

CRYPTO-BOX[®] SC

White Paper for Software Developers

Table of Contents

Table of Contents	1
1. Introduction	2
1.1. CRYPTO-BOX [®] SC: a new generation of MARX USB security hardware	2
1.2. SmarxOS and CRYPTO-BOX [®] SC	3
1.2.1. CRYPTO-BOX [®] SC AES encryption extension	3
1.2.2. Using hardware based RSA	3
2. Evaluation: step-by-step	3
2.1. Installing CRYPTO-BOX [®] SC driver and SmarxOS [®] COM object	4
2.2. Testing	4
2.2.1. CBU compatibility level	4
2.2.2. Testing CBU SC specific features	4
3. CRYPTO-BOX [®] SC and RSA encryption.....	5
3.1. CRYPTO-BOX [®] USB and RSA encryption	5
3.1.1. PKCS#1 padding rules	5
3.1.2. Padding rules used in CBU RSA implementation	6
3.2. RSA for CBU: Conclusion	6
3.3. CRYPTO-BOX [®] SC: hardware RSA implementation	6
4. SmarxOS API: new and extended calls and structures.....	7
4.1. Introducing CBIOS 1.5	7
4.2. CBIOS 1.5: RSA encryption support.....	7
4.3. CBIOS 1.5: CBU RSA calls.....	7
4.3.1. CBU SC: hardware support for SmarxOS system RSA keypairs (Distributor and Client)	7
4.3.2. CBU SC: ignoring login password parameter for CBU RSA calls.....	8
4.4. CBIOS 1.5: CBU SC RSA calls.....	8
4.4.1. CBIOS API: CBU SC RSA new API calls.....	9
4.4.2. New CBIOS structures related to RSA.....	14
4.5. CBIOS 1.5: AES encryption support.....	15
4.6. CBIOS 1.5: CBU SC AES calls	15
4.6.1. CBIOS API: CBU SC AES new API calls	16
4.6.2. New CBIOS structures related to AES	19
4.7. New CBIOS constants	20
4.8. CBU SC specific CBIOS error codes	20
5. Appendix A: CBU SC and CBU timing	21
6. Appendix B: Contact, Support and Distributors	23

1. Introduction

1.1. CRYPTO-BOX[®] SC: a new generation of MARX USB security hardware

The CRYPTO-BOX SC (CBU SC) introduces a brand new generation of MARX USB security hardware. It uses the modern Atmel AT90SC family chipset - low-power, high-performance secure micro controllers built around SecureAVR[®] 8-/16-bit RISC architecture. This smart choice makes the device architecture extremely flexible. Reloadable firmware allows implementing in hardware the necessary logic and functionality, as a result obtaining required type of security hardware from a standard smart card in USB form-factor to a smart USB token – CBU SC (protection and licensing device).

The CBU SC firmware provides 100% backward compatibility with the existing CRYPTO-BOX USB (CBU), adding such important features as:

- **Reprogrammable and remotely upgradable firmware:**
 - o more standard security features will be added (CBU SC firmware will be continuously improved and updated by MARX: new encryption algorithms and other useful universal security logic will be added);
 - o customer specific functionality can be added to the firmware upon request;
 - o secure remote firmware update (the secure remote update technology is embedded to the CBU SC firmware from the very beginning)
- **Hardware based RSA encryption:**
 - o 512/1024/2048 RSA encryption is supported in hardware;
 - o variable number of RSA keypairs, reprogrammable by customers (a new dedicated memory zone RAM4 is used for RSA key-pairs, by default containing space for 8 keypairs: 3 holding SmarxOS system objects and 5 available for application purposes)
 - o top security and highest level of access control for RSA keypairs:
 - the values can never be read;
 - a special login (UPW/APW) may be required (depending on the access rights preset for this key-pair upon creation) in order to:
 - use the key for encryption;
 - change its value (full keypair);
 - o a special encryption mode fully conforming to the PKCS#1 padding rules (allows to exchange RSA encrypted data with any standard cryptographic provider, like: OpenSSL, Windows CryptoAPI, .NET Cryptography, etc.)
- **Extended hardware implementation of AES encryption:**
 - o While CBU includes only three dedicated AES keys (Fixed, Private and Session), the CBU SC additionally contains a special new memory zone (RAM5) which can securely hold as many extra AES keys as required;
 - o Moreover, besides OFB (output feedback) mode of AES encryption supported by CBU, CBU SC allows to choose between OFB and CBC (cipher block chaining) encryption modes for AES keys stored in RAM5 memory zone;
 - o Top security and highest access control level for AES keys in RAM5 (similar to RSA access control):
 - the values can never be read;
 - a special login (UPW/APW) may be required (depending on the access rights preset for the key upon its creation) in order to:
 - use the key for encryption;
 - change its value.
- **CBU SC is ultra-fast:** up to 100 times faster than CBU, depending on the operation:
 - o approx. 120Kb/sec for read/write memory operations comparing to 2.6/1.3Kb/sec for CBU
 - o approx. 11Kb/sec for hardware based AES encryption comparing to 2.6Kb/sec for CBU
 - o more details on CBU SC timing and its comparison to CBU can be found in Appendix A
- **32KB of fast access internal memory:** approx. 120Kb/sec for read/write memory operations (see Appendix A for more details)

1.2. SmarxOS and CRYPTO-BOX[®] SC

The CBU SC is fully supported by SmarxOS from the very beginning. Being 100% backward compatible with CBU, CBU SC can simply replace it for all existing applications and solutions where CBU is currently used. Only re-compilation (or re-protection in case of AutoCrypt) is required. In fact even this simple replacement will mean: ultra-fast CBU SC with 32KB of RAM. Plus hardware based RSA implementation, which automatically increases security of Remote License Management. The WEB API, OLM (Online License Management) and RU (Remote Update) MARX solutions are built on top of RSA.

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

1.2.1. CRYPTO-BOX[®] SC AES encryption extension

CBU SC AES encryption implementation dramatically extends the number of hardware supported AES keys. In addition to three predefined CBU keys (Session, Private and Fixed), the CBU SC allows customers to create as many AES keys in a special memory zone RAM5 as required and to specify access rights for them.

This new functionality will be utilized by AutoCrypt to protect more than one application with one crypto-box using separate encryption. In case of CBU all protected applications share the same value of AES key for their hardware encryption. For CBU SC it is possible to have separate AES key for every application/set of applications.

