RFP API Reference (Remote Update API)

Document: 0-24May010ks(CBIOS RFP API Reference).odt
Last update: 1 September 2017 by Steffen Kaetsch
Environment: C++ (Microsoft Visual Studio)

Executive summary

This document describes the RFP API as part of the Smarx API for the CRYPTO-BOX. It provides functions for updating data objects with licensing information stored in CRYPTO-BOX memory on the end-user side.

Table of Contents

1. Smarx®OS Remote Update Technology

1.1. Overview

Smarx OS Remote Update API is designed for updating any set of Data Objects, programmed into a CRYPTO-BOX after automatic protection or implementation with the API. Using Remote Update API, you may extend the expiration period of evaluation or demo versions, change limitations, turn features on/off and even add new licensing options. There are no limitations on how to use Remote Update API: you can update everything you need in any manner you want.

Alternatively to using Smarx OS Remote Update API functions, the SxAF (Smarx Application Framework) or RU_Tool Command Line Utilitiy can be used. For these tools no programming efforts are required. See the RUMS Application Notes for more details.

This document contains the RFP API reference only. Please read chapter 15 in the Smarx Compendium to get an introduction to the Remote Update Technology.

If you need more information first on how to start implementing the CRYPTO-BOX with API: Our White Paper “Implementation with API” provides a general introduction and overview about all available APIs for the CRYPTO-BOX, including the new object oriented Smarx API.

