CryptoServer

Application Interface
(CSAPI)

Office Aachen
Germanusstrasse 4
D-52080 Aachen

Phone: ++49 241-1696-200

Telefax: ++49 241-1696-222

Internet: www.utimaco.de

E-Mail: info.sp@utimaco.de

Document-Number: 2002-0005

Document-Version: 2.0.1

Date: 22nd August 2008

State of Release: released

Author: Dipl. Inf. Rainer Herbertz

Table of Contents

1 Introduction

In this document the application interface of Utimaco's hardware security module **CryptoServer** is described. To get an overview about the CryptoServer's security architecture and its complete software architecture, read [CSSSA] and [CSSWA] before.

The **CryptoServer Application Interface (CSAPI)** is a programming interface to access the external functions of the CryptoServer security module from an application running on a host. With the CSAPI it is possible either to access a CryptoServer plugged in a slot of the local computer or to access a CryptoServer LAN over TCP/IP. The CSAPI is available for the programming language C.

The CSAPI can use different logical protocols with a variable number of protocol layers to communicate with the CryptoServer. The protocol stack can be configured by the application. For this reason the CSAPI consists of a protocol independent part and a couple of C modules, one for every implemented protocol layer.

1.1 Change History

Document Version	*CSAPI Version*	*Date*	*Description of Changes*
1.0.0	0.9.5	15-11-2002	Initial released version, approved by ZKA 18-12-2002
1.0.1	1.0.0	08-01-2003	• chapter 1.1 "Change History" initialized • chapter 2.9: new function code "CSA_IOC_GET_INFOX"
1.1.0	1.1.0	10-04-2003	• Secure messaging layer in chapter 3.4.
1.1.1	1.1.12	28-05-2004	• Chapter 2.9: new function codes "CSA_IOC_SET/GET_CTIMEOUT"
1.1.2	1.2.4	17-11-2004	• Chapter 3.4: field 'id' expanded to 2 byte, field 'Flags' introduced.
1.1.3	1.2.6	05-01-2005	• Chapter 2.9: CSA_AUTH_CSL structure and CSA_IOC_PING_(ON/OFF) introduced
1.1.4	1.3.0	06-07-2005	• Chapter 3.4: MAC calculation of secure messaging changed.
1.2.0	1.4.11	30-06-2006	• name of hardware security module changed from "CryptoServer 2000" or "CS2" to "CryptoServer" or "CryptoServer CS", and from "CSLAN" to "CryptoServer LAN"
2.0.0	1.4.16	22-06-2007	• Description of command blocks (byte streams) moved to CMDS documentation, see [CSCMDS]. • Changed authentication mechanism "RSA Signature" in section 3.2.1. • New authentication mechanisms "HMAC Password" and "ECDSA Signature" in sections 3.2.5 and 3.2.6. • New mechanism for secure messaging (AES session key) in section 3.4

2.0.1	1.6.3	22-08-2008	• New function codes for `cs_ioctl()` in section 2.9.

2 Core Functions of the CSAPI for C

The CSAPI for the programming language C consists of the following parts:

- The header file '`csapi.h`' that must be included in the application program which shall run on the host.
- One header file '`csa_xxx.h`' for every layer of the protocol stack, where *xxx* is the name of the protocol layer, see chapter 3.
- The library that must be linked to the application program.

There are different versions of the library:

- `csapi.a` and `csapi.so` for Linux / Solaris
- `csapi.lib` and `csapi.dll` for Windows

The C functions of the CSAPI are described in the following sections.

2.1 Function 'cs_open()'

This function opens a connection to a CryptoServer.

```
#include <csapi.h>

long cs_open (char   *device,
              int    *p_handle)
```

Parameters:

Name	I/O	Description
device	I	Pointer to an ASCII C string that contains the address of the CryptoServer. The address may be a local device or a LAN address, see below.
p_handle	O	Pointer to an integer that receives a handle to the open connection. This handle must be passed to all other CSAPI functions.

Examples for possible addresses of the CryptoServer (parameter 'device'):

Address	Description
PCI:/dev/cryptoserver0	Local CryptoServer in a Linux system.
PCI:0	Local CryptoServer in a Windows system.
TCP:kslan01	Hostname of a CryptoServer LAN
TCP:194.168.4.107	IP-Address of a CryptoServer LAN
TCP:57@kslan01	Host name and port number of a CryptoServer LAN
TCP:57@194.168.4.107	IP-Address and port number of a CryptoServer LAN
UDP:4711@kslan02	Host name and port number of a CryptoServer LAN using a UDP connection
PIPE:/etc/ks-pipe	UNIX stream pipe (with name /etc/ks-pipe)

The prefix "TCP:" is the default and can be omitted. If the string begins with a "/", "PCI:" is assumed.

A timeout for opening TCP connections can be set by:

```
cs_ioctl(0,CSA_IOC_SET_CTIMEOUT,timeout);
```

before calling cs_open(). If no timeout is set, an operating system default timeout is used.

The function returns 0 on success or an error code, see 2.11.

2.2 Function 'cs_close()'

This function closes a connection to the CryptoServer.

```
#include <csapi.h>

void cs_close (int  handle)
```

Parameters:

Name	I/O	Description
handle	I	Handle to an open connection that was returned by a `cs_open()` call.

2.3 Functions 'cs_push_xxx()'

Each of this functions pushes a protocol layer onto the protocol stack ('xxx' is the name of the protocol layer). One or more protocol layers must be pushed onto the protocol stack after a `cs_open()` call and before data can be transmitted to the CryptoServer.[1]

All possible protocol layers are explained in the following chapter 3.

```
#include <csapi.h>
#include <csa_xxx.h>

long cs_push_xxx (int  handle)
```

Parameters:

Name	I/O	Description
`handle`	I	Handle to an open connection that was returned by a `cs_open()` call.

The function returns 0 on success or an error code, see 2.11.