In case of API based protection developers will use as many AES keys as necessary by the product licensing logic. Also such MARX technologies as: Document Protection, Data Protection, Media Protection, and WEB based communication will benefit from this functionality, providing higher level of security and customization.

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

1.2.2. Using hardware based RSA

CBU SC brings hardware based RSA and secure storage of RSA key-pairs (RAM4 area). With a special encryption mode fully conforming to the PKCS#1 padding rules the developers will be able to integrate CBU SC hardware based RSA to a wide range of standard and customer specific security solutions.

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

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

- 1) in addition to currently supported driver level RSA implementation CBIOS API 1.5 also supports hardware based RSA
- 2) a special encryption mode fully compatible with PKCS#1 padding rules is supported in addition to the existing RSA support with non-standard padding mode
- 3) CBIOS 1.5 also includes software based RSA for both padding modes (identical to CBU2 hardware implementation) - it allows to consider CBU as RSA key storage for solutions with entry level security requirements
- 4) a new data object - "RSA keypair" will be introduced in DO API helping developers with RSA integration

2. Evaluation: step-by-step

CRYPTO-BOX SC support is integrated to the SmarxOS[®] Protection Kit (PPK)

2.1. Installing CRYPTO-BOX[®] SC driver and SmarxOS[®] COM object

During SmarxOS Protection Kit installation, drivers for CRYPTO-BOX SC and CRYPTO-BOX USB are installed automatically by smart installation utility called CBUSetup.exe, which is included to SmarxOS PPK and available online for re-distribution to end-users. It detects the platform (Windows 32 or 64 bit), installs suitable driver and registers required ActiveX/COM object(s).



We strongly recommend to ship CBUSetup.exe with your software. Please refer to SmarxOS Compendium, chapter 8 for more detailed information on CBUSetup and on distributing your software to your end-users.

When you attach the CBU SC to a USB port of your computer the very first time it will be recognized by Windows as "USB CrypToken II". The driver which was installed with CBUSetup.exe will be found by the system automatically.



When you attach the CBU SC to another USB port of your computer the very first time the system will resolve it automatically.

2.2. Testing

2.2.1. CBU compatibility level

Start evaluation with CBU compatibility level. Launch standard CBIOS test for CBU included to the Kit, for example:

```
< SmarxOS PPK Root > \SmarxOS\API\Win\Samples\CBIOS\C + + \MSVS2005 (Static VC)\Release\
```

and run it in local mode with CBU SC and/or CBU hardware attached to the computer.

You will notice that CBU SC is 100% backward compatible to CBU but tremendously fast.

Now it's time for CBIOS network evaluation. Install CBU SC driver on some other computer (or several computers) on your local network to be used as CBIOS Server(s). See section 2.1 for details.

Copy the contents of the < SmarxOS PPK Root > \SmarxOS\Network\Win\x86\ to those computer(s) and launch CBIOS Server (the CBIOSsrv.exe file). Attach CBU SC (and/or CBU) hardware to those computers with running CBIOS Server.

Now run the sample on your local computer and choose the network mode. You will see that CBU SC is fully compatible with CBIOS network mode.



Your local computer can be used for network mode evaluation too - just launch the CBIOS server on it and attach CBU SC (and/or CBU) hardware.

2.2.2. Testing CBU SC specific features

At this point it makes sense to look through chapters 3 and 4 of this document, describing RSA and AES encryption and all new and extended/updated CBIOS API calls and related data structures currently included to the Evaluation Kit.

Evaluating hardware based RSA and AES encryption

Launch CBU SC specific CBIOS test included to the Kit:

```
< SmarxOS PPK Root > \SmarxOS\API\Win\Samples\CBIOS\C + + \MSVS2005 (Static VC CBU2)\Release\
```

Here you can test CBU SC hardware based RSA and AES encryption.

Provided RSA test scenarios include:

- RSA keypair generation;
- Its programming to CBU SC dedicated RSA memory;
- Performing:
 - hardware based encryption and decryption (option 3);
 - hardware based encryption and software based decryption (option 4);
 - CBU SC hardware RSA and standard WinCrypt compatibility demo (option 6).

AES test scenarios include:

- CBU SC hardware AES (option 5);
- Standard WinCrypt and CBU SC hardware AES compatibility demo (option 7);

All tests can be done in local or network mode. All supplementary CBIOS calls (RSA keypair generation, setting access rights, getting info on RSA keypair, etc.) are demonstrated.

3. CRYPTO-BOX® SC and RSA encryption

3.1. CRYPTO-BOX® USB and RSA encryption

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

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

3.1.1. PKCS#1 padding rules

D: data

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

Encryption

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

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

Decryption

- Public Key decryption (message was encrypted with private key):
 - after decryption checks for block starting with 00 || 01
 - checks for 0xFF padding bytes
 - gets beginning of data from 0x00 separator
- Private Key decryption (message was encrypted with public key):
 - after decryption checks for block starting with 00 || 02
 - gets beginning of data from 0x00 separator

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

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

3.1.2. Padding rules used in CBU RSA implementation

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

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

- Encryption:

00 || 01 || L || P || 00 || D

P: all bytes have value 0xFF

- CBU Decryption

- after decryption checks for block starting with 00 || 01

- gets L

- checks for 0xFF padding bytes

- checks for 0x00 separator

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

3.2. RSA for CBU: Conclusion

As described above, the CBU RSA implementation is not fully compatible to standard RSA implementations conforming to the PKCS#1 padding rules. For CBU2 PKCS#1 padding mode is fully supported, in addition to CBU RSA compatible padding.

For CBU MARX provides RSA implementation sources with CBU compatible padding for such popular environments as Java, PHP, and C#.NET as a part of MARX WEB API solution. These sources are based on standard RSA implementation adjusted for CBU RSA padding rules:

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

The CBU RSA implementation is also fully supported in CBIOS API and can be efficiently used for virtually any server side environment to establish secure encrypted channel with remote client as well as for any other RSA based solutions.

3.3. CRYPTO-BOX[®] SC: hardware RSA implementation

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

For compatibility purposes CBU SC firmware RSA implementation uses CBU padding rules. In addition it also includes standard PKCS#1 padding extending RSA support to any standard scenario. The CBIOS API related calls include a special parameter (dwMode) reserved for defining padding rules: CBU / PKCS#1 modes.

See section 4.2 for more details on CBU SC RSA support in CBIOS API.

