

CRYPTO-BOX[®] SC

White Paper for Software Developers

Table of Contents

| | |
|----------------------------------------------------------------------------------------------|----|
| Table of Contents | 2 |
| 1 Introduction | 3 |
| 1.1 CRYPTO-BOX® SC: a new generation of MARX USB security hardware | 3 |
| 1.2 SmarxOS and CRYPTO-BOX® SC..... | 3 |
| 1.2.1 CRYPTO-BOX® SC AES encryption extension | 4 |
| 1.2.2 Using hardware based RSA..... | 4 |
| 1.3 The contents of CRYPTO-BOX® SC Evaluation Kit | 4 |
| 2 Evaluation: step-by-step | 5 |
| 2.1 Installing CRYPTO-BOX® SC driver | 5 |
| 2.2 Testing | 5 |
| 2.2.1 CBU compatibility level..... | 5 |
| 2.2.2 Testing CBU SC specific features..... | 6 |
| 3 CRYPTO-BOX® SC and RSA encryption..... | 6 |
| 3.1 CBU and RSA encryption..... | 6 |
| 3.1.1 PKCS#1 padding rules | 6 |
| 3.1.2 Padding rules used in CBU RSA implementation | 7 |
| 3.2 RSA for CBU: Conclusion | 7 |
| 3.3 CRYPTO-BOX® SC: hardware RSA implementation..... | 7 |
| 4 SmarxOS API: new and extended calls and structures..... | 8 |
| 4.1 Introducing CBIOS 1.5..... | 8 |
| 4.2 CBIOS 1.5: RSA encryption support | 8 |
| 4.3 CBIOS 1.5: CBU RSA calls..... | 8 |
| 4.3.1 CBU SC: hardware support for SmarxOS system RSA keypairs (Distributor and Client)..... | 8 |
| 4.3.2 CBU SC: ignoring login password parameter for CBU RSA calls | 8 |
| 4.4 CBIOS 1.5: CBU SC RSA calls | 9 |
| 4.4.1 CBIOS API: CBU SC RSA new API calls | 9 |
| 4.4.2 New structures | 15 |
| 4.4.3 New constants | 16 |
| 4.4.4 Error codes..... | 17 |

1 Introduction

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

CBU SC introduces a brand new generation of MARX USB security hardware. It utilizes 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 smart USB token (protection and licensing device).

The CBU SC firmware provides 100% backward compatibility with the existing 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 memory zone RAM4 is used for RSA key-pairs, by default containing space for 8 keypairs)
 - 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 (will be added soon *)
- **Extended hardware implementation of AES encryption (will be added soon *):**
 - 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 needed;
 - o Top security and highest level for access control 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.
- **CBU2 is ultra-fast:** 10 to 100 times faster than CBU, depending on the operation;
- **32Kb of fast access RAM.**



* A special RSA encryption mode fully conforming to the PKCS#1 padding rules and extended AES encryption are under development at the moment.

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) will be 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. Both 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 some brand new features and options for SmarxOS customers, in particular:

1.2.1 CRYPTO-BOX[®] SCAES encryption extension

CBU SC AES encryption extends the number of hardware supported AES keys. In addition to three predefined CBU keys (Session, Private and Fixed), CBU SC allows customers to create more AES keys in a special memory zone RAM5 and to specify access rights for them. This new functionality will be utilized by AutoCrypt in case when using CBU SC device to protect more than one application.

In case of CBU all protected applications will 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 technologies as:

- Document protection;
- Data Protection;
- Audio and Video protection;
- WEB based communication

will benefit from this functionality, providing higher level of security and customization.

To help developers to incorporate this extended AES functionality to their programs the DO API will include a new 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 use them also for software protection and licensing needs.

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

- 1) in addition to currently supported CBU driver level RSA implementation CBIOS API 1.5 will also support hardware based RSA
- 2) a special encryption mode fully compatible with PKCS#1 padding rules will be supported in addition to the existing RSA non-standard padding mode
- 3) CBIOS 1.5 will also include software based RSA for both padding modes (identical to CBU SC hardware implementation) - it will allow 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

1.3 The contents of CRYPTO-BOX[®] SC Evaluation Kit

The initial version of CBU SC Evaluation Kit contains the following folders:

- **Doc** - with this document
- **Driver** - includes Win32 driver for CBU SC (see 2.1 for the installation info)