If more than one protocol layer is used for a connection to the CryptoServer, the order of the different `cs_push_xxx()` calls is important. The top level layer must be pushed last.
***Example:** If the protocol layer CMDS is used, the call to `cs_push_cmds()` must be done after all other `cs_push_xxx()` calls.*

Every wanted protocol layer has to be pushed only once, even if this layer will be used twice or more within some CryptoServer commands.
***Example:** The AUTH layer should be used twice for all CryptoServer commands that have to be authenticated according to the 2-persons-rule. But nevertheless the AUTH layer has only to be pushed once on the protocol stack, i. e. `cs_push_auth()` has only once to be performed.*

[1] This has to be done only once at the beginning after the `cs_open()` call and has not to be repeated before every `cs_send()` or `cs_exec()` command.

2.4 Functions 'cs_pop_all()'

This functions removes all protocol layers from the protocol stack that were pushed with the `cs_push_xxx()` functions.

```
#include <csapi.h>

long cs_pop_all (int  handle)
```

Parameters:

Name	I/O	Description
`handle`	I	Handle to an open connection that was returned by a `cs_open()` call.

The function returns 0 on success or an error code, see 2.11.

2.5 Function 'cs_send()'

This function sends a data block to the CryptoServer. First a byte string is created by calling a constructor function from every protocol layer that was pushed onto the protocol stack. One or more protocol layers must be supplied with appropriate parameters. Then this byte string is transmitted to the CryptoServer. All possible parameters for the specific protocol layer XXX are explained in the following chapter 3.

```
#include <csapi.h>

long cs_send (int   handle,
              void  *p_param)
```

Parameters:

Name	I/O	Description
handle	I	Handle to an open connection that was returned by a cs_open() call.
p_param	I	Linked list of CSA_PARAM_*XXX* structures, see below. The order of the structures in the list does not care.

```
typedef struct
{
    int   tag;         // tag CSA_TAG_XXX for the specific layer
    void  *p_next;     // pointer to the next CSA_PARAM_YYY structure
                       // or NULL
    ...                // more specific parameters for protocol layer
                       // XXX here, see the chapter "Usage of the
                       // protocal layer XXX"
} CSA_PARAM_XXX;
```

For every protocol layer *XXX* there may be one structure in the list with specific parameters for this layer. The tag field of the parameter structure for layer *XXX* must contain the value CSA_TAG_*XXX* to be recognized.

The presence of a parameter structure may be optional or required depending on the needs of the specific protocol layer:

- *If the CMDS layer was pushed, the CMDS parameter structure is mandatory.*
- *If the AUTH layer was pushed, the AUTH parameter structure is optional (depending on the specific command) and can also be used more than once.* ***Example:*** *If a specific command has to be authenticated according to the 2-person's rule, two CSA_PARAM_AUTH structures can/must be given.*
- *If the CHNL layer was pushed, the CHNL parameter structure is optional.*
- *If the SM layer was pushed, the SM parameter structure is optional.*
- *If the BL layer was pushed, the BL parameter structure is mandatory.*

The structure CSA_PARAM_*XXX* and the tag value CSA_TAG_*XXX* are defined in the header file csa_*xxx*.h.

The function returns 0 on success or an error code, see 2.11.

2.6 Function 'cs_recv()'

This function receives a data block from the CryptoServer. The received byte string is analyzed by a function from every protocol layer that was pushed onto the protocol stack. Relevant data is extracted from the byte string and returned to the application via the parameter structures. Some protocol layers must be supplied with appropriate parameters. All possible data returned by a specific protocol layer are explained in the following chapter 3.

The function allocates memory for the byte string received from the CryptoServer. This memory must be freed by the application via the `cs_free()` function call (see 2.8) after the application has proceeded the data. For that reason the top level protocol layer contains a pointer '`p_answ`' that can be given to the `cs_free()` function. If the function `cs_recv()` returns an error code, no memory is allocated.

```
#include <csapi.h>

long cs_recv (int   handle,
              void  *p_param)
```

Parameters:

Name	I/O	Description
`handle`	I	Handle to an open connection that was returned by a `cs_open()` call.
`p_param`	I/O	Linked list of `CSA_PARAM_`*`XXX`* structures. The order of the structures in the list does not care.

```
typedef struct
{
    int   tag;             // tag CSA_TAG_XXX for the specific layer
    void  *p_next;         // pointer to the next CSA_PARAM_YYY
    ...                    // structure or NULL
    unsigned char *p_answ; // answer data (top level layer only)
    ...                    // more specific parameters for protocol
                           // layer XXX here, see the chapter "Usage
                           // of the protocal layer XXX"
} CSA_PARAM_XXX;
```

For every protocol layer *XXX* there may be one structure in the list with specific parameters for this layer, see also the explanations in 2.5. The `tag` field of the parameter structure for layer *XXX* must contain the value `CSA_TAG_`*`XXX`* to be recognized. The presence of a parameter structure may be optional or required depending on the needs of the specific protocol layer. The structure `CSA_PARAM_`*`XXX`* and the tag value `CSA_TAG_`*`XXX`* are defined in the header file `csa_xxx.h`.

The function returns 0 on success or an error code, see 2.11.

2.7 Function 'cs_exec()'

This function sends a data block of an external command to the CryptoServer and receives the answer. This is done by calling the functions `cs_send()` and `cs_recv()` one after the other. See the description of the functions `cs_send()` and `cs_recv()` above.

```
#include <csapi.h>

long cs_exec (int   handle,
              void  *p_param)
```

Parameters:

Name	I/O	Description
`handle`	I	Handle to an open connection that was returned by a `cs_open()` call.
`p_param`	I/O	Linked list of `CSA_PARAM_`*`XXX`* structures, see 2.5 and 2.6 (description of the commands `cs_send()` and `cs_recv()`). The order of the structures in the list does not care.

For every protocol layer *`XXX`* there may be one structure in the list with specific parameters for this layer. The `tag` field of the parameter structure for layer *`XXX`* must contain the value `CSA_TAG_`*`XXX`* to be recognized. The presence of a parameter structure may be optional or required depending on the needs of the specific protocol layer, see 2.5. The structure `CSA_PARAM_`*`XXX`* and the tag value `CSA_TAG_`*`XXX`* are defined in the header file `csa_`*`xxx`*`.h`.

The function returns 0 on success or an error code, see 2.11.