4. SmarxOS API: new and extended calls and structures

4.1. Introducing CBIOS 1.5

The CBIOS ver.1.5 is the first CBIOS kernel supporting both CBU and CBU SC. It uses CBU SC specific features and contains serious improvements of CBIOS internal architecture for local and network modes.

4.2. CBIOS 1.5: RSA encryption support

The CBIOS API (ver.1.5) supports all CBU related RSA calls “as is” for both CBU and CBU SC hardware.

4.3. CBIOS 1.5: CBU RSA calls

The following four groups of CBIOS API calls cover existing CBU RSA functionality:

- 1) Software generation and creation of RSA keypairs:

CBIOS_GenerateKeyPairRSA() – generates RSA keypair in computer memory and writes its public and private keys to specified offset in CBU RAM1/2/3 memory;
CBIOS_PrepareRSAKey() – convert externally obtained RSA key to CBU RSA format, can be later stored to CBU memory with **CBIOS_SetKeyPublicRSA()** / **CBIOS_SetKeyPrivateRSA()**

- 2) Reading/writing values of RSA keys to/from CBU memory:

CBIOS_GetKeyPublicRSA() / **CBIOS_GetKeyPrivateRSA()**
CBIOS_SetKeyPublicRSA() / **CBIOS_SetKeyPrivateRSA()**

- 3) CBU RSA data encryption/decryption:

CBIOS_EncryptRSA() / **CBIOS_DecryptRSA()**

- 4) CBU RSA data encryption/decryption using predefined SmarxOS system RSA keypairs (Client/Distributor):

CBIOS_EncryptInternalRSA() / **CBIOS_DecryptInternalRSA()**

As mentioned above this functionality is fully supported for CBU SC. The next two sections address differences in CBU SC interpretation of CBU RSA calls. It's important to mention that these differences only improve RSA calls processing, preserving full compatibility.

4.3.1. CBU SC: hardware support for SmarxOS system RSA keypairs (Distributor and Client)

The first difference between CBU and CBU SC is internal implementation of SmarxOS system RSA keypairs: Distributor and Client. These two predefined RSA keypairs are included to every SmarxOS formatted CBU or CBU SC unit. They are used by MARX WEB API/OLM, Remote Update scenarios, as well as by any customer specific implementations requiring establishing of secure client/server channel.

As mentioned in the previous section CBIOS API includes two calls working with these system SmarxOS keypairs:

CBIOS_EncryptInternalRSA() / **CBIOS_DecryptInternalRSA()**

For CBU SC these calls are based on hardware RSA implementation and SmarxOS system keypairs are stored in specially reserved cells of RAM4 (RSA dedicated memory zone), making impossible any access to their values, other than their usage for encryption/decryption.

4.3.2. CBU SC: ignoring login password parameter for CBU RSA calls

Those CBU RSA CBIOS API calls which work with CBU hardware contain a special parameter (bPass), allowing UPW/APW password submission for this very call. When required, this parameter value is used internally during the call processing for CBU login/logout while accessing its RAM1/2 memory zones storing RSA key and/or data buffer. CBU SC low-level implementation assumes UPW/APW login to be performed in advance (prior to RSA related call) in case if it is required. Thus this parameter value is not needed and is ignored in case of CBU SC.

4.4. CBIOS 1.5: CBU SC RSA calls

Besides CBIOS set of functions representing CBU RSA implementation, CBIOS 1.5 also introduces the following brand new functions covering CBU SC specific RSA functionality:

- 1) Software generation of RSA keypairs for CBU SC:
CBIOS_GenerateKeyPairRSAEx() – generates CBU SC RSA keypair in computer memory
- 2) Writing RSA keypairs to special cells in CBU SC dedicated memory (RAM4):
CBIOS_CBU2_SetKeyRSA()
- 3) Setting/getting/controlling access rights for RSA keys in CBU SC:
CBIOS_CBU2_SetKeyInfoRSA() / **CBIOS_CBU2_GetKeyInfoRSA ()**
CBIOS_CBU2_LockKeyRSA()
- 4) Hardware based RSA data encryption./decryption for CBU SC:
CBIOS_CBU2_EncryptRSA() / **CBIOS_CBU2_DecryptRSA()**
- 5) Software based RSA data encryption/decryption (CBU SC compatible):
CBIOS_EncryptRSAEx() / **CBIOS_DecryptRSAEx()**

The above set of CBIOS calls introduces also two new structures specific to CBU SC: **CBIOS_RSA_KEY**, **CBIOS_RSA_KEY_INFO**. The following subsections describe CBU SC specific RSA calls and related data structures, constants and error codes in detail.

4.4.1. CBIOS API: CBU SC RSA new API calls

4.4.1.1. CBIOS_GenerateKeyPairRSAEx

Generates RSA keypair

```
DWORD WINAPI CBIOS_GenerateKeyPairRSAEx( CBIOS_RSA_KEY* pRSAKeyPair,
                                          CBIOS_RSA_KEY* pRSAPublicKey,
                                          WORD bits,
                                          BYTE* randomPool);
```

pRSAKeyPair	[in] Pointer to a structure receiving generated RSA keypair
pRSAPublicKey	[in] Pointer to a structure receiving public part of generated RSA keypair
bits	[in] RSA key size
randomPool	[in] Array of random bytes. Length of array must be at least CBIOS_RAND_POOL_SIZE(bits)
[return]	CBIOS Error code

Description: This function generates CBU SC RSA keypair of 512/1024/2048 bits size (**bits**) and saves the resulting keypair to the **pRSAKeyPair** structure.

In addition it saves the public part of the keypair separately to the **pRSAPublicKey** structure. The generation is software based and is supposed to be done on trusted computer. If **pRSAPublicKey** is null, then this parameter is ignored.

Further the resulting keypair can be saved to one of the cells in CBU SC dedicated memory (RAM4 zone) with the **CBIOS_CBU2_SetKeyRSA()** call. If necessary its access rights can be adjusted with the **CBIOS_CBU2_SetKeyInfoRSA()** function. Then it can be used for CBU SC RSA encryption: **CBIOS_EncryptRSAEx()/CBIOS_DecryptRSAEx()**.