- **API** - currently contains two MSVS2005 sample projects for Win32 platform, demonstrating CBIOS (ver.1.5) functionality:
 - **CBU_Classic_Sample** - standard CBIOS API sample; works for CBU and CBU SC hardware
 - **CBU_SC_Sample** - demonstrates CBU SC specific features (at the moment includes hardware based RSA only)
- **Network Server** - includes a special build of CBIOS Network Server for CBIOS networking evaluation, supporting CBU and CBU SC

2 Evaluation: step-by-step

If you received CBU SC Evaluation Kit as a part of SmarxOS PPK, then it is automatically installed together with the PPK and is located in the following folder:

<SmarxOS PPK> \CBU SC Evaluation Kit\

If you received it separately, then copy the Kit to any local folder on your computer.

2.1 Installing CRYPTO-BOX® SC driver

When you attach CBU SC to a USB port of your computer the very first time it will be recognized by Windows as "USB CryptToken II". At this point it is necessary to install CBU SC driver.



As for now only **Win32** version of the driver is available.

Point the "New Hardware Found" wizard to a directory containing the latest CBU SC driver (files: cbusb2.sys, cbusb2.inf):

< CBU SC Evaluation Kit Root > \Driver\

The latest available version as for now is: **CBUSB2 Ver.1.0** (dated 10Dec2008 - 1.0.8.1210). Now the CBU SC hardware is resolved properly (as CBUSB2 device) and ready for evaluation.



When you attach the CBU SC to another USB port of your computer the very first time the system may ask you to resolve the driver once more. This is because of the current version of CBU SC driver is not digitally signed by Microsoft.

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:

< CBU2 Evaluation Kit Root > \API\Win32\MSVS2005\CBIOS\static(VC)\CBU_Classic_Sample\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 and 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 < CBU SC Evaluation Kit Root > \Network Server\Win32\ to those computer(s) and launch CBIOS Server (the CBIOSrv.exe file). Attach CBU SC (and/or CBU) hardware to those computers with running CBIOS Server.

Now run the sample and choose the network mode. You will see that CBU2 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 (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 encryption and all new and extended/updated CBIOS API calls and related data structures currently included to the Evaluation Kit.

Evaluating hardware based RSA encryption

Launch CBU SC specific CBIOS test included to the Kit:

< CBU SC Evaluation Kit Root > \API\Win32\MSVS2005\CBIOS\static(VC)\CBU2_Sample\Release\

Here you can test CBU SC hardware based RSA encryption. Provided test scenarios include:

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

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

3 CRYPTO-BOX® SC and RSA encryption

3.1 CBU and RSA encryption

The RSA encryption for CBU hardware is implemented inside CBU system level software: driver and low-level library. This implementation is supported by SmarxOS and MPI APIs and used in various MARX solutions and technologies, such as: WEB API/OLM and Remote Update.

One problem with the RSA implementation in the CBU driver is that it does not fully conform to the 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

Historically it happened so, that CBU driver uses 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 implementations 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

Although CBU RSA implementation is not fully compatible with standard RSA implementations because of different padding rules, however MARX provides RSA implementation sources with CBU compatible padding for such popular environments as Java, PHP, and C# 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. However 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 CBU2 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 utilizes 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 box. They are used by MARX WEB API/OLM, Remote Update scenarios, as well as by any customer specific implementations requiring 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: **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

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.

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` included to the CBU SC Evaluation Kit illustrates this call usage.

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` included to the CBU2 Evaluation Kit illustrates this call usage.

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 CBU2 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` included to the CBU SC Evaluation Kit illustrates this call usage.

CBIOS_CBU2_GetKeyInfoRSA

Returns access rights and information on RSA keypair stored in CBU2 RSA cell

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

| | |
|--------------------|---------------------------------------------------------------------------|
| dwKeyIndex | [in] Index of RSA cell in CBU2 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** included to the CBU2 Evaluation Kit illustrates this call usage.

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 CBU2 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.

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 CBU2 RAM4 (starting from 0) |
| dwMode | [in] Defines key type: CBIOS_RSA_PUBL_KEY - public key CBIOS_RSA_PRIV_KEY - private key NOTE: in future it will also define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet) |
| pInBuffer | [in] Pointer to input buffer in computer memory (points to byte array to encrypt). |
| dwInBufferLen | [in] Size in bytes of pInBuffer. The size of the data buffer is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to: - 53 bytes for 512 bit RSA; - 117 bytes for 1024 bit RSA; - 245 bytes for 2048 bit RSA |
| 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. Initial value is limited to 512 bytes. |
| [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 which part of the keypair should be used: public or private. In future it will also define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet).