2.8 Function 'cs_free()'

This function is used to free memory allocated by a `cs_recv()` or `cs_exec()` function call.

```
#include <csapi.h>

void cs_free (void *p_mem)
```

Parameters:

Name	I/O	Description
`p_mem`	I	Reference to the allocated memory returned by the top level layer of the protocol stack. The pointer is found in '`p_answ`' of the structures CSA_PARAM_CMDS, CSA_PARAM_BL, CSA_PARAM_DUMY and CSA_PARAM_KS.

2.9 Function 'cs_ioctl()'

This function is used to set specific parameters or execute special action on an open connection to a CryptoServer.

```
#include <csapi.h>

long cs_ioctl (int   handle,
               int   function,
               void  *p_param)
```

Structure for CSA_IOC_GET_INFOX:

```
typedef struct
{
    unsigned int  len;      // (I/O) size of buffer / length of data
    unsigned char *buff;    // (O) pointer to buffer
} CSA_INFOX;
```

Structure for CSA_IOC_RESET, CSA_IOC_RESET_TO_BL and CSA_IOC_RESTART:

```
typedef struct
{
    unsigned int magic;
    char *password;                  // password for authentication
    int (*calc_auth)(
          char          *passwd,     //I: Password / Keyfile / device string
          int           l_ch,        //I: length of challenge
          unsigned char *p_ch,       //I: challenge
          int           sfc,         //I: SFC for hash calculation
          int           l_cmd,       //I: length of command data
          unsigned char *p_cmd,      //I: command data for hash calculation
          int           *p_l_sec,    //I/O: sizeof buffer p_sec /
                                     //     length of calculated hash / sign
          unsigned char *p_sec       //O: calculated hash / signature
         )
} CSA_AUTH_CSL;
```

Parameters:

Name	I/O	Description
handle	I	Handle to an open connection that was returned by a cs_open() call. May be 0, if one of the functions CSA_IOC_SET/GET_CTIMEOUT or CSA_IOC_PING_ON/ CSA_IOC_PING_OFF is used (global setting).
function	I	Specifies the function to be performed, see below.
p_param	I/O	Pointer to parameter(s) for the function or pointer to a location where information returned by the function is stored.

Known functions are:

Function	Type of Parameter	Description
CSA_IOC_RESET	CSA_AUTH_CSL (I) (optional, see below)	Perform reset of the CryptoServer device.
CSA_IOC_RESET_TO_BL	CSA_AUTH_CSL (I) (optional, see below)	Perform reset of the CryptoServer device and make boot loader ready.
CSA_IOC_RESTART	CSA_AUTH_CSL (I) (optional, see below)	Perform reset of the CryptoServer device and boot up CryptoServer.
CSA_IOC_PING_ON	none	Send a dummy command inside a cs_open to a CryptoServer LAN (default)
CSA_IOC_PING_OFF	none	Don not send a dummy command inside a cs_open to a CryptoServer LAN (for backward capability of the KryptoServer LAN)
CSA_IOC_SET_TIMEOUT	int timeout (I)	Set timeout (in milliseconds)
CSA_IOC_GET_TIMEOUT	int timeout (O)	Get timeout (in milliseconds)
CSA_IOC_SET_CTIMEOUT	int timeout (I)	Set timeout (in milliseconds) for opening an IP connection to the CryptoServer LAN (global setting, handle not used).
CSA_IOC_GET_CTIMEOUT	int timeout (O)	Get timeout (in milliseconds) for opening an IP connection to the CryptoServer LAN (global setting, handle not used).
CSA_IOC_MSG_MODE	int mode (I/O)	Set message mode (only for CryptoServer LAN) mode = 0: write messages into local log file mode = 1: discard messages mode = 2: forward messages to the application.
CSA_IOC_GET_INFOX	CSA_INFOX *info (I/O)	Return driver information (state) as ASCII text.
CSA_IOC_WAIT_CMDS_READY	none	Wait until CMDS has initialized
CSA_IOC_START_OS	none	Start OS and wait until CMDS gets read
CSA_IOC_GET_MODEL	int model (O)	Returns CryptoServer model: 1: CryptoServer 2000 + CryptoServer CS 2: CryptoServer Se

Remarks to CSA_IOC_RESET, CSA_IOC_RESET_TO_BL and CSA_IOC_RESTART

If these commands are sent over a TCP connection to a CryptoServer LAN the optional parameter CSA_AUTH_CSL can be passed. The usage of this parameter depends on the configuration of the CryptoServer LAN (see [CSLAN] for details). The parameter must provide pointers to a

- password – The plaintext password used for authentication against the CryptoServer LAN (root password) or a specifier for the RSA key for signature authentication.
- calc_auth() – A pointer to a function that calculates the authentication token for the CryptoServer LAN

If the configuration of the CryptoServer LAN does not require authentication for these commands, the parameter mustn't be passed and can be set to NULL.

The function returns 0 on success or an error code, see 2.11.

2.10 Function 'cs_set_msg_handler()'

The CryptoServer can output debug or fatal error messages via the PCI interface. Normally these messages are discarded for a locally installed CryptoServer or written to a log file in a CryptoServer LAN. This function is used to register a function in the host application that receives all the messages.

```
#include <csapi.h>

long cs_set_msg_handler (int   handle,
                         void  (*p_msg)(unsigned char *, int))
```

Parameters:

Name	I/O	Description
`handle`	I	Handle to an open connection that was returned by a `cs_open()` call.
`p_msg`	I	Pointer to a function that is called for every message sent from the CryptoServer. A pointer to the message data and the size of the message data is given as parameters to the function. If `p_msg` is NULL the message handling is set back to the old behavior.

The function returns 0 on success or an error code, see 2.11.