Example: the **Test_RSA_HW()** function of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.2. CBIOS_CBU2_SetKeyRSA

Writes RSA keypair to one of CBU SC RSA cells

```
DWORD WINAPI CBIOS_CBU2_SetKeyRSA( DWORD dwKeyIndex,
                                   CBIOS_RSA_KEY* pRSAKeyPair,
                                   CBIOS_RSA_KEY_INFO* pRSAKeyInfo );
```

dwKeyIndex	[in] Index of RSA cell in CBU SC RAM4 zone for this keypair (starting from 0)
pRSAKeyPair	[in] Pointer to RSA keypair to write
pRSAKeyInfo	[in] Pointer to RSA keypair information (optional)
[return]	CBIOS Error code

Description: This function writes CBU SC RSA keypair (**pRSAKeyPair**) to one of CBU SC cells (**dwKeyIndex**) in dedicated memory (RAM4 zone).

The **pRSAKeyInfo** structure can optionally set access rights on this keypair, including:

- its usage for encryption,
- obtaining access rights info and/or changing the keypair value/access rights.

See section 4.4.2 for more details on how to define access rights and limitations in **CBIOS_RSA_KEY_INFO** structure.

Example: the `Test_RSA_HW ()` function of `CBU2_Sample.c` (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.3. CBIOS_CBU2_SetKeyInfoRSA

Sets access rights and limitations for RSA keypair stored in CBU SC RSA cell

```
DWORD WINAPI CBIOS_CBU2_SetKeyInfoRSA(    DWORD          dwKeyIndex,
                                           CBIOS_RSA_KEY_INFO*  pRSAKeyInfo );
```

dwKeyIndex	[in] Index of RSA cell in CBU SC RAM4 holding the keypair (starting from 0)
pRSAKeyInfo	[in] Pointer to new RSA keypair information
[return]	CBIOS Error code.

Description: This function changes access rights and limitations for CBU SC RSA cell (**dwKeyIndex**) to values defined in the **pRSAKeyInfo** structure.

It can change access rights for this keypair on: its usage for encryption, obtaining access rights info and/or changing the keypair value/access rights. See section 4.4.2 for more details on how to define access rights and limitations in `CBIOS_RSA_KEY_INFO` structure.

Example: the `Test_RSA_HW ()` function of `CBU2_Sample.c` (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.4. CBIOS_CBU2_GetKeyInfoRSA

Returns access rights and information on RSA keypair stored in CBU SC RSA cell

```
DWORD WINAPI CBIOS_CBU2_GetKeyInfoRSA(    DWORD          dwKeyIndex,
                                           CBIOS_RSA_KEY_INFO*  pRSAKeyInfo );
```

dwKeyIndex	[in] Index of RSA cell in CBU SC RAM4 storing the keypair (starting from 0)
pRSAKeyInfo	[in] Pointer to structure that will receive RSA keypair information
[return]	CBIOS Error code.

Description: This function retrieves access rights and limitations data for CBU SC RSA cell (**dwKeyIndex**) to the **pRSAKeyInfo** structure. See section 4.4.2 for more details on how to define/interpret access rights and limitations in `CBIOS_RSA_KEY_INFO` structure.

Example: the `Test_RSA_HW (void)` function of `CBU2_Sample.c` (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.5. CBIOS_CBU2_LockKeyRSA

Locks specified RSA keypair in one of CBU SC RSA cells

```
DWORD WINAPI CBIOS_CBU2_LockKeyRSA( DWORD dwKeyIndex );
```

dwKeyIndex	[in] Index of CBU SC RSA cell storing the keypair
[return]	CBIOS Error code.

Description: This function locks the CBU SC RSA cell (**dwKeyIndex**), so its keypair can not be used for further RSA encryption/decryption operations.

4.4.1.6. CBIOS_CBU2_EncryptRSA

CBU SC hardware based RSA encryption

```

DWORD WINAPI CBIOS_CBU2_EncryptRSA(  DWORD          dwKeyIndex,
                                     DWORD          dwMode,
                                     PVOID          pInBuffer,
                                     DWORD          dwInBufferLen,
                                     PVOID          pOutBuffer,
                                     DWORD*         pdwOutBufferLen );

```

dwKeyIndex	[in] Index of RSA keypair storage cell in CBU SC RAM4 (starting from 0)
dwMode	[in] Defines key type and encryption padding mode, the value is bitwise combination of: CBIOS_RSA_PUBL_KEY - encrypt with public key CBIOS_RSA_PRIV_KEY - encrypt with private key and CBIOS_RSA_MARX_PADDING - use CBU RSA padding CBIOS_RSA_RSAREF_PADDING - use PKCS#1 compliant padding
pInBuffer	[in] Pointer to input buffer in computer memory (points to byte array to encrypt).
dwInBufferLen	[in] Size in bytes of pInBuffer
pOutBuffer	[in] Pointer to buffer that will receive encrypted data from pInBuffer.
pdwOutBufferLen	[in, out] Pointer to a variable containing the pOutBuffer size (in bytes). On return it contains number of encrypted bytes Or required size of pOutBuffer
[return]	CBIOS Error code.

Description: This function performs CBU SC hardware based RSA encryption using **dwKeyIndex** cell in RAM4 CBU SC memory as RSA key holder.

The **dwMode** parameter defines 1) which part of the keypair should be used: public or private, and 2) RSA padding mode: CBU RSA/PKCS#1.

The input data (byte array **pInBuffer** of length **dwInBufferLen**) will be encrypted and the result of this operation will be placed to the **pOutBuffer** of length defined by **pdwOutBufferLen**. On return the **pdwOutBufferLen** will contain the number of encrypted bytes or required size of pOutBuffer (if size of the pOutBuffer is not enough).

This function should be considered in combination with CBU SC hardware or software based RSA decryption. In other words some data being encrypted with public/private RSA key with this function can be decrypted with private/public RSA key of the same keypair using CBU SC hardware (**CBIOS_CBU2_DecryptRSA** - see 4.4.1.7) or software (**CBIOS_DecryptRSAEx** - see 4.4.1.9) RSA decryption.

