White Paper

CRYPTO-BOX® for Insiders WP-6

Subject: Professional Software Protection with the CRYPTO-BOX

Version: Smarx OS PPK 8.4 and higher

Last Update: 25 July 2019

Target Operating Systems: Windows, Linux, macOS

Applicable for Product: CRYPTO-BOX® SC / XS / Versa

CRYPTO-BOX® for Insiders - How to deal with hackers, cracking or memory dumps

This white paper provides suggestions, ideas and techniques for a secure integration of the CRYPTO-BOX into your software. Please consider these hints to when you develop applications that use the CRYPTO-BOX for software protection and software licensing via the Smarx OS API.

Even if you are working with the API integration already, or if you are planning to revise your API implementations soon, this document might give you useful tips and references.

CRYPTO-BOX® Key Features

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

Table of Contents

How to deal with hackers, cracking or memory dumps

1. The Case

Customers may ask: "Is there a way to remove the possibility to be emulated? We suspect that some of our end users have done it already."

Websites offering such illegal 'services' exist since many years, but this is no proof that the CRYPTO-BOX was cracked in general! And it is even no proof that your particular application was or can be cracked by them.

Most solutions for cracking resp. removing protection are based on the following approaches:

2. Attempts to defeat software protection might comprise of these attacks

2.1. Recording the communication with the dongle and trying to emulate it (replay attack)

This can be done by replacing either the functions library of the CRYPTO-BOX or the CRYPTO-BOX driver with a faked version which returns certain values at certain inquiries.

Some examples how to avoid that:

- Do not only expect fixed responses, instead use the encryption capabilities to generate dynamic responses from the CRYPTO-BOX. The encryption engine is the most powerful unit of the CRYPTO-BOX, take advantage of it. Because the encryption key never leaves the hardware, and without having the correct encryption key, the information cannot be decrypted.
- Using our static libraries for implementation instead of DLLs or verifying their integrity

2.2. In case of AutoCrypt: Dumping the application

If you use AutoCrypt for protecting your application, there is always a certain point where - after all checks of our protective shell (wrapper) were done - the protected application is decrypted into the RAM. At this point, a cracker can dump the RAM content and create a working copy of the application. Although we use anti-debugging and other countermeasures to defeat running debuggers and dumpers, there are possibly new avenues from the "other side" to hide such a debugger so that it is not detected.

We are continuously working on improving AutoCrypt, therefore you should regularly upgrade to the latest AutoCrypt version.

2.3. Analyzing the source code and use debugging/disassembling techniques to remove the protection

This approach requires the most efforts and therefore is usually quite expensive. This cannot be avoided completely - no protection is 100% secure. But you can make the work of the cracker very difficult, so that it will require much efforts to crack your application (and therefore it will be time-consuming and too expensive). And please note: "No automatic removal" is possible in this case. Very few hackers will deal with it - if at all.

Here are some further hints which should be considered during implementation with our API:

a) UPW Encryption

We suggest to encrypt the UPW of the CRYPTO-BOX with the Rijndael Fixed Key. To use the Rijndael Fixed Key encryption, no UPW submission is required.

Even such a simple sequence of CBIOS calls will provide a reasonable level of protection:

```
...
ScanBoxes()
...
OpenByIndex()
...
CryptFixed()  // using fixed AES encryption for some program critical data (in case of large amount of data hardware based encryption should be applied to encrypt some value to be further used as a key for software based encryption)
...
```

b) Using Notifications

Furthermore, CBIOS notifications can be used for immediate detection of CRYPTO-BOX unplugging. In particular plugging the key should start license validation and proceed to full operating mode if it is valid. Unplugging the key normally should fire popup dialog asking to plug it back or the program will exit/switch to limited mode.

Check our sample code for more details.

c) Further Ideas

UPW security is a good start, but you should also think about some more sophisticated procedure to protect your application: not only password security, but also encryption, and verifying program integrity.

Consider AES/RSA based encryption of dynamic contents for validation purposes rather than reading any static information from the CRYPTO-BOX (such as GetBoxInfo/GetBoxInfoAdv output of which could be emulated). It is practically impossible to emulate encryption results in case of dynamic contents. Only a CRYPTO-BOX with valid values of corresponding encryption keys can do it.