2.11 Error Codes of the Core Functions

The core functions of the CSAPI can produce the following error codes:

Error	Value	Description
E_CSA_CORE_NO_LEN	0xB9000000	Can't determine length of data in bottom level protocol layer.
E_CSA_CORE_HANDLE	0xB9000001	Invalid handle.
E_CSA_CORE_INVAL	0xB9000002	Invalid argument.
E_CSA_CORE_MEM	0xB9000003	Can't allocate memory.
E_CSA_CORE_STACK	0xB9000004	Malformed protocol stack.
E_CSA_CORE_SIZE	0xB9000005	Data block too big.
E_CSA_CORE_TOO_MANY	0xB9000006	Too many open connections
E_CSA_CORE_BLK_LEN	0xB9000009	can't calculate block length
E_CSA_CORE_EMPTY	0xB900000A	empty command block
E_CSA_CORE_BAD_ANSW	0xB900000B	malformed answer block from CryptoServer LAN
E_CSA_CORE_V24_CTRL	0xB900000C	can't set V24 device
E_CSA_CORE_NO_V24	0xB900000D	V24 mode not activated
E_CSA_CORE_V24_CRC	0xB900000E	V24 CRC error on read
E_CSA_CORE_FMT_LEN	0xB9000010	bad length within format string (scanf)
E_CSA_CORE_BAD_CMD	0xB9000011	bad format of command block
E_CSA_CORE_BAD_OUT	0xB9000012	bad parameter structure (scanf)
E_CSA_CORE_BAD_FMT	0xB9000013	bad format string (scanf)

The UNIX specific part of CSAPI can produce the following error codes:

Error	Value	Description
E_CSA_LX_PATH	0xB9010001	Path name too long.
E_CSA_LX_PORT	0xB9010002	Bad port number.
E_CSA_LX_ADDR	0xB9010003	Bad IP address.
E_CSA_LX_HOSTNAME	0xB9010004	Bad host name.
E_CSA_LX_TERM	0xB9010005	Connection terminated by remote host.
E_CSA_LX_MEM	0xB9010006	Can’t allocate memory.
E_CSA_LX_TIMEOUT	0xB9010007	Timeout occurred.
E_CSA_LX_INVAL	0xB9010008	Invalid argument.
E_CSA_LX_ADDRLEN	0xB9010009	No space for sockaddress (internal error).
E_CSA_LX_BLKSIZE	0xB901000A	Bad block size received.
E_CSA_LX_NOT_RDY	0xB901000B	no ready message from CMDS
E_CSA_LX_CRIT_TEMP	0xB901000C	CryptoSever exceeds critical temperature
E_CSA_LX_PROC	0xB901000D	error on /proc file
E_CSA_LX_DEV	0xB901000E	can't stat device file
E_CSA_LX_OPEN	0xB9011xxx	Can’t open device (xxx = system ‘errno’)

Error	Value	Description
E_CSA_LX_SOCKET	0xB9012xxx	Can't create socket (xxx = system 'errno')
E_CSA_LX_CONNECT	0xB9013xxx	Can't get connection (xxx = system 'errno')
E_CSA_LX_POLL	0xB9014xxx	Error while polling (xxx = system 'errno')
E_CSA_LX_READ	0xB9015xxx	Read error (xxx = system 'errno')
E_CSA_LX_WRITE	0xB9016xxx	Write error (xxx = system 'errno')
E_CSA_LX_IOCTL	0xB9017xxx	Error on ioctl call (xxx = system 'errno')
E_CSA_LX_LOCK	0xB9018xxx	Error on ioctl locking call (xxx = system 'errno')
E_CSA_LX_RECV	0xB9019xxx	tcp receive error
E_CSA_LX_SEND	0xB901Axxx	tcp send error

The UNIX error code 'xxx' can be found in '/usr/include/errno.h' in UNIX systems. Additional error codes from the device driver are:

Error	Value	Description
ECS2TOOMANY	0x700	too many CryptoServer devices found
ECS2TIMEOUT	0x701	no response from CryptoServer
ECS2BADLEN	0x702	bad length
ECS2NACK	0x703	command rejected by CryptoServer
ECS2NOREQ	0x704	no request pending
ECS2STATE	0x705	unknown state
ECS2RESET	0x706	interrupted by reset
ECS2TEMPALARM	0x707	high temperature
ECS2TXCRC	0x708	transmit CRC error
ECS2RXCRC	0x709	receive CRC error
ECS2DOWN	0x70A	CryptoServer is down
ECS2PANIC	0x70B	CS2 issued panic message
ECS2NOCRC	0x70C	no CRC in CS answer
ECS2BADPLEN	0x70D	bad packet length
ECS2BADSEQ	0x70E	bad sequence number
ECS2PCITX	0x71x	PCI error while transmit (x = PCI Chip error code)
ECS2PCIRX	0x72x	PCI error while receive (x = PCI Chip error code)
ECS2NORST	0x73x	reset failed (x = CryptoServer reset phase)

The WINDOWS specific part of CSAPI can produce the following error codes:

Error	Value	Description
E_CSA_WIN_PATH	0xB9020001	path name too long
E_CSA_WIN_PORT	0xB9020002	bad port number
E_CSA_WIN_ADDR	0xB9020003	bad IP address
E_CSA_WIN_HOSTNAME	0xB9020004	bad host name
E_CSA_WIN_TERM	0xB9020005	connection terminated by remote host
E_CSA_WIN_MEM	0xB9020006	can't allocate memory