Example: The **Test_RSA_HW ()** and **Test_RSA_HW_SW ()** functions of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.7. CBIOS_CBU2_DecryptRSA

CBU SC hardware based RSA decryption

```

DWORD WINAPI CBIOS_CBU2_DecryptRSA(  DWORD          dwKeyIndex,
                                     BYTE          bKeyType,
                                     PVOID          pInBuffer,
                                     DWORD          dwInBufferLen,
                                     PVOID          pOutBuffer,
                                     DWORD*         pdwOutBufferLen );

```

dwKeyIndex	[in] Index of RSA keypair storage cell in CBU SC RAM4 (starting from 0)
dwMode	[in] Defines key type and encryption padding mode, the value is bitwise combination of: CBIOS_RSA_PUBL_KEY - decrypt with public key CBIOS_RSA_PRIV_KEY - decrypt with private key and CBIOS_RSA_MARX_PADDING - use CBU RSA padding CBIOS_RSA_RSAREF_PADDING - use PKCS#1 compliant padding
pInBuffer	[in] Pointer to input buffer in computer memory (points to byte array to decrypt).
dwInBufferLen	[in] Size in bytes of pInBuffer.
pOutBuffer	[in] Pointer to buffer that will receive decrypted data from pInBuffer.
pdwOutBufferLen	[in, out] Pointer to a variable containing the pOutBuffer size (in bytes). On return it contains number of decrypted bytes, or required size of pOutBuffer.
[return]	CBIOS Error code.

Description: This function performs CBU SC hardware based RSA decryption using **dwKeyIndex** cell in RAM4 CBU SC memory as RSA key holder. The **dwMode** parameter defines 1) which part of the keypair should be used: public or private; 2) RSA padding mode: CBU RSA/PKCS#1.

The input data (byte array **pInBuffer** of length **dwInBufferLen**) will be decrypted and the result of this operation will be placed to the **pOutBuffer** of length defined by **pdwOutBufferLen**. On return the **pdwOutBufferLen** will contain the number of decrypted bytes or required size of pOutBuffer (if size of pOutBuffer is not enough).

This function should be considered in combination with CBU SC hardware or software based RSA encryption. In other words some data being decrypted with public/private RSA key with this function can be encrypted with private/public RSA key of the same keypair using CBU SC hardware (**CBIOS_CBU2_EncryptRSA** - see 4.4.1.6) or software (**CBIOS_EncryptRSAEx** - see 4.4.1.8) RSA encryption.

Example: The **Test_RSA_HW ()** and **Test_RSA_HW_SW ()** functions of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.8. CBIOS_EncryptRSAEx

Software based RSA encryption

```

DWORD WINAPI CBIOS_EncryptRSAEx(
    CBIOS_RSA_KEY*
    DWORD
    PVOID
    DWORD
    PVOID
    DWORD*
    pRSAKey,
    dwMode,
    pInBuffer,
    dwInBufferLen,
    pOutBuffer,
    pdwOutBufferLen );
    
```

pRSAKeyPair	[in] Points to RSA keypair (for private encryption) or public key (for public encryption).
dwMode	[in] Defines key type and encryption padding mode, the value is bitwise combination of: For compatibility with CBU SC hardware RSA encryption the key type may be specified explicitly: CBIOS_RSA_PUBL_KEY - encrypt with public key CBIOS_RSA_PRIV_KEY - encrypt with private key and CBIOS_RSA_MARX_PADDING - use CBU RSA padding

	CBIOS_RSA_RSAREF_PADDING - use PKCS#1 compliant padding
pInBuffer	[in] Pointer to input buffer in computer memory (points to byte array to encrypt)
dwInBufferLen	[in] Size in bytes of pInBuffer.
pOutBuffer	[in] Pointer to buffer that will receive encrypted data from pInBuffer
pdwOutBufferLen	[in, out] Pointer to a variable containing the pOutBuffer size (in bytes). On return it contains number of encrypted bytes or required size of pOutBuffer
[return]	CBIOS Error code.

Description: This function performs software based RSA encryption using **pRSAKeyPair** as a pointer to the structure containing RSA private or public key. The software based RSA encryption is 100% compatible with CBU SC hardware based RSA.

The **dwMode** parameter optionally (for compatibility with CBU SC hardware encryption) defines which part of the keypair should be used: public or private. It can also define RSA padding mode: CBU RSA/PKCS#1.

The input data (byte array **pInBuffer** of length **dwInBufferLen**) will be encrypted and the result of this operation will be placed to the **pOutBuffer** of length defined by **pdwOutBufferLen**. On return the **pdwOutBufferLen** will contain the number of encrypted bytes or required size of pOutBuffer (if size of pOutBuffer is not enough).

This function should be considered in combination with CBU SC hardware or software based RSA decryption. In other words some data being encrypted with public/private RSA key with this function can be decrypted with private/public RSA key of the same keypair using CBU SC hardware (**CBIOS_CBU2_DecryptRSA** - see 4.4.1.7) or software (**CBIOS_DecryptRSAEx** - see 4.4.1.9) RSA decryption.

Example: The **Test_RSA_HW ()** and **Test_RSA_HW_SW ()** functions of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.4.1.9. CBIOS_DecryptRSAEx

Software based RSA decryption

```

DWORD WINAPI CBIOS_DecryptRSAEx(
    CBIOS_RSA_KEY*
    DWORD
    PVOID
    DWORD
    PVOID
    DWORD*
    pRSAKey,
    dwMode,
    pInBuffer,
    dwInBufferLen,
    pOutBuffer,
    pdwOutBufferLen );
    
```

pRSAKeyPair	[in] Points to RSA keypair (for private decryption) or public key (for public decryption).
dwMode	[in] Defines key type and decryption padding mode, the value is bitwise combination of: For compatibility with CBU SC hardware RSA encryption the key type may be specified explicitly: CBIOS_RSA_PUBL_KEY - decrypt with public key CBIOS_RSA_PRIV_KEY - decrypt with private key and CBIOS_RSA_MARX_PADDING - use CBU RSA padding CBIOS_RSA_RSAREF_PADDING - use PKCS#1 compliant padding
pInBuffer	[in] Pointer to input buffer in computer memory (points to byte array to decrypt)
dwInBufferLen	[in] Size in bytes of pInBuffer

pOutBuffer	[in] Pointer to buffer that will receive decrypted data from pInBuffer
pdwOutBufferLen	[in, out] Pointer to a variable containing the pOutBuffer size (in bytes). On return it contains number of decrypted bytes or required size of pOutBuffer
[return]	CBIOS Error code

Description: This function performs software based RSA decryption using **pRSAKeyPair** as a pointer to the structure containing RSA private or public key. The software based RSA encryption is 100% compatible with CBU SC hardware based RSA.

The **dwMode** parameter optionally (for compatibility with CBU SC hardware decryption) defines which part of the keypair should be used: public or private. It can also define RSA padding mode: CBU RSA/PKCS#1.