1.2. Remote Update API Calls: detailed description

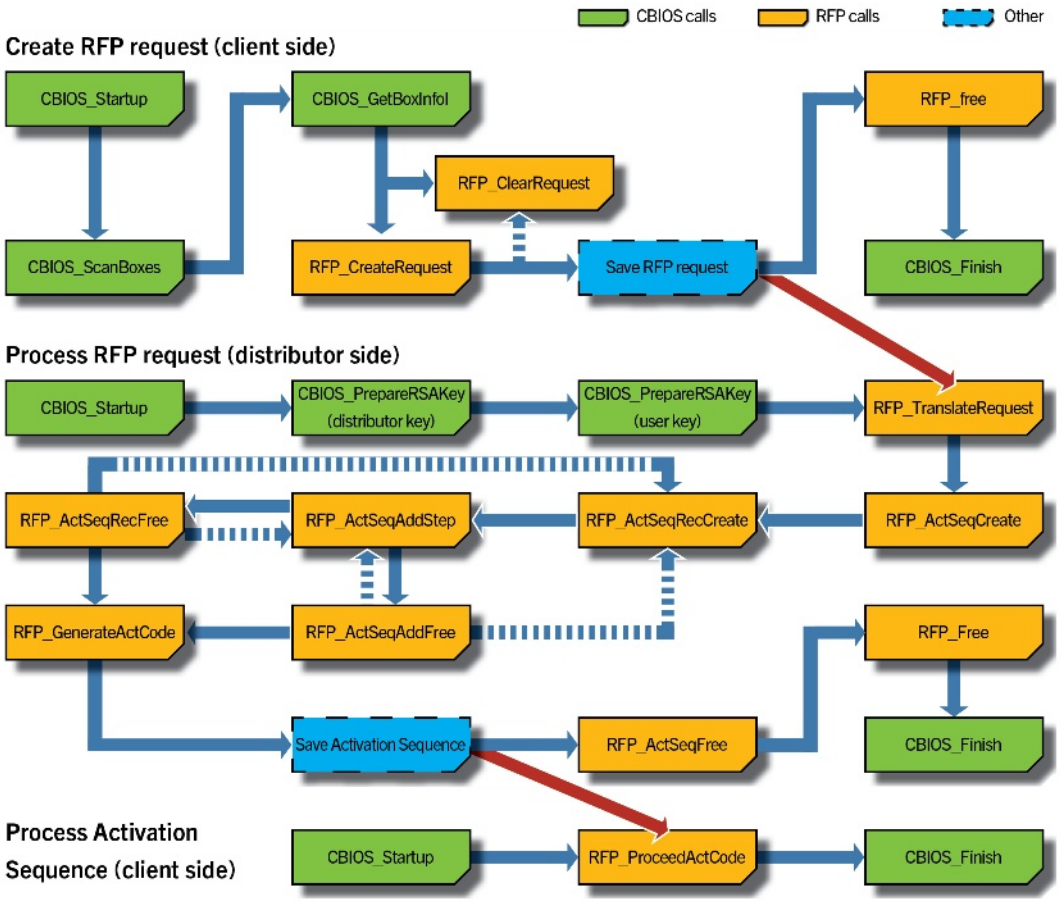

Fig. 1.1:
RFP API calls - overview

RFP_ErrorCode WINAPI RFP_GetVersion (DWORD* pdwMajor, DWORD* pdwMinor);
(end-user side and software vendor side function)

Returns current RFP API version to pdwMajor and pdwMinor out parameters

Parameters:

DWORD* pdwMajor	OUT: Pointer on major version
DWORD* pdwMinor	OUT: Pointer on minor version

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_CreateRequest (DWORD dwBoxName, BYTE ppRequestData, DWORD* pdwRequestDataSize, const RFP_Request* pRequest, DWORD dwMaskNotEncrypted, RFP_PASSW bUPWPass);**
(*end-user side function)*

This generates a Remote Update request and programs the transaction mark to a secure CRYPTO-BOX area. The **RFP_Request** structure field **dwTrMark** is generated as the result of a **RFP_CreateRequest** execution. If **dwTrMark** parameter was set before, its value would be ignored, even if the flag **FRP_REQUEST_MASK_TRANS_MARK** was set.

Parameters:

DWORD dwBoxName	IN: CRYPTO-BOX BoxName
BYTE ** ppRequestData	OUT: Pointer to the output RequestData buffer (allocated dynamically, must be released with **RFP_Free()** function)
DWORD * pdwRequestDataSize	OUT: Pointer to the size of the allocated buffer
const RFP_Request * pRequest	IN: Pointer to Request structure
DWORD dwMaskNotEncrypted	IN: Non-Encrypted parameters mask
RFP_PASSW bUPWPass	IN: User Password

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ClearRequest (DWORD dwBoxName,
RFP_PASSW bUPWPass);
(end-user side function)

This clears Remote Update request and transaction mark in the secure CRYPTO-BOX area.

Parameters:

DWORD dwBoxName	IN: CRYPTO-BOX BoxName
RFP_PASSW bUPWPass	IN: User Password

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_PrepareRequest (RFP_RequestHandle* phRequest //[out]
, const RFP_Request* pRequest //[in]
, DWORD dwMaskNotEncrypted); //[in];
(end-user side function)

This creates Request object and returns its pointer in phRequest out parameter.

Parameters:

RFP_RequestHandle* phRequest	OUT: Pointer to created Request object handle
const RFP_Request * pRequest	IN: Pointer to Request structure
DWORD dwMaskNotEncrypted	IN: Non-Encrypted parameters mask

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_AddRequestParam (RFP_RequestHandle hRequest //[in]
, const GUID* pguidParam //[in]
, BYTE* pParamData //[in]
, DWORD dwParamDataSize //[in]
, BOOL bNotEncrypted); //[in]
(end-user side function)

This adds a parameter to the Request object. Call this function for each parameter you want to add.

Parameters:

RFP_RequestHandle hRequest	IN: Request object handle
const GUID* pguidParam	IN: Identifier (GUID) of parameter.
BYTE* pParamData	IN: Pointer to parameter data memory
DWORD dwParamDataSize	IN: size of parameters data (in bytes)
BOOL bNotEncrypted	IN: If TRUE, the corresponding parameter of Request object is not encrypted and will be available on distributor side (software vendor) before calling RFP_TranslateRequest using corresponding RSA keys.

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_MakeRequest (DWORD dwBoxName //[in]
, RFP_RequestHandle hRequest //[in]
, BYTE ppRequestData //[out]**
, DWORD* pdwRequestDataSize //[out]
, RFP_PASSW bUPWPass); //[in]
(end-user side function)

This repeats RFP_CreateRequest functionality for the object previously prepared by the RFP_PrepareRequest and RFP_AddRequestParam Request functions. For a detailed description see: RFP_CreateRequest function.

