6.** EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY, CONDITION OR REPRESENTATION REGARDING MARX'S PRODUCTS, SERVICES OR PERFORMANCE, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Notice to users

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. MARX products 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 products 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. This device has been found to be harmless for residential electrical installations. However, we cannot guarantee that interference with radio communications will not occur in a particular installation.

12. Index

.NET 181, 182, 188

A

Access Codes 108, 147, 186
 Activation code 72, 73
 AES/Rijndael 28, 50, 91, 103, 176
 Alaska Xbase++ 181
 Anti-debugging 96, 97
 Anti-disassembling 96
 API 186, 187, 188, 189
 Apple MacOS 87
 Apple Project Builder C/C++ 181
 Arguments, function calls 186
 Asymetrix Toolbook 182
 Authentication 59, 186
 AutoCAD 182
 AutoCrypt Wizard 41, 70, 173
 Automatic Softw. Protection 41

B

Barcode 36
 Beta testing programs 21
 Binary shell 186
 Blowfish 91
 Borland C/C++ 181
 Borland Delphi 181
 Borland Pascal 181

C

C# 181
 C++ 181
 C++Builder (Borland) 181
 CBNET server 47, 107, 160
 CBNetServer 68, 69, 107
 CBProg 45, 56, 61, 63, 69, 70
 CBS3/9-Pin 179
 CBSSetup 79, 80, 82, 83

Checksums 92
 CITRIX 60, 87, 182, 186

Citrix Metaframe 182
 CLARION 182
 Clipper 181
 Code Warrior 181
 Compression 48, 95, 186
 Counter 186
 CRYPTO-BOX 87
 CRYPTO-BOX 560/Net 32, 33, 81, 82
 CRYPTO-BOX Card 23, 84
 CRYPTO-BOX ID codes 61
 CRYPTO-BOX Serial 33, 34, 77, 83
 CRYPTO-BOX USB 28, 78
 CRYPTO-BOX Versa 33, 81
 CRYPTO-BOX Serial 179
 CrypToken® 184, 186, 191
 Customer-specific solutions 36

D

Data matrix code 36
 Data objects 37, 94, 148
 dBASE IV/5/5.5/7.5/dB2K 181
 Debugger 50, 51
 Decryption 113, 118, 134, 163
 DES 185, 187
 Developer ID 112, 138, 146
 Device Driver 77, 187
 Digital signature 32, 187
 Disassembly 103
 Distribution 77, 83
 DLL 78, 79, 81, 83, 84
 DoD 187
 Downloads, support 35
 DOS 37, 90, 181
 DRM, Digital Rights Mgmt. 85