The CRYPTO-BOX SC model has even more capabilities: It offers hardware based RSA and extended AES support (multiple key slots) allowing implementation of high level protection (professional security). See Compendium, chapter 10.10 for further details on CRYPTO-BOX SC specific functions, and CBU SC API samplecode in the PPK.

Some examples what can be done:

- Add some program hash/signature which can be calculated by the application and compared to the value stored in the box and encrypted with AES (eg. Rijndael Session key).
- Spread protection routines in the code. So the cracker never knows, if all protection was removed. Integrate license validation to those program features and/or functions which are specific to full operating mode or used only sometimes (like: printing reports, performing application specific transactions and more).
- Use long testing routines and implement dummy calls.
- Store parameters and program variables in the internal CRYPTO-BOX memory.
- Add timer based validation.

3. Seeing the challenge from the hacker's point of view, and what makes hacker's life difficult

1) The hacker knows his tools, he's good with disassembling code. He will see where fixed responses are obtained (and he'll emulate or bypass them) but he does not know memory contents of the CRYPTO-BOX, nor calculated results obtained using the intrinsic algorithm of every CRYPTO-BOX. He will also have great difficulty to understand why the (allegedly un-)protected software still does not work as with the CRYPTO-BOX, if the implementation was done properly.

2) Such illegal services are nearly always located in former Soviet states, or in far east - not accessible with legal means. Again, they are proficient with their tools, but they DO NOT know the protected individual application software! Here is the lever MARX customers can and must use to get the utmost from a strong smartcard-based device like the CRYPTO-BOX.

3) Any automatic protection has limits. We know that our **AutoCrypt** enjoys great popularity, in particular because it's fast and easy to apply. We all know from experience that "software development rarely is completed as scheduled", with the consequence that software protection more often than not is 'added to the menu' the day (or night) before release date, or the start of the exhibition or presentation. This, of course, does not lead to a professional protection level, and the capabilities of the CRYPTO-BOX are not exploited to the fullest. Far from it.

4) We recommend a 'multi-tier-protection' using the API. First comes an 'entry check' at program start - mainly as an information for honest users. "Attach the CRYPTO-BOX" may be a suitable message. Let's assume this 'query' was successfully removed by an attacker - and the protected application is still running' (ostensibly now 'cracked') - albeit in a different way, unbeknowst to the user and hacker. Let's also assume our customer heeded our advice and added sophisticated software protection in subsequent program modules, often invoked later, much later: When building monthly totals, compiling print reports, monthly backups, even mouse movements or the year-end report - you get the picture. At these occasions, in an irregular manner and not predictable by anyone who does not know how this specific application software works and how the CRYPTO-BOX is used, e.g. delivering values to the program or calculating results, the protected program will behave differently devoid of the CRYPTO-BOX. We do recommend NOT to abort the program at once (no Yes/No decision), but rather let it run further, even with wrong results, until much later there may be a cryptic or hidden error message (meaningless for the user, but descriptive for our customer and for us) to prove what's going on. At this time, the 'crack service' usually has been paid (way too early) and the dishonest user will get tired after repeat and repeat failure of the (ostensibly now 'unprotected') program. This leads to the desired outcome: Paid licenses!

5) Be creative with memory contents, use the memory of the CRYPTO-BOX in various ways! Use certain stored values much later than the first protection steps or -levels. No one except you, the developer, is able to know memory content in advance.

6) Do not completely repeat the entire protection scenario when it comes to updates/upgrades! Variation is the key to success. Always create new challenges for hackers and dishonest users. Experience shows that everyone gets tired at some point, or simply can't or does not want to invest more energy, time and money into futile software cracking attempts. One aptly real-world example comes to mind, regarding 'media protection', here: Transmission of soccer games. The protection scheme used was such, that for the second half of the game a different encryption key was used - impossible to crack within 45 minutes.

We also offer implementation consulting, such as reviewing your code and providing advice.
Contact us for an offer:

(+1) 770-904-0369 (U.S.A., Canada, North & South America)

(+49) 8403-9295-0 (Germany, Europe, Worldwide)

support@marx.com

CRYPTO-BOX® Data Sheet

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

CRYPTO-BOX Certifications

Evaluation Kit
www.marx.com/eval

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

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

www.marx.com