The input data (byte array **pInBuffer** of length **dwInBufferLen**) will be decrypted and the result of this operation will be placed to the **pOutBuffer** of length defined by **pdwOutBufferLen**. On return the **pdwOutBufferLen** will contain the number of decrypted bytes or required size of pOutBuffer (if size of pOutBuffer is not enough).

This function should be considered in combination with CBU SC hardware or software based RSA encryption. In other words some data being decrypted with public/private RSA key with this function can be encrypted with private/public RSA key of the same keypair using CBU SC hardware (**CBIOS_CBU2_EncryptRSA** – see 4.4.1.6) or software (**CBIOS_EncryptRSAEx** – see 4.4.1.8) RSA decryption.

4.4.2. New CBIOS structures related to RSA

4.4.2.1. CBIOS_RSA_KEY

This structure contains RSA keypair or RSA public part.

```
typedef struct {
    DWORD dwStructSize;
    WORD wModulusLen; // in bytes, must be a multiple of four bytes
    WORD wPrivExpLen; // in bytes, must be a multiple of four bytes
    BYTE ubModulus[CBIOS_CBU2_MAX_RSA_KEY_LEN]; // most significant byte first
    BYTE ubPrivExp[CBIOS_CBU2_MAX_RSA_KEY_LEN]; // most significant byte first
} CBIOS_RSA_KEY;
```

dwStructSize	[in] Size of structure in bytes.
wModulusLen	[in,out] RSA keypair modulus length in bytes.
wPrivExpLen	[in,out] RSA keypair exponent length in bytes.
ubModulus	[in,out] Modulus bytes. Most significant byte first.
ubPrivExp	[in,out] Exponent bytes. Most significant byte first.

The **CBIOS_RSA_KEY** structure is used in the following CBIOS calls:

- **CBIOS_GenerateKeyPairRSAEx()**
- **CBIOS_CBU2_SetKeyRSA()**
- **CBIOS_EncryptRSAEx()**
- **CBIOS_DecryptRSAEx()**

4.4.2.2. CBIOS_RSA_KEY_INFO

This structure defines access rights and limitations for CBU SC RSA keypair: its usage for encryption, obtaining access rights info and/or changing the keypair value/access rights.

```
typedef struct {
    DWORD dwStructSize;
    DWORD dwAccess;    // low nibble:    SetKey/SetKeyInfo access rights,
                      // high nibble:    Encrypt/Decrypt and GetKeyInfo
    WORD  wBits;
} CBIOS_RSA_KEY_INFO;
```

dwStructSize	[in] Size of structure in bytes.
dwAccess	[in,out] RSA keypair access flags. Lower 4 bits define access rights for changing RSA keypair (SetKey and SetKeyInfo operations). Higher 4 bits define access rights for performing RSA encryption, decryption and obtaining keypair information. Supported values: CBIOS_CBU2_ACCESS_NEVER CBIOS_CBU2_ACCESS_ALWAYS CBIOS_CBU2_ACCESS_UPW CBIOS_CBU2_ACCESS_APW CBIOS_CBU2_ACCESS_LOCK
wBits	[out] RSA key size in bits (key strength).

Comment: The **dwAccess** member of this structure defines required access rights or returns current access rights for RSA keypair. For any CBU SC RSA keypair access rights can be set for:

- encryption/decryption and obtaining information on this keypair (key strength and current access rights)
- changing the keypair value

Supported values and their meaning:

- 1) **CBIOS_CBU2_ACCESS_NEVER** – this value (if being set) can not be reset even by MARX distributor. MARX distributor can only reprogram the whole RAM4 zone of this CBU SC unit, so all current values of RSA keypairs (if any) will be lost
- 2) **CBIOS_CBU2_ACCESS_ALWAYS** – this value means free access (no limitations).
- 3) **CBIOS_CBU2_ACCESS_UPW** – UPW login is required
- 4) **CBIOS_CBU2_ACCESS_APW** – APW login is required
- 1) **CBIOS_CBU2_ACCESS_LOCK** – the access will be locked, only MARX distributor can unlock it.

The **CBIOS_RSA_KEY_INFO** structure is used in the following CBIOS calls:

- **CBIOS_CBU2_SetKeyRSA()**
- **CBIOS_CBU2_SetKeyInfoRSA()**
- **CBIOS_CBU2_GetKeyInfoRSA()**

4.5. CBIOS 1.5: AES encryption support

The CBIOS API (ver.1.5) supports CBU AES calls “as is” for CBU SC hardware. The CBU SC AES implementation supports two block cipher modes: **CBC** and **OFB** (OFB – is the only encryption mode implemented in CBU firmware).

4.6. CBIOS 1.5: CBU SC AES calls

The following CBIOS API calls cover CBU SC extended AES functionality:

- 1) Writing AES keys to special cells in CBU SC dedicated memory (RAM5):

CBIOS_CBU2_SetKeyAES()

2)Setting/getting/controlling access rights for AES keys in CBU SC RAM5 memory zone:

CBIOS_CBU2_SetKeyInfoAES() / **CBIOS_CBU2_GetKeyInfoAES ()**
CBIOS_CBU2_LockKeyAES()

3)Hardware based AES data encryption./decryption for CBU SC:

CBIOS_CBU2_CryptAES()

The above set of CBIOS calls introduces also two new structures: **CBIOS_AES_KEY**, **CBIOS_AES_KEY_INFO**. Next subsections describe CBU SC specific AES calls and related data structures, constants and error codes in detail.

4.6.1. CBIOS API: CBU SC AES new API calls

4.6.1.1. CBIOS_CBU2_SetKeyAES

Writes AES key to one of CBU SC AES cells

```
DWORD WINAPI CBIOS_CBU2_SetKeyAES (    DWORD          dwKeyIndex,
                                       CBIOS_AES_KEY*   pAESKey,
                                       CBIOS_AES_KEY_INFO * pAESKeyInfo );
```

dwKeyIndex	[in] Index of AES cell in CBU SC RAM5 zone for this key (starting from 0)
pAESKey	[in] Pointer to AES key value in computer memory
pAESKeyInfo	[in] Pointer to a structure with AES key information (optional)
[return]	CBIOS Error code.

Description: This function writes CBU SC AES key value (**pAESKey**) to one of AES key holding cells (**dwKeyIndex**) in dedicated CBU SC memory zone (RAM5).

The **pAESKeyInfo** structure can optionally set access rights for this key, including: its usage for encryption; obtaining access rights info and/or changing the key value/access rights. See section 4.6.2.2 for more details on how to define access rights and limitations using CBIOS_AES_KEY_INFO structure.