Error	Value	Description
E_CSA_WIN_TIMEOUT	0xB9020007	timeout occurred
E_CSA_WIN_INVAL	0xB9020008	invalid argument
E_CSA_WIN_ADDRLEN	0xB9020009	no space for sockaddress (internal error)
E_CSA_WIN_BLKSIZE	0xB902000A	bad block size received
E_CSA_WIN_CMDS_NOT_RDY	0xB902000B	no ready message from CMDS
E_CSA_WIN_CRIT_TEMP	0xB902000C	CryptoServer exceeds critical temperature
E_CSA_WIN_DCI_INVALID_PARAM	0xB9020010	DCI: invalid parameter
E_CSA_WIN_DCI_INVALID_HANDLE	0xB9020011	DCI: invalid handle value
E_CSA_WIN_DCI_CREATE_MUTEX	0xB9020013	DCI: error creating mutex
E_CSA_WIN_DCI_LOCK	0xB9020014	DCI: unable to set lock
E_CSA_WIN_DCI_LOCK_TIMEOUT	0xB9020015	DCI: timeout while waiting for mutex
E_CSA_WIN_LOCK_HANDLE	0xB9020016	no valid mutex object
E_CSA_WIN_OPEN	0xB90201xx	TCP: can't open device
E_CSA_WIN_SOCKET	0xB90202xx	TCP: can't create socket
E_CSA_WIN_CONNECT	0xB90203xx	TCP: can't get connection
E_CSA_WIN_POLL	0xB90204xx	TCP: error while polling
E_CSA_WIN_READ	0xB90205xx	TCP: read error
E_CSA_WIN_WRITE	0xB90206xx	TCP: write error
E_CSA_WIN_INIT	0xB90207xx	tcp: init error
E_CSA_WIN_IOCTL	0xB90208xx	tcp: ioctl error
E_CSA_WIN_ECS2NACK	0xB9020703	command rejected by CS2
E_CSA_WIN_ECS2NOREQ	0xB9020704	no request pending
E_CSA_WIN_ECS2PCITX	0xB9020710	PCI error while transmit
E_CSA_WIN_ECS2PCIRX	0xB9020720	PCI error while receive
E_CSA_WIN_INIT	0xB90207xx	TCP: init error
E_CSA_WIN_IOCTL	0xB90208xx	TCP: I/O control error
E_CSA_WIN_TCP_CREATE_MUTEX	0xB9020900	TCP: error creating mutex
E_CSA_WIN_TCP_LOCK	0xB9020901	TCP: unable to set lock
E_CSA_WIN_DCI_OPEN	0xB9021xxx	DCI: can't open device
E_CSA_WIN_DCI_READ_RLEN	0xB9022001	DCI: read returned wrong length
E_CSA_WIN_DCI_READ_TMOUT	0xB90220B5	DCI: read timeout
E_CSA_WIN_DCI_READ	0xB9022xxx	DCI: read error
E_CSA_WIN_DCI_WRITE_RLEN	0xB9023001	DCI: write returned wrong length
E_CSA_WIN_DCI_WRITE_TMOUT	0xB90230B5	DCI: write timeout
E_CSA_WIN_DCI_WRITE	0xB9023xxx	DCI: write error
E_CSA_WIN_DCI_IOCTL_TMOUT	0xB90240B5	DCI: I/O control timeout
E_CSA_WIN_DCI_IOCTL	0xB9024xxx	DCI: I/O control error

The Windows error codes for the TCP-errors ‘xx’ can be found in ‘winsock.h’, supplied by Microsoft in their software development kits.

3 Description of the Protocol Layers

This chapter describes the specific protocol layers that can be used with the CSAPI. These are:

- CMDS – always the top level layer on the protocol stack for normal CryptoServer operations (CryptoServer in state 'operational', i. e. operating system SMOS is running)
- AUTH – mandatory for all commands that have to be authenticated
- CHNL – optional
- SM – optional
- BL – used for communication with the boot loader of the CryptoServer (transport layer of the boot loader, see also [CSBL])

3.1 Protocol Layer 'CMDS'

Required push function (must be executed as last push function if more than one protocol layer is used):

```
cs_push_cmds()
```

Header files to be included in the application:

```
#include <csapi.h>
#include <csa_cmds.h>
```

Parameter structure of this layer CMDS:

```
typedef struct
{
    int             tag;
    void            *p_next;
    int             fc;
    int             sfc;
    int             l_cmd;
    unsigned char   *p_cmd;
    unsigned char   *p_answ;
    int             l_answ;
    long            error;
} CSA_PARAM_CMDS;
```

Description of the parameters that have to be set before the `cs_send()` function is called:

Member	Description
`tag`	Constant value `CSA_TAG_CMDS`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Function Code of the module of the requested function (module ID). If `fc` == -1, then a zero length command block is created and the parameters '`sfc`', '`l_cmd`' and '`p_cmd`' are ignored.
`sfc`	Number of the requested function (Sub Function Code).
`l_cmd`	Length of the data block (pure command data) that should be sent to the CryptoServer.

Member	Description
p_cmd	Pointer to the data block (pure command data) that should be sent to the CryptoServer.

Description of the parameters that are set after a successful call to the `cs_recv()` or `cs_exec()` function:

Member	Description
p_answ	Pointer to the data block (pure answer data) that was received from the CryptoServer.
l_answ	Length of the data block (pure answer data) that was received from the CryptoServer.
error	Error code returned by the CryptoServer or 0, if no error occurred.

If the return value of the function `cs_recv()/cs_exec()` was 0, memory was allocated by this function and must be freed by the application, even if '`l_answ`' is 0 or '`error`' is not 0. The memory is freed by giving the pointer '`p_answ`' to the function `cs_free()`.

3.1.1 Error Codes of the Protocol Layer 'CMDS'

Error	Value	Description
E_CSA_CMDS_ALEN	0xB9000200	Length error of answer block
E_CSA_CMDS_CLEN	0xB9000201	Length error of command data
E_CSA_CMDS_PARAM	0xB9000202	Missing parameter structure
E_CSA_CMDS_TAG	0xB9000203	Bad tag of answer block

3.2 Protocol Layer 'AUTH'

The protocol layer 'AUTH' is used to authenticate a CryptoServer command or to login and logoff into the CryptoServer. Therefore AUTH uses the user table which is administrated by the CMDS module, see [CSCMDS], where for each user a name, his permissions and specific data for the authentication mechanism is stored (e. g. the password). AUTH supports several **authentication mechanisms**:

- Clear Password,
- SHA-1 Hashed Password,
- HMAC Password,
- RSA Signature (according to [PKCS#1]) and
- RSA Smart Card Signature,
- ECDSA Signature

see [CSCMDS].

All mechanisms except the "Clear Password" and the "RSA Smart Card Signature" mechanisms work with a challenge-response mechanism.[2] For this, the 'Get Challenge' command block format must be used to request the challenge before the authentication can be done.

The AUTH layer can be used several times for one command (e. g. twice if authentication according to the 2-persons-rule is necessary for a specific command).