Parameters:

DWORD dwBoxName	IN: CRYPTO-BOX BoxName
RFP_RequestHandle hRequest	IN: Handle to Request object
BYTE ** ppRequestData	OUT: Pointer to the output RequestData buffer (allocated dynamically, must be released using the RFP_Free() function)
DWORD * pdwRequestDataSize	OUT: Pointer to the size of the allocated buffer
const RFP_Request * pRequest	IN: Pointer to Request structure
RFP_PASSW bUPWPass	IN: User Password

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_RequestFree (RFP_RequestHandle hRequest);
(end-user side and software vendor side function)

This removes the Request object and releases allocated memory. The function must be called when the Request object is not needed anymore.

Parameters:

RFP_RequestHandle hRequest	IN: Request object handle

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_Free(BYTE* lpMem);
(end-user side and software vendor side function)

This releases the memory allocated for the RFP request.

Parameters:

BYTE * lpMem	IN: Pointer on RequestData

Return:

0	Success
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_TranslateRequest(RFP_Request* pRequest , BYTE* pRequestData, DWORD dwRequestSize, BYTE* pbRSADistribPrivate , BYTE* pbRSAUserPublic);
(software vendor side function)

This translates Remote Update request and obtains non-encrypted data (always) and encrypted data if proper encryption keys are submitted.

Parameters:

RFP_Request * pRequest	OUT: Pointer to Request structure
BYTE * pRequestData	IN: RequestData buffer
DWORD dwRequestSize	IN: Size of RequestData buffer
BYTE * pbRSADistribPrivate	IN: Pointer to Distributor RSA Private Key data. If NULL: Only non encrypted request data are obtained.
BYTE * pbRSAUserPublic	IN: Pointer to end-user RSA Public Key data. If NULL: Only non encrypted request data are obtained.

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_LoadRequest (RFP_RequestHandle* phRequest //[out]
, BYTE* pRequestData //[in]
, DWORD dwRequestSize); //[in]
(software vendor side function)

This loads a request from the binary array to the Request object.

Parameters:

RFP_RequestHandle* phRequest	OUT: Pointer to Request object handle
BYTE * pRequestData	IN: RequestData buffer
DWORD dwRequestSize	IN: Size of RequestData buffer

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_TranslateRequestFromHandle
(RFP_Request* pRequest //[out]
, RFP_RequestHandle hRequest //[in]
, BYTE* pbRSADistribPrivate //[in]
, BYTE* pbRSAUserPublic); //[in]
(software vendor side function)

This function is similar to RFP_TranslateRequest() but gets a Request object instead of a binary array. If this function is called with RSA keys, it decrypts data in Request object. As the Request object is in “decrypted” state, RFP_GetRequestParam calls for encrypted parameters will be successful.

Parameters:

RFP_Request * pRequest	OUT: Pointer to Request structure
RFP_RequestHandle hRequest	IN: Request object handle
BYTE * pbRSADistribPrivate	IN: Pointer to Distributor RSA Private Key data. If NULL: Only non encrypted request data are obtained
BYTE * pbRSAUserPublic	IN: Pointer to User RSA Public Key data. If NULL: only non encrypted request data are obtained

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_GetRequestParam (RFP_RequestHandle hRequest //[in]
, const GUID* pguidParam //[in]
, BYTE pParamData //[in]**
, DWORD* dwParamDataSize); //[in]

(software vendor side function)

This returns the parameters from the Request to the pParamData memory buffer.

Parameters:

RFP_RequestHandle hRequest	IN: Request object handle
const GUID* pguidParam	IN: Identifier (GUID) of requested parameter
BYTE** pParamData	OUT: Pointer to the buffer, where parameter data will be stored. Must be released later using the RFP_Free function
DWORD* dwParamDataSize	OUT: Pointer to the buffer size

Return:

0	Successful
RFP_ERR_PARAM_DOES_NOT_EXISTS	Requested parameter does not exists in the request
RFP_ERR_PARAM_IS_ENCRYPTED	Requested parameter exists but is encrypted. Call RFP_TranslateRequestFromHandle function using corresponding RSA keys to decrypt it.
Other error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqCreate(ActSeqHandle* phActSeq , RFP_TrMark pTrKey, BYTE* pDOMap, DWORD dwDOMapSize);
(software vendor side function)

This creates an activation sequence. An Activation Sequence consists of a set of records, called Activation Sequence Records. Each record corresponds to updates executed for one partition in CRYPTO-BOX memory.