Example: the **Test_AES ()** function of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.6.1.2. CBIOS_CBU2_SetKeyInfoAES

Sets access rights and limitations for AES key stored in CBU SC AES cell

```
DWORD WINAPI CBIOS_CBU2_SetKeyInfoAES(    DWORD          dwKeyIndex,
                                           CBIOS_AES_KEY_INFO* pAESKeyInfo );
```

dwKeyIndex	[in] Index of AES cell in CBU SC RAM5 holding the key (starting from 0)
pAESKeyInfo	[in] Pointer to new AES key information
[return]	CBIOS Error code.

Description: This function changes access rights and limitations for CBU SC AES cell (**dwKeyIndex**) to values defined in the **pAESKeyInfo** structure.

It can change access rights for this key on: its usage for encryption, obtaining access rights info and/or changing the key value/access rights. See section 4.6.2.2 for more details on how to define access rights and limitations in CBIOS_AES_KEY_INFO structure.

Example: the **Test_AES ()** function of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.6.1.3. CBIOS_CBU2_GetKeyInfoAES

Returns access rights and information on AES key stored in CBU SC AES cell

```
DWORD WINAPI CBIOS_CBU2_GetKeyInfoAES(    DWORD          dwKeyIndex,
                                           CBIOS_AES_KEY_INFO* pAESKeyInfo );
```

dwKeyIndex	[in] Index of AES cell in CBU SC RAM5 zone storing the key (starting from 0)
pAESKeyInfo	[in] Pointer to the structure that will receive AES key information
[return]	CBIOS Error code.

Description: This function retrieves access rights and limitations data for CBU SC AES cell (**dwKeyIndex**) to the **pAESKeyInfo** structure. See section 4.6.2.2 for more details on how to define/interpret access rights and limitations in CBIOS_AES_KEY_INFO structure.

Example: the **Test_AES (void)** function of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) illustrates this call usage.

4.6.1.4. CBIOS_CBU2_LockKeyAES

Locks specified AES key in one of CBU SC AES cells

```
DWORD WINAPI CBIOS_CBU2_LockKeyAES( DWORD dwKeyIndex );
```

dwKeyIndex	[in] Index of CBU SC AES cell storing the key (starting from 0)
[return]	CBIOS Error code.

Description: This function locks the CBU SC AES cell (**dwKeyIndex**), so this key can not be used for further AES encryption/decryption operations.

4.6.1.5. CBIOS_CBU2_CryptAES

CBU SC hardware based AES encryption

```
DWORD WINAPI CBIOS_CBU2_CryptAES (  DWORD dwKeyIndex,
                                   DWORD dwMode,
                                   PVOID pIV,
                                   PVOID pInBuffer,
                                   PVOID pOutBuffer,
                                   DWORD dwBufferLen );
```

dwKeyIndex	[in] Index of AES key storage cell in CBU SC RAM5 (starting from 0)
dwMode	[in] Defines encryption mode. One of the following values: CBIOS_AES_OFB - Encrypt/decrypt with OFB mode CBIOS_AES_CBC_ENCRYPT - Encrypt with CBC mode CBIOS_AES_CBC_DECRYPT - Decrypt with CBC mode
pIV	Pointer to AES initialization vector value in computer memory
pInBuffer	[in] Pointer to input buffer in computer memory (points to byte array to encrypt).
pOutBuffer	[in] Pointer to a buffer that will receive encrypted data from pInBuffer.
dwBufferLen	[in] Size in bytes of pInBuffer/pOutBuffer .
[return]	CBIOS Error code.

Description: This function performs CBU SC hardware based AES encryption using **dwKeyIndex** cell in RAM5 CBU SC memory as AES key holder.

The **dwMode** parameter defines which mode of encryption/decryption should be used: CBC or OFB.

The input data (byte array **pInBuffer** of length **dwBufferLen**) will be encrypted and the result of this operation will be placed to the **pOutBuffer**.

The AES algorithm logic assumes that encrypted buffer size is always the same as source buffer size. In case of OFB mode usage this size must be multiple of the key size (16 bytes).

Example: The **Test_AES ()** and **Test_AES_SW ()** functions of **CBU2_Sample.c** (MSVS 2005 or 2008 static sample included to the SmarxOS PPK) demonstrate this call usage..

4.6.2. New CBIOS structures related to AES

4.6.2.1. CBIOS_AES_KEY

This structure contains AES key or AES public part.

```
typedef struct {
    DWORD dwStructSize;
    BYTE  ubKey[CBIOS_AES_KEY_LEN];
} CBIOS_AES_KEY;
```

dwStructSize	[in] Size of structure in bytes.
ubKey	[in,out] AES key

The **CBIOS_AES_KEY** structure is used in the following CBIOS calls:

- CBIOS_CBU2_SetKeyAES()
- CBIOS_ryptAESEx()

4.6.2.2. CBIOS_AES_KEY_INFO

This structure defines access rights and limitations for CBU SC AES key: its usage for encryption, obtaining access rights info and/or changing the key value/access rights.

```
typedef struct {
    DWORD dwStructSize;
    DWORD dwAccess;    // low nibble:   SetKey/SetKeyInfo access rights,
                      // high nibble:   Encrypt/Decrypt and GetKeyInfo
    WORD  wBits;
} CBIOS_AES_KEY_INFO;
```

dwStructSize	[in] Size of structure in bytes.
dwAccess	[in,out] AES key access flags. Lower 4 bits define access rights for changing AES key (SetKey and SetKeyInfo operations). Higher 4 bits define access rights for performing AES encryption, decryption and obtaining key information. Supported values: CBIOS_CBU2_ACCESS_NEVER CBIOS_CBU2_ACCESS_ALWAYS CBIOS_CBU2_ACCESS_UPW CBIOS_CBU2_ACCESS_APW CBIOS_CBU2_ACCESS_LOCK
wBits	[out] AES key size in bits (key strength).

Comment: The **dwAccess** member of this structure defines required access rights or returns current access rights for AES key. For any CBU SC AES key access rights can be set for:

- encryption/decryption and obtaining information on this key (key strength and current access rights)
- changing the key value

Supported values and their meaning:

- 1) **CBIOS_CBU2_ACCESS_NEVER** – this value (if being set) can not be reset even by MARX distributor. MARX distributor can only reprogram the whole RAM4 zone of this CBU SC unit, so all current values of AES keys (if any) will be lost

- 2) **CBIOS_CBU2_ACCESS_ALWAYS** – this value means free access (no limitations).
- 3) **CBIOS_CBU2_ACCESS_UPW** – UPW login is required
- 4) **CBIOS_CBU2_ACCESS_APW** – APW login is required
- 5) **CBIOS_CBU2_ACCESS_LOCK** – the access will be locked, only MARX distributor can unlock it.