Required push function:

```
cs_push_auth()
```

Header files to be included in the application:

```
#include <csapi.h>
#include <csa_auth.h>
```

Parameter structure CSA_PARAM_AUTH of this layer:

```
typedef struct
{
    int             tag;
    void            *p_next;
    int             fc;
    int             mech;
    char            user[8];
    long            perm;
    unsigned char   login;
    unsigned char   ch_id;
    unsigned char   challenge[8];
```

[2] The "RSA Smart Card Signature" authentication mechanism works internally (inside the CryptoServer respectively inside the smart card) also with a challenge/response mechanism, but the CSAPI is not affected from this.

```
union
{
    struct
    {
        int    l_sign;
        int    (*p_get_size)()
        int    (*p_sign)()
        void   *p_data;
        int    (*p_sign)(void         *p_prvdata,
                         unsigned int  l_data,
                         unsigned char *p_data,
                         unsigned int  l_challenge,
                         unsigned char *p_challenge,
                         unsigned int  *p_len,
                         unsigned char *p_sign);
    } rsa_sign;

    struct
    {
        unsigned char  passwd[16];
    } clr_pwd;

    struct
    {
        unsigned char  passwd[16];
        int            (*p_hash)(unsigned int  l_data,
                                 unsigned int  *p_data,
                                 unsigned int  l_challenge,
                                 unsigned int  *p_challenge,
                                 unsigned int  l_passwd,
                                 unsigned int  *p_passwd,
                                 unsigned char *p_hash);
    } sha1_pwd;

    struct
    {
    } rsa_sc;

    struct
    {
        int            halgo;
        unsigned int   hsize;
        unsigned int   l_passwd;
        unsigned char  *p_passwd;
        int            (*p_hmac)(int           halgo,
                                 unsigned int  l_data,
                                 unsigned int  *p_data,
                                 unsigned int  l_challenge,
                                 unsigned int  *p_challenge,
                                 unsigned int  l_passwd,
                                 unsigned int  *p_passwd,
                                 unsigned char *p_hmac);
    } hmac_pwd;
```

```
            struct
            {
                int    l_sign_max;
                int    (*p_sign)(void         *p_prvdata,
                                 unsigned int  l_data,
                                 unsigned char *p_data,
                                 unsigned int  l_challenge,
                                 unsigned char *p_challenge,
                                 unsigned int  *p_len,
                                 unsigned char *p_sign);
                void   *p_data;
            } ecdsa;
        } u;
    } CSA_PARAM_AUTH;
```

For the CSAPI, the usage of a parameter structure `CSA_PARAM_AUTH` is optional, i. e. the parameter structure must not be given in a `cs_send()` or `cs_exec()` command even if the AUTH layer was pushed onto the protocol stack. If an authentication is mandatory for a specific CryptoServer command (according the specification of the respective firmware module) and the authentication is missing, then the CryptoServer will return an error.

In the following subchapters the parameters will be explained which have to be set before a specific AUTH-command can be performed. The other parameters will not be used then.

3.2.1 Authentication with RSA Signature

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_RSA_SIGN.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login
`ch_id`	Challenge ID from the last “Get Challenge” response.
`challenge`	Challenge data from the last “Get Challenge” response.
`rsa_sign.l_sign`	Maximum size of the signature in bytes.
`rsa_sign. p_get_size`	Obsolete. Must be set to NULL.
`rsa_sign.p_sign`	Obsolete. Must be set to NULL.
`p_data`	Pointer that is transferred to the function ‘p_sign2’ (first argument). May be used arbitrarily by the application, to store session specific data (e. g. about a smart card session).
`rsa_sign. p_sign2`	Pointer to a function that calculates the PKCS#1 signature (according to [PKCS#1]) with the SHA-1 hash value from “`data || challenge`” and stores it into ‘`p_sign`’. The byte stream `data` is the command block of the upper level protocol layer(s) and will be constructed automatically by the AUTH layer from the parameter structures of the upper level layers. The function must return the actual size of the calculated signature via the ‘`p_len`’ parameter. The function must return 0 on success or an error code.

3.2.2 Authentication with Clear Password

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_CLR_PWD.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login
`clr_pwd.passwd`	Password of the user.

3.2.3 Authentication with SHA-1 Hashed Password

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_SHA1_PWD.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login
`ch_id`	Challenge ID from the last “Get Challenge” response.
`challenge`	Challenge data from the last “Get Challenge” response.
`sha1_pwd.passwd`	Password of the user.
`sha1_pwd.p_hash`	Pointer to a function that calculates the SHA-1 hash value from “`data || challenge || passwd`” and stores it into ‘`p_hash`’. The byte stream `data` is the command block of the upper level protocol layer(s) and will be constructed automatically by the AUTH layer from the parameter structures of the upper level layers. The function must return 0 on success or an error code.

3.2.4 Authentication with RSA Smart Card

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_RSA_SC.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login

3.2.5 Authentication with HMAC Password

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_HMAC_PWD.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login
`ch_id`	Challenge ID from the last “Get Challenge” response.
`challenge`	Challenge data from the last “Get Challenge” response.
`hmac_pwd.halgo`	Specifies the hash algorithm used by the HMAC calculation.
`hmac_pwd.hsize`	Size of the HMAC to be calculated in bytes.
`hmac_pwd.l_passwd`	Length of the password in bytes.
`hmac_pwd.p_passwd`	Pointer to the password of the user. The password must be located in a static memory area and must be valid until the call to the `cs_send()` or `cs_exec()` function returns.
`hmac_pwd.p_hmac`	Pointer to a function that calculates the HMAC value over “`data || challenge`” using “`passwd`” as a key and stores it into ‘`p_hmac`’. The byte stream `data` is the command block of the upper level protocol layer(s) and will be constructed automatically by the AUTH layer from the parameter structures of the upper level layers. The function must return 0 on success or an error code.

3.2.6 Authentication with ECDSA Signature

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_AUTHENTICATE.
`mech`	Constant value CSA_AUTH_MECH_ECDSA.
`user`	Name of the user who should authenticate the command or wants to be logged in (as given in the user table).
`login`	0 = single command authentication, 1 = user login
`ch_id`	Challenge ID from the last “Get Challenge” response.
`challenge`	Challenge data from the last “Get Challenge” response.
`rsa_sign. l_sign_max`	Maximum size of the signature in bytes.
`rsa_sign.p_sign`	Pointer to a function that calculates the ECDSA signature with the hash value from “`data || challenge`” and stores it into ‘`p_sign`’. The byte stream `data` is the command block of the upper level protocol layer(s) and will be constructed automatically by the AUTH layer from the parameter structures of the upper level layers. The function must return the actual size of the calculated signature via the ‘`p_len`’ parameter. The function must return 0 on success or an error code.
`p_data`	Pointer that is transferred to the function ‘p_sign’ (first argument). May be used arbitrarily by the application, to store session specific data (e. g. about a smart card session).