Parameters:

ActSeqHandle * phActSeq	OUT: Pointer to created activation sequence handle
RFP_TrMark pTrKey	IN: Pointer to Transaction Mark structure translated from Request
BYTE * pDOMap	IN: Pointer to DataObjects Map buffer
DWORD dwDOMapSize	IN: Size of DataObjects Map buffer

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqRecCreate(ActSeqRecHandle* phActSeqRec , ActSeqHandle hActSeq, WORD wAppId);
(software vendor side function)

This creates an activation record for the proper application update.

Parameters:

ActSeqRecHandle * phActSeqRec	OUT: Pointer to created activation record handle
ActSeqHandle hActSeq	IN: Activation sequence handle
WORD wAppId	IN: Application (Partition) ID

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqRecCreateEx(ActSeqRecHandle* phActSeqRec , ActSeqHandle hActSeq, WORD wAppId, DWORD dwMode, DWORD dwSizeRAM1, DWORD dwSizeRAM2, DWORD dwSizeRAM3, DWORD dwOldSizeRAM1, DWORD dwOldSizeRAM2, DWORD dwOldSizeRAM3);
(software vendor side function)

This creates an extended activation record. Depending on mode specified this activation record can do application (partition) update, upgrade, creation or deletion.

Parameters:

ActSeqRecHandle * phActSeqRec	OUT: Pointer to created activation record handle
ActSeqHandle hActSeq	IN: Activation sequence handle
WORD wAppId	IN: Application (Partition) ID
DWORD dwMode	IN: Operation mode: – PARTITION_MODE_UPDATE (0x00) Update existing partition - classic mode (equivalent to what RFP_ActSeqRecCreate is doing). – PARTITION_MODE_OVERWRITE (0x01) Overwrite existing partition. Partition will be recreated using new RAM sizes specification. Will not create partition if there were not this partition. – PARTITION_MODE_CREATE (0x02) Create partition. – PARTITION_MODE_DELETE (0x03) Delete partition bitwise summed with Check mode: – PARTITION_MODE_CHECK_FAILS (0x00) Operation will fail if current operation mode can not be applied. E.g. If partition geometry mismatches or If partition already exists and PARTITION_MODE_CREATE mode is specified etc. (see **Use cases** for details). – PARTITION_MODE_CHECK_IGNORE (0x10) Do nothing is specified mode can not be applied. Can be useful, for example, if you want to create partition but not sure if it exists or not and you **do not** want operation to be failed.
DWORD dwSizeRAM1	IN: New RAM1 size. These new (RAM) sizes are used when PARTITION_MODE_OVERWRITE or PARTITION_MODE_CREATE mode specified. Otherwise they will be ignored.
DWORD dwSizeRAM2	IN: New RAM2 size.
DWORD dwSizeRAM3	IN: New RAM3 size.
DWORD dwOldSizeRAM1	IN: Old RAM1 size (existing partition might have). These old RAM sizes are used with all but PARTITION_MODE_CREATE mode. If all of these sizes are set to IGNORE_RAM_SIZE_CHECK (0xFFFF) then size checks will be passed automatically.
DWORD dwOldSizeRAM2	IN: Old RAM2 size (existing partition might have).
DWORD dwOldSizeRAM3	IN: Old RAM3 size (existing partition might have).

Use cases:
dwMode =
(PARTITION_MODE_UPDATE | PARTITION_MODE_CHECK_FAILS) - Classic mode;
(PARTITION_MODE_CREATE | PARTITION_MODE_CHECK_FAILS) - Create new partition, but fail if it already exists;
(PARTITION_MODE_OVERWRITE | PARTITION_MODE_CHECK_IGNORE) - Overwrite existing partition, but do nothing if it does not exist or it's (old) RAM sizes mismatch.
(PARTITION_MODE_ DELETE | PARTITION_MODE_CHECK_FAILS) - Delete existing partition, but fail if it does not exist or it's (old) sizes mismatch. If, nevertheless, all old RAM sizes are set to 0xFFFF (IGNORE_RAM_SIZE_CHECK) partition will be deleted.