The **CBIOS_AES_KEY_INFO** structure is used in the following CBIOS calls:

- CBIOS_CBU2_SetKeyAES()**
- CBIOS_CBU2_SetKeyInfoAES()**
- CBIOS_CBU2_GetKeyInfoAES()**

4.7. New CBIOS constants

Box type

New field of **CBIOS_BOX_INFO_ADV** structure (wType)

CBIOS_BOX_UNKNOWN	Unknown CRYPTO-BOX type
CBIOS_BOX_CBU	CRYPTO-BOX USB (CBU)
CBIOS_BOX_CBU2	CRYPTO-BOX SC (CBU SC)

4.8. CBU SC specific CBIOS error codes

CBIOS_CBU2_ERR_CONFIG_LOCKED	Box configuration is locked (Reserved for future usage)
CBIOS_CBU2_ERR_LOAD_FIRMWARE	Firmware upload error (Reserved for future usage)
CBIOS_CBU2_ERR_RAM4_NOT_EXIST	No RAM for RSA keys
CBIOS_CBU2_ERR_RAM5_NOT_EXIST	No RAM for AES keys (Reserved for future usage)
CBIOS_CBU2_ERR_KEY_LOCKED	RSA Key is locked
CBIOS_CBU2_ERR_KEY_NOTUSED	RSA Key is not used (not set)
CBIOS_CBU2_ERR_CRYPTOP_FAILED	Crypto operation failed
CBIOS_CBU2_ERR_NO_SPACE_IN_RAM4	Not enough space in ram 4
CBIOS_CBU2_ERR_NO_SPACE_IN_RAM5	Not enough space in ram 5 (Reserved for future usage)

5. Appendix A: CBU SC and CBU timing

Configuration used for tests:

Processor: AMD Sempron® Processor 3200 + 1815.00 MHz
RAM: 2Gb 333 MHz
Motherboard: ASUS A8N-VM CSM
OS: Windows XP SP3 x86
CBU2 driver: CBUSB2.SYS 1.0.9.421
CBU driver: CBUSB.SYS 2.0.7.214

Timing for: read/write operations

Function	Box Type\ Firmware	Box Size	Memory buffer size						
			100	1K	2K	4K	8K	16K	24K
CBIOS_ReadRAM / WriteRAM	CBU SC 2.2	32K	3/4	11/10	20/18	36/35	68/66	132/130	196/194
	CBU1 2.3		45/85	394/765	778/1512	1546/3007	3082/5997	6155/11976	9227/17954

Conclusions:

- * CBU SC and CBU read/write operation timing does not depend on OS;
- ** CBU SC and CBU read/write operation speed doesn't depend on the buffer size
- *** The read and write operations speed is almost the same for CBU SC

CBU SC: Read/Write: ~ 120 Kb/sec (< +-10%)

CBU: Read/Write: ~ 2.6 / 1.3 Kb/sec (< +-10%)

CBU SC \ CBU Read\Write ratio: 46x\92x

Timing for: symmetric encryption (AES)

Function	Box Type \ Firmware	Box Size	Key	Memory buffer size				
				100	1K	10K	30 K	60K
CBIOS_CryptFixed / Private / Session	CBU SC 2.2	32K	Fixed	11	92	892	2694	5288
	CBU 2.3			45	387	3844	11548	23047
	CBU SC 2.2		Private	12	92	898	2654	5291
	CBU 2.3			45	387	3860	11536	23047
	CBU SC 2.2		Session	12	91	898	2649	5285
	CBU 2.3			45	398	3845	11525	23049
CBIOS_CBU2_CryptAES	CBU SC 2.2	OFB	12	90	882	2642	5283	
		*CBC	25	208	2044	6125	12246	

NOTE: Time is measured for both operations (encrypt + decrypt)

Conclusions:

- * CBU SC and CBU encryption operation timing does not depend on OS;
- ** CBU SC and CBU encryption operation speed doesn't depend on the buffer size
- *** For CBC mode the time value should be divided by 2.

CBU SC: ~ 11 Kb/sec (< +-10%)

CBU: ~ 2.6 Kb/sec (< +-10%)

CBU SC \ CBU symmetric encryption ratio: 4.2x

Timing for: asymmetric encryption (RSA)

Function	Firmware	Box Size	Key	Memory buffer size				
				100	1K	10K	30 K	52K
CBIOS_Encrypt + DecryptRSA	CBU SC 2.2	32K	priv + pub	45	290	2687	8191	13952
	CBU 2.3			165	430	2976	8151	14443
	CBU SC 2.2		pub + priv	43	282	2767	8157	13737
	CBU 2.3			164	431	2861	8380	14588
CBIOS_CBU2_Encrypt + DecryptRSA	CBU SC 2.2		priv + pub	123	1065	10478	31043	54732
			pub + priv	124	1066	10477	31568	54275

Conclusions:

CBU SC hardware based asymmetric encryption operation timing does not depend on OS.

Its value is: ~ **0.95 Kb/sec** (< + -10%)

For CBU software based asymmetric encryption (RSA) implementation depends on OS and system overall productivity. Its value for the system used for tests is: ~ **3.5 Kb/sec**

For asymmetric encryption there is no reason to calculate the ratio value, because of different implementation.

6. Appendix B: Contact, Support and Distributors

USA

MARX CryptoTech LP
3355 Annandale Lane, Suite 2
Suwanee, GA 30024
USA
www.cryptotech.com

Sales: sales@cryptotech.com
Support: support@cryptotech.com
Phone: (+ 1) 770-904-0369
Fax: (+ 1) 770-904-3893
E-Mail: contact@cryptotech.com

Germany

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

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

Italy

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

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

Poland

Microplan Polska Sp. z o.o.
Polwiejska 3
PL-61-885 Poznan
Poland
www.microplan.pl

Sales: Grzegorz Bigos
Phone: + 48 (0) 61 8518916
Fax: + 48 (0) 61 8518774
E-Mail: marx@microplan.pl

0-15Apr011ks(CBU SC Notes for Developers).doc