3.2.7 Get Challenge

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_GET_CHALLENGE.

Description of the parameters that are set after a successful call to the `cs_recv()` function:

Member	Description
`ch_id`	Challenge ID.
`challenge`	Challenge data.

3.2.8 Logoff a User

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_LOGOFF_USER.
`user`	Name of the user who should be logged off.

3.2.9 Logoff a Group of Users

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_LOGOFF_GROUP.
`perm`	Integer that contains 8 nibbles, each must be 0_H or F_H The high order nibble corresponds to the user group 7, the low order nibble to user group 0. If a nibble is F_H, all users who belong to the corresponding group are logged off. **Example:** 0x00F000F0 logs off all users that belong to user groups 1 or 5.

3.2.10 Logoff All Users

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_AUTH`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`fc`	Constant value CSA_AUTH_CMD_LOGOFF_ALL.

3.2.11 Error Codes of the Protocol Layer 'AUTH'

Error	Value	Description
E_CSA_AUTH_ALEN	0xB9000400	Length error of answer block.
E_CSA_AUTH_BAD_FC	0xB9000401	Invalid function code (parameter 'fc' in structure CSA_PARAM_AUTH).
E_CSA_AUTH_BAD_ANSW	0xB9000402	Malformed answer block.
E_CSA_AUTH_BAD_MECH	0xB9000403	Invalid authentication mechanism.
E_CSA_AUTH_HASH_ERR	0xB9000404	Error in hash function.
E_CSA_AUTH_SIGN_ERR	0xB9000405	Error in signature function.
E_CSA_AUTH_HMAC_ERR	0xB9000406	error in HMAC function

3.3 Protocol Layer 'CHNL'

The protocol layer 'CHNL' is a simple format to add a channel number to a CryptoServer command.

Required push function:

```
cs_push_chnl()
```

Header files to be included in the application:

```
#include <csapi.h>
#include <csa_chnl.h>
```

Parameter structure `CSA_PARAM_CHNL` of this layer:

```
typedef struct
{
    int             tag;
    void            *p_next;
    long            channel;
} CSA_PARAM_CHNL;
```

The usage of a parameter structure `CSA_PARAM_CHNL` is optional, i. e. the parameter structure must not be given in a `cs_send()` or `cs_exec()` command even if the CHNL layer was pushed onto the protocol stack. If the structure is not present, the CHNL layer will set a channel number '0' automatically.

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_CHNL`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`channel`	Channel number. Can be freely assigned by the application.

Description of the parameters that are set after a successful call to the `cs_recv()` or `cs_exec()` function:

Member	Description
`channel`	Channel number, echoed by the CryptoServer.

3.3.1 Error Codes of the Protocol Layer 'CHNL'

Error	Value	Description
`E_CSA_CHNL_ALEN`	`0xB9000300`	Length error of answer block.
`E_CSA_CHNL_TAG`	`0xB9000301`	Bad tag of answer block.

3.4 Protocol Layer 'SM'

The protocol layer 'SM' is used for Secure Messaging between a CryptoServer and the host. A session key must be requested from the CryptoServer with the CMDS command "Get Session Key" (see [CSCMDS]) before Secure Messaging can be used. With an active SM layer every command and answer block sent to / received from the CryptoServer is encrypted and protected with a MAC. SM currently supports two mechanisms for Secure Messaging with either a 16 bytes Triple-DES key or a 32 bytes AES key.

Required push function:

```
cs_push_sm()
```

Header files to be included in the application:

```
#include <csapi.h>
#include <csa_sm.h>
```

Parameter structure CSA_PARAM_SM of this layer:

```
typedef struct
{
    int            tag;
    void           *p_next;
    int            mech;
    unsigned int   id;
    unsigned char  seq_ct[16];
    union
    {
        struct                      // DES encryption
        {
            unsigned char key[16];
            int    (*p_crypt)(
                    int key_len,          // in bytes
                    unsigned char *key,   // key (16 Bytes)
                    int mode,             // 1 = encrypt, 0 = decrypt
                    unsigned char *iv,    // input and output IV
                    int data_len,         // multiple of 8
                    unsigned char *data, // input data
                    unsigned char *res); // output data
            int    (*p_mac)(
                    int key_len,          // in bytes
                    unsigned char *key,   // key (16 Bytes)
                    unsigned char *iv,    // input and output IV
                    int data_len,         // multiple of 8
                    unsigned char *data, // input data
                    unsigned char *mac); // output
         } des_encr;
```

```
            struct                          // AES encryption
            {
                unsigned char   key[32];
                unsigned char   mkey[32];
                int     (*p_crypt)(
                         int key_len,           // in bytes
                         unsigned char *key,    // key (32 Bytes)
                         int mode,              // 1 = encrypt, 0 = decrypt
                         unsigned char *iv,     // input and output IV
                         int data_len,          // multiple of 16
                         unsigned char *data,   // input data
                         unsigned char *res);   // output data
                int     (*p_mac)(
                         int key_len,           // in bytes
                         unsigned char *key,    // key (32 Bytes)
                         unsigned char *iv,     // input IV
                         int data_len,          // multiple of 16
                         unsigned char *data,   // input data
                         unsigned char *res);   // output MAC
            } aes_encr;

        } u;
    } CSA_PARAM_SM;
```

For the CSAPI, the usage of a parameter structure `CSA_PARAM_SM` is optional, i. e. the parameter structure must not be given in a `cs_send()` or `cs_exec()` command even if the SM layer was pushed onto the protocol stack.