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqAddRec(ActSeqHandle hActSeq , ActSeqRecHandle hActSeqRec);
(software vendor side function)

This adds an activation record for application (partition) updates to the activation sequence.

Parameters:

ActSeqHandle hActSeq	IN: Activation sequence handle
ActSeqRecHandle hActSeqRec	IN: Activation record handle

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqFree(ActSeqHandle hActSeq);
(software vendor side function)

This removes the Activation Sequence object and releases occupied memory. The function must be called when the Activation Sequence is not needed anymore.

Parameters:

ActSeqHandle hActSeq	IN: Activation sequence handle

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqRecFree(ActSeqRecHandle hActSeqRec);
(software vendor side function)

This removes an Activation Record object that was created but not added to the Activation

Sequence. If the record was already added to the Activation Sequence, there is no need to call it as the record will be released automatically when **RFP_ActSeqAddRec** is called.

Parameters:

ActSeqRecHandle hActSeqRec	IN: Activation record handle

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ActSeqAddStep(ActSeqRecHandle hActSeqRec , DWORD dwDoID, DWORD dwOperation, BYTE* pData, DWORD dwDataSize , DWORD dwParameter);
(software vendor side function)

This adds one step to the update record (for one DataObject). The parameters **dwOperation**, **pData**, **dwDataSize**, **dwParameter** must receive values according the **TEOS_DoOperation** (DO API) calling rules.

Parameters:

ActSeqRecHandle hActSeqRec	IN: Activation record handle
DWORD dwDoID	IN: DataObjects ID
DWORD dwOperation	IN: DataObjects operation
BYTE * pData	IN: Pointer to DataObject data
DWORD dwDataSize	IN: Size of DataObject data
DWORD dwParameter	IN: Reserved

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_GenerateActCode (BYTE ppActCode , DWORD* pdwActCodeSize, ActSeqHandle hActSeq, BYTE* pbRSADistribPrivate , BYTE* pbRSAUserPublic);**
(software vendor side function)

This generates encrypted activation code, to be sent to the end user.

Parameters:

BYTE ** ppActCode	OUT: Pointer to buffer, where activation code will be stored. Must be released later with **RFP_Free** function.
DWORD * pdwActCodeSize	OUT: Pointer to buffer size
ActSecGandle hActSeq	IN: Activation sequence handle
BYTE * pbRSADistribPrivate	IN: Pointer to Distributor RSA Private Key data
BYTE * pbRSAUserPublic	IN: Pointer to User RSA Public Key data

Return:

0	Successful
error code	See <rfp.h> for details

RFP_ErrorCode WINAPI RFP_ProceedActCode(DWORD dwBoxName , BYTE* pActCode, DWORD dwActCodeSize, RFP_PASSW bAPWPass , RFP_PASSW bUPWPass);
(end-user side function)

This executes the activation code on end user side. As a result, proper DataObjects in the specified partitions will be updated.

Parameters:

DWORD dwBoxname	IN: CRYPTO-BOX BoxName
BYTE * pActCode	IN: Pointer to Activation Code buffer
DWORD dwActCodeSize	IN: Size of buffer
RFP_PASSW bAPWPass	IN: Admin Password. If not needed, NO_PASSWORD must be submitted.
RFP_PASSW bUPWPass	IN: User Password

Return:

0	Successful
error code	See <rfp.h> for details

2. Contact and Support

USA
MARX CryptoTech LP
489 South Hill Street
Buford, GA 30518
USA
www.marx.com

Sales: sales@marx.com
Support: support@marx.com
Phone: (+1) 770-904-0369
Fax: (+1) 678-730-1804
E-Mail: contact@marx.com

Germany
MARX Software Security GmbH
Vohburger Str. 68
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 9295-40
E-Mail: contact-de@marx.com

3. Alphabetical Index

0-24May010ks(CBIOS RFP API Reference).odt