DrvVersion 139

Dummy calls 93, 187

E

Embedded systems 87, 90

Emulation 187

Encryption 37, 112, 162

Encryption key 112, 162

Error codes 113

ESD, Electronic Sw Distr. 3, 5, 187

Evaluation period 4, 9, 12

Execution counter 37, 49, 50, 58

Expiration date 37, 49, 107, 148

F

Fixed Key 119

Foxpro 2.5/2.6/3.0 181

Function call 187

G

GNU C++ 181

H

Hardware key 186, 187

Hash function 32, 140

Hash value 112, 140, 141

HashBuffer 140, 141

HashBufLength 140

Hash-function 187

Hexadecimal values 187

HTML-Format 182

I

IBM CSet 2++ (OS/2) 181

ID codes 150, 165, 167

IDA Pro Debugger 96, 103

IDEA algorithm 91, 119, 128, 188

Identifier 109

IRM, Information Rights Mgmt. 85

Internet download 7

Internet Explorer 189

J

Java 181

K

Kernel 188

Kylix 181

L

Labview 182

Lahey Fortran 181

Legacy ports 182

License Activation Wizard 6, 8

License Control System 24, 61,

License counter 61, 63, 68, 69, 70

License management 188

Linux 87, 89, 90

Logitech Modula 181

Lotus Notes 182

LPT port 188

M

MAAS 22, 45, 74

MacOS 87, 90

Macro-Assembler 181

MacroMedia Director 85, 182

Managed components 89

Marketing strategy 7

MARX algorithm 91, 119

MARX Data Objects 105

MARX Distributors 183

MARX Programming Interface (MPI) 21, 59

MarxProbe 64

MD4_ALGORITHM 140

Metaware High C/C++ 181

Metrowerks Code Warrior (Win + Mac) 181

MicroFocus Cobol 181

- Microprocessor 176, 188
 - Microsoft .NET 89
 - Microsoft Access 182
 - Microsoft Assembler (MASM) 181
 - Microsoft C# (.NET-Umgebung) 181
 - Microsoft C/C++ 181
 - Microsoft Cobol 181
 - Microsoft Outlook 182
 - Microsoft Word 182
 - MPI 188
 - MPI Function calls 106, 110, 117
 - MPI_Channel 157
 - MPI_Close 117
 - MPI_Decrypt 122
 - MPI_DecryptEx 118, 122
 - MPI_Encrypt 128
 - MPI_EncryptEx 124, 128,
 - MPI_EraseEncryptionKey 130
 - MPI_ErasePassword 132
 - MPI_GenerateKeyPairRSA 134
 - MPI_GetBoxInfo 136
 - MPI_GetBoxType 137
 - MPI_GetDeveloperId 138
 - MPI_GetDrvVersion 139
 - MPI_GetHash 140
 - MPI_GetLastError 142
 - MPI_GetLicenseInfo 143
 - MPI_GetNetworkParameter 144
 - MPI_GetRandomSequence 145
 - MPI_GetSerialNr 146
 - MPI_Open 109, 117, 139, 142, 147
 - MPI_ProcessDataObject 148
 - MPI_ReadID 150
 - MPI_Readmem 151
 - MPI_SearchFirst 105, 117,
 - MPI_SearchNext 117, 139
 - MPI_SampleFunction 116
 - MPI_SetChannel 157
 - MPI_SetNetworkParameter 107,
 - MPI_SubmitEncryptionKey 118
 - MPI_SubmitPassword 105, 117
 - MPI_SubmitPasswordEx 167
 - MPI_WhatIsAvailable 169
 - MPI_WriteMem 171
 - MS Basic 7.1 PDS 181
 - MS CAPI 182
 - MS Fortran Power Station 181
 - MS Pascal 181
 - MS Quick Basic 181
 - MS Visual Basic DOS 181
 - MS Visual Basic WIN 181
- N**
- NDP C 181
 - NetBIOS 60
 - NetWare 182, 188
 - Network licenses 47, 61, 63, 70
 - Noise Generator 186, 188, 190
 - Novell 87, 182, 188, 191
 - Novell NetWare 38, 60
- O**
- OEM solutions 25
 - Output Feedback Mode 29
- P**
- PARADOX 182
 - Parallel port 188, 189
 - Password Manager 30
 - Pay-per-Use 15, 16
 - PDF 85, 182
 - Periodic check 50, 51
 - Piracy 188
 - PKCS#11 189
 - PKI 186, 189
 - Power Basic 181
 - PowerBuilder 5.01 181
 - Private Key 119

Professional Protection Kit (PPK) 38, 39
Public Key 119

Q

QNX 60, 87, 89
Quick C 181
Quickstart 41

R

Random generator 29, 92, 145
REALIA Cobol 181
Remote Programming 14, 24, 68
Return Codes 113
Rijndael algorithm 119, 120, 125
RSA 91

S

S/MIME 189
Secure Compress 52
Serial Evaluation Kit 173
serial number 29, 30, 112, 138, 146
Serial port 189
Session Key 119
Shell, protective 189
Site Licensing 13
Smart card 187, 189
SoftICE 96, 97, 98
Software Leasing 15
Software Protection 41, 105
Solaris 60, 87, 89
SourceBuffer 119
SourceBufLength 140
SPBase 181
SQL Windows 182
static libraries 78, 83
SUN Solaris 87, 89
Support 35
Symantec C/C++ 181

T

TCP/IP 47, 60
Technical Data 176, 178, 179, 180
Technical support 35
TEOS 37, 38, 60, 189
Terminal Server 60
Token 189
TOKEY 30
Tracing 103
Trademarks 191, 193
Transaction key 72, 73
Trialware 189
Triple DES 185
Troubleshooting 64
True Physical Noise Generator 28
TRW2000 96
Turbo Basic 181
Turbo C/C++ 181
Turbo Debugger 96
Turbo Pascal 181
Two-factor authentication 3

U

UNIX 87, 89, 90
Usage Counter 107, 114
USB port 189

V

Virtual PC for MacOS 182
Visual C/C++ 181
Visual Fortran 181
Visual Foxpro 181
Visual Java 181
Visual Objects (CA) 18

W

Watcom C/C++ 9.5/10.5/11.0 181
Watcom Fortran 181
White Noise Rand.Gen. 28

WIN Kernel Debugger 97
WINDEV 182
Windows, supported versions 37
Windows 2000/2003 Term. Server 87
Wrapper 18

X

X.509 22, 190

**0-01MAR04(MPI_Manual.qxp
(0-01MAR04(MPI_Manual.pdf)**



Software Security

Establish secure distribution channels and assure revenue for every license! Includes automatic protection of Windows applications and manual implementation of your custom protection strategies for Windows, MacOS, Linux, UNIX, Solaris and more.



Data Protection

Secure distribution of documents (Office, PDF, Macromedia Director files). Protect your valuable digital content with the CRYPTO-BOX and limit access to authorized users only. Ideal for distribution via Internet and CDROM.



Web Security

Secure Authentication, Online Identification and Access Control for all users. Restrict and allow login into your web site, your subscription service or any kind of online business!



AudioVideo RTE™

Protection of digital media (video and audio) from piracy and unauthorized duplication with the CRYPTO-BOX USB.

CRYPTO-BOX® USB and CrypToken® USB

- AES/Rijndael algorithm, implemented in hardware
- 4-64 kByte secure memory - ideal for certificates, passwords, ...
- Low profile designer metal case
- Shielded against radiation, water resistant
- Support for MS-CAPI (Crypto-API)
- For Windows, MacOS, Linux, UNIX, Solaris, ...
- TEOS (Token Embedded Operating System)
- OEM versions available - more on request



[actual size]

Securing the Digital WorldSM

www.marx.com

MARX Software Security
2900 Chamblee Tucker Rd. N.E., Building 9
Atlanta, GA 30341 USA
Tel.: (+1) 770-986-8887
Fax: (+1) 770-986-8891
info@marx.com