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_SM`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`mech`	Value CSA_SM_MECH_3DES_ENCM for DES mechanism or CSA_SM_MECH_AES for AES mechanism.
`id`	Session ID returned by the CMDS function “Get Session Key”, see [CSCMDS]. The flag CSA_SM_FLAG_END_SESSION may be added to ‘id’. In this case the session key in invalidated after execution of the command.
`seq_ct`	Initial sequence counter returned by the CMDS function “Get Session Key”, see [CSCMDS].
`des_encr.key`	Triple DES session key (length 16 bytes) returned by the CMDS function “Get Session Key”, see [CSCMDS].
`des_encr.p_crypt`	Pointer to a Triple DES CBC encryption / decryption function.
`des_encr.p_mac`	Pointer to a DES MAC calculation function.

Member	Description
aes_encr.key	AES session key (length 32 bytes) returned by the CMDS function "Get Session Key", see [CSCMDS].
aes_encr.mkey	AES MAC key, derived from the AES session key, see [CSCMDS].
aes_encr.p_crypt	Pointer to an AES CBC encryption / decryption function.
aes_encr.p_mac	Pointer to an AES MAC calculation function.

3.4.1 Error Codes of the Protocol Layer 'SM'

Error	Value	Description
E_CSA_SM_ALEN	0xB9000600	Length error of answer block.
E_CSA_SM_BAD_ANSW	0xB9000601	Malformed answer block
E_CSA_SM_BAD_MECH	0xB9000602	Invalid SM mechanism
E_CSA_SM_NO_DATA	0xB9000603	Zero length data
E_CSA_SM_DES_ERR	0xB9000604	en- / decryption / MAC error
E_CSA_SM_UNWRAP	0xB9000605	Secure Messaging unwrap error

3.5 Protocol Layer 'BL'

The protocol layer 'BL' is a simplified version of the layer 'CMDS' that is used for communication with the boot loader of the CryptoServer (transport layer of the boot loader, see [CSBL]).[3] It must be the top level layer on the protocol stack if the boot loader is running (and thus other firmware modules are not yet active).

Required push function:

```
cs_push_bl()
```

Header files to be included in the application:

```
#include <csapi.h>
#include <csa_bl.h>
```

Parameter structure of this layer:

```
typedef struct
{
    int             tag;
    void            *p_next;
    int             l_cmd;
    unsigned char   *p_cmd;
    unsigned char   *p_answ;
    int             l_answ;
} CSA_PARAM_BL;
```

Description of the parameters that have to be set before a call to the `cs_send()` or `cs_exec()` function:

Member	Description
`tag`	Constant value `CSA_TAG_CMDS`.
`p_next`	Pointer to the parameter structure of the next protocol layer or NULL.
`l_cmd`	Length of the data block that should be sent to the CryptoServer.
`p_cmd`	Pointer to the data block that should be sent to the CryptoServer.

Description of the parameters that are set after a successful call to the `cs_recv()`or `cs_exec()` function:

Member	Description
`p_answ`	Pointer to the data block that was received from the CryptoServer.
`l_answ`	Length of the data block that was received from the CryptoServer.

[3] To identify the boot loader, the Function Code (module ID) in this "simplified CMDS version" is set to 0x0007. The SFC is not set.

If the return value of the function `cs_recv()` was 0, memory was allocated by this function and must be freed by the application, even if '`l_answ`' is 0. The memory is freed by giving the pointer '`p_answ`' to the function `cs_free()`, see also section 2.8.

3.5.1 Error Codes of the Protocol Layer 'BL'

Error	Value	Description
E_CSA_BL_ALEN	0xB9000500	Length error of answer block.
E_CSA_BL_CLEN	0xB9000501	Bad length error of command data.
E_CSA_BL_PARAM	0xB9000502	Missing parameter structure.
E_CSA_BL_TAG	0xB9000503	Bad tag of answer block.

4 Sample Program Using 'CMDS' Layer

```
#include <stdio.h>
#include "csapi.h"
#include "csa_cmds.h"

int main()
{
  int cs_handle;
  long err;
  CSA_PARAM_CMDS cs_para;

/* connect to the CryptoServer */

  if((err = cs_open("288@192.168.4.207",&cs_handle)) != 0) {
        printf("Can't connect to CryptoServer !, error %lx\n",err);
        return(1);
        }
  if((err = cs_push_cmds(cs_handle)) != 0) {
        printf("Can't push protocol layer !, error %lx\n",err);
        return(1);
        }

/* Execute command "Get time" */

  cs_para.tag = CSA_CMDS_TAG;
  cs_para.p_next = NULL;
  cs_para.fc = 0x87;
  cs_para.sfc = 6;
  cs_para.l_cmd = 0;                    /* command block is empty */
  cs_para.p_cmd = NULL;
  if((ret = cs_exec(cs_handle,&cs_para)) != 0) {
        printf("Communication error to CryptoServer (%lx)\n",err);
        return(1);
        }

  if(cs_para.error != 0) {
        printf("Error %08x occured\n",cs_para.error);
        cs_free(cs_para.p_answ);
        return(1);
        }

if(cs_para.l_answ != 18) {
        printf("Unexpected length of answer block\n");
        cs_free(cs_para.p_answ);
        return(1);
        }

  /* Output time (answer block contains date and time in ASCII */

  printf("CryptoServer Time: %.18s\n",cs_para.p_answ);

  cs_free(cs_para.p_answ);
  cs_close(cs_handle);

  return(0);
}
```

5 References

	Title / Company	Doc.-No.
[CSADM]	CryptoServer – Firmware Module ADM – Interface Specification / Utimaco Safeware AG	2002-0004
[CSBL]	CryptoServer CS – Boot Loader Design Specification / Utimaco Safeware AG	2002-0007
[CSCMDS]	CryptoServer – Firmware Module CMDS – Interface Specification / Utimaco Safeware AG	2002-0008
[CSSMOS]	CryptoServer – Operating System SMOS – Design Specification / Utimaco Safeware AG	2002-0016
[CSSSA]	CryptoServer CS – System Security Architecture / Utimaco Safeware AG	2000-0027
[CSSWA]	CryptoServer CS – Software Architecture / Utimaco Safeware AG	2000-0018
[CSLAN]	CryptoServer LAN – Operating Manual / Utimaco Safeware AG	2003-0002
[PKCS#1]	PKCS#1: RSA Encryption Standard v1.5, November 1993 / RSA Laboratories, http://www.rsasecurity.com/rsalabs/pkcs	