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.

NOTE: The input buffer length is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to:

- 53 bytes for 512 bit RSA;
- 117 bytes for 1024 bit RSA;
- 245 bytes for 2048 bit RSA

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 CBU2 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** included to the CBU2 Evaluation Kit illustrate this call usage.

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 CBU2 RAM4 (starting from 0) |
| dwMode | [in] Key type: CBIOS_RSA_PUBL_KEY - public key CBIOS_RSA_PRIV_KEY - private key NOTE: in future it will also define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet) |
| pInBuffer | [in] Pointer to input buffer in computer memory (points to byte array to decrypt). |
| dwInBufferLen | [in] Size in bytes of pInBuffer. The size of the data buffer is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to: - 53 bytes for 512 bit RSA; - 117 bytes for 1024 bit RSA; - 245 bytes for 2048 bit RSA |
| 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. Initial value is limited to 512 bytes. |
| [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 which part of the keypair should be used: public or private. In future it will also define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet).

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.

NOTE: The input buffer length is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to:

- 53 bytes for 512 bit RSA;
- 117 bytes for 1024 bit RSA;
- 245 bytes for 2048 bit RSA

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** included to the CBU SC Evaluation Kit illustrate this call usage.

CBIOS_EncryptRSAEx

Software based RSA encryption

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

| | |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pRSAKeyPair | [in] Points to RSA keypair (for private encryption) or public key (for public encryption). |
| dwMode | [in] Reserved, in future it will define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet) For compatibility with CBU SC hardware RSA encryption the key type may be specified explicitly: CBIOS_RSA_PUBL_KEY - public key CBIOS_RSA_PRIV_KEY - private key |
| pInBuffer | [in] Pointer to input buffer in computer memory (points to byte array to encrypt) |
| dwInBufferLen | [in] Size in bytes of pInBuffer. The size of the data buffer is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to: - 53 bytes for 512 bit RSA; - 117 bytes for 1024 bit RSA; - 245 bytes for 2048 bit RSA |
| 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. Initial value is limited to 512 bytes. |
| [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 is reserved for future usage - it will define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet). For compatibility with CBU SC hardware encryption it can specify the key type explicitly: public or private.

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.

NOTE: The input buffer length is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to:

- 53 bytes for 512 bit RSA;
- 117 bytes for 1024 bit RSA;
- 245 bytes for 2048 bit RSA

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** included to the CBU SC Evaluation Kit illustrate this call usage.

CBIOS_DecryptRSAEx

Software based RSA decryption

```

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

| | |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pRSAKeyPair | [in] Points to RSA keypair (for private decryption) or public key (for public decryption). |
| dwMode | [in] Reserved, in future it will define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet) For compatibility with CBU SC hardware RSA encryption the key type may be specified explicitly: CBIOS_RSA_PUBL_KEY - public key CBIOS_RSA_PRIV_KEY - private key |
| pInBuffer | [in] Pointer to input buffer in computer memory (points to byte array to decrypt) |
| dwInBufferLen | [in] Size in bytes of pInBuffer. The size of the data buffer is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to: - 53 bytes for 512 bit RSA; - 117 bytes for 1024 bit RSA; - 245 bytes for 2048 bit RSA |
| 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. Initial value is limited to 512 bytes. |

| | |
|----------|-------------------|
| [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 is reserved for future usage - it will define RSA padding mode: CBU RSA/PKCS#1 (PKCS#1 padding is not implemented yet). For compatibility with CBU SC hardware decryption it can specify the key type explicitly: public or private.

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.

NOTE: The input buffer length is limited by the RSA key size (RSA strength) according to the following formula: <Key length in bytes> - 11, i.e. up to:

- 53 bytes for 512 bit RSA;
- 117 bytes for 1024 bit RSA;
- 245 bytes for 2048 bit RSA

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 structures

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()**

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 box, 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()**

4.4.3 New constants

Box type

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

| | |
|--------------------------|------------------|
| CBIOS_BOX_UNKNOWN | Unknown box type |
| CBIOS_BOX_CBU | CBU box |
| CBIOS_BOX_CBU2 | CBU SC box |

4.4.4 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_CRYPTTO_